Mobile Application Development

Sub Code : 22617

EDITION : 2020

- SIMPLIFIED & CONCEPTUAL APPROACH

INCLUDES-I SCHEME PATTERN
SAMPLE PAPERS

MSBTE I SCHEME PATTERN
T. Y. DIPLOMA SEM VI
COMPUTER ENGINEERING PROGRAM GROUP
(CO/CM/IF/CW)

TECHNICAL

Vrushali R. Sonar

SUBJECT CODE : 22617

As per Revised Syllabus of

MSBTE - I SCHEME

T.Y. Diploma Semester - VI
Computer Engineering Program Group
(CO / CM / IF / CW)

MOBILE APPLICATION DEVELOPMENT

Vrushali R. Sonar
M.E. (IT), B.E. (Computer),
Lecturer, AISSMS's Polytechnic,
Pune.

Narendra S. Joshi
Ph.D. (Pursuing), M.E. (CSE),
SOCSE Department
Assistant Professor,
Sandip University, Nashik.

Aniruddha D. Talole
M.E. (Computer Engg.), B.E. (Computer),
Lecturer(Computer Technology Department),
K. K. Wagh Polytechnic, Nashik.

TECHNICAL PUBLICATIONS
SINCE 1993
An Up-Thrust for Knowledge

MOBILE APPLICATION DEVELOPMENT

Subject Code : 22617

T.Y. Diploma Semester - VI
Computer Engineering Program Group (CO / CM / IF / CW)

First Edition : January 2020

Published by :

TECHNICAL PUBLICATIONS
An Up-Thrust for Knowledge

Amit Residency, Office No.1, 412, Shaniwar Peth, Pune - 411030, M.S. INDIA
Ph.: +91-020-24495496/97, Telefax : +91-020-24495497
Email : sales@technicalpublications.org Website : www.technicalpublications.org

Printer :
Yogiraj Printers & Binders
Sr.No. 10/1A,
Ghule Industrial Estate, Nanded Village Road,
Tal. - Haveli, Dist. - Pune - 411041.

Price : ₹ 200/-

ISBN 978-93-89750-07-2

MSBTE I

PREFACE

The importance of **Mobile Application Development** is well known in various engineering fields. Overwhelming response to our books on various subjects inspired us to write this book. The book is structured to cover the key aspects of the subject **Mobile Application Development**.

The book uses plain, lucid language to explain fundamentals of this subject. The book provides logical method of explaining various complicated concepts and stepwise methods to explain the important topics. Each chapter is well supported with necessary illustrations, practical examples and solved problems. All chapters in this book are arranged in a proper sequence that permits each topic to build upon earlier studies. All care has been taken to make students comfortable in understanding the basic concepts of this subject.

The book not only covers the entire scope of the subject but explains the philosophy of the subject. This makes the understanding of this subject more clear and makes it more interesting. The book will be very useful not only to the students but also to the subject teachers. The students have to omit nothing and possibly have to cover nothing more.

We wish to express my profound thanks to all those who helped in making this book a reality. Much needed moral support and encouragement is provided on numerous occasions by our whole family. We wish to thank the **Publisher** and the entire team of **Technical Publications** who have taken immense pain to get this book in time with quality printing.

Any suggestion for the improvement of the book will be acknowledged and well appreciated.

Authors

Vrushali R. Sonar
Narendra S. Joshi
Aniruddha D. Talole

Dedicated to Readers of Book.

SYLLABUS

Mobile Application Development (22617)

Teaching Scheme			Credit (L+T+P)	Examination Scheme											
				Theory						Practical					
L	T	P		Paper Hrs.	ESE		PA		Total		ESE		PA		Total	
					Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min
3	-	4	7	3	70	28	30*	00	100	40	25#	10	25	10	50	20

Unit	Unit Outcomes (UOs) (in cognitive domain)	Topics and Sub - topics
Unit - I **Android and its Tools**	1a. Explain the given basic terms related to Android system. 1b. Explain with sketches Android architecture for the given application. 1c. Indentify tools and software required for developing the given Android application with justification. 1d. Explain significance of the given component in Android architecture.	1.1 Introduction to Android, open handset alliance, Android Ecosystem. 1.2 Need of Android, Featutes of Android. 1.3 Tools and software required for developing an Android Application. 1.4 Android Architecture.
Unit - II **Installation and configuration of Android**	2a. Describe function of the given component to operate the specified IDE. 2b. Explain the given term related to virtual machine. 2c. Explain the given basic term related to Android development tools. 2d. Describe the features of given android emulator. 2e. Describe the steps to configure the given android development environment.	2.1 Operating system, Java JDK, Android SDK. 2.2 Android Development Tools (ADT) 2.3 Android Virtual Devices (AVDs) 2.4 Emulators 2.5 Dalvik Virtual Machine, Difference between JVM and DVM. 2.6 Steps to install configure Android Studio and SDK.

Unit-III **UI** **Components and Layouts**	3a. Explain with relevant analogy the given Directory Structure. 3b. Describe the steps to use the given Android rich UI componet. 3c. Describe the steps to use the given type of Layout. 3d. Develop the given basic Android application.	3.1 Control flow, Directory Structure. 3.2 Components of a screen, Fundamental UI design. 3.3. Linear Layout ; Absolute Layout, Frame Layout, Table Layout ; Relative Layout.
Unit-IV **Designing User Interface with View**	4a. Develop rich user Interfaces for the given Android application. 4b. Develop Android application using the given view. 4c. Explain the significance of the given display Alert. 4d. Develop the given application using time and date picker.	4.1 Text view, Edit Text; Button Image Button; Toggle Button; Radio Button and Radio Grop; Checkbox; Progress Bar 4.2 List view ; Grid view ; Image view ; Scroll view ; Custom Toast Alert. 4.3 Time and Date Picker.
Unit-V **Activity And Multimedia with databases**	5a. Apply the given Intents and service in Application development. 5b. Use Fragment to generate the given multiple activities. 5c. Develop programs to play the given multimedia. 5d. Write the query to perform the given database management operation.	5.1 Intent, Intent_Filter 5.2 Activity, Lifecycle; Broadcast Lifecycle. 5.3 Content Provider ; Fragments. 5.4 Service : Features of Service, Android platform service, Defining new service, Service Lifecycle, Permission, example of service. 5.5 Android System Architecture, Multimedia framework, Play Audio and Video, Text to speech, Sensors, Async tasks. 5.6 Audio capture, Camera. 5.7 Bluetooth, Animation. 5.8 SQLite Database, necessity of SQLite, Creation and connection of the database, extracting value from cursors, Transactions.

Unit-VI **Security and Application Deployment**	6a. Explain the given location based service. 6b. Write the steps to customize the given permissions for users. 6c. Explain features of the given android scecurity service. 6d. Write the steps to publish the given android App.	6.1 SMS Telephony 6.2 Location Based Sevices : Creating the project, Getting the maps API key, Displaying the map, Displaying the zoom control, Navigating to a specific location, Adding markers, Getting location, Geocoding and reverse Geocoding, Getting Location data, Monitoring Location. 6.3 Android Security Model, Declaring and Using Permissions, Using Custom Permission. 6.4 Application Deployment : Creating Small Application, Signing of application, Deploying app on Google Play Store, Become a Publisher, Developer Console.

TABLE OF CONTENTS

Unit - I

Chapter - 1 Android and its Tools (1 - 1) to (1 - 12)

1.1 Introduction to Android . . . . . 1 - 1
- 1.1.1 What is Android ? . . . . . 1 - 1
- 1.1.2 Key Platform Components. . . . . 1 - 1
- 1.1.3 Android Versions. . . . . 1 - 2
- 1.1.4 Open Handset Alliance (OHA) . . . . . 1 - 3
- 1.1.5 Android Ecosystem . . . . . 1 - 4
- 1.1.6 Need of Android. . . . . 1 - 5

1.2 Android Features. . . . . 1 - 6

1.3 Tools and Software Required for Developing an Android Application . . . . . 1 - 7

1.4 Android Architecture . . . . . 1 - 8

Unit - II

Chapter - 2 Installation and Configuration of Android (2 - 1) to (2 - 20)

2.1 The Android Operating System . . . . . 2 - 1
- 2.1.1 How do Android Apps Work ? . . . . . 2 - 1
- 2.1.2 Java Development Kit (JDK) . . . . . 2 - 3
- 2.1.3 Android SDK . . . . . 2 - 4

2.2 Android - Developer Tools . . . . . 2 - 4

2.3 Android Virtual Device (AVD) . . . . . 2 - 5

2.4 Android Emulator . . . . . 2 - 7

2.5 Dalvik Virtual Machine (DVM) . . . . . 2 - 7

2.6 Steps to Install and Configure Android Studio and SDK. . . . . 2 - 9
- 2.6.1 The development environment . . . . . 2 - 9
- 2.6.2 Installing the Java Development Kit (JDK) 2 - 9
- 2.6.3 Setting up Android Studio. . . . . 2 - 11
- 2.6.4 Installing Additional Android SDK Packages. . . . . 2 - 15
- 2.6.5 Installation of Emulators. . . . . 2 - 18

Unit - III

Chapter - 3 Components and Layouts (3 - 1) to (3 - 66)

3.1 The Development Process. . . . . 3 - 1
- 3.1.1 Creating the Project . . . . . 3 - 2
- 3.1.2 Directory and File Structure of an Android Studio Project. . . . . 3 - 12

3.2 Components of Android Application . . . . . 3 - 16
- 3.2.1 Creating the user Interface . . . . . 3 - 18

3.3 Layouts . . . . . 3 - 19
- 3.3.1 Android Linear Layout . . . . . 3 - 21
- 3.3.2 Android Absolute Layout. . . . . 3 - 33
- 3.3.3 Android Frame Layout . . . . . 3 - 36
- 3.3.4 Android Table Layout . . . . . 3 - 43
- 3.3.5 Android Relative Layout . . . . . 3 - 52

Unit - IV

Chapter - 4 Designing User Interface with View (4 - 1) to (4 - 34)

4.1 The Android View . . . . . 4 - 1
- 4.1.1 Basic Views . . . . . 4 - 1
  - 4.1.1.1 Text View . . . . . 4 - 2
  - 4.1.1.2 Declare UI Elements in XML . . . . . 4 - 2
- 4.1.2 Edit Text. . . . . 4 - 2
- 4.1.3 Button . . . . . 4 - 6
  - 4.1.3.1 Button Attributes . . . . . 4 - 7
- 4.1.4 Image Button . . . . . 4 - 7
  - 4.1.4.1 Image Button Attributes . . . . . 4 - 8
- 4.1.5 Toggle Button . . . . . 4 - 8
- 4.1.6 Radio Button . . . . . 4 - 13
- 4.1.7 Radio Group. . . . . 4 - 15
- 4.1.8 Check Box . . . . . 4 - 17
  - 4.1.8.1 Check Box Attributes . . . . . 4 - 18

4.1.9 Progress Bar . . . . . 4 - 19
4.1.9.1 Indeterminate . . . . . 4 - 19
4.1.9.2 Result-based . . . . . 4 - 20
4.2 List View . . . . . 4 - 21
4.2.1 Grid View . . . . . 4 - 24
4.2.1.1 Methods of Grid View . . . . . 4 - 25
4.2.1.2 Android Grid View Attributes . . . . . 4 - 25
4.2.2 Image View . . . . . 4 - 26
4.2.3 Scroll View . . . . . 4 - 28
4.2.3.1 Vertically Scrolling . . . . . 4 - 28
4.2.3.2 Horizontally Scrolling . . . . . 4 - 28
4.2.4 Custom Toast Alert . . . . . 4 - 29
4.2.4.1 Creating a Toast . . . . . 4 - 30
4.3 Time and Date Picker . . . . . 4 - 31
4.3.1 Date Picker Properties . . . . . 4 - 31
4.3.2 Time Picker Properties . . . . . 4 - 32

Unit - V

Chapter - 5 Activity and Multimedia with Databases (5 - 1) to (5 - 58)

5.1 Intent . . . . . 5 - 1
5.1.1 Use of Intent . . . . . 5 - 1
5.1.2 Types of Intent in Android . . . . . 5 - 1
5.1.3 Intent Filter . . . . . 5 - 2
5.1.3.1 Work of Intent . . . . . 5 - 3
5.2 Activity Life Cycle . . . . . 5 - 4
5.2.1 Understanding the Activity Lifecycle . . . . . 5 - 7
5.2.2 Broadcast Life Cycle . . . . . 5 - 8
5.2.2.1 Types of Broadcasts . . . . . 5 - 8
5.3 Content Provider . . . . . 5 - 14
5.3.1 Creating a Content Provider . . . . . 5 - 16
5.3.2 Fragments . . . . . 5 - 22
5.3.2.1 Creating a Fragment . . . . . 5 - 22
5.3.2.2 Types of Fragments . . . . . 5 - 23
5.4 Service . . . . . 5 - 25
5.4.1 Android Services Lifecyle Diagram . . . . . 5 - 26
5.4.2 Features of Service . . . . . 5 - 27
5.4.2.1 Android Platform Service . . . . . 5 - 27
5.4.3 Defining New Service . . . . . 5 - 27
5.4.3.1 Create a Service . . . . . 5 - 27
5.4.4 Service Life Cycle . . . . . 5 - 28
5.4.5 Permissions . . . . . 5 - 31
5.5 Android System Architecture . . . . . 5 - 31
5.5.1 Multimedia Framework . . . . . 5 - 33
5.5.2 Play Audio and Video . . . . . 5 - 33
5.5.2.1 Playing Audio Files . . . . . 5 - 34
5.5.2.2 Playing Video Files . . . . . 5 - 34
5.5.3 Text to Speech . . . . . 5 - 34
5.5.4 Sensors . . . . . 5 - 36
5.5.5 Async Tasks . . . . . 5 - 38
5.6 Audio Capture . . . . . 5 - 39
5.6.1 Camera . . . . . 5 - 39
5.7 Bluetooth . . . . . 5 - 41
5.7.1 Animation . . . . . 5 - 43
5.8 SQLite Database . . . . . 5 - 47
5.8.1 Necessity of SQLite Database . . . . . 5 - 48
5.8.2 Creation and Connection of the Database . 5 - 48
5.8.3 Extracting Value from Cursor . . . . . 5 - 48
5.8.4 Transactions . . . . . 5 - 57

Unit - VI

Chapter - 6 Security and Application Deployment (6 - 1) to (6 - 16)

6.1 SMS Telephony . . . . . 6 - 1
6.1.1 Sending and Receiving SMS Messages . . . 6 - 1
6.2 Location Based Service . . . . . 6 - 1
6.2.1 Location API's . . . . . 6 - 1
6.2.1.1 Classes and Interfaces of Location Based Services . . . . . 6 - 2
6.2.2 Creating the Project . . . . . 6 - 2
6.2.2.1 Creating an SMS App . . . . . 6 - 2
6.2.3 Getting the Maps API Key . . . . . 6 - 2
6.2.3.1 What is a Google Maps API Key ? . . 6 - 2
6.2.3.2 How to get a Google Maps API Key ? . . . . . 6 - 2
6.2.4 Displaying the Map . . . . . 6 - 4
6.2.4.1 Google Map - Layout File . . . . . 6 - 7
6.2.4.2 Google Map - Android Manifest File 6 - 7
6.2.4.3 Customizing Google Map . . . . . 6 - 8

6.2.4.4 Changing Map Type . . . . . 6 - 8
6.2.5 Displaying the Zoom Control . . . . . 6 - 8
6.2.5.1 Enable/Disable Zoom . . . . . 6 - 8
6.2.5.2 Important Methods of Zoom Controls . . . . . 6 - 8
6.2.6 Navigating to Specific Location . . . . . 6 - 8
6.2.6.1 Android Google Map Displaying Current Location . . . . . 6 - 8
6.2.6.2 Callback Methods in Google Map. . . 6 - 9
6.2.7 Adding Markers . . . . . 6 - 9
6.2.8 Getting Location, Getting Location data, Monitoring Location . . . . . 6 - 9
6.2.9 Geocoding and Reverse Geocoding. . . . . 6 - 10
6.3 Android Security Model . . . . . 6 - 12
6.3.1 Declaring and using Permissions. . . . . 6 - 12
6.3.1.1 Define a Custom App Permission . . 6 - 12
6.3.1.2 App Signing. . . . . 6 - 12
6.3.1.3 User IDs and File Access. . . . . 6 - 13
6.3.1.4 Create a Permission Group . . . . . 6 - 13
6.3.2 Using Custom Permission . . . . . 6 - 13
6.4 Application Deployment. . . . . 6 - 13
6.4.1 Creating Small Application . . . . . 6 - 13
6.4.2 Signing of Application. . . . . 6 - 13
6.4.2.1 Signing in Debug Mode. . . . . 6 - 13
6.4.2.2 Signing in Release Mode. . . . . 6 - 14
6.4.3 Deploying App on Google Play Store, Become a Publisher, Developer Console. 6 - 14

Solved Sample Papers (S - 1) to (S - 6)

UNIT - I

1 Android and its Tools

1.1 Introduction to Android

1.1.1 What is Android ?

- Android is an operating system and programming platform developed by Google for smartphones and other mobile devices (such as tablets).
- Android is a mobile operating system that is based on a modified version of Linux with a Java programming interface.
- It was originally developed by a start-up of the same name, Android, Inc. In 2005, as part of its strategy to enter the mobile space, Google purchased Android and took over its development work.
- Android was developed by the **Open Handset Alliance (OHA)**, which is led by Google. The **Open Handset Alliance (OHA)** is consortium of multiple companies like Samsung, Sony, Intel and many more to provide a service and deploy handsets using android platform.
- Android gives a chance to reuse the application components and the replacement of native applications.
- The most distinguished feature of Android is that it gives equal opportunities to native apps and third party apps to use its resources.

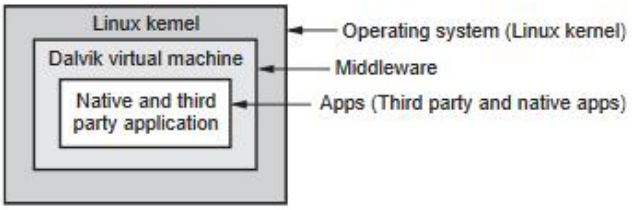


Fig. 1.1.1 Android environment

1.1.2 Key Platform Components

Like any technology stack, the Android platform can be broken down into areas of responsibility to make it easier to understand. The main divisions of the Android platform are shown below.

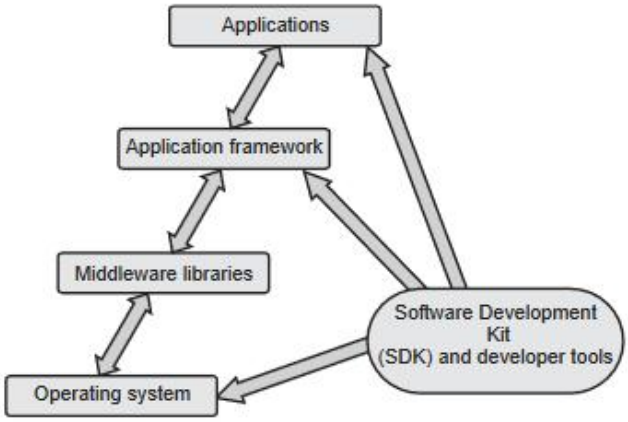


Fig. 1.1.2 The major components of the Android platform

The architectural diagram in Fig. 1.1.2 shows that the Android platform can be broken down into five sections :

- Applications
- Application framework
- Middleware libraries
- Operating system
- SDK and developer tools

Applications are pretty obvious. But several different types of applications are available on most Android devices and the distinction is subtle. Core open source applications are included as part of Android itself, such as the Browser, Camera, Gallery, Music, Phone and more. These are typically included with every Android device. There are also non-open source Google apps that are included with most official builds, including Market, Gmail, Maps, YouTube and more. Many carrier or handset

manufacturer specific Applications are included on specific builds (such as AT and T's own music player, Verizon's own Navigator, or Sprint's TV). And, third-party applications are available in the Android Market, which can be either open source or proprietary. These include independent Google applications such as Goggles and Listen, official apps from popular services like Twitter and Facebook, and thousands of other choices.

1.1.3 Android Versions

Google did not attach any high-calorie code name to its initial versions 1.0 and 1.1 of the Android Operating System. The code names of android ranges from A to N currently, such as Aestro, Blender, Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwitch, Jelly Bean, KitKat, Lolipop and Marshmallow. Let's understand the android history in a sequence.

Fig. 1.1.3 Android versions

Release Date	Version	API Level	Version Name
September 23, 2008	Android 1.0	1	Apple Pie
February 9, 2009	Android 1.1	2	Banana Bread
April 30, 2009	Android 1.5	3	Cupcake
September 15, 2009	Android 1.6	4	Donut
October 26, 2009	Android 2.0	5	Eclair
December 3, 2009	Android 2.0.1	6	
January 12, 2009	Android 2.1	7	
May 20, 2010	Android 2.2	8	Froyo
January 18, 2011	Android 2.2.1	8	
January 22, 2011	Android 2.2.2	8	
November 21, 2011	Android 2.2.3	8	

December 6, 2010	Android 2.3	9	Gingerbread
February 9, 2011	Android 2.3.1	9	
July 25, 2011	Android 2.3.3	10	
September 2, 2011	Android 2.3.4	10	
February 22, 2011	Android 3.0.x	11	Honeycomb
May 10, 2011	Android 3.1.x	12	
July 15, 2011	Android 3.2.x	13	
October 18,2011	Android 4.0	14	Ice Cream Sandwich
October 19, 2011	Android 4.0.1	14	
November 28, 2011	Android 4.0.2	14	
December 16, 2011	Android 4.0.3	15	
February 4, 2012	Android 4.0.4	15	
July 9, 2012	Android 4.1	16	Jelly Bean
July 23, 2012	Android 4.1.1	16	
October 9, 2012	Android 4.1.2	16	
November 13, 2012	Android 4.2	17	
November 27, 2012	Android 4.2.1	17	
February 11, 2013	Android 4.2.2	17	
July 24, 2013	Android 4.3	18	
October 31, 2013	Android 4.4	19	Kitkat
June 23, 2014	Android 4.4.1, 4.4.2, 4.4.3, 4.4.4	19	
October 17, 2014	Android 5.0	21	Lollipop
March 09, 2015	Android 5.1	22	
October 5, 2015	Android 6.0	23	Marshmallow
December 7, 2015	Android 6.0.1	23	
August 22, 2016	Android 7.0	24	Nougat
October 4, 2016	Android 7.1	25	
August 21, 2017.	Android 8.0	26	Oreo
December 5, 2017	Android 8.1	27	Android Oreo (Go edition)
August 6, 2018	Android 9.0	28	Pie
September 3, 2019	Android 10.0	29	Android Q

1.1.4 Open Handset Alliance (OHA)

- The Open Handset Alliance (OHA) is a consortium whose goal is to develop open standards for mobile devices, promote innovation in mobile phones and provide a better experience for consumers at a lower cost.

- The Open Handset Alliance is made up of telecom-related entities, including wireless carriers, semiconductor companies, handset manufacturers and software companies. However, several major wireless companies and manufacturers are absent from the coalition, including Nokia, Symbian, Apple, RIM, Microsoft, Verizon and Cingular.
- Google and 33 other companies announced the formation of the Open Handset Alliance on November 5, 2007. According to the joint press release from that day :

This alliance shares a common goal of fostering innovation on mobile devices and giving consumers a far better user experience than much of what is available on today's mobile platforms.

- By providing developers a new level of openness that enables them to work more collaboratively, Android will accelerate the pace at which new and compelling mobile services are made available to consumers.
- For us as mobile application developers, that means we are free to develop whatever creative mobile applications we can think of, free to market them (or give them, at our option) to Android mobile phone owners, and free to profit from that effort any way we can. Each member of the Open Handset Alliance has its own reasons for participating and contributing its intellectual property, and we are free to benefit.
- The Open Handset Alliance integrates contributed software and other intellectual property from its member companies and makes it available to developers through the open source community.
- Software is licensed through the Apache V2 license, which you can see at http://www.apache.org/licenses/LICENSE-2.0.txt. Use of the Apache license is critical, because it allows handset manufacturers to take Android code, modify it as necessary and then either keep it proprietary or release it back to the open source community, at their option.
- The original Alliance members include handset manufacturers (HTC, LG, Motorola, Samsung), mobile operators (China Mobile Communications, KDDI, DoCoMo, Sprint/Nextel, T-Mobile, Telecom talia, Telefonica), semiconductor companies (Audience, Broadcom, Intel, Marvell, NVidia Qualcomm, SiRF, Synaptics), software companies (Ascender, eBay, esmertec, Google, LivingImage, LiveWire, Nuance, Packet Video, SkyPop, SONiVOX), and commercialization companies (Aplix, Noser, TAT, Wind River). The Alliance includes the major partners needed to deliver a platform for mobile phone applications in all of the major geographies.
- The Alliance releases software through Google's developer website (http://developer .android.com). The Android SDK for use by application software developers can be downloaded directly from that website.
- The OHA introduced a mobile device operating system called Android Symbian operating system.

1.1.5 Android Ecosystem

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

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

- **Users** buy devices and applications
- **Equipment makers** sell devices, sometimes bundled with applications
- **Developers** buy devices, then make and sell applications.
- An open source platform for mobile, embedded and wearable devices .
- Google is the principle maintainer
- Other companies contribute to the system.
- Each device manufacturer can customize Android to suit their needs

This new ecosystem is built and developed in order to drive the usage of apps. The end-users gain access to several mobile apps just because of their popularity. Thus, the ecosystem of mobile applications enables the enterprises and the independent developers to develop attractive, feature-rich and unique mobile applications which will perform rapidly and can essentially garner the attention of the market.

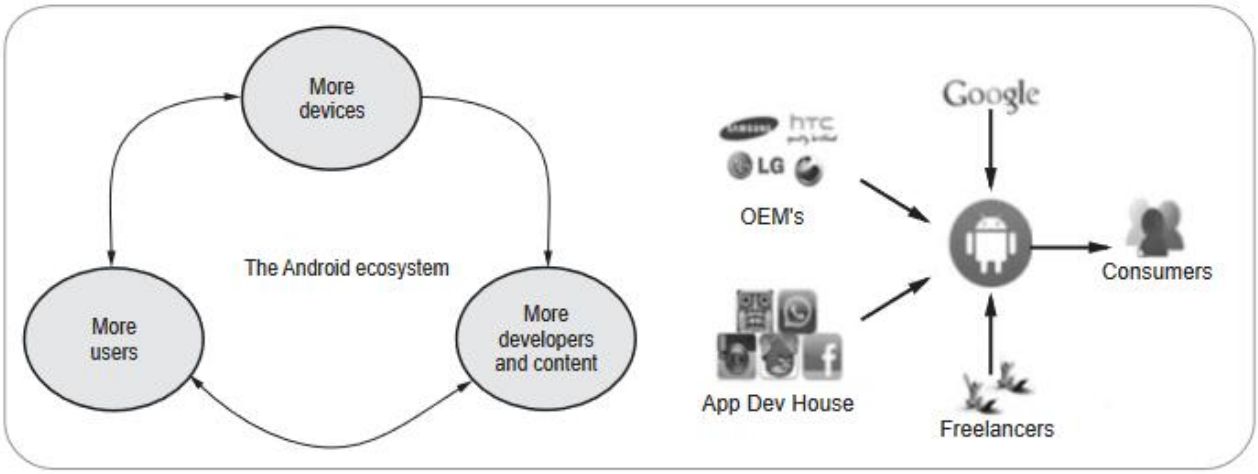


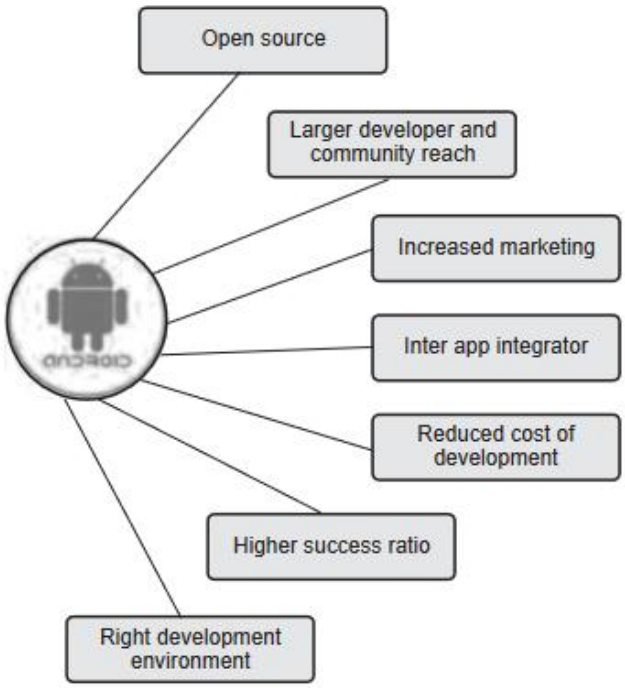
Fig. 1.1.4 The Android ecosystem thrives with device compatibility

Android's source code has been used as the basis of different ecosystems, most notably that of Google which is associated with a suite of proprietary software called Google Mobile Services (GMS), that frequently comes pre-installed on the devices. This includes core apps such as Gmail, the digital distribution platform Google Play and associated Google Play Services development platform, and usually apps such as the Google Chrome web browser. These apps are licensed by manufacturers of Android devices certified under standards imposed by Google. Other competing Android ecosystems include Amazon.com's Fire OS, or LineageOS. Software distribution is generally offered through proprietary application stores like Google Play Store or Samsung Galaxy Store, or open source platforms like Aptoide or F-Droid, which utilize software packages in the APK format.

1.1.6 Need of Android

There are many reasons to choose android operating system like

1. Open source
2. Large developer and community reach
3. Increased marketing
4. Inter App integration
5. Reduced cost of development
6. Higher success ratio
7. Rich development environment
8. Variety
9. Widgets
10. Multi-tasking
11. Google integration



1. **Reduced cost of development :** The development tools like Android studio, Android SDK, JDK, and Eclipse IDE etc. are free to download for the android mobile application development.

2. **Open Source :** The Android OS is an open-source platform based on the Linux kernel and multiple open-source libraries. In this way developers are free to contribute or extend the platform as necessary for building mobile apps which run on Android devices.
3. **Multi-Platform Support :** In market, there are a wide range of hardware devices powered by the Android OS, including many different phones and tablet. Even development of android mobile apps can occur on Windows, Mac OS or Linux.
4. **Multi-Carrier Support :** World wide a large number of telecom carriers like Airtel, Vodafone, Idea Cellular, AT and T Mobility, BSNL etc. are supporting Android powered phones.
5. **Open Distribution Model :** Android Market place (Google Play store) has very few restrictions on the content or functionality of an android app. So the developer can distribute theirs app through Google Play store and as well other distribution channels like Amazon's app store.
6. **Multi-tasking :** Multi-tasking is available on virtually all smartphone platforms but hardly does any operating system do it better than android. Some manufacturers, e.g. Samsung allows for multi-window tasking. This allows users to view multiple apps at the same time.
7. **Variety :** Android has provided a medium that has allowed smartphone manufacturers like Samsung, HTC, ZTE, Sony and Motorola to allow their imagination run wild. On Android platform, there is something for everyone regardless of taste or budget. Android offers flexibility that is not obtainable with other platforms.
8. **Widgets :** Widget is another thing that is working absolutely well on Android devices. It has worked so excellently, other operating systems and mobile app development are starting to adopt something similar, e.g. the Live Tile System in Windows phone. Widget allows users to access all the necessary information they want right from the screen, at a glance, without having to launch an app.
9. **Google Integration :** Google, the current owners of Android, are highly smart in business. Android devices work seamlessly with all of Google's products; Google Docs, Gmail, YouTube, Google Music, Google Maps, Google Chrome, Google+ etc. Google is regarded as the king of the web. This is a major attraction for most people, as having an android device gives them instant access to all of this important software.

1.2 Android Features

Android is a powerful open source operating system which provides a lot of great features like :

- It's an open source and we can customize the OS based on our requirements.
- It supports a multi-tasking, we can move from one task window to another and multiple applications can run simultaneously.
- It will give a chance to reuse the application components and the replacement of native applications.
- It has an extensive support for multimedia hardware control to perform playback or recording using camera and microphone.
- By using WIFI technology we can pair with other devices using apps.
- It support a connectivity for GSM, CDMA, WIFI, NFC, Bluetooth, etc. for telephony or data transfer. It will allow us to make or receive a calls / SMS messages and we can send or retrieve a data across mobile networks.
- Android have a multiple APIs to support a location-based services such as GPS.
- We can perform all data storage related activities by using light weight database SQLite.
- It have a wide range of media supports like AVI, MKV, FLV, MPEG4 etc. to play or record variety of audio / video and having a different image formats like JPEG, PNG, GIF, BMP, MP3, etc.
- It has an integrated open source webkit layout based web browser to support HTML5, CSS3.

- We can access the hardware components like Camera, GPS and Accelerometer.
- It has a support for 2D/3D Graphics.

1.3 Tools and Software Required for Developing an Android Application

Following are Tools and software required to begin with Android development :

1. Windows (XP or later), Linux (any recent Linux distribution) and Mac OS X (10.4.9 or later)
2. Android Studio or Eclipse
3. Android SDK
4. Java

System requirements :

1. Windows/Linux/Mac powered PC

Operating system is the soul of the PC. So the better your processor is, the easier will be developing Android over it. There is some basic system requirements which if followed could really ease Android development .

Following are the ideal system requirements for developing Android Apps

Windows

- Microsoft® Windows® 7/8/10 (32 or 64-bit)
- 3 GB RAM minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator
- 2 GB of available disk space minimum, 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 × 800 minimum screen resolution
- For accelerated emulator : Intel® processor with support for Intel® VT-x, Intel® EM64T (Intel® 64), and Execute Disable (XD) Bit functionality

Mac

- Mac® OS X® 10.10 (Yosemite) or higher, up to 10.12 (macOS Sierra)
- 3 GB RAM minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator
- 2 GB of available disk space minimum, 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 × 800 minimum screen resolution

Linux

- GNOME or KDE desktop Tested on Ubuntu® 14.04 LTS, Trusty Tahr (64-bit distribution capable of running 32-bit applications)
- 64-bit distribution capable of running 32-bit applications
- GNU C Library (glibc) 2.19 or later
- 3 GB RAM minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator
- 2 GB of available disk space minimum, 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 × 800 minimum screen resolution
- For accelerated emulator : Intel® processor with support for Intel® VT-x, Intel® EM64T (Intel® 64), and Execute Disable (XD) Bit functionality, or AMD processor with support for AMD Virtualization™ (AMD-V™)

Recommended Processor

More than i3, i5 or i7 developers should be concerned about the speed of the processor and number of cores. 2 GHz Quad core Intel i3 processor would be enough to develop on Android. I have personally used it and did not face any difficulty.

2. IDE (Eclipse or Android Studio)

Although the SDK can be used to write Android programs in the command prompt, the most common method is by using an Integrated Development Environment (IDE). Eclipse is quite reputed and trusted IDE. And a lot of people use it for Android development too. But now a days Android Studio is preferable, especially for a beginner.

Because

i) Android Studio is a Google product - these are the same guys who develop Android.

ii) Android Studio uses Grade Build system. Which is quite faster than Eclipse's Apache Ant

iii) Android Studio's Autocomplete feature quite better than that of Eclipse

iv) Designing UI has always been complex but Android Studio has completely changed this.

v) The new interface design in Android Studio is faster, responds to changes more rapidly and has more customization options than Eclipse

3. Android SDK

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

- Required libraries
- Debugger
- An emulator
- Relevant documentation for the Android Application Program Interfaces (APIs)
- Sample source code
- Tutorials for the Android OS

4. Java

Java Development Kit (JDK) for developing on Android.

1.4 Android Architecture

The Android operating system is built on top of a modified Linux kernel. The software stack contains Java applications running on top of a virtual machine. Components of the system are written in Java, C, C++, and XML. Android operating system is a stack of software components which is roughly divided into five sections

1) Linux kernel

2) Native libraries (middleware)

3) Android runtime

4) Application framework

5) Applications

1) Linux kernel

It is the heart of android architecture that exists at the root of android architecture. Linux kernel is responsible for device drivers, power management, memory management, device management and resource access. This layer is the foundation of the Android Platform.

Positioned at the bottom of the Android software stack, the Linux Kernel provides a level of abstraction between the device hardware and the upper layers of the Android software stack. Based on Linux version 2.6, the kernel provides pre-emptive multitasking, low-level core system services such as memory, process and power management in addition to providing a network stack and device drivers for hardware such as the device display, Wi-Fi and audio.

- Contains all low level drivers for various hardware components support.
- Android Runtime relies on Linux Kernel for core system services like,
- Memory, process management, threading etc.
- Network stack
- Driver model
- Security and more.

2) 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.

In addition to a set of standard Java development libraries (providing support for such general purpose tasks as string handling, networking and file manipulation), the Android development environment also includes the Android Libraries. These are a set of 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.

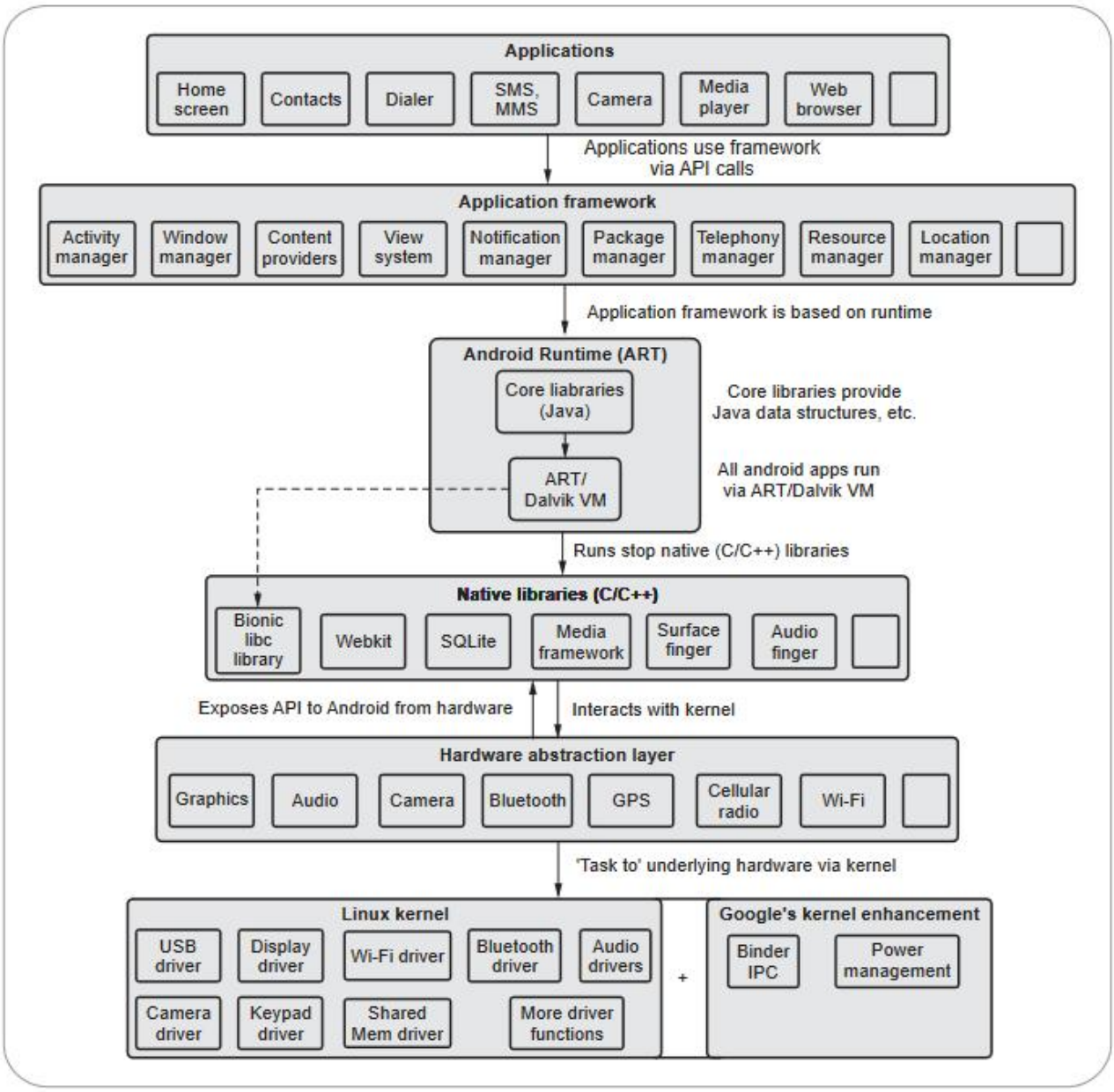


Fig. 1.4.1 Android architecture

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.graphics** : A low-level 2D graphics drawing API including colors, points, filters, rectangles and canvases.
- **android.hardware** : Presents an API providing access to hardware such as the accelerometer and light sensor.

- **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.media :** Provides classes to enable playback of audio and video.
- **android.net :** A set of APIs providing access to the network stack. Includes android.net.wifi, which provides access to the device's wireless stack.
- **android.print :** Includes a set of classes that enable content to be sent to configured printers from within Android applications.
- **android.provider :** A set of convenience classes that provide access to standard Android content provider databases such as those maintained by the calendar and contact applications.
- **android.text' :** Used to render and manipulate text on a device display.
- **android.util :** A set of utility classes for performing tasks such as string and number conversion, XML handling and date and time manipulation.
- **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.

C/C++ Libraries

The Android runtime core libraries outlined in the preceding section are Java-based and provide the primary APIs for developers writing Android applications. It is important to note, however, that the core libraries do not perform much of the actual work and are, in fact, essentially Java "wrappers" around a set of C/C++ based libraries. When making calls, for example, to the android.opengl library to draw 3D graphics on the device display, the library actually ultimately makes calls to the OpenGL ES C++ library which, in turn, works with the underlying Linux kernel to perform the drawing tasks.

C/C++ libraries are included to fulfil a wide and diverse range of functions including 2D and 3D graphics drawing, Secure Sockets Layer (SSL) communication, SQLite database management, audio and video playback, bitmap and vector font rendering, display subsystem and graphic layer management and an implementation of the standard C system library (libc).

In practice, the typical Android application developer will access these libraries solely through the Java based Android core library APIs. In the event that direct access to these libraries is needed, this can be achieved using the Android Native Development Kit (NDK), the purpose of which is to call the native methods of non-Java or Kotlin programming languages (such as C and C++) from within Java code using the Java Native Interface (JNI).

- **SQLite** Library used for data storage and light in terms of mobile memory footprints and task execution.
- **WebKit** Library mainly provides Web Browsing engine and a lot more related features.
- The **surface manager** library is responsible for rendering windows and drawing surfaces of various apps on the screen.
- The **media framework** library provides media codecs for audio and video.
- The **OpenGl** (Open Graphics Library) and **SGL** (Scalable Graphics Library) are the graphics libraries for 3D and 2D rendering, respectively.
- The **FreeType** Library is used for rendering fonts.

3) Android runtime

In android runtime, there are core libraries and DVM (Dalvik Virtual Machine) which is responsible to run android application. DVM is like JVM but it is optimized for mobile devices. It consumes less memory and provides fast performance. 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.

When an Android app is built within Android Studio it is compiled into an intermediate bytecode format

(referred to as DEX format). When the application is subsequently loaded onto the device, the Android Runtime (ART) uses a process referred to as Ahead-of-Time (AOT) compilation to translate the bytecode down to the native instructions required by the device processor. This format is known as Executable and Linkable Format (ELF).

Each time the application is subsequently launched, the ELF executable version is run, resulting in faster application performance and improved battery life.

This contrasts with the Just-in-Time (JIT) compilation approach used in older Android implementations whereby the bytecode was translated within a Virtual Machine (VM) each time the application was launched.

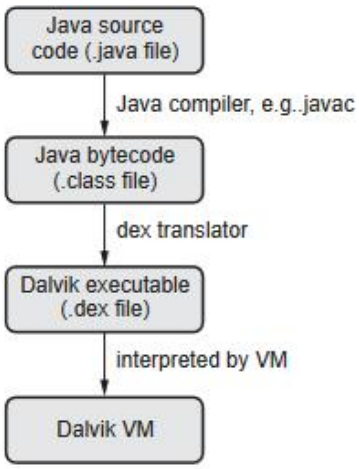


Fig. 1.4.2 Android runtime

- Dalvik is a specialized virtual machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU.
- Android apps execute on Dalvik VM, a "clean-room" implementation of JVM
- Dalvik optimized for efficient execution
- Dalvik: register-based VM, unlike Oracle's stack-based JVM
- Java .class bytecode translated to Dalvik EXecutable (DEX) bytecode, which Dalvik interprets

4) Android framework

On the top of Native libraries and android runtime, there is android framework. Android framework includes Android API's such as UI (User Interface), telephony, resources, locations, Content Providers (data) and package managers. It provides a lot of classes and interfaces for android application development.

The Application Framework is a set of services that collectively form the environment in which Android applications run and are managed. This framework implements the concept that Android applications are constructed from reusable, interchangeable and replaceable components. This concept is taken a step further in that an application is also able to publish its capabilities along with any corresponding data so that they can be found and reused by other applications.

The Android framework includes the following key services :

- **Activity Manager :** Manages the life cycle of an applications and maintains the back stack as well so that the applications running on different processes has smooth navigations.
- **Package Manager :** Keeps track of which applications are installed in your device.
- **Window Manager :** Manages windows which are java programming abstractions on top of lower level surfaces provided by surface manager.
- **Telephony Managers :** Manages the API which is used to build the phone applications
- **Content Providers :** Provide feature where one application can share the data with another application, like phone number, address, etc.
- **View Manager :** Buttons, Edit text, all the building blocks of UI, event dispatching etc.
- **Resource Manager :** Provides access to non-code embedded resources such as strings, colour settings and user interface layouts.
- **Notifications Manager :** Allows applications to display alerts and notifications to the user.
- **Telephony Manager :** Provides information to the application about the telephony services available on the device such as status and subscriber information.

- **Location Manager :** Provides access to the location services allowing an application to receive updates about location changes

5) Applications

On the top of android framework, there are applications. These comprise both the native applications provided with the particular Android implementation (for example web browser and email applications) and the third party applications installed by the user after purchasing the device. All applications such as home, contact, settings, games, browsers are using android framework that uses android runtime and libraries.

Android runtime and native libraries are using Linux kernel. Any applications that you write are located at this layer.

❑❑❑

UNIT - II

2 Installation and Configuration of Android

2.1 The Android Operating System

- Android is an open-source mobile operating system. It is a variant of Linux hence providing extensive security, modularity and productivity at the mobile device level.
- Originally, Android was created by a company called Android Inc. Google acquired this company in 2005. After then, Google made it open source.
- Android has seven major releases each having several minor revisions. In order to follow these versions easier, developers name them with cookie names. The popular versions of Android are Kitkat (Android 4.4), Lollipop (Android 5.1) and Marshmallow (Android 6.0), Nougat (Android 7.0) is also gaining popularity.
- Android is utilized not only in smartphones but also in tablets, netbooks, digital television boxes, handheld game devices and even in single board computers such as UDOO. UDOO is a single-board computer with an integrated Arduino Due compatible microcontroller, designed for computer science education, the world of Makers and the Internet of Things.

2.1.1 How do Android Apps Work ?

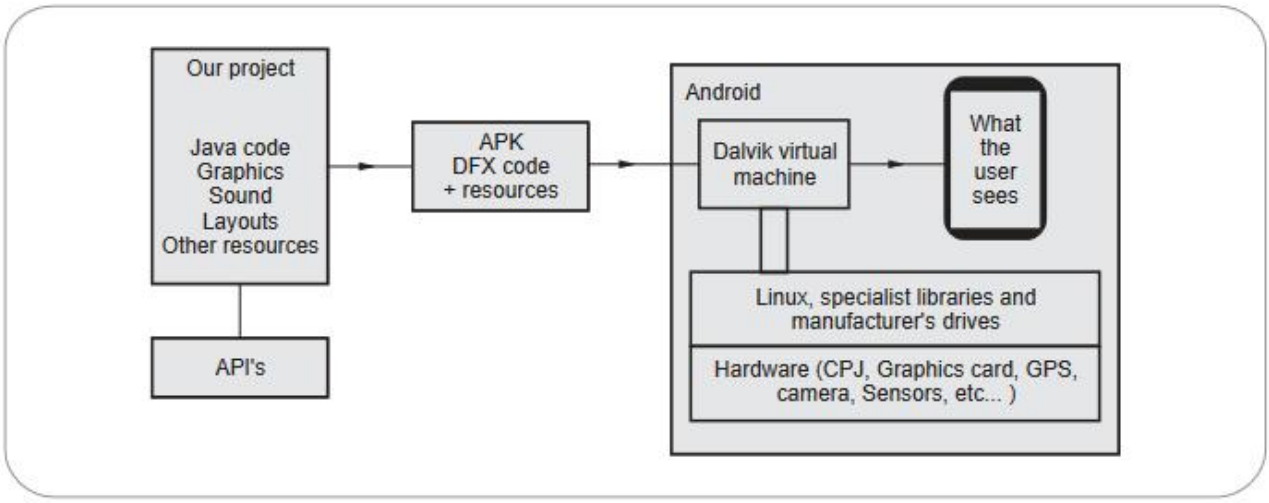


Fig. 2.1.1 Working of Android Apps

- There are different ways the programs run on various platforms. The lowest level software can be written in machine code that runs directly on the microprocessor. This is shown in Fig. 2.1.2. Since it is difficult to develop complex applications in machine code, operating systems are used. Operating systems provide a communication and control layer between the application software and the hardware as shown in Fig. 2.1.3. If we want to develop a native application for running on a specific

hardware / operating system, we have to do this using a compiler and linker. Compiler and linker takes the source code and creates the executable file that actually runs on the operating system as shown in Fig. 2.1.4. For example, if we want to develop an application in C++ programming language, we have to utilize the compilation/linking process.

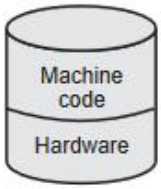


Fig. 2.1.2 Machine code - hardware relation

- The main advantage of native applications is their speed. However, the disadvantage is the incompatibility across different platforms. For example, we cannot run a native Windows application on Ubuntu and vice versa. Virtual machine concept is developed to overcome this limitation. Virtual machine is software that runs on the operating system and provides an abstraction to the developer as shown in Fig. 2.1.5. The application software runs on top of the virtual machine.

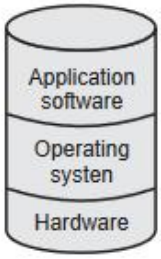


Fig. 2.1.3 Operating system layer between the hardware and the app

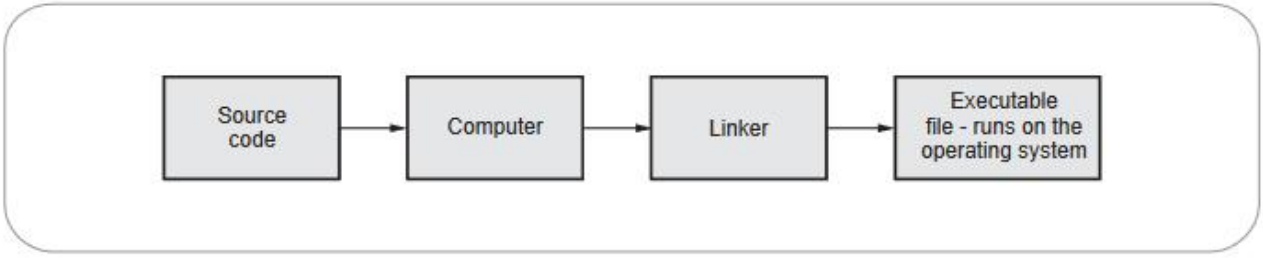


Fig 2.1.4 Creating a native executable from the source code

- Therefore, as long as a computer has the virtual machine running, the application software can run on that computer independent of the hardware and the operating system. A good example is the Java Virtual Machine (JVM). JVM runs on almost all operating systems and platforms. Therefore, when we develop Java software, it will be run on the JVM independent of the operating system/platform. The obvious advantage of developing apps that run on virtual machines can then be stated as : "develop once and run on all platforms". However, applications running on virtual machines are slower than native applications. General development process of virtual machine applications is summarized in Fig. 2.1.6.

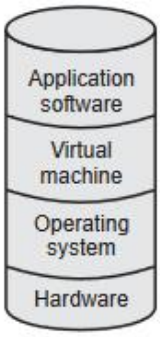


Fig. 2.1.5 Virtual machine between the app and the operating system

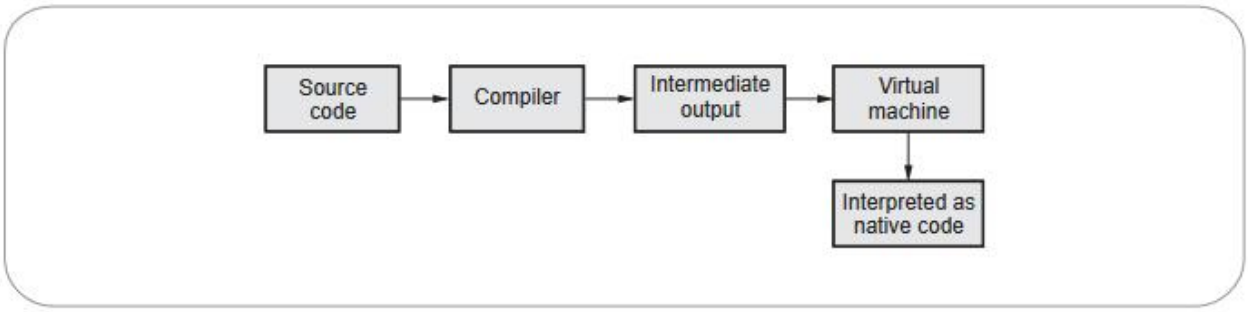


Fig 2.1.6 Creating an intermediate code from the source code - intermediate code is interpreted by the virtual machine

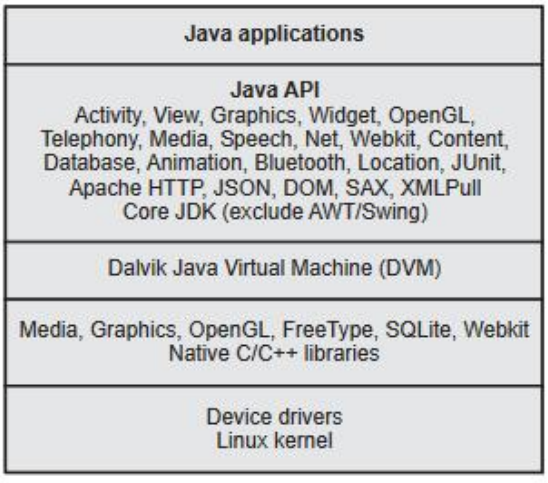
- Similar to Java applications, Android applications also run on a JVM. There are two special virtual machines used in Android : Dalvik Virtual Machine (DVM) and Android RunTime (ART). These are specialized JVMs which can run on low system resources. The .apk files (executables of Android apps) actually run on these virtual machines. DVM has been the default runtime environment (~ virtual machine) until the Lollipop release (Android 5.0). ART is introduced by Android 4.0 and has been the default VM as of Android 5.0. DVM and ART basically do the same job : running Android apps independent of the platform. The main advantage of ART over DVM is the utilization of a concept called Ahead of Time (AOT) compilation instead of Just in Time (JIT) approach. In AOT, apps are compiled during installation hence they load faster with lower CPU usage. On the other hand, JIT compilation provides lower storage space consumption with relatively longer loading times.

2.1.2 Java Development Kit (JDK)

- The Android SDK was developed using the Java programming language. Similarly, Android applications are also developed using Java. As a result, the Java Development Kit (JDK) is the first component that must be installed. Android development requires the installation of version 7 or above of the Standard Edition of the Java Platform Development Kit. Java is provided in both development (JDK) and runtime (JRE) packages.

How Java and Android work together ?

- After writing a program in Java for Android, we click on a button to change our code into another form that is understood by Android. This other form is called **Dalvik EXecutable (DEX)** code, and the transformation process is called **compiling.**
- Compiling takes place on the development machine after we click on that button.
- The part of the Android system that **executes** (runs) our compiled DEX code is called the **Dalvik Virtual Machine (DVM).** The DVM itself is a piece of software written in another language that runs on a specially adapted version of the Linux operating system. So what the user sees of Android, is itself just an app running on another operating system.



- The purpose of the DVM is to hide the complexity and diversity of the hardware and software that Android runs on but, at the same time, its purpose is to expose all of its useful features. This exposing of features generally works in two ways. The DVM itself must have access to the hardware, which it does, but this access must be programmer friendly and easy to use. The way the DVM allows us access is indeed easy to use because of the Android **Application Programming Interface (API).**

2.1.3 Android SDK

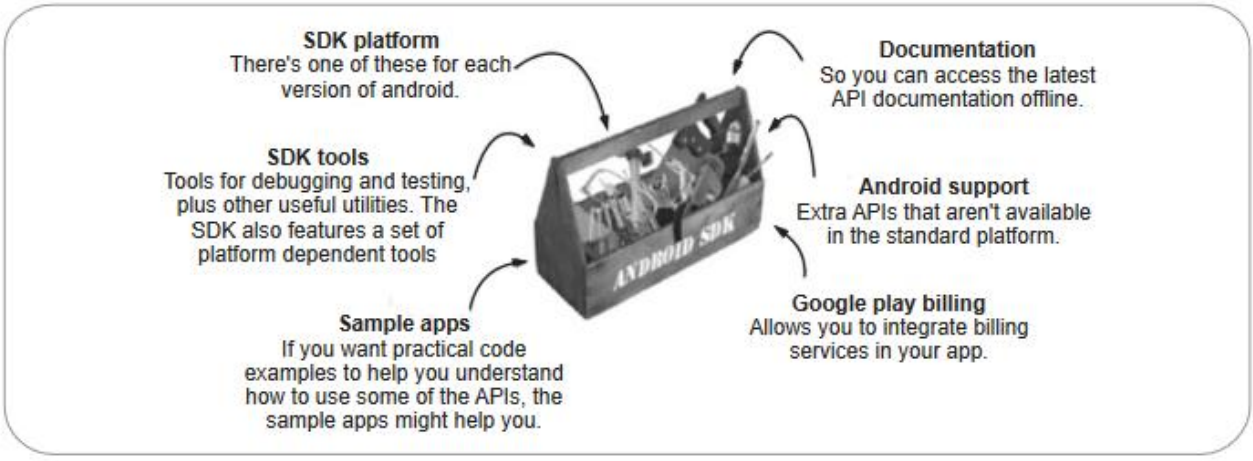


Fig. 2.1.7 Android SDK

- The Android SDK is a software development kit for Android development that contains the necessary components to build Android applications. It comes as part of the Android Studio and includes a comprehensive set of development tools including a debugger, software libraries of prewritten code, a device emulator to run your Android applications, documentation, sample code, and tutorials. These tools are used to create apps that look great and take advantage of the hardware capabilities.
- The Android Virtual Device manager that comes with the Android SDK enables to test application with any version of Android.
- The Android SDK and Gradle tooling incorporate the needed tools to build, compile, and package Android applications.
- For debugging Android applications, the Android Debug Bridge (adb) tool that enables to connect to any virtual or real Android device.
- Here's the list of the important packages in the Android SDK :
- SDK Tools • SDK Platform Tools • SDK Platform • System Image • SDK Samples

2.2 Android - Developer Tools

- The android developer tools are used to create interactive and powerful application for android platform. The tools can be generally categorized into two types.
- SDK tools • Platform tools

SDK tools :

SDK tools are generally platform independent and are required no matter which android platform you are working on. When you install the Android SDK into your system, these tools get automatically installed. The list of SDK tools has been given below -

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

2.3 Android Virtual Device (AVD)

An Android Virtual Device (AVD) is an emulator configuration that allows developers to test the application by simulating the real device capabilities, without the necessity to install the application on a physical Android based device. An AVD may be configured to emulate a variety of hardware features including options such as screen size, memory capacity and the presence or otherwise of features such as a camera, GPS navigation support or an accelerometer. As part of the standard Android Studio installation, a number of emulator templates are installed allowing AVDs to be configured for a range of different devices. Additional templates may be loaded or custom configurations created to match any physical Android device by specifying properties such as processor type, memory capacity and the size and pixel density of the screen. When launched, an AVD will appear as a window containing an emulated Android device environment. Fig. 2.3.1, for example, shows an AVD session configured to emulate the Google Nexus 6 API 15 model. New AVDs are created and managed using the Android Virtual Device Manager, which may be used either in command-line mode or with a more user-friendly graphical user interface.

Fig. 2.3.1 Sample AVD

An AVD contains a hardware profile, system image, storage area, skin, and other properties.

AVD and app features :

Hardware Profile Property	Description
Device Name	Name of the hardware profile. The name can contain uppercase or lowercase letters, numbers from 0 to 9, periods (.), underscores (_), parentheses (()), and spaces. The name of the file storing the hardware profile is derived from the hardware profile name.
Device Type	Select one of the following : Phone/Tablet Wear OS Android TV Chrome OS Device Android Automotive
Screen Size	The physical size of the screen, in inches, measured at the diagonal. If the size is larger than your computer screen, it's reduced in size at launch.
Screen Resolution	Type a width and height in pixels to specify the total number of pixels on the simulated screen.
Round	Select this option if the device has a round screen, such as some Wear OS devices.
Memory : RAM	Type a RAM size for the device and select the units, one of B (byte), kB (kilobyte), MB (megabyte), GB (gigabyte) or TB (terabyte).
Input : Has Hardware Buttons (Back/Home/Menu)	Select this option if your device has hardware navigation buttons. Deselect it if these buttons are implemented in software only. If you select this option, the buttons won't appear on the screen. You can use the emulator side panel to "press" the buttons, in either case.
Input: Has Hardware Keyboard	Select this option if your device has a hardware keyboard. Deselect it if it doesn't. If you select this option, a keyboard won't appear on the screen. You can use your computer keyboard to send keystrokes to the emulator, in either case.
Navigation Style	Select one of the following : None - No hardware controls. Navigation is through the software. D-pad - Directional Pad support. Trackball Wheel These options are for actual hardware controls on the device itself. However, the events sent to the device by an external controller are the same.
Supported Device States	Select one or both options : Portrait - Oriented taller than wide. Landscape - Oriented wider than tall. If you select both, you can switch between orientations in the emulator. You must select at least one option to continue.
Cameras	To enable the camera, select one or both options : Back-Facing Camera - The lens faces away from the user. Front-Facing Camera - The lens faces toward the user. Later, you can use a webcam or a photo provided by the emulator to simulate taking a photo with the camera.
Sensors : Accelerometer	Select if the device has hardware that helps the device determine its orientation.
Sensors: Gyroscope	Select if the device has hardware that detects rotation or twist. In combination with an accelerometer, it can provide smoother orientation detection and support a six-axis orientation system.
Sensors : GPS	Select if the device has hardware that supports the Global Positioning System (GPS) satellite-based navigation system.

Sensors: Proximity Sensor	Select if the device has hardware that detects if the device is close to your face during a phone call to disable input from the screen.
Default Skin	Select a skin that controls what the device looks like when displayed in the emulator. Remember that specifying a screen size that's too big for the resolution can mean that the screen is cut off, so you can't see the whole screen.

2.4 Android Emulator

Android Emulator is used to run, debug and test the android application. If you don't have the real device, it can be the best way to run, debug and test the application.

An emulator is an application that supports a virtual device which prototypes, develops and tests the applications without using a real device. It acts as all hardware and software features of device. It cannot make a real phone call. It runs a full Android system stack, down to the kernel level. It includes a set of preinstalled applications (such as the dialer, messenger) which can be accessed from our applications.

Each **AVD** is an independent device. It has its own storage and data. In each configuration, an Android platform is specified which is to be run in the emulator along with the set of hardware options, emulator skin, etc. when emulator is launched, we specify the AVD configuration that we want to load. As soon as we launch the emulator with an AVD configuration, it automatically loads the user data and SD card data from the AVD directory. By default, cache, user data and SD card is stored in AVD directory. AVD manager manages all the emulators.

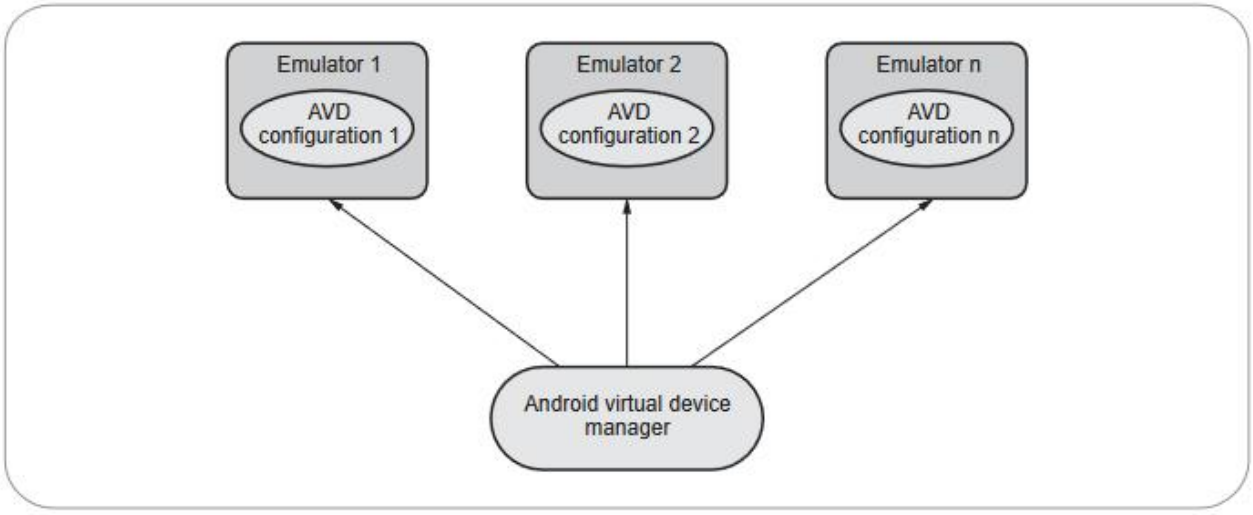


Fig. 2.4.1 Relation between AVD Manager, AVD and Emulator

2.5 Dalvik Virtual Machine (DVM)

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 purpose built virtual machine designed specifically for android which was developed by Dan Bornstein and his team. Strictly it was developed for mobile devices. Dalvik virtual machine uses register based architecture. With this architecture dalvik virtual machine has few advantages over JAVA virtual machine such as :

1. Dalvik uses its own 16 bit instruction set than java 8 bit stack instructions, which reduce the dalvik instruction count and raised its interpreter speed.
2. Dalvik use less space, which means an uncompressed **.dex** file is smaller in size(few bytes) than compressed java archive file(.jar file). 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.

The compiling and packaging process :

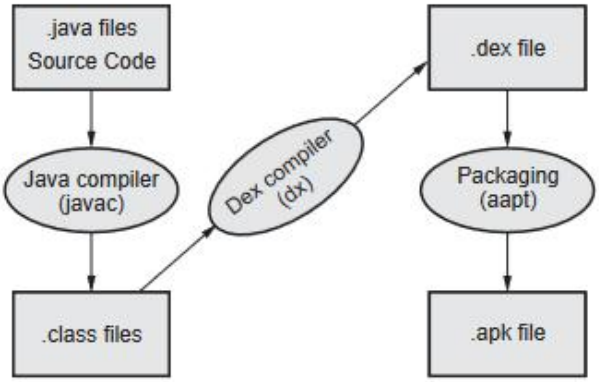


Fig. 2.5.1 Compiling and packaging process

- As shown in Fig. 2.5.1, 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.

Role of Dalvik Virtual Machine :

In java we write and compile java program using java compiler and run that bytecode on the java virtual machine. On the other side, In android we still write and compile java source file (bytecode) on java compiler, but at that point we recompile it once again using dalvik compiler to dalvik bytecode (dx tool converts java .class file into .dex format) and this dalvik bytecode is then executed on the dalvik virtual machine.In android Architecture there is DVM above Linux kernel.

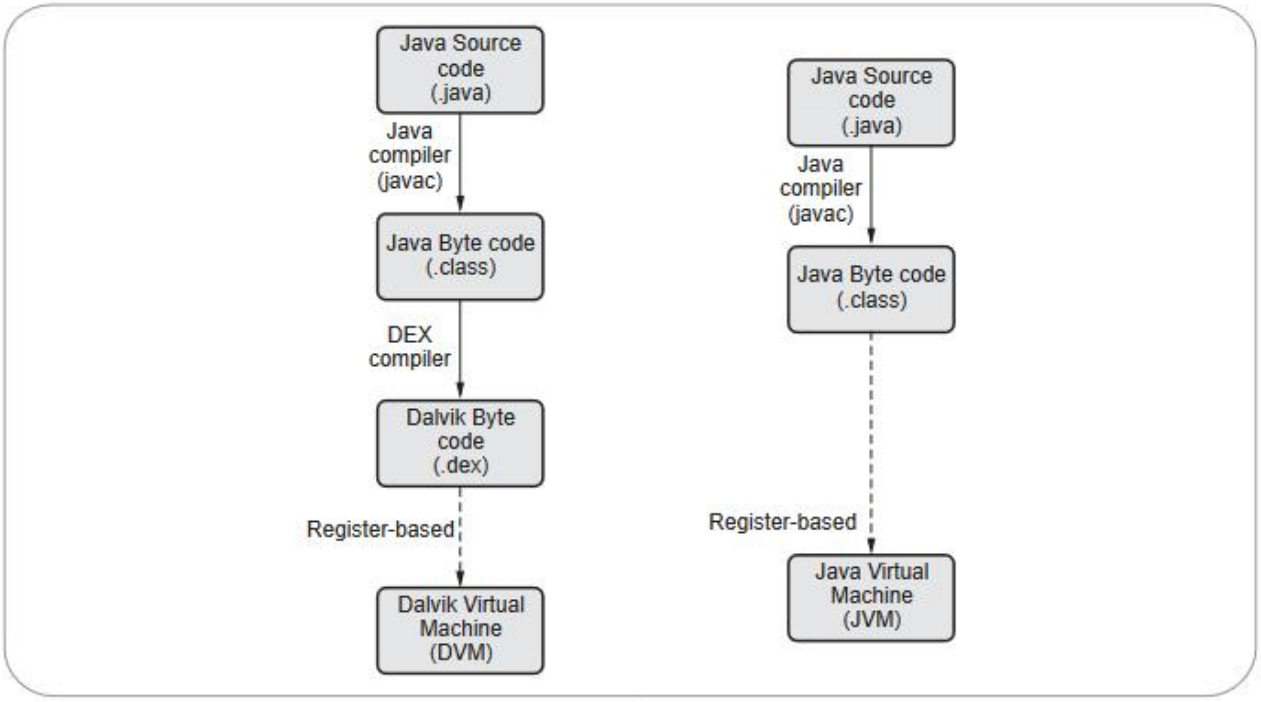
Every single application runs as one process and every process has its own instance of DVM. Every time user click on app icon then above process will execute.

Key Feature :

- Android dalvik virtual machine is a registered based Virtual machine.
- With using DVM, android device can run multiple instances of VM efficiently.
- **Duplicate String** and other **constants** used in multiple class files are included only once in the .dex Output to conserve space.
- Java byte code is also converted into alternative instruction set used by DVM.
- An uncompressed .dex file is smaller in size as compare to compressed .jar derived from same .class file.
- The Dalvik executable may be modified again when installed onto the mobile devices.
- DVM was slimmed down to use less space.
- The Dalvik VM executes file in Dalvik executable (.dex) format which is optimized for minimal memory foot print.
- DVM not only can run multiple VM instances but creation of new VM must be fast as well.

Comparison of Java VM and the Dalvik virtual machine :

Sr. No.	DVM (Dalvik Virtual Machine)	JVM (Java Virtual Machine)
1.	It is Register based which is designed to run on low memory.	It is Stack based.
2.	DVM uses its own byte code and runs ".Dex" file. From Android 2.2 SDK Dalvik has got a Just in Time compiler	JVM uses java byte code and runs ".class" file having JIT (Just In Time).
3.	DVM has been designed so that a device can run multiple instances of the VM efficiently. Applications are given their own instance.	Single instance of JVM is shared with multiple applications.
4.	DVM supports Android operating system only.	JVM supports multiple operating systems.
5.	For DVM very few Re-tools are available.	For JVM many Re-tools are available.
6.	There is constant pool for every application.	It has constant pool for every class.
7.	Here the executable is APK.	Here the executable is JAR.



2.6 Steps to Install and Configure Android Studio and SDK

2.6.1 The development environment

A **development environment** is a term that refers to having everything you need in order to develop, set up, and be ready to go in one place.

Android Studio is an **integrated development environment (IDE)** that takes care of all the complexities of compiling our code and linking with the JDK and the Android API. Once we have installed the JDK and Android Studio, we can do everything we need inside this application.

2.6.2 Installing the Java Development Kit (JDK)

The Android SDK was developed using the Java programming language. Similarly, Android applications are also developed using Java. As a result, the Java Development Kit (JDK) is the first component that must be installed.

Android Studio development requires the installation of version 8 of the Standard Edition of the Java Platform Development Kit. Java is provided in both development (JDK) and runtime (JRE) packages. For the purposes of Android development, the JDK must be installed.

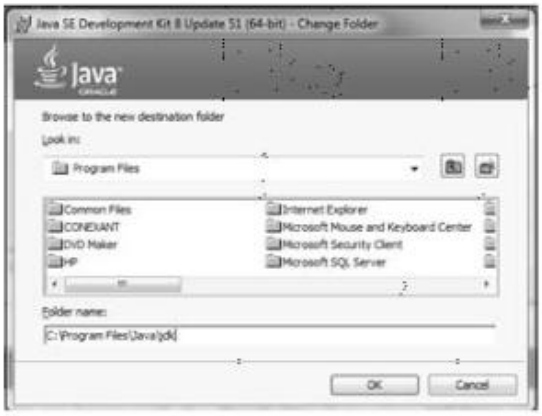
Windows JDK Installation :

For Windows systems, the JDK may be obtained from Oracle Corporation's website using the following URL : *http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html*

1. Assuming that a suitable JDK is not already installed on your system, download version 8 of the JDK package that matches the destination computer system. Once downloaded, launch the installation executable and follow the on screen instructions to complete the installation process.
2. There will be a series of windows that will guide us through the installation process. The most we have to do is just click on Next to proceed. The following is a screenshot of the first window that you will see during the installation :

3. Now, click on Next and you will see this window :

4. click on the **Change** button and you will see this window :



5. Browse to the hard drive where you will be installing all of your development tools. Then simplify the names of the folders in which you will install the JDK to just Java\JDK\.

Folder name :

D:\Java\JDK	

6. After choosing installation location and folder names, click on Next.

7. Click on Next.
8. Next, you will see the window that says 3 Billion Devices Run Java.

9. Now, you will see the final screen.

10. Click on **Close**, and we are almost done installing the JDK.
11. Now, we will make sure that Windows (and all its applications) know where to find the JDK. Right-click on your My Computer icon and **Properties** | **Advanced system settings** | **Environment Variables** | **New (under System variables** and not under **User variables**). Now you can see the **New System Variable** dialog as follows :

12. As shown in the above screenshot, type JAVA_HOME in **Variable name** and enter D:\Java\JDK in the **Variable value** field. If you've installed the JDK somewhere else, then the file path you enter in the **Variable value** field will need to point to where you've placed it. Be sure to type it correctly, make sure that the slashes \ are the right way around, and don't add any extra slashes.
13. Click on **OK** to save your new settings. Now, click on **OK** again to clear the **Advanced system settings** window.

2.6.3 Setting up Android Studio

1. Visit https://developer.android.com/sdk/index.html and click on the **Download Android Studio for Windows** button. If at the time of reading this the link has changed, simply Google Download Android Studio.
2. Next, you will see the **Terms and Conditions** page as shown in the following screenshot :

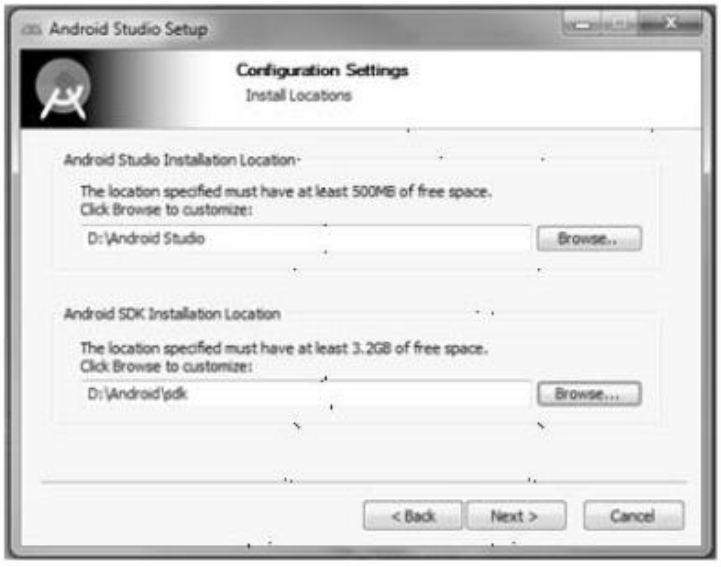
3. Click on the "**I have read and agree with the above terms and conditions**" checkbox as highlighted in the above screenshot.
4. Now, click on the **DOWNLOAD ANDROID STUDIO FOR WINDOWS** button. Wait for the download to complete.

5. Open the folder where you have downloaded Android Studio. Right-click on the android-studio-bundle-141.1980579 -windows.exe file and select **Run as administrator**. Your file will most likely have a different name based on whichever is the current version of Android Studio at the time.
6. When you get the **Do you want the following program to be allowed to make changes to this computer** message, click on **Yes**. You will see the first window of the installation process.
7. Let's step through the setup process a window at a time. Pictured next is the first window that you will see when you start the setup process :

8. Click on Next. Now we can see a few options, as in this next screenshot :

9. Make sure that all the options have a tick next to them, and then click on Next.
10. The next window is the license agreement. Click on "I Agree". Take a look at the next screenshot that shows you the Install Locations window :

11. In this step, we want to install the Android Studio IDE and Android SDK to the same hard drive where we installed the JDK. So you might just be able to click on Next at this point. However, if you've installed the JDK to another drive, then we need to change the drive and the folders we use at this step too. This isn't strictly essential, but it can avoid problems for some users.
12. For Android Studio Installation Location, choose the root of the drive where you've installed the JDK followed by \Android Studio. So in my case, this will be D:\Android Studio. For Android SDK Installation Location, choose the same hard drive and simply add Android\sdk as the location. So if, like me, you've installed the JDK on D:, then choose D:\Android\sdk. The next screenshot makes this clear :



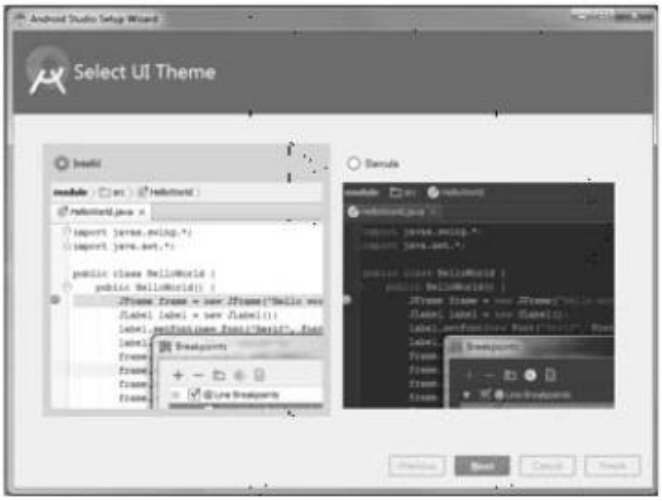
13. Click on Next when you have selected your installation locations.
14. Next, you might see the Emulator Setup window as pictured in the next figure. If you do, then accept the default settings and click on Next; otherwise, you can skip to step 15. Don't worry if you don't see this screen, it is a minor issue to do with running the Android emulators a bit more smoothly. Most of the time, you will probably want to use a real device anyway.

15. The next window asks you to choose a start menu folder, just as when we install any new Windows app. You might want to make a note of this location. Click on Install to accept the default settings, and Android Studio will begin to install itself and extract the SDK to the appropriate folder that we selected earlier. This might take some time.
16. When you see the Installation Complete window, click on Next. Now, you will see the following window :

17. Click on Finish to bring up the second to last window of the installation process. Assuming that this is your first time using Android Studio, click on the "I do not have a previous version of Android Studio" or "I do not want to import my settings" radio button and click on OK.

18. Now, in the next screenshot, you get to choose the theme that Android Studio will use. If you like conventional black texts on white background appearance, then choose IntelliJ, and if you want a

cool dark style, choose Darcula. You can alter any of these schemes from within Android Studio if you change your mind later.

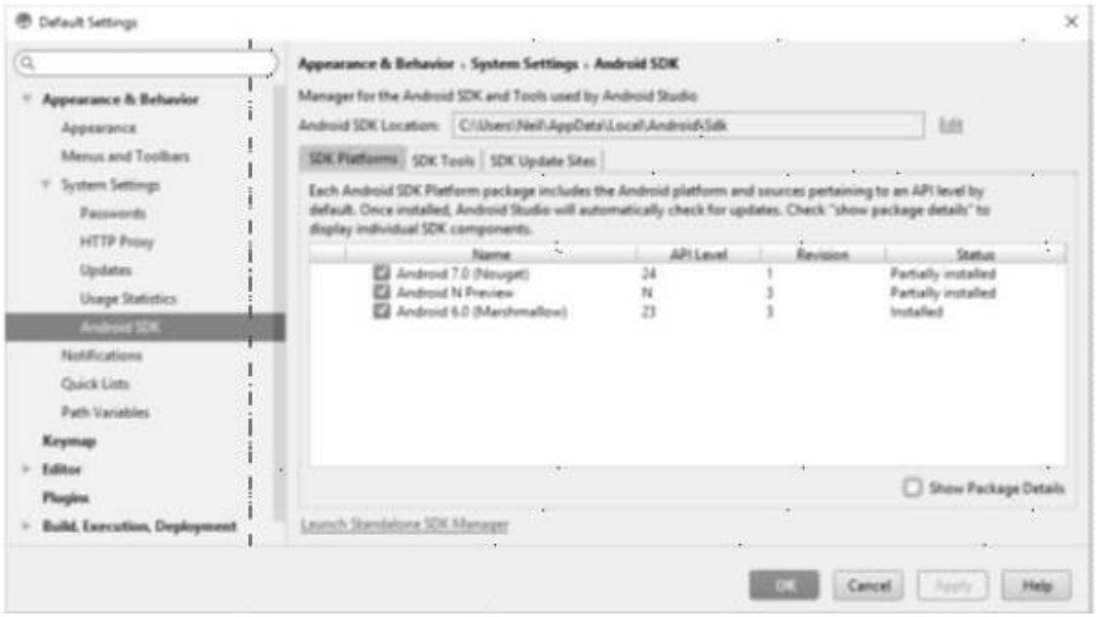


19. Click on Next when you have chosen your theme.
20. Now Android Studio will connect to the Internet and download some of the Android tools that we will be using soon. Again, this could take a while.
21. When the Downloading Components window has done its work, it will present you with a Finish button, Click on it.
22. Finally, we are presented with the Welcome to Android Studio screen. This screen, among other things, allows us to start a new project or open an existing project. Take a look at the next screenshot :

2.6.4 Installing Additional Android SDK Packages

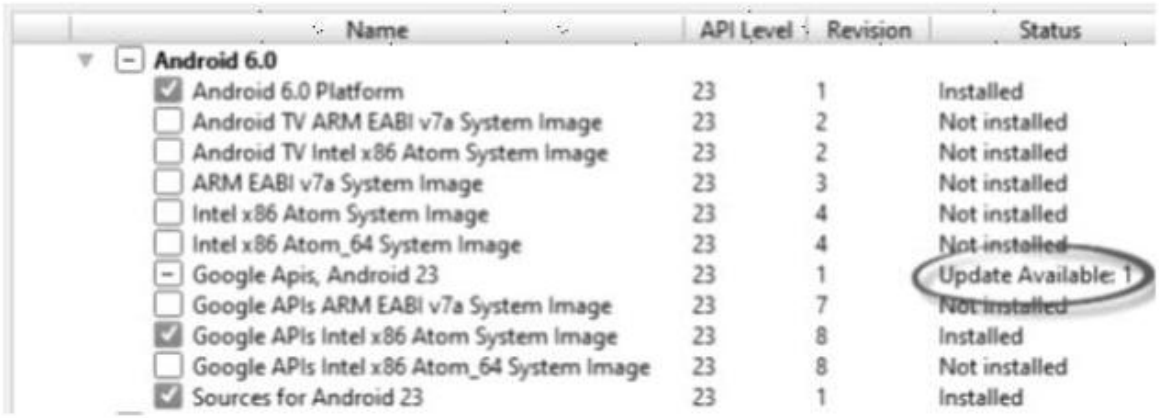
The steps performed so far have installed Java, the Android Studio IDE and the current set of default Android SDK packages. Before proceeding, it is worth taking some time to verify which packages are installed and to install any missing or updated packages.

This task can be performed using the Android SDK Settings screen, which may be launched from within the Android Studio tool by selecting the Configure -> SDK Manager Option from within the Android Studio welcome dialog. Once invoked, the Android SDK screen of the default settings dialog will appear as shown below :



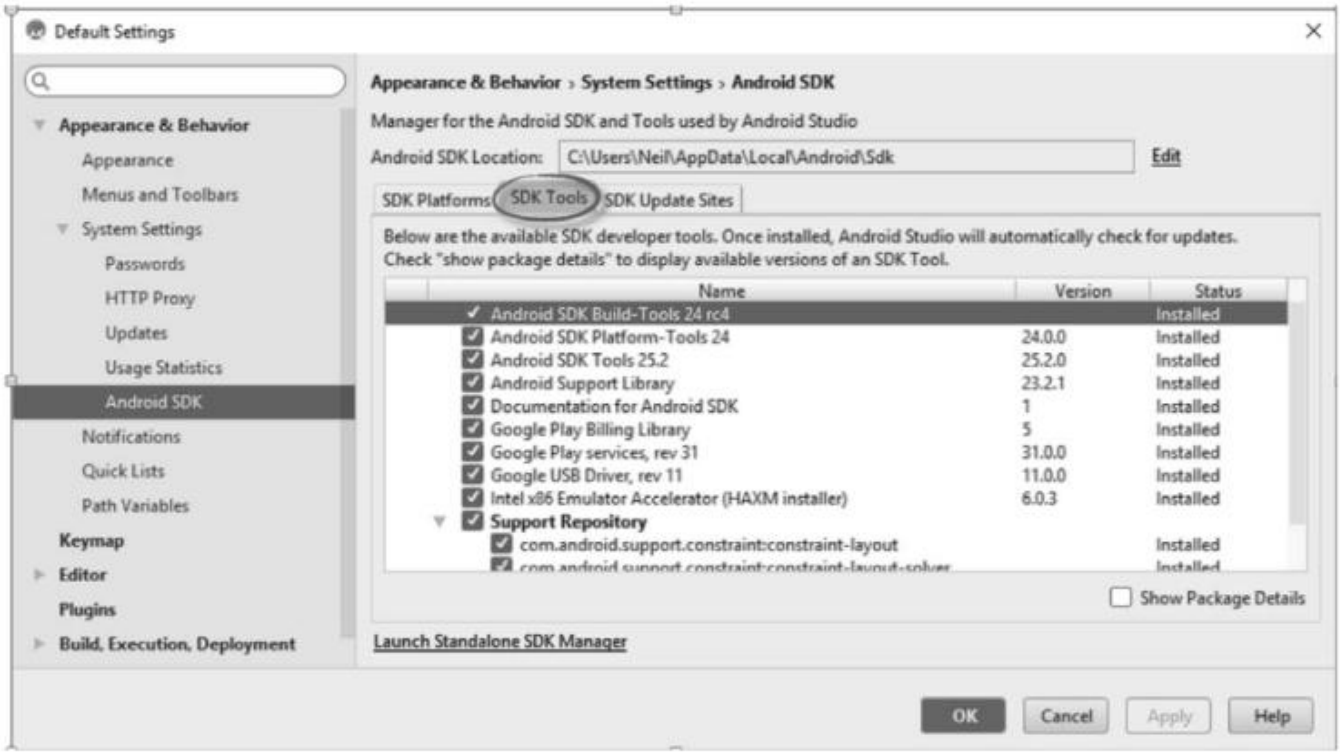
Immediately after installing Android Studio for the first time it is likely that only the latest released version of the Android SDK has been installed. To install preview or older versions of the Android SDK simply select the checkboxes corresponding to the versions and click on the Apply button.

It is also possible that updates will be listed as being available for the latest SDK. To access detailed information about the packages that are available for update, enable the Show Package Details option located in the lower right hand corner of the screen. This will display information similar to that shown in Figure.



The above figure highlights the availability of an update. To install the updates, enable the checkbox to the left of the item name and click on the Apply button.

In addition to the Android SDK packages, a number of tools are also installed for building Android applications. To view the currently installed packages and check for updates, remain within the SDK settings screen and select the SDK Tools tab as shown below :



Setting up an Android Studio Development Environment

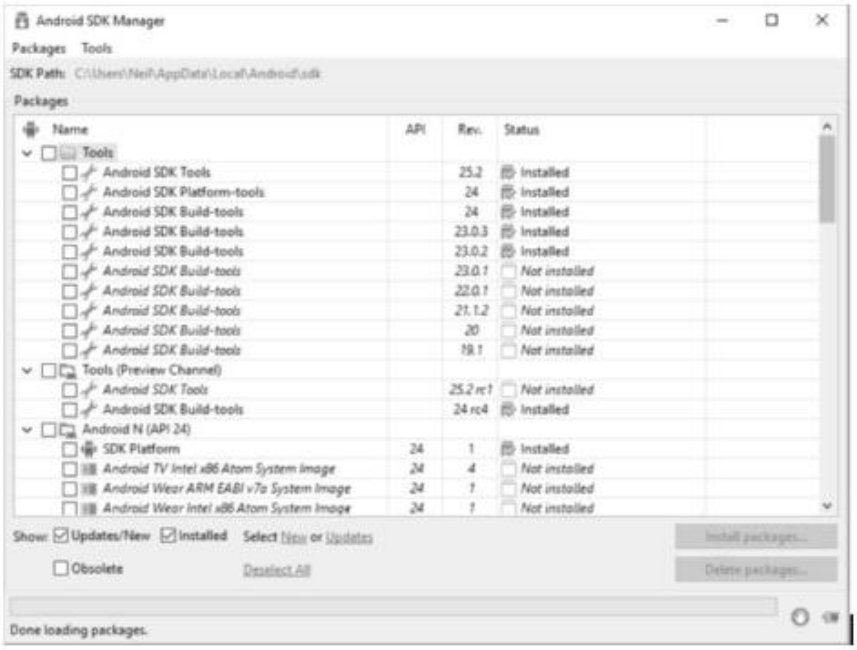
Within the Android SDK Tools screen, make sure that the following packages are listed as Installed in the Status column :

- Android SDK Build-tools
- Android SDK Tools
- Android SDK Platform-tools
- Android Support Repository
- Android Support Library
- Google Repository
- Google USB Driver (Windows only)
- Intel x86 Emulator Accelerator (HAXM installer)

In the event that any of the above packages are listed as Not Installed or requiring an update, simply select the checkboxes next to those packages and click on the Apply button to initiate the installation process.

Once the installation is complete, review the package list and make sure that the selected packages are now listed as Installed in the Status column. If any are listed as Not installed, make sure they are selected and click on the Install packages... button again.

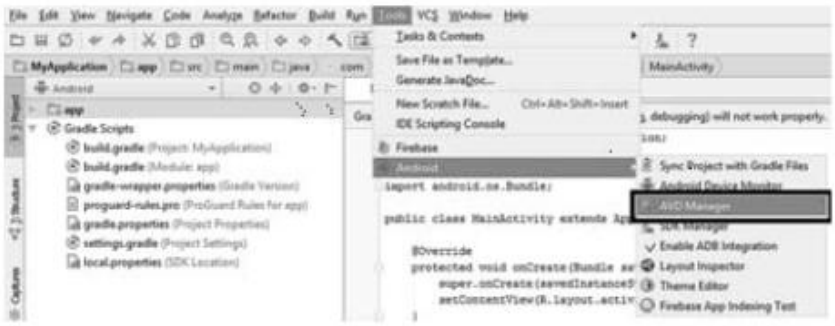
An alternative to using the Android SDK settings panel is to access the Standalone SDK Manager which can be launched using the link in the lower left hand corner of the settings screen. The Standalone SDK Manager provides a similar list of packages together with options to perform update and installation tasks :



2.6.5 Installation of Emulators

Emulators are software that mimics the behaviour of real devices. When we develop an app, we obviously won't have all the possible devices (Android phones, tablets, etc.) available at hand. Because of this, we run the apps on emulators for testing on various devices. Emulators are also called as "Android Virtual Devices (AVDs)" in Android Studio. When Android Studio is first installed, there is no default AVD. We need to create one before testing our apps.

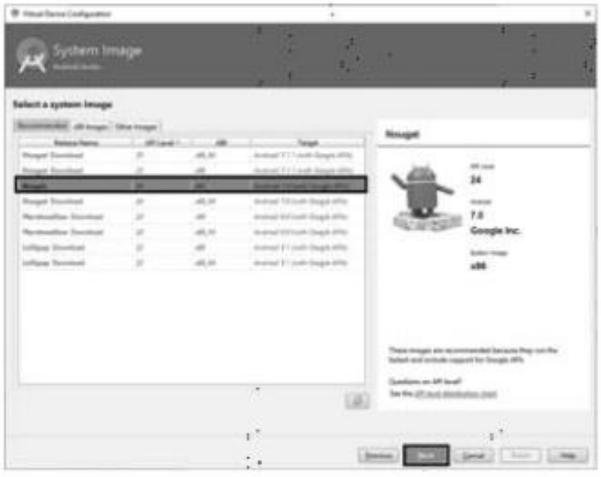
1. For this, select Tools → Android → AVD Manager as shown below :



2. When AVD Manager appears, there won't be any AVDs created or installed. Please click on the **Create a Virtual Device** button as shown below :

3. AVD Manager will show a detailed window.

4. You can select various devices with different screen sizes and other hardware properties. You can select device groups from the left pane as TV, Phone, etc. Phone group is the default selection. In this group, Nexus 5 is also selected by default. When you click "Next", you'll be presented by choices for the Android version of the AVD as shown below.



5. The recommended targets start from Android 5.1. We can Android 7.0 with Goole APIs (Nougat) as shown in the figure. Then, please click "Next" and give a name you like to the AVD.

6. After clicking "Finish", the AVD is created and shown in the AVD Manager as

7. You can now try your Android apps on this AVD, which will accurately mimic the behaviour of a real Nexus 5 phone. We can run the AVD by clicking the "Play" button shown inside the square in above Figure
8. The virtual device will appear as in Figure below which you can use like a real Nexus 5 phone.

❑❑❑

UNIT - III

3 Components and Layouts

3.1 The Development Process

An Android app project begins with an idea and a definition of the requirements necessary to realize that idea. As the project progresses, it goes through design, development, and testing.

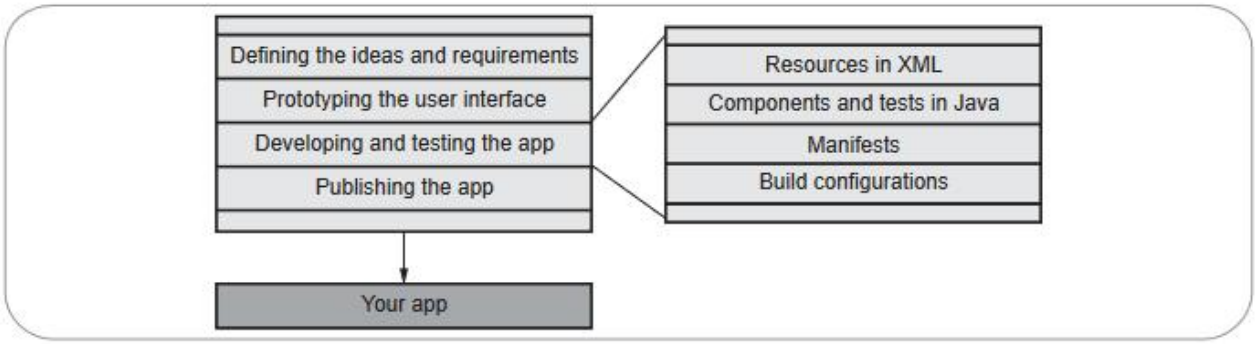


Fig. 3.1.1 The development process

The above diagram is a high-level picture of the development process, with the following steps :

1. **Defining the idea and its requirements :** Most apps start with an idea of what it should do,bolstered by market and user research. During this stage the app's requirements are defined.
2. **Prototyping the user interface :** Use drawings, mock ups and prototypes to show what the user interface would look like, and how it would work.
3. **Developing and testing the app :** An app consists of one or more activities. For each activity you can use Android Studio to do the following, in no particular order:
   - **Create the layout :** Place UI elements on the screen in a layout, and assign string resources and menu items, using the Extensible Markup Language(XML).
   - **Write the Java code :** Create source code for components and tests, and use testing and debugging tools.
   - **Register the activity :** Declare the activity in the manifest file.
   - **Define the build :** Use the default build configuration or create custom builds for different versions of your app.
   - **Publishing the app :** Assemble the final APK (package file) and distribute it through channels such as the Google Play.

General steps of app development for developing our first app :

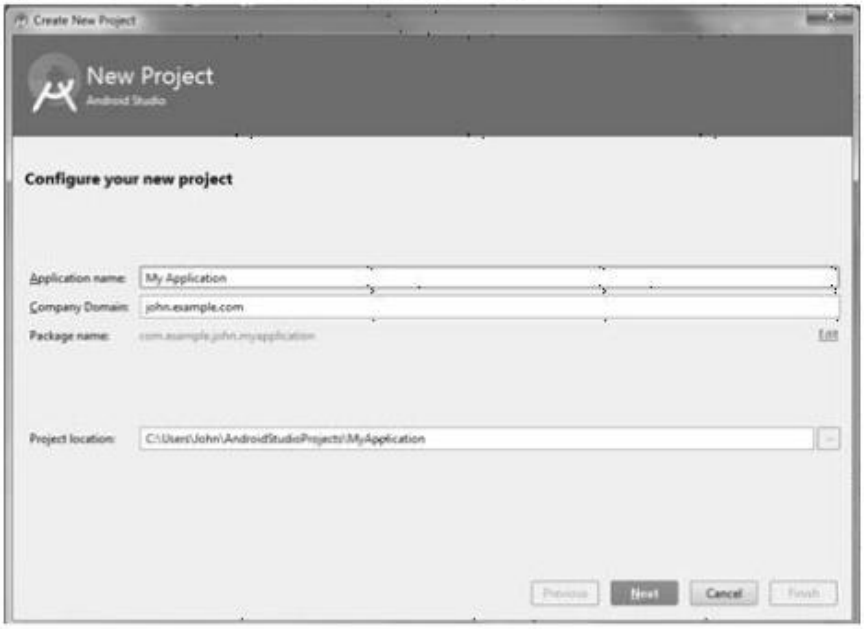
1. Creating an Android Studio project.
2. Setting up the User Interface (UI) of the app.
3. Connecting the UI components such as buttons, textboxes, etc. to the Java code.
4. Coding in Java - the actual programming part.

5. Building the project : This means creating the executable (file that actually runs on device or the emulator). This is not difficult as it sounds; Android Studio does the entire job with a single click
6. Trying the app on an emulator.
7. Running the app on a real Android device (optional).
8. Publishing the app on Google Play (optional).

3.1.1 Creating the Project

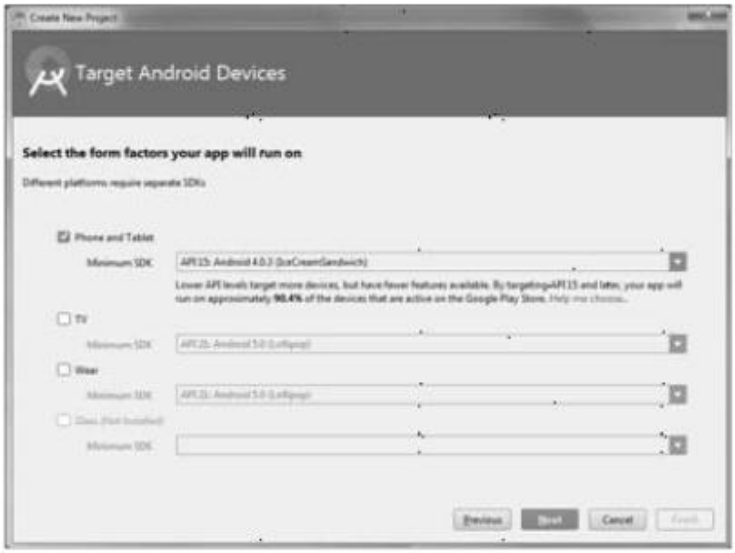
1. Start Android Studio by clicking on its icon in the start menu.
2. If you get the **Windows Firewall has blocked some features of this program** message, as pictured in the next screenshot, click on **Allow access :**

3. Now, you will see the **Welcome to Android Studio** start menu ,click on **Start a new Android Studio project.** You will see the **New Project** screen pictured next.



4. First, give name to application in the **Application name** field.
   You can choose any name you like for your first application, but just be aware that Android Studio will soon generate a whole bunch of code and files for us and that the name you choose will be reflected in them. If you want your code and files to be identical to those that we will be examining shortly, call your application Hello Android.
5. Next, the **Company Domain** field. This is where you will enter the details of your company. It is a convention and very practical (because it is unique) to use the domain name of your company website. Unlike the application name, using a different company domain will have almost zero effect on the code and files. Choose and enter a company domain name.
6. Now, the **Package name** field. It has been automatically derived from the previous two fields. Remember that a package is a collection of the Java classes that are our code files, and our apps can comprise one or more packages if we wish them to, but they must comprise at least one package. We can edit the package name by clicking on the edit link, but we have no need to do so here.
7. Finally, in the **Project location** field, you can accept the default settings or browse to the location where you would like to store all your Android projects. You have the option to change this for each project that you create.

8. Click on the Next button to continue and you will see the Target Android Devices window:

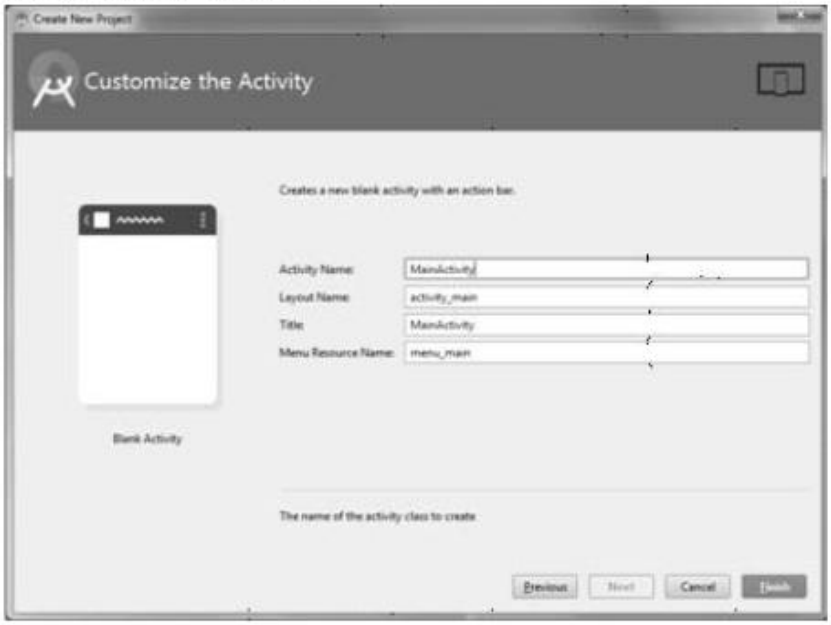


9. Here, we can see that we have the option of developing an application for a phone, tablet, TV, and wear. Wear is the range of Android-enabled smartwatches. In the preceding screenshot, in the grayed-out part at the bottom of the window, we can also see that we can develop for Glass, Google's trendy Android-enabled glasses, although we would need to install additional files to do this. In this book, we will be developing for phones and tablets, so the already selected option is just what we need. The only other thing we need to choose on this screen is the Minimum SDK.

We already know that the Android SDK is a collection of packages of code that we will be using to develop our apps. Like any good SDK, the Android SDK is regularly updated, and each time it gets a significant update, the version number is increased. Simply put, the higher the version number, the newer the features you get to use; the lower the version number, the more devices our app will work on. For now, the default setting API 15: Android 4.0.3 (IceCreamSandwich) will give us lots of great features and at least 90% compatibility with the Android devices that are currently in use.

10. Click on the Next button. Now we can see the Add an activity to Mobile window :

11. An Activity class is a special Java class and every Android app must have at least one. It is the part of the code where our app will start when it is launched by the user, and this also handles any interaction with the user. The options on this screen provide different ready-made templates of the Activity class code in order to give programmers a fast start when creating various types of apps. As we are starting from scratch, the most appropriate option for us is Blank Activity. Make sure that Blank Activity is selected by clicking on it and then clicking on Next. Now, take a look at the Customize the Activity window :



12. The Customize the Activity window gives us four things. It is perfectly possible to leave them all at their default settings. Activity Name is the name of the class that will contain our code. Name the activity MyActivity.
13. The next field is Layout Name. Android UI layouts are usually defined in a separate XML text file, not with our Java code. The layout name is what this file will be called. Name the layout my_layout.
14. The next field is Title. This is different than the Activity Name field and will be used by Android on the device's screen as the name of the app. Name the title My App.
15. Finally the Menu Resource Name. Menus are the pop-up options that you get on Android when you click on the menu button. They might also be shown on the topmost bar of the app known as the action bar. This varies depending on the version of Android a device is running. These are considered to be part of the UI as well and are usually defined in this separate XML file. Name the menu file my_menu.
16. Click on the Finish button, and Android Studio will now create a project for us based on our choices.

Lets get familier with the main sections of Android Studio

1. **Main Sections of the IDE :** Android Studio is a sophisticated tool and it has dozens of properties to make app development easier. The main sections of Android Studio is as shown in Figure 3.1.2

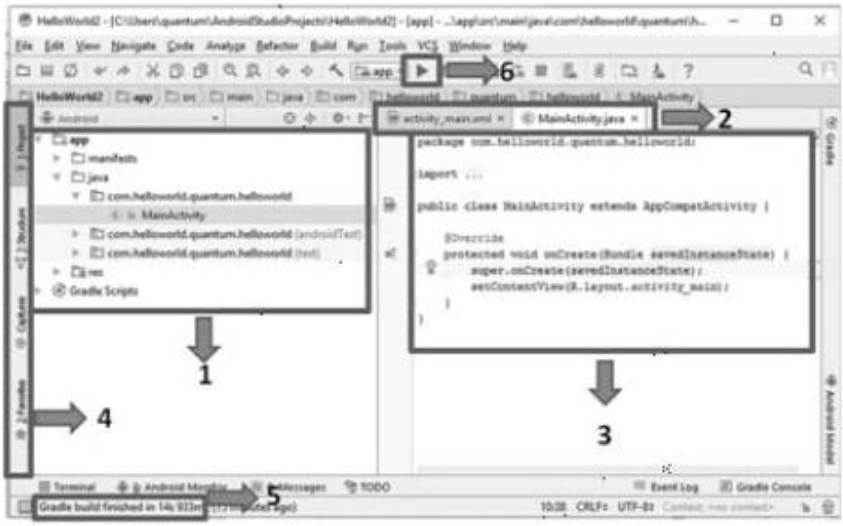


Fig. 3.1.2 Basics sections of Android Studio

Section 1. The project files and folders can be viewed from here. In addition, new files can be added from this pane. We need to double-click on the filenames here to open them in the middle pane. The project structure will be explained in detail in the next subsection.

Section 2. The opened files can be activated from the tabs located here for viewing in the middle pane

Section 3. This is the middle pane. Contents of the active files can be viewed and changed from here. For the project shown in Figure 3.1.2, the file called "MainActivity.java" is the active tab in Section 2 therefore the middle pane in Section 3 shows the contents of this "MainActivity.java" file.

Section 4. This section is also controlled via tabs. The developer can switch project files, structures, captures and favourites for viewing in the left pane.

Section 5. The current or previous compilation, building or debugging processes are shown here. For the snapshot of Figure 3.1.2, it is indicated that the "Gradle build finished in 14 seconds". Gradle is the build system of Android Studio. Therefore, the message says that the building engine completed its previous task in 14 seconds.

Section 6. This is the Run button of Android Studio. When we set up the user interface and write the Java code of a project, we click this button to make the Android Studio build the project (which means creating the executable file from project files) and then we can run it on an emulator or on a real device.

2. **The Tool Windows :** In addition to the project view tool window, Android Studio also includes a number of other windows which, when enabled, are displayed along the bottom and sides of the main window. The tool window quick access menu can be accessed by hovering the mouse pointer over the button located in the far left hand corner of the status bar (Fig.3.1.3) without clicking the mouse button(pointed by the arrow).

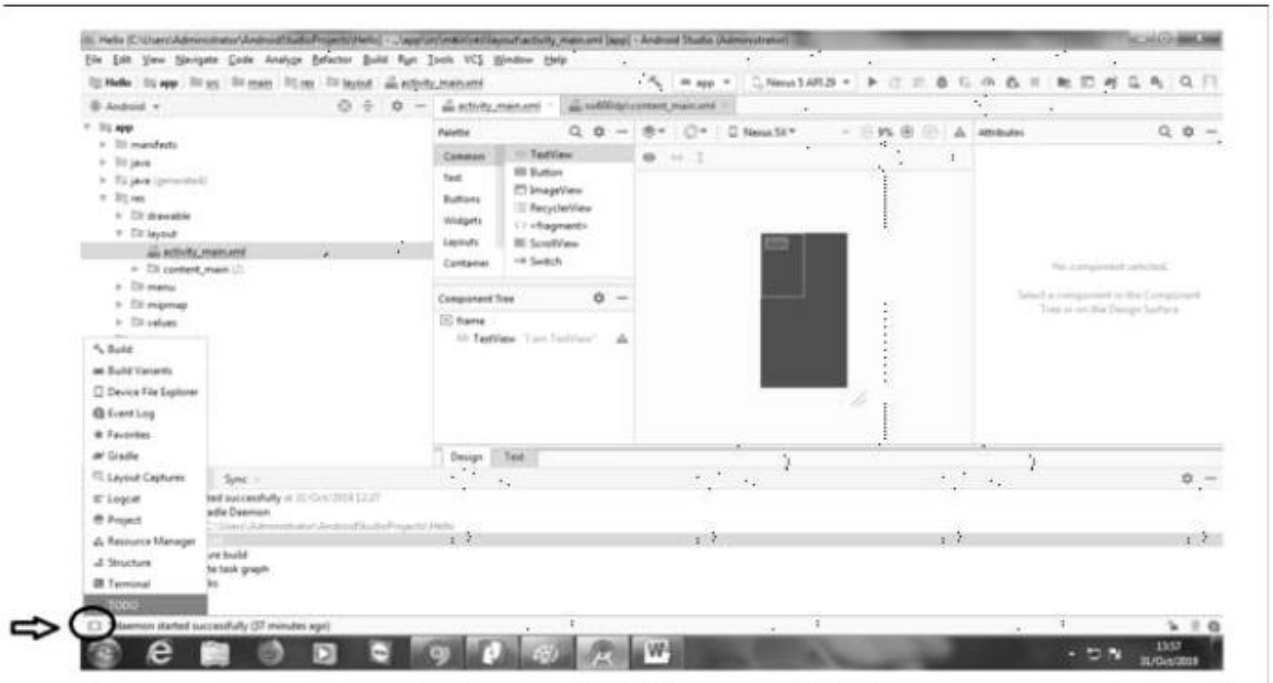


Fig. 3.1.3 The Tool Window of Android Studio

Selecting an item from the quick access menu will cause the corresponding tool window to appear within the main window. Alternatively, a set of tool window bars can be displayed by clicking on the quick access menu icon in the status bar. These bars appear along the left, right and bottom edges of the main window (as indicated by the arrows in Fig 3.1.4) and contain buttons for showing and hiding each of the tool windows. When the tool window bars are displayed,a second click on the button in the status bar will hide them.

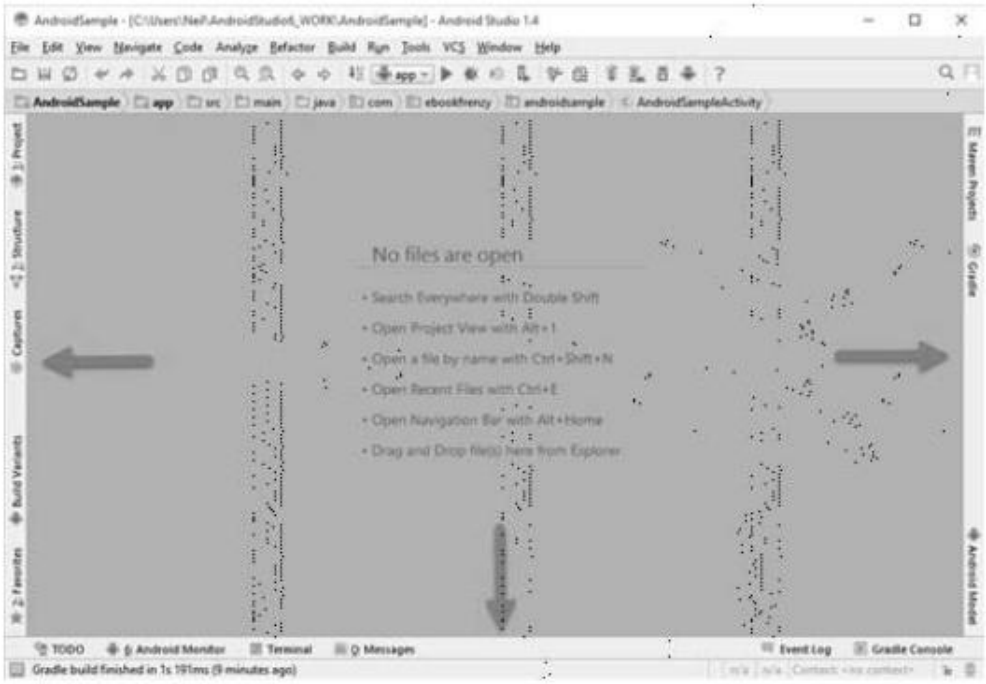


Fig. 3.1.4 Tool window bars

Clicking on a button will display the corresponding tool window while a second click will hide the window. Buttons prefixed with a number (for example 1: Project) indicate that the tool window may also be displayed by pressing the Alt key on the together with the corresponding number. The location of a button in a tool window bar indicates the side of the window against which the window will appear when displayed. These positions can be changed by clicking and dragging the buttons to different locations in other window tool bars.Each tool window has its own toolbar along the top edge. The buttons within these toolbars vary from one tool to the next, though all tool windows contain a settings option, represented by the cog icon,which allows various aspects of the window to be changed.

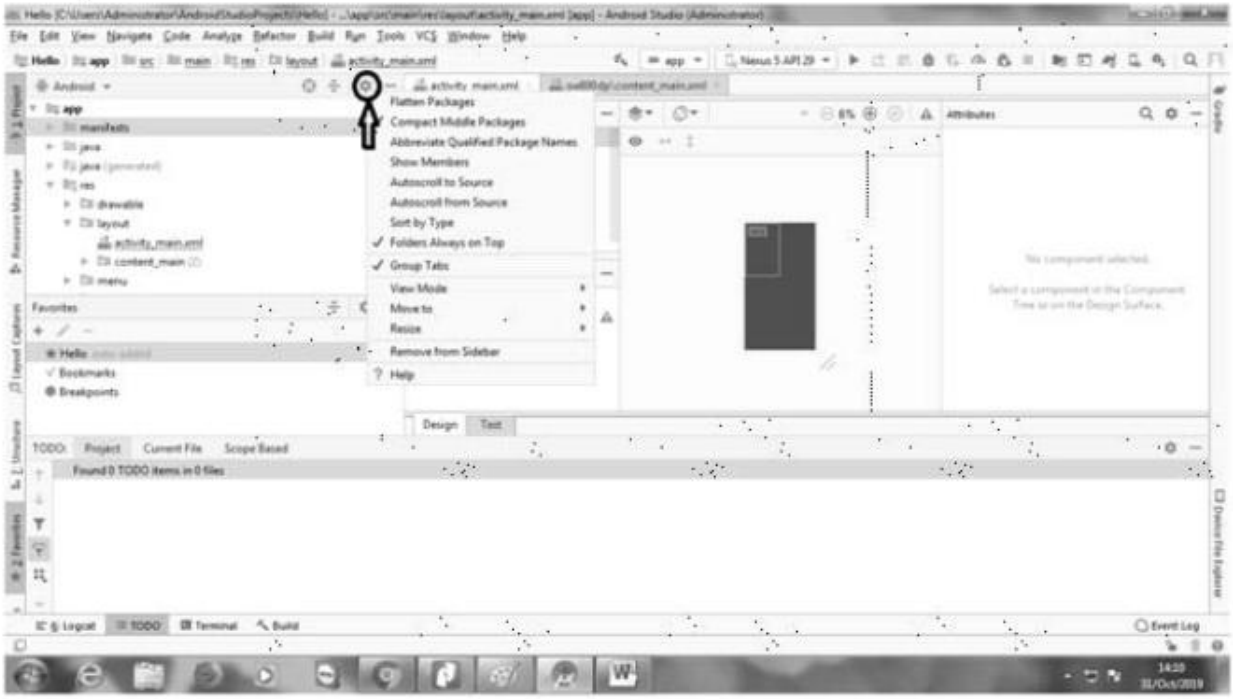


Fig. 3.1.5 Settings menu for the project view tool window

Fig. 3.1.5 shows the settings menu for the project view tool window(pointed by arrow). Options are available, for example, to undock a window and to allow it to float outside of the boundaries of the Android Studio main window and to move and resize the tool panel.

All of the windows also include a far right button on the toolbar providing an additional way to hide the tool window from view. A search of the items within a tool window can be performed simply by giving that window focus by clicking in it and then typing the search term (for example the name of a file in the Project tool window). A search box will appear in the window's tool bar and items matching the search highlighted.Android Studio offers a wide range of window tool windows, the most commonly used of which are as follows:

Project - The project view provides an overview of the file structure that makes up the project allowing for quick navigation between files. Generally, double clicking on a file in the project view will cause that file to be loaded into the appropriate editing tool.

Structure - The structure tool provides a high level view of the structure of the source file currently displayed in the editor. This information includes a list of items such as classes, methods and variables in the file. Selecting an item from the structure list will take you to that location in the source file in the editor window.

Captures - The captures tool window provides access to performance data files that have been generated by the monitoring tools contained within the Android Monitor tool window.

Favorites - A variety of project items can be added to the favorites list. Right-clicking on a file in the project view, for example, provides access to an Add to Favorites menu option. Similarly, a method in a source file can be added as a favourite by right-clicking on it in the Structure tool window. Anything added to a Favorites list can be accessed through this Favorites tool window.

Build Variants - The build variants tool window provides a quick way to configure different build targets for the current application project (for example different builds for debugging and release versions of the application, or multiple builds to target different device categories).

TODO - As the name suggests, this tool provides a place to review items that have yet to be completed on the project. Android Studio compiles this list by scanning the source files that make up the project to look for comments that match specified TODO patterns. These patterns can be reviewed and changed by selecting the File -> Settings... menu option and navigating to the TODO page listed under Editor.

Messages - The messages tool window records output from the Gradle build system (Gradle is the underlying system used by Android Studio for building the various parts of projects into runnable applications) and can be useful for identifying the causes of build problems when compiling application projects.

Android Monitor - The Android Monitor tool window provides access to the Android debugging system. Within this window tasks such as monitoring log output from a running application, taking screenshots and videos of the application,stopping a process and performing basic debugging tasks can be performed. The tool also includes real-time GPU, networking, memory and CPU usage monitors.

Android Model - The Android Model tool window provides a single location in which to view an exhaustive list of the different options and settings configured within the project. These can range from the more obvious settings such as the target Android SDK version to more obscure values such as build configuration rules.

Terminal - Provides access to a terminal window on the system on which Android Studio is running. On Windows systems this is the Command Prompt interface, while on Linux and Mac OS X systems this takes the form of a Terminal prompt.

Run - The run tool window becomes available when an application is currently running and provides a view of the results of the run together with options to stop or restart a running process.If an application is failing to install and run on a device or emulator, this window will typically provide diagnostic information relating to the problem.

Event Log - The event log window displays messages relating to events and activities performed within Android Studio. The successful build of a project, for example,or the fact that an application is now running will be reported within this tool window.

Gradle Console - The Gradle console is used to display all output from the Gradle system as projects are built from within Android Studio. This will include information about the success or otherwise of the build process together with details of any errors or warnings.

Gradle - The Gradle tool window provides a view onto the Gradle tasks that make up the project build configuration. The window lists the tasks that are involved in compiling the various elements of the project into an executable application. Right-click on a top level Gradle task and select the Open Gradle Configmenu option to load the Gradle build file for the current project into the editor.

3. **Fundamental User Interface Design :** Anroid Studio provides an easy way of designing user interfaces. The file named "activity_main.xml" located under the "res/layout" folder contains all the layout information of the current activity. If we try to open an .xml file outside of Android Studio, it is opened by a text editor or a browser. However, when we open an .xml file in Android Studio, it reads the .xml file and shows the corresponding activity layout with its components. In order to open the activity_main.xml in Android Studio, double-click on it in the project explorer and the activity layout will be displayed in the middle pane as shown below :

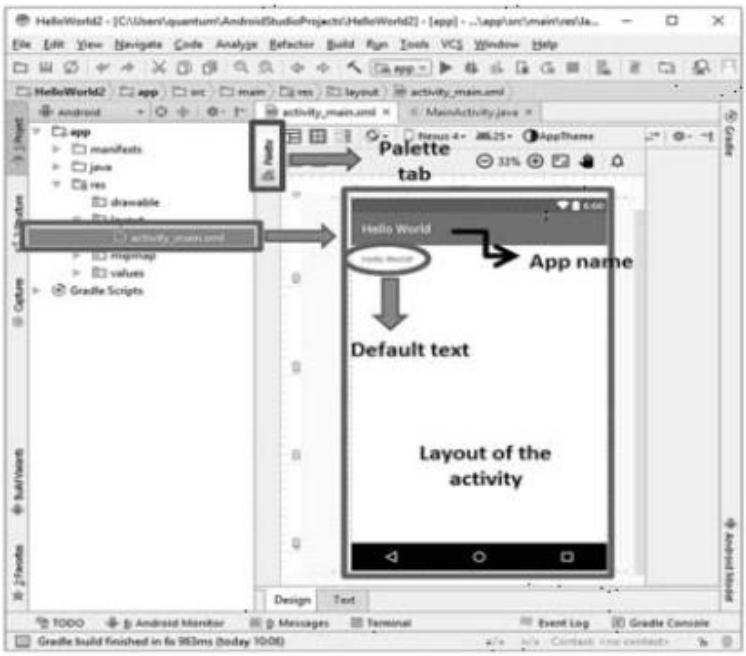


Fig. 3.1.6 Layout of the activity

As you can see, the layout of the activity is shown in the middle pane. The name of the app appears at the top of the activity. The default empty activity contains a default text which is shown inside the circle in the above figure. At the left top of the middle pane, there exists a tab called "Palette" indicated inside the rectangle in the figure. When we click on this tab, the palette shown in Figure Fig. 3.1.7 appears from which we can add all possible user interface objects and layout templates to the activity.

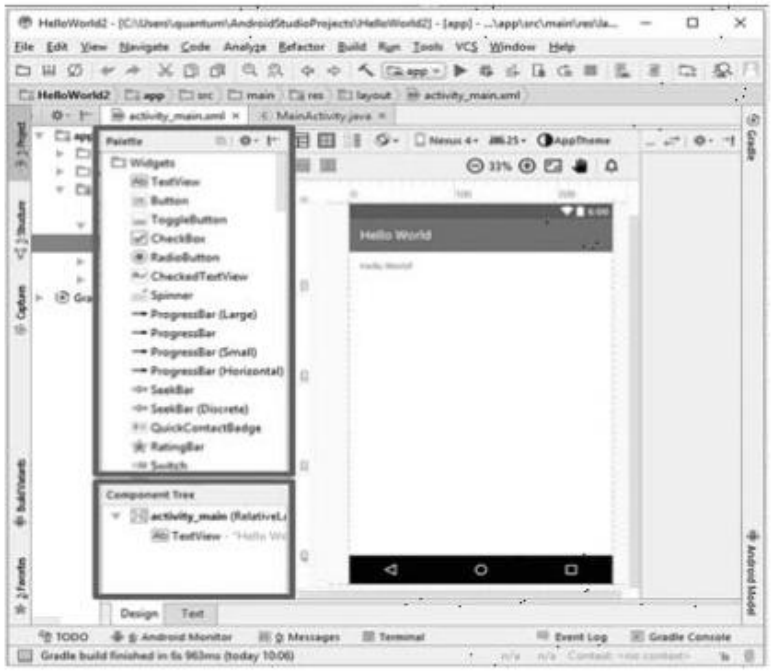


Fig. 3.1.7 The component palette

When the palette tab is clicked, two panes are opened: the Palette shown by the upper rectangle and the Component Tree pane inside the lower rectangle in Figure Fig. 3.1.7. The Palette contains several groups like Widgets, Text Fields and Layouts. We can easily drag and drop these components to the user interface. On the other hand, the Component Tree lists the activity's components in a hierarchical manner. As you can see from Fig. 3.1.7, Android Studio already placed a "Hello World" text at the top left of the view.

Let's position this text, comprised of a **TextView** widget, to the middle of the view. For this, select this TextView and then drag and drop to the middle by the help of the guiding lines as shown below :

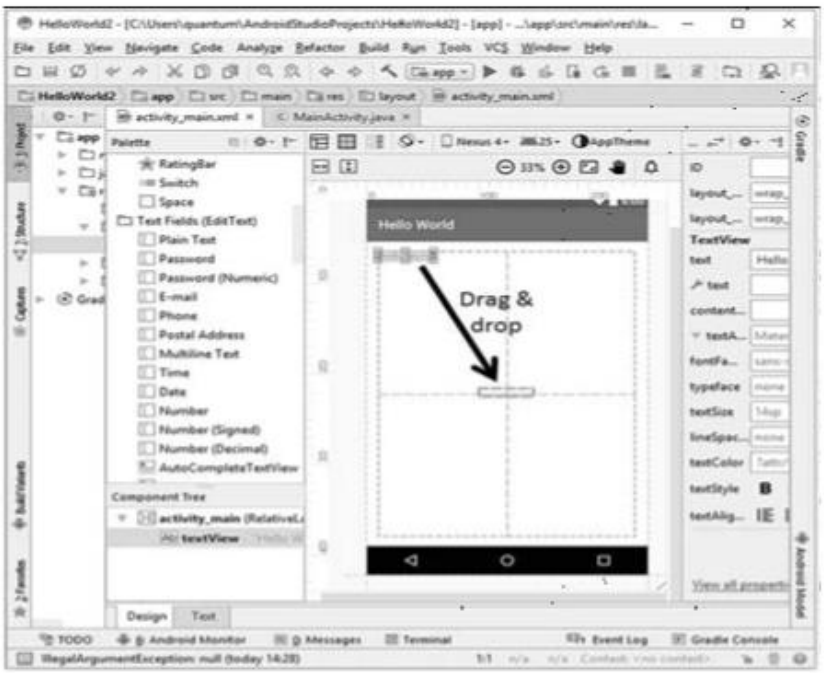


Fig. 3.1.8 Drag and drop operation on the TextView

After the drag and drop operation, the TextView will be kept selected. We can now change the properties of the TextView using the Properties pane which is at the right of the Layout view as shown inside the rectangle in Fig. 3.1.9. Click the arrow shown inside the circle in this figure to open the basic editable properties of the TextView.

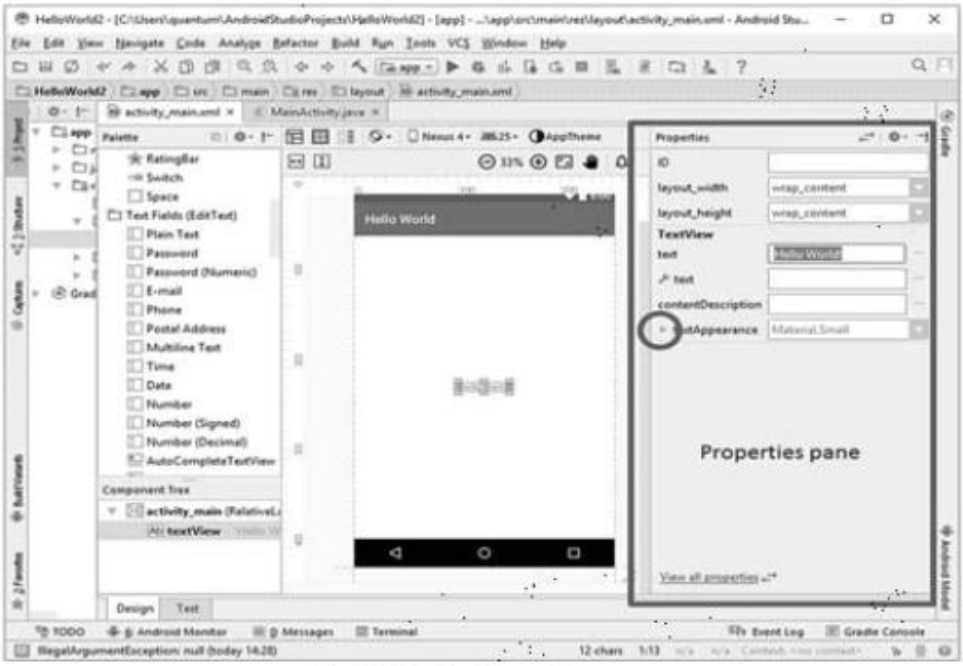


Fig. 3.1.9 The Properties pane

The editable properties of the TextView component are shown inside the rectangle in Figure.

4. **Building the Project and Running on an Emulator :** To run our first Android app on an emulator follow the steps given below :

   i) Building the project, ii) Selecting the emulator

   iii) Run our app on the emulator.

Fig. 3.1.10 The editable properties of the TextView

In order to build and run the project, please click the "Run" button as indicated by the arrow in Fig. 3.1.10 The emulator and device selection dialog shown in Fig. 3.1.11 will appear. When the emulator starts running, you will see a Nexus 5 screen as shown in Fig. 3.1.12.

Fig. 3.1.11 Selecting the target for running our first app

Fig. 3.1.12 The Nexus 5 emulator

3.1.2 Directory and File Structure of an Android Studio Project

The file structure of an Android project can be viewed in various forms in Android Studio. The button just at above the left pane (shown by the arrow) is used to open the selection box for choosing the preferred method of viewing the file hierarchy as shown in Fig. 3.1.13. The default file viewer is the "Android" mode.

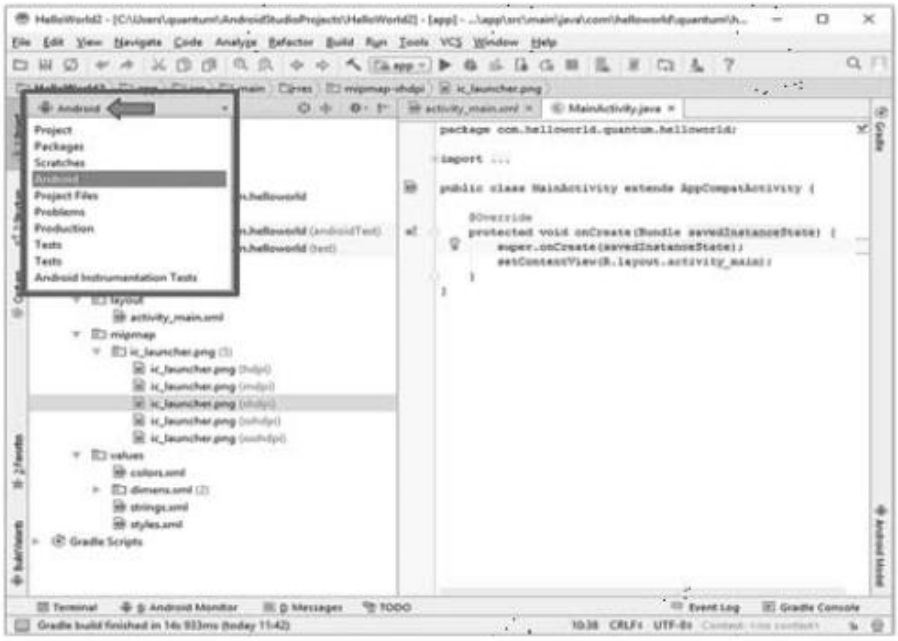


Fig. 3.1.13 Switching among different ways of viewing files and folders

When the selection is the "Android" mode, the default files and folders shown in Fig. 3.1.14

Fig. 3.1.14 Default folder and file structure of an Android project

You can use the arrows (shown inside the circle in the figure)for viewing the contents of folders. The default folders (shown inside the rectangles in Fig. 3.1.14) and their contents are explained as follows :

1. **manifests folder :** This folder has the AndroidManifest.xml file inside. This file contains the configuration parameters of the project such as permissions, services and additional libraries.

2. **java folder :** The source code files written in Java programming language reside in this folder. You can see that the java file of the activity named "MainActivity.java" is automatically created in this folder.
3. **res folder :** The resource files are contained in this folder. Resources basically mean all the needed files except the source code. The media, image and layout files residing in the resources folder are accessed via Java code written in MainActivity.java.

The floder structure includes different types of files

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

- **Java and XML soure files :** These are the activity and layout files for your app.
- **Android-generated Java files :** There are some extra Jave files you don't need to touch that Android STudio generates for you automatically.
- **Resource files :** These include default image files for icons, styles your app might use and any common String values your app might want to look up.
- **Android libraries :** In the wizard, ou specified the minimum SDK version you want your app to be compatible with. Android Studio makes sure your app includes the relevant Android libraries for that version.
- **Configuration files :** The configuration files tell Android what's actually in the app and how it should run.

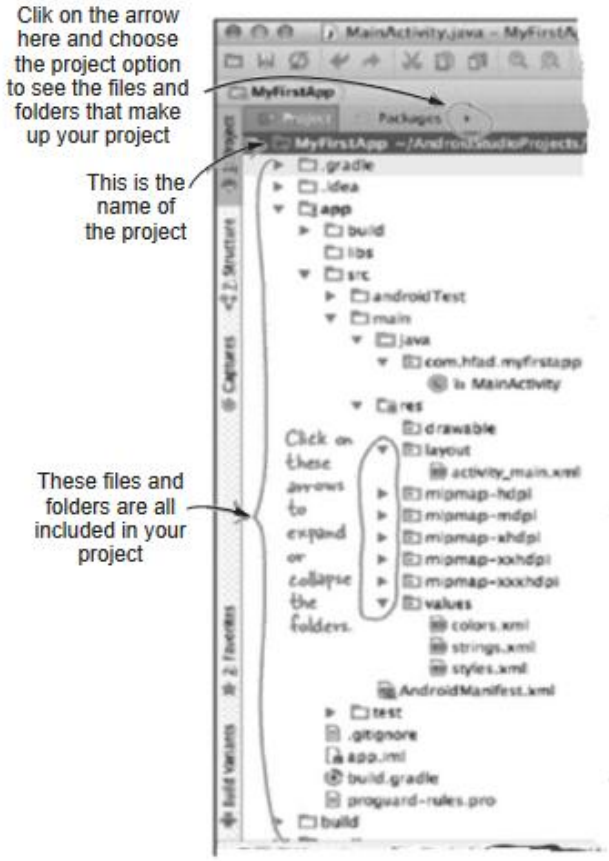


Fig. 3.1.15 File structure of an Android project

Some of the key files and directory(folders) are as shown in Fig. 3.1.16 : (See Fig. on next page)

Let's see each folder in detail :

- src - Java source files associated with your project. This includes the Activity "controller" files as well as your models and helpers.
- res - Resource files associated with your project. All graphics, strings, layouts, and other resource files are stored in the resource file hierarchy under the res directory.
- res/layout - XML layout files that describe the views and layouts for each activity and for partial views such as list items.
- res/values - XML files which store various attribute values. These include strings.xml, dimens.xml, styles.xml, colors.xml, themes.xml, and so on.
- res/drawable - Here we store the various density-independent graphic assets used in our application.
- res/drawable-hdpi - Series of folders for density specific images to use for various resolutions.
- res/mipmap - most commonly used for application icons.

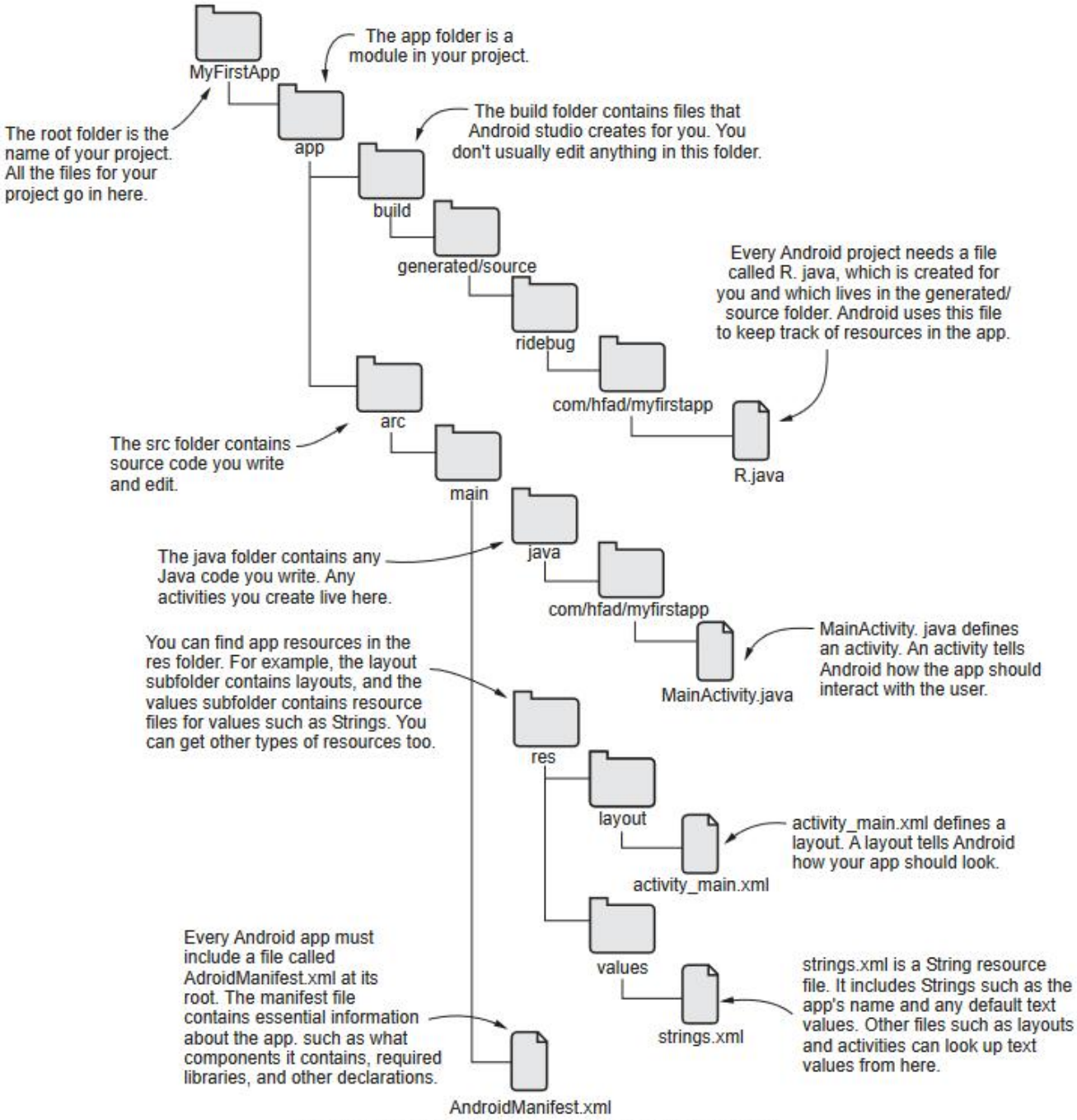


Fig. 3.1.16 Directory structure of an Android project

The most frequently edited files are :

- res/layout/activity_foo.xml - This file describes the layout of the activity's UI. This means the placement of every view object on one app screen.
- src/.../FooActivity.java - The Activity "controller" that constructs the activity using the view, and handles all event handling and view logic for one app screen.
- AndroidManifest.xml - This is the Android application definition file. It contains information about the Android application such as minimum Android version, permission to access Android device capabilities such as internet access permission, ability to use phone permission, etc.

Other less edited folders include :

- gen - Generated Java code files, this library is for Android internal use only.
- assets - Uncompiled source files associated with your project; Rarely used.
- bin - Resulting application package files associated with your project once it's been built.
- libs - Before the introduction of Gradle build system, this directory was used for any secondary libraries (jars) you might want to link to your app.

3.2 Components of Android Application

Almost all popular applications are interactive. These applications interact with the user, and, depending on the data supplied by the user, desired actions and/or processing are performed. The user interface controls thus play a major role in getting feedback from the user. Application components are the ones which when combined together, offers you a brilliant Android application. So, these components exactly act as the building blocks of an Android application. The information regarding all the application components is provided in the manifest file, which is **AndroidManifest.xml**. This file will help you in understanding the use of each and every application component and how do they interact with each other.

Android provides four important components to build any android application.

- Activities
- Services
- Intent and broadcast receivers
- Content Providers

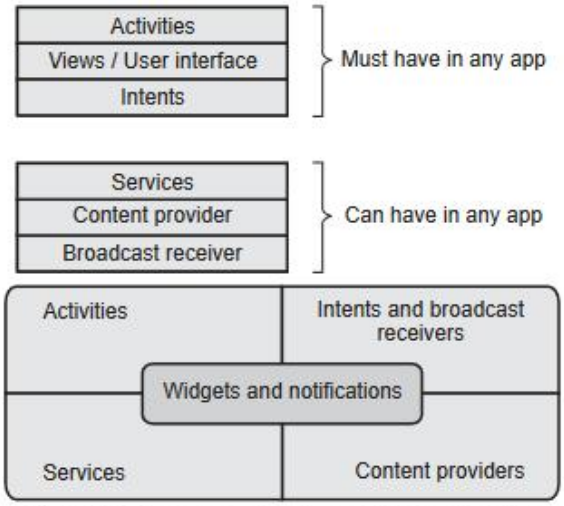


Fig. 3.2.1 Components of android application

1. **Activities :** Activities are said to be the presentation layer of our applications. An activity is the first stepping stone is building an Android user application. The UI of our application is build around one or more extensions of the Activity class.
   An activity in android is like your computer welcome screen which presents single user display. In other words, Activity in android represents single screen with a user interface. We can understand Activity in terms of web applications For example: We creates numbers of web pages to build complete web application, similarly on the other hand android application consist of several Activities to run as a complete application. There is one "main" activity. All other activities are child activities. There is a stack called **back stack**. Whenever, there is a new window is started, previous activity is pushed to the back stack and it is stopped until the new activity is done. As soon as the back key of your device is pressed, new activity is popped out of stack and destroyed. Now previous activity resumes.
2. **Services :** These are like invisible workers of our app. These components run at backend, updating your data sources and Activities, triggering Notification and also broadcast Intents. They also perform some tasks when applications are not active. This component is responsible for handling the time taking operations which generally runs in the background of the operating system (in this case, Android). The simple example of the service component is that when you play music on your mobile phone, you will be able to use other applications too. A service can take two forms:
   a. **Started :** After a service starts, it can run indefinitely and usually performs single operation. No result is returned to user. For example, uploading a file. After the task is completed, it should terminate itself.
   b. **Bound :** In this case, a component is bound to a service so that a particular task can be completed. This type of service provides a client-server like interface. Requests can be send, receive requests, and return result to the user. Inter process communication is achieved through this service.

3. **Intents and Broadcast Receivers -** 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. It binds individual components to each other. Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents. 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. another example is, when your device boots up or switched on, the system generates a broadcast to all apps. There should be a procedure or should be something which can receive these broadcasts. These receptors are called broadcast receivers. There are two types of broadcasts:
   a. **Normal Broadcasts :** These are asynchronous in nature. Many receivers can be activated at the same time which doesn't have any defined order. But they are very efficient.
   b. **Ordered Broadcasts :** They are synchronous in nature. Broadcast received by one receiver passes it to other receivers. Broadcasts are delivered to receiver on one-to-one and sequential basis. Either receiver will pass result to another receiver or it may completely destroy the broadcast.
4. **Content Providers -** It is used to manage and persist the application data also typically interact with SQL database. They are also responsible for sharing the data beyond the application boundaries. The Content Providers of a particular application can be configured to allow access from other applications, and the Content Providers exposed by other applications can also be configured. With content providers we can save data in SQLite database, on the web or any other persistent storage location, where application can easily access the data. This component is useful in reading and writing private data. For example: we can read and write important reminders or notes in database(within an application).
5. **Android Widgets and Notifications : Android App widgets** are the small application views. These views can be embedded into other applications. They can receive updates on periodic basis. A widget is a quick view of your app's functionality and data. This view is accessible from home screen of your device. Now widgets are of following types :

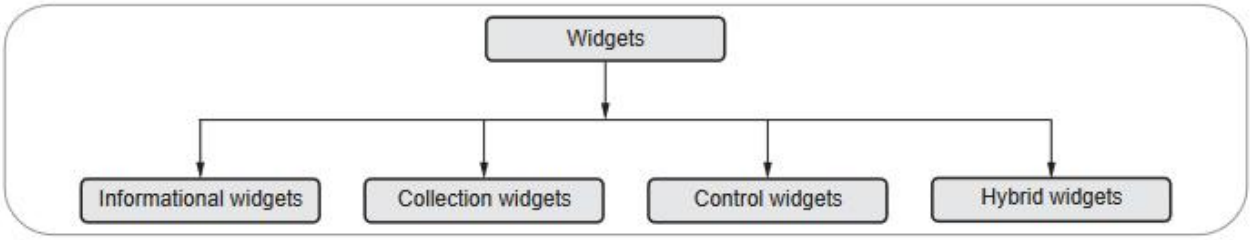


Fig. 3.2.2 Types of widgets

   a. **Informational Widget :** These Android widgets are going to display only that information to user which is important and dynamic in nature. Example the information displayed on your home screen saying time and weather condition is a widget of this type.
   b. **Collection Widgets :** These Android widgets scroll in top-to-down direction. Collection of information of same type and then enabling user to open any one of them to full detail. Example is your e-mail app which will display all the mails in your inbox and then allow you to open any one o them.
   c. **Control Widgets :** Displays the most frequently used functionalities which user might want to control from home screen. For example in a video app, you can pause, play, stop, go to previous track, move to next track is an example of control widget.

d. **Hybrid Widgets** : These Android widgets combine features of all of the above three.

- **Notification,** as the name says keeps the user aware of events going on. User is kept informed like any news channel. For e.g, everyone of us know about facebook or whatsapp, now notification system of app is responsible for informing you about any new friend request, chat request, or a new message from say, dvs or xyz, etc.

There are a few other application components that you should be aware of. These application components include fragments, views, layouts, intents, resources, and manifest. All of these components are used for the creation of above components.

S.No.	Application Components	Description
1	Fragments	Represents the fragments of a user interface in the Activity component
2	Views	Includes the user interface elements like buttons, drop-down lists, etc.
3	Layouts	Controls the screen format based on different hierarchies of the views Takes care of the appearance of the views on the screen
4	Intents	Wires the messages of different components together
5	Resources	Includes external elements like drawable or editable pictures, strings, and constants

3.2.1 Creating the user Interface

There are three approaches to creating user interfaces in Android.

1. You can create user interfaces entirely in Java code
2. Entirely in XML
3. Combining both methods (defining the user interface in XML and then referring and modifying it through Java code).

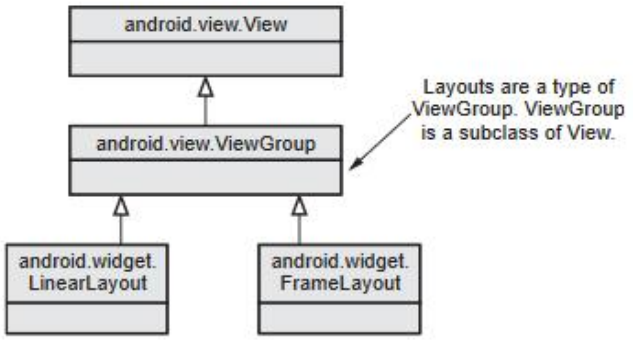
The third approach, the combined approach, is highly preferred.

The graphical user interface for an Android app is built using a hierarchy of View and ViewGroup objects.

- A View is an interactive UI component (or widget or control), such as button and text field. It controls a *rectangular area* on the screen. It is responsible for drawing itself and handling events (such as clicking, entering texts). Android provides many ready-to-use Views such as TextView, EditText, Button, RadioButton, etc, in package android.widget. You can also create your custom View by extending android.view.View.
- A ViewGroup is an *invisible container* used to *layout* the View components. Android provides many ready-to-use ViewGroups such as LinearLayout, RelativeLayout, TableLayout and GridLayout in package android.widget. You can also create your custom ViewGroup by extending from android.view.ViewGroup.

Views and ViewGroups are organized in a single tree structure called view-tree. You can create a view-tree either using programming codes or describing it in a XML layout file.

Layout view groups



The views for a screen are organized in a hierarchy. At the root of this hierarchy is a ViewGroup that contains the layout of the entire screen. The view group's child screens can be other views or other view groups as shown in the following figure.

1. The *root* view group.
2. The first set of child views and view groups whose parent is the root.

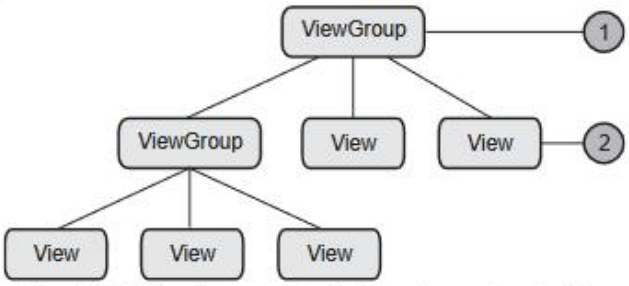


Fig. 3.2.3 ViewGroup objects form branches in the layout and contain other View objects.

Commonly Used Layouts And Controls :

The views or the controls that we want to display in an application are arranged in an order or sequence by placing them in the desired layout. The layouts also known as **Containers or ViewGroups.** These are used for organizing the views or controls in the required format. Android provides the following layouts

- **LinearLayout** : In this layout, all elements are arranged in a descending column from top to bottom or left to right. Each element contains properties to specify how much of the screen space it will consume. Depending on the orientation parameter, elements are either arranged in row or column format.
- **RelativeLayout :** In this layout, each child element is laid out in relation to other child elements. That is, a child element appears in relation to the previous child. Also, the elements are laid out in relation to the parent.
- **AbsoluteLayout :** In this layout, each child is given a specific location within the bounds of the parent layout object. This layout is not suitable for devices with different screen sizes and hence is deprecated.
- **FrameLayout** : This is a layout used to display a single view. Views added to this are always placed at the top left of the layout. Any other view that is added to the FrameLayout overlaps the previous view; that is, each view stacks on top of the previous one.
- **TableLayout :** In this layout, the screen is assumed to be divided in table rows, and each of the child elements is arranged in a specific row and column.
- **GridLayout** : In this layout, child views are arranged in a grid format, that is, in the rows and columns pattern. The views can be placed at the specified row and column location. Also, more than one view can be placed at the given row and column position.

The following is list of some commonly used controls in Android applications :

- **TextView** : A read-only text label. It supports multiline display, string formatting, and automatic word wrapping.
- **EditText** : An editable text box that also accepts multiline entry and word-wrapping.
- **ListView** : A ViewGroup that creates and manages a vertical list of views, displaying them as rows within the list.
- **Spinner** : A TextView and an associated list of items that allows us to select an item from the list to display in the text box.
- **Button** : A standard command button.
- **CheckBox** : A button allowing a user to select (check) or unselect (uncheck).
- **RadioButton** : A mutually exclusive button, which, when selected, unselects all other buttons in the group.

3.3 Layouts

Layouts are basically containers for other items known as Views , which are displayed on the screen. Layouts help manage and arrange views as well. Layouts are defined in the form of XML files that cannot be changed by our code during runtime.

Following table shows the layout managers provided by the Android SDK.

Layout Manager	Description
LinearLayout	Organizes its children either horizontally or vertically
RelativeLayout	Organizes its children relative to one another or to the parent
AbsoluteLayout	Each child control is given a specific location within the bounds of the container
FrameLayout	Displays a single view; that is, the next view replaces the previous view and hence is used to dynamically change the children in the layout
TableLayout	Organizes its children in tabular form
GridLayout	Organizes its children in grid format

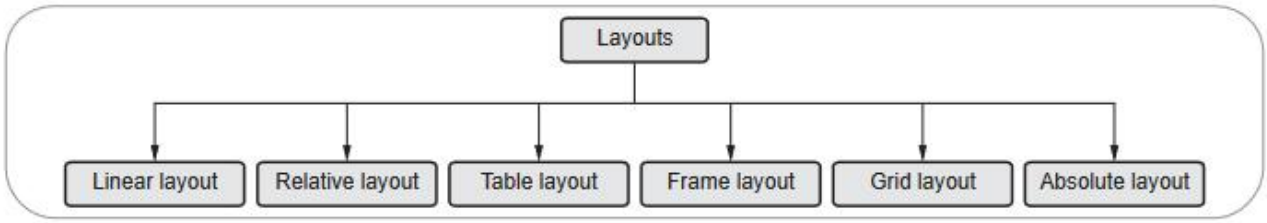


Fig. 3.3.1 Layouts supported by android system

Let's get familiar with the **layout editor** :

The layout editoris used to edit the layout file. You can drag and drop view objects into a graphical pane, and arrange, resize, and specify properties for them. You immediately see the effect of changes you make.

To use the layout editor, open the XML layout file. The layout editor appears with the **Design** tab at the bottom highlighted.

(If the **Text** tab is highlighted and you see XML code, click the **Design** tab.) For the Empty Activity template, the layout is as shown in the figure below.

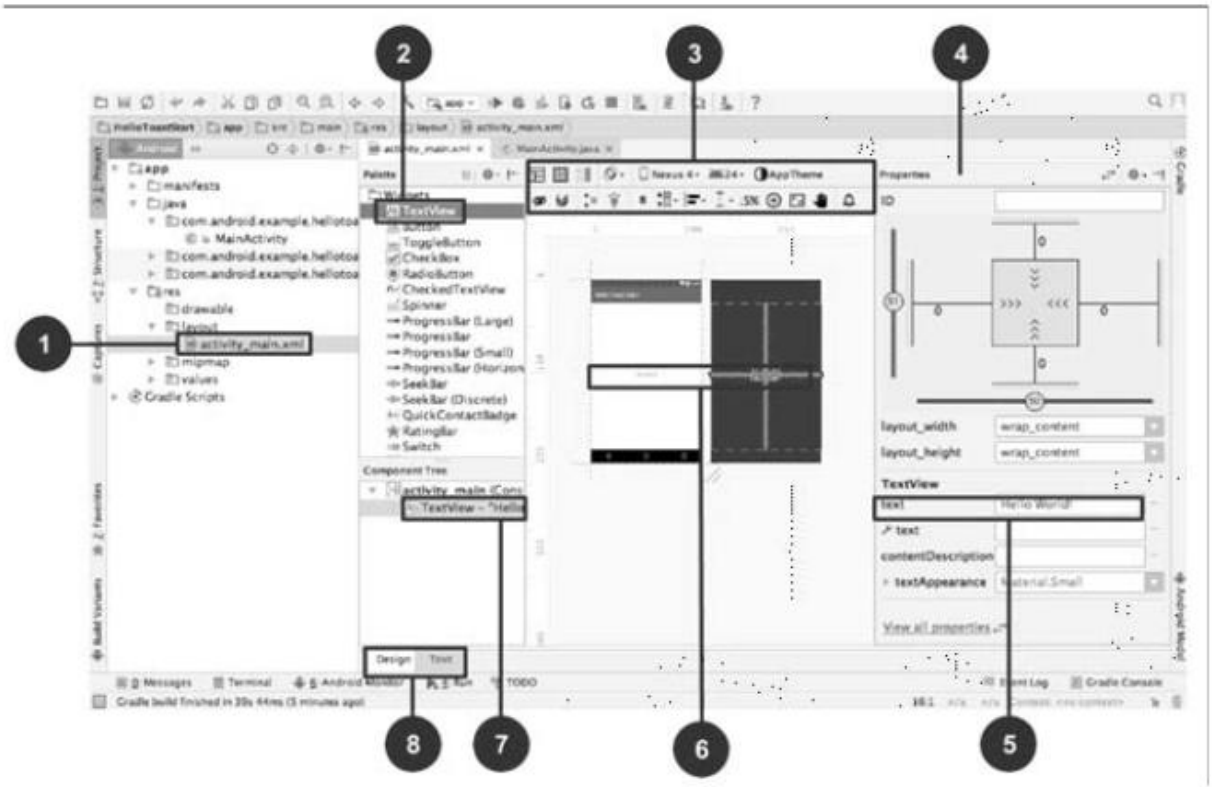


Fig. 3.3.2 The layout editor window

In the Fig. above :

1. **XML layout file :** The XML layout file, typically named **activiy_main.xml** file.Double-click it to open the layout editor.
2. **Palette of UI elements (views) :** The Palette pane provides a list of UI elements and layouts.Add an element or layout to the UI by dragging it into the design pane.

3. **Design toolbar :** The design pane tool bar provides buttons to configure your layout appearance in the design pane and to edit the layout properties.See the figure 3.3.6 for details.
4. **Properties pane :** The Properties pane provides property controls for the selected view.
5. **Property control :** Property controls correspond to XML attributes. Shown in the figure is the Text property of the selected TextView, set to Hello World!.
6. **Design pane :** Drag views from the Palette pane to the design pane to position them in the layout.
7. **Component Tree :** The Component Tree pane shows the view hierarchy. Click a view or view Group in this pane to select it.The figure shows the TextView selected.
8. **Design** and **Text** tabs Click **Design** to see the layout editor, or **Text** to see XML code.The layout editor's design toolbar offers a row of buttons that let you configure the appearance of the layout :

Fig. 3.3.3 Design toolbar

In the figure above :

1. **Design, Blueprint** and **Both :** Click the **Design** icon (first icon) to display a color preview of your layout. Click the **Blueprint** icon (middle icon) to show only outlines for each view. You can see both views side by side by clicking the third icon.
2. **Screen orientation :** Click to rotate the device between landscape and portrait.
3. **Device type and size :** Select the device type (phone/tablet, Android TV, or Android Wear) and Screen configuration (size and density).
4. **API version :** Select the version of Android on which to preview the layout.
5. **App theme :** Select which UI theme to apply to the preview.
6. **Language :** Select the language to show for your UI strings. This list displays only th languages available in the string resources.
7. **Layout Variants :** Switch to one of the alternative layouts for this file, or create a new one.

3.3.1 Android Linear Layout

- In android, **LinearLayout** is a **ViewGroup** subclass which is used to render all child **View** instances one by one either in **Horizontal** direction or **Vertical** direction based on the orientation property.
- Linear layout in Android allow us to arrange components horizontally in a single column or vertically in a single row. Vertically or horizontally direction depends on attribute **android : orientation.**
- Linear layout is simple and easy to use, it creates a scroll bar if the length of the window exceeds the length of the screen. Vertically linear layout has only one item per row.
- Linear layout has many different attributes which can be used to customize linear layout according to needs.
- There are two types of linear layout orientation :
  1. Vertical 2. Horizontal

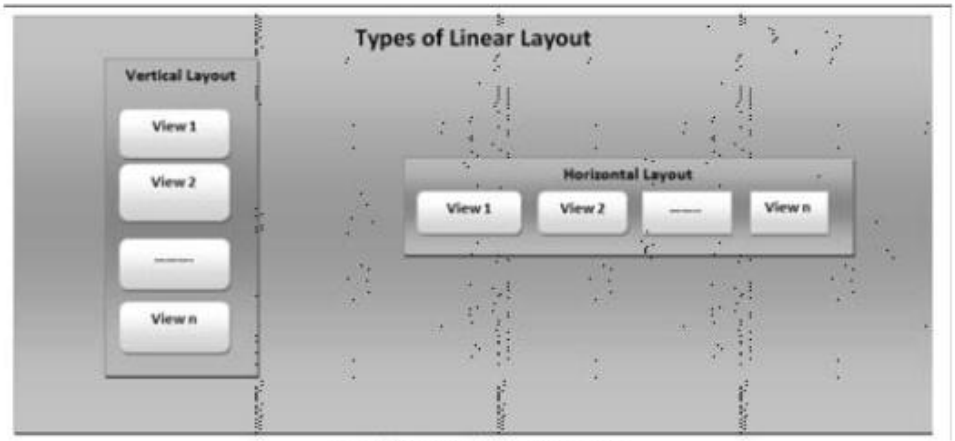
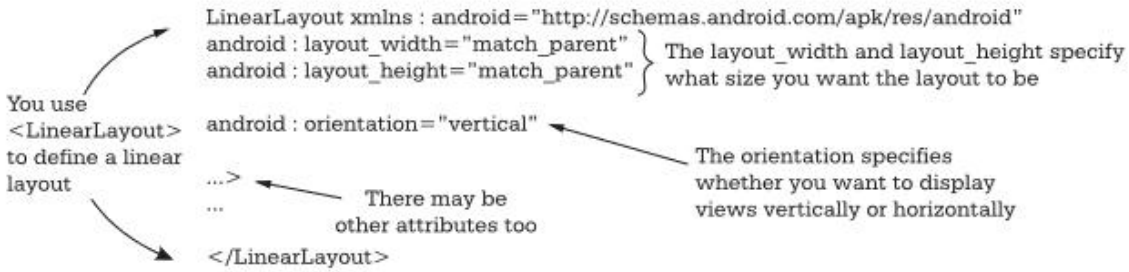


Fig. 3.3.4 Types of layouts

1. **Vertical :** In this all the child are arranged vertically in a line one after the other.
2. **Horizontal :** In this all the child are arranged horizontally in a line one after the other.

Android LinearLayout Declaration :



In above code snippet, the orientation is defined as vertical, so this aligns all its child layout / views vertically.

Android LinearLayout Example

Following is the example of creating a **LinearLayout** with different controls in android application.

Steps :

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

```
activity_main.xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingLeft="20dp"
   android:paddingRight="20dp"
   android:orientation="vertical" >
   <EditText
```

```
        android:id="@+id/txtTo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="To"/>
    <EditText
        android:id="@+id/txtSub"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Subject"/>
    <EditText
        android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="Message"/>
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="Send"/>
</LinearLayout>
```

3. After creation of layout, load the XML layout resource from activity **onCreate()** callback method,for that open main activity file **MainActivity.java** from **\java\com.vrs.linearlayout** path and write the code like as shown below.

Output :

MainActivity.java

```
package com.vrs.linearlayout;
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);
    }
}
```

Attributes of linear layout

Following are some attributes of linear layout of Android.

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

1. Orientation :

- The orientation attribute used to set the childs/views horizontally or vertically.
- In Linear layout default orientation is vertical.
- The valid values for this attribute are horizontal and vertical.
- If the value of the android:orientation attribute is set to vertical , the children in the linear layout are arranged in a column layout, one below the other.
- If the value of the android:orientation attribute is set to horizontal, the controls in the linear layout are arranged in a row format, side by side.
- The orientation can be modified at runtime through the setOrientation() method. That is, by supplying the values HORIZONTAL or VERTICAL to the setOrientation() method, we can arrange the children of the LinearLayout in row or column format, respectively as shown below :

For vertical -

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:orientation="vertical"> <!-- Vertical Orientation set -->
<!-- Put Child Views like Button here -->
</LinearLayout>
```

For horizontal -

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:orientation="horizontal"> <!-- Horizontal Orientation set -->
<!-- Child Views are here -->
</LinearLayout>
```

Example 1 : Orientation vertical :

```
<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=".LinearLayout" >

        <Button
            android:text="BUTTON"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
        />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Text 1"
            android:paddingTop="10px"/>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Text 2"
            android:paddingTop="10px"/>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Text 3"
            android:paddingTop="10px"/>

</LinearLayout>
```

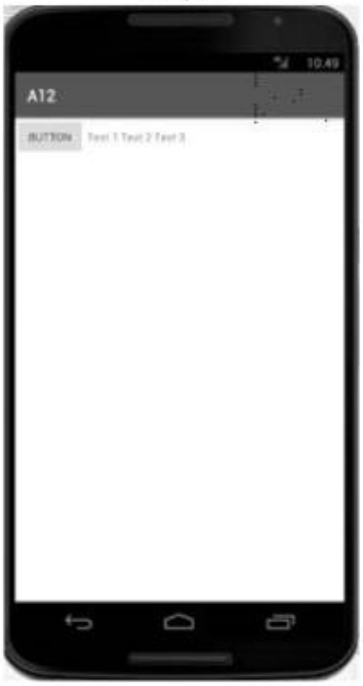
Output

Example 2 : Orientation Horizontal :

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

        <Button
            android:text="BUTTON"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
        />
```

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Text 1"
        android:paddingTop="10px"
        android:paddingLeft="10px"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Text 2"
        android:paddingTop="10px"
        android:paddingLeft="10px"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Text 3"
        android:paddingTop="10px"
        android:paddingLeft="10px"/>
</LinearLayout>
```

Output



2. gravity :

- The gravity attribute is an optional attribute for aligning the content within a control. For example, to align the text of a control to the right, we set the value of its android:gravity attribute to right .
- The valid options for android:gravity include left, center, right, top, bottom, center_horizontal , center_vertical , fill_horizontal , and fill_vertical.
- The task performed by few of the above options is as follows :
  - **center_vertical** : Places the object in the vertical center of its container, without changing its size
  - **fill_vertical :** Grows the vertical size of the object, if needed, so it completely fills its container
  - **center_horizontal :** Places the object in the horizontal center of its container,without changing its size
  - **fill_horizontal :** Grows the horizontal size of the object, if needed, so it completely fills its container
  - **center** : Places the object in the center of its container in both the vertical and horizontal axis, without changing its size

Example :

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:gravity="right"
android:orientation="horizontal">
<!--Button Child View Here--->
```

```
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="click2"
android:id="@+id/click2"
android:background="#0e7d0d" />

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="click1"
android:id="@+id/click"
android:background="#761212" />
</LinearLayout>
```

Output

3. layout_weight :

- The layout weight attribute specify each child control's relative importance within the parent linear layout.
- The weight attribute affects the size of the control. That is, we use weight to assign the capability to expand or shrink and consume extra space relative to the other controls in the container.
- The values of the weight attribute range from 0.0 to 1.0 , where 1.0 is the highest value. Let's suppose a container has two controls and one of them is assigned the weight of 1. In that case, the control assigned the weight of 1 consumes all the empty space in the container, whereas the other control remains at its current size. If we assign a weight of 0.0 to both the controls, nothing happens and the controls maintain their original size.
- If both the attributes are assigned the same value above 0.0, both the controls consume the extra space equally. Hence, weight lets us apply a size expansion ratio to the controls.

Example :

```
<?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="horizontal">
<!--Add Child View Here--->
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Weight 2"
android:background="#5b39c6"
android:layout_margin="5dp"
android:id="@+id/button"
android:layout_weight="2" />
<Button
android:layout_width="wrap_content"
```

Output

```
android:layout_height="wrap_content"
android:background="#00ff00"
android:layout_margin="5dp"
android:layout_weight="1"
android:text="Weight 1" />
</LinearLayout>
```

4. WeightSum :

- weightSum is the sum up of all the child attributes weight.
- This attribute is required if we define weight property of the childs.

Example :

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:weightSum="3"
android:orientation="horizontal">
<!--Add Child View Here--->
</LinearLayout>
```

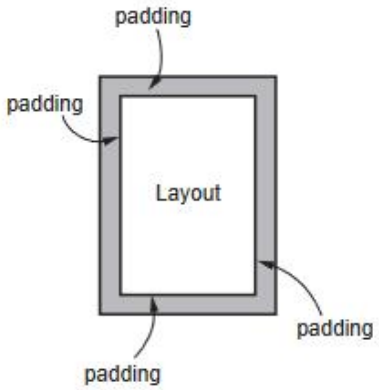
5. layout_width and layout_height :

- The default height and width of a control are decided on the basis of the text or content that is displayed through it.
- To specify a certain height and width to the control, the android:layout_width and android:layout_height attributes is used.
- The values for the height and width attributes can be specified in the following three ways :
  - By supplying specific dimension values for the control in terms of px (pixels), dip/dp (device independent pixels), sp (scaled pixels), pts (points), in (inches), and mm (millimeters). For example, the android:layout_width="20px" attribute sets the width of the control to 20 pixels.
  - By providing the value as wrap_content. When assigned to the control's height or width, this attribute resizes the control to expand to fit its contents. For example, when this value is applied to the width of the TextView , it expands so that its complete text is visible.
  - By providing the value as match_parent. When assigned to the control's height or width, this attribute forces the size of the control to expand to fill up all the available space of the enclosing container.
- For layout elements, the value wrap_content resizes the layout to fit the controls added as its children. The value match_parent makes the layout expand to take up all the space in the parent layout.
- It will control the width and height of the linear layout.
  - If you want to give full width and height then set layout_width and layout_height as **match_parent.**
  - Set layout_width and layout_height as **wrap_content** if you want to have width of linearlayout as per requirements of children widgets.
  - You can set layout_width in units also. For example **10dp,20dp** etc.
  - The higher the **dp number** the more width is there.

6. Applying the padding Attribute :

- The padding attribute is used to increase the whitespace between the boundaries of the control and its actual content.
- Through the android:padding attribute, we can set the same amount of padding or spacing on all four sides of the control.
- Similarly, by using the android:paddingLeft , android:paddingRight , android:paddingTop , and android:paddingBottom attributes, we can specify the individual spacing on the left, right, top, and bottom of the control, respectively.
- For example-1) To add padding of 16dp to all edges of the layout :

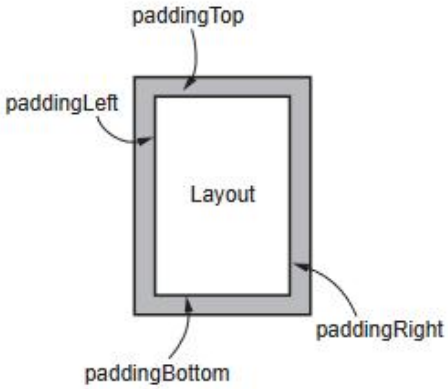
```
< LinearLayout ...
android:padding="16dp:> ← This ads the same padding to all edges of the layout
< LinearLayout>
```



2) Different amounts of padding to different edges

```
< LinearLayout> ...
androdi : paddingBottom="16dp"
androdi : paddingLeft="16dp"
androdi : paddingRight="16dp"
androdi : paddingTop="16dp"
< LinearLayout>
```

} Add padding to the individual edges



- To set the padding at runtime, we can call the setPadding() method.

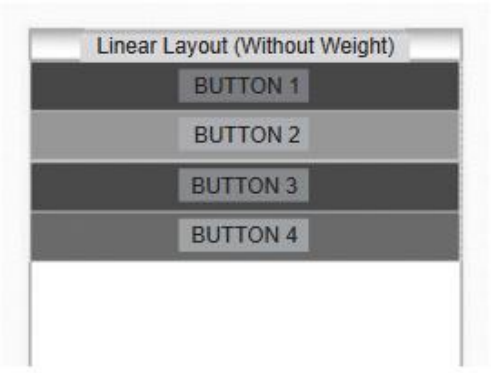
Example 1 : First we will design Android Linear Layout without using weight property

In this example we have used one TextView and 4 Button. The orientation is set to vertical.

Below is the code of activity_main.xml

```
<!-- Vertical Orientation is set -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">
   <!-- Text Displayed At Top -->

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textAppearance="?android:attr/textAppearanceLarge"
      android:text="Linear Layout (Without Weight)"
      android:id="@+id/textView"
      android:layout_gravity="center_horizontal" />

   <!-- Button Used -->

   <Button
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Button 1"
      android:background="#009300" />

   <Button
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Button 2"
      android:background="#e6cf00" />

   <Button
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Button 3"
      android:background="#0472f9" />

   <Button
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Button 4"
      android:background="#e100d5" />
</LinearLayout>
```

Output



Example 2 : In this example of linear layout we have used weight property.

Below is the code of activity_main.xml with explanation included

```
<!-- Vertical Orientation is set  with weightSum-->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
```

```
    android:weightSum="5"
    android:orientation="vertical">

    <!-- Text Displayed At Top -->

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Linear Layout (With Weight)"
        android:id="@+id/textView"
        android:layout_gravity="center_horizontal"
        android:layout_weight="0"/>

    <!-- Button Used -->

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Button 1"
        android:background="#009300"
        android:layout_weight="1"/>

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Button 2"
        android:background="#e6cf00"
        android:layout_weight="1"/>

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Button 3"
        android:background="#0472f9"
        android:layout_weight="1"/>

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Button 4"
        android:background="#e100d5"
        android:layout_weight="1"/>
</LinearLayout>
```

Output

Example 3 : Mixed Example With Horizontal And Vertical Orientation Properties :

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

```
android:layout_gravity="center"
android:orientation="vertical"
tools:context=".LinearLayout" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:paddingLeft="10px"
        android:paddingTop="20px"
        android:text="LOGIN" />

        <LinearLayout
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal"
            android:paddingTop="20px">

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:paddingLeft="10px"
                android:paddingTop="20px"
                android:text="Username" />

            <EditText
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="0.50"
                android:layout_marginLeft="40px"
                android:paddingTop="20px" />

        </LinearLayout>

        <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:paddingTop="20px"
         >

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Password"
                android:paddingTop="20px"
                android:paddingLeft="10px"/>

            <EditText
```

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center_horizontal"
android:layout_marginLeft="40px"
android:layout_weight="0.50"
android:paddingTop="20px" />

</LinearLayout>

<Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_gravity="center"
   android:paddingLeft="20px"
   android:paddingTop="10px"
   android:text="BUTTON" />
</LinearLayout>
```

Output

3.3.2 Android Absolute Layout

Android Absolute Layout is a layout type that allows to position the views in the screen using the X, Y coordinates.

- Absolute Layout enables you to specify the exact location of its children.
- In order to align the views in the screen using absolute layout is very difficult because we have to manually specify the locations of the screen accurately. That is why Absolute layouts are less flexible and harder to maintain than other types of layouts in android.
- Each child in an AbsoluteLayout is given a specific location within the bounds of the container.
- Such fixed locations make AbsoluteLayout incompatible with devices of different screen size and resolution.
- The controls in AbsoluteLayout are laid out by specifying their exact X and Y positions. The coordinate 0,0 is the origin and is located at the top-left corner of the screen.

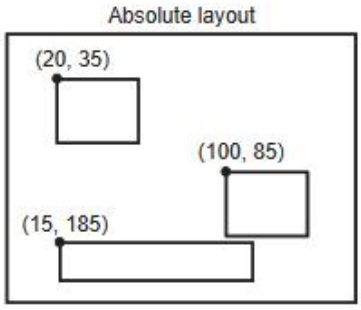


Fig. 3.3.5

Android AbsoluteLayout Attributes :

Following are the important attributes of android AbsoluteLayout.

android:id : This is the ID which uniquely identifies the layout.

android:layout_x : This specifies the x-coordinate of the view.

android:layout_y : This specifies the y-coordinate of the view.

Absolute Layout xml snippet

```
<AbsoluteLayout
android:layout_width="50dp"
android:layout_height="wrap_content
android:layout_x="25px"
```

```
android:layout_y="29px">
</AbsoluteLayout>
```

Android AbsoluteLayout Syntax Code :

```
<AbsoluteLayout android:id="@+id/absoluteLayout"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   xmlns:android="http://schemas.android.com/apk/res/android">
   </AbsoluteLayout>
```

Example1 :

activity_main.xml

```
<AbsoluteLayout android:id="@+id/absoluteLayout"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   xmlns:android="http://schemas.android.com/apk/res/android">

   <Button
      android:layout_width="100dp"
      android:layout_height="wrap_content"
      android:id="@+id/btn1"
      android:text="Login"
      android:textColor="@color/colorPrimary"
      android:layout_x="50px"
      android:layout_y="361px" />
   <Button
      android:layout_width="100dp"
      android:layout_height="wrap_content"
      android:id="@+id/btn2"
      android:textColor="@color/colorPrimary"
      android:text="Register"
      android:layout_x="350px"
      android:layout_y="361px" />

</AbsoluteLayout>
```

The graphical representation of the above code is as follows.

Example 2 : Layout File activity_absolute_layout_app.xml on Arranging Controls in the AbsoluteLayout Container

```
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Student Registration Form"
      android:textSize="20sp"
      android.textStyle="bold"
      android:layout_x="90dip"
```

```
        android:layout_y="2dip"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Student ID:"
        android:layout_x="5dip"
        android:layout_y="40dip" />
    <EditText
        android:id="@+id/student_is"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:minWidth="100dip"
        android:layout_x="110dip"
        android:layout_y="30dip" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Student Name:"
        android:layout_x="5dip"
        android:layout_y="90dip"/>
    <EditText
        android:id="@+id/student_name"
        android:layout_width="200dip"
        android:layout_height="wrap_content"
        android:minWidth="200dip"
        android:layout_x="110dip"
        android:layout_y="80dip"
        android:scrollHorizontally="true" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Marks:"
        android:layout_x="5dip"
        android:layout_y="140dip" />
    <EditText
        android:id="@+id/marks"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:minWidth="100dip"
        android:layout_x="110dip"
        android:layout_y="130dip" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/click_btn"
        android:text="Add New Student"
        android:layout_x="80dip"
        android:layout_y="190dip" />
</AbsoluteLayout>
```

The controls in activity_absolute_layout_app.xml are as follows :

- The Student Registration Form TextView is set to appear 90dip from the left and 2dip from the top side of the container. The size of the text is set to 20sp, and its style is set to bold .
- The Student ID TextView is set to appear 5dip from the left and 40dip from the top side of the container.
- The student_id EditText control is set to appear 110dip from the left and 30dip from the top side of the container. The minimum width of the control is set to 100dp.
- The Student Name TextView control is set to appear 5dip from the left and 90dip from the top side of the container.
- The student_name EditText control is set to appear 110dip from the left and 80dip from the top side of the container. The minimum width of the control is set to 200dip, and its text is set to scroll horizontally when the user types beyond its width.
- The Marks TextView is set to appear 5dip from the left and 140dip from the top side of the container.
- The marks EditText control is set to appear 110dip from the left and 130dip from the top side of the container. The minimum width of the control is set to 100dip .
- The click_btn Button, Add New Student, is set to appear 80dip from the left and 190dip from the top side of the container.

If we don't specify the x, y coordinates of a control in AbsoluteLayout, it is placed in the origin point, that is, at location 0,0. If the value of the x and y coordinates is too large, the control does not appear on the screen. The values of the x and y coordinates are specified in any units, such as sp, in, mm, and pt. After specifying the locations of controls in the layout file activity_absolute_layout_app.xml, we can run the application. There is no need to make any changes in the file AbsoluteLayoutAppActivity.java.

Output

3.3.3 Android Frame Layout

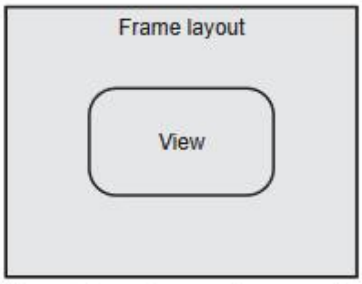
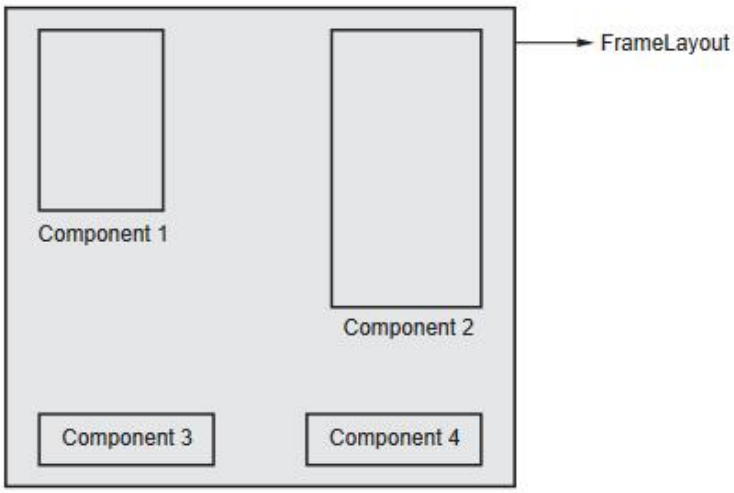
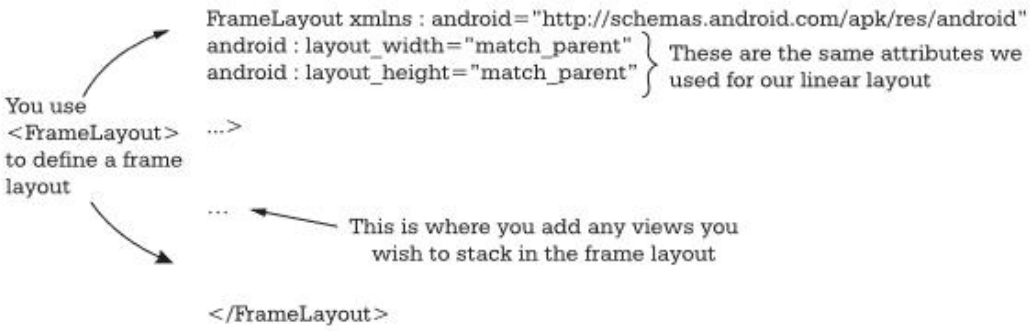


Fig. 3.3.6 Frame layout supports one view inside it

- Frame layout is the simple layout representation used to organise the control within the user interface of android application. When we want to display a single item on the display and block out the other items then frame layout tool is used. Sometimes it is difficult to organize child views in a way that is scalable to different screen sizes without the children over lapping each other, in that case the frame layout is used a it can hold a single child view and can block the other view.
- We can also add multiple children using frame layout and also control their position within this by simply assigning the gravity tags to each child. In this child view are drawn in the stack with the feature of stack the recently added is on the top and also last one added is removed from the top. The size of the frame layout depends on the size of its largest child whether it is visible or not.
- Using different tags in frame layout you can also determine whether to measure all the children or just those that are in the visible or in the invisible state when measuring. These can be simply defined within the XML layout resources and in Java' s for programmatically application. Understanding layout and when to use these layouts is important for good android application design. When this layout is used correctly, it becomes the fundamental layout in which many interesting android application user interfaces can be designed.
  - When the child content is too large to draw within the bounds of the layout, it uses the view scroll view to organise the control within the interface.
  - When we need a frame around underlying views for our interface, then frame layout provides the foreground draw able facility in addition to the normal ackground,this task is done with the help od XML' s attributes.
  - If multiple child view occurs within the frame layout then they are drawn in such a order such that the last one child view on the top, which is done using the frame layout.
  - Frame layout are the normal layout of choice especially when you want to use the overlapped view of the design.
- Frame Layout is one of the simplest layout to organize view controls. They are designed to block an area on the screen.
- Frame Layout should be used to hold child view, because it can be difficult to display single views at a specific area on the screen without overlapping each other.
- Multiple children can be addes to a FrameLayout and control their position by assigning gravity to each child, using the **android:layout_gravity** attribute.
- FrameLayout is used to display a single View.
- The View added to a FrameLayout is placed at the top-left edge of the layout. Any other View added to the FrameLayout overlaps the previous View ; that is, each View stacks on top of the previous one.



- Define frame layout using the <FrameLayout> element like this :



Attributes of frame layout :

Lets see different properties of Frame Layout which will be used while designing Android App UI :

1. android:id

- This attribute will give the unique identity to the frame layout.
- When you want to access the frame layout from the JAVA class, this id will give you the reference via **findViewById()** method.
- Value of this attribute should be unique across the whole android app to reduce the complexity.

Example to use id:

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/frame"
android:layout_width="200dp"
android:layout_height="300dp"
android:foreground="@color/colorAccent"
tools:context=".MainActivity">
```

2. android:foreground

- Foreground defines the drawable to draw over the content and this may be a color value.
- Possible color values can be in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb". This all are different color code model used.

Example: To set the blue color for foreground of frameLayout so the ImageView and other child views of this layout will not be shown.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:id="@+id/framelayout"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_gravity="center"
android:foregroundGravity="fill"
android:foreground="#4363d8"><!--foreground color for a FrameLayout-->

<LinearLayout
android:orientation="vertical"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
>

<!-- Imageview will not be shown because of foreground color which is drawn over it-->

<ImageView
android:layout_width="200dp"
android:layout_height="200dp"
android:layout_marginBottom="10dp"
android:src="@mipmap/ic_launcher"
android:scaleType="centerCrop"
/>
<!--Textview will not be shown because of foreground color is drawn over it-->

<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:gravity="center_horizontal"
android:text="vrs"/>

</LinearLayout>
</FrameLayout>
```

Output

3. android:foregroundGravity

- This defines the gravity to apply to the foreground drawable.
- Default value of gravity is fill.
- We can set values in the form of "top", "center_vertical" , "fill_vertical", "center_horizontal", "fill_horizontal", "center", "fill", "clip_vertical", "clip_horizontal", "bottom", "left" or "right" .
- It is used to set the gravity of foreground. We can also set multiple values by using "|". Ex: fill_horizontal|top .Both the fill_horizontal and top gravity are set to framelayout.

4. android:visibility

- We can apply this attribute to any child of the framelayout.

- Possible values for visibility are visible, invisible and gone.
  - Visible means that the view is present and we can also see it.
  - Invisible means that the view is present and we can not see it.
  - Gone means that view is not present and so we can not see it.

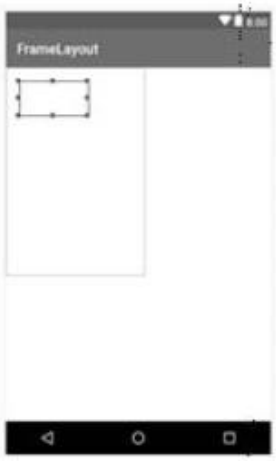
Example :

```
android:visibility="visible"
```

write the below code in **activity_main.xml** file

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 android:id="@+id/frame"
 android:layout_width="200dp"
 android:layout_height="300dp"
 tools:context=".MainActivity">

<TextView
 android:layout_width="100dp"
 android:layout_height="50dp"
 android:visibility="invisible"
 android:textSize="20sp"
 android:layout_marginTop="20dp"
 android:layout_marginLeft="20dp"
 android:background="@color/colorPrimary"
 android:textColor="#fff"
 android:text="I am TextView" />

</FrameLayout>
```

Otuput



You can see that Textview is present **(blue border rectangle)** but we can not see it.

5. android:measureAllChildren :

- This determines whether to measure all children including gone state visibility or just those which are in the visible or invisible state of measuring visibility.
- The default value of measureallchildren is false. We can set values in the form of Boolean i.e. "true" OR "false".
- This may also be a reference to a resource (in the form "@[package:]type:name") or theme attribute (in the form "?[package:][type:]name") containing a value of this type.
- If measureallchildren is set true then it will show actual width and height of frame layout even if the views visibility is in gone state.

Example : To set the ImageView visibility to gone and measureAllChildren to true.

activity_main.xml

Note: Make sure you have image in Drawable folder.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:id="@+id/frame"
  android:orientation="vertical" android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:measureAllChildren="true"
    >
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="gone"
        android:src="@drawable/ic_launcher"/>

</FrameLayout>
```

code of MainActivity.java . Here we have used Toast to display height and width on screen.

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.demo);
        FrameLayout frame=(FrameLayout)findViewById(R.id.frame);
        frame.measure(View.MeasureSpec.UNSPECIFIED, View.MeasureSpec.UNSPECIFIED);
        int width = frame.getMeasuredWidth();
        int height = frame.getMeasuredHeight();
        Toast.makeText(getApplicationContext(),"width="+width+"  height="+height,Toast.LENGTH_SHORT).show();
    }
}
```

Otuput

Explanation of Example : It measures all the children in the layout. For ex: If we setVisiblity of an view be gone and set measuresAllChildren property to be true, then also it will also count to that view which is not visible, but if we set the measuresAllChildren property to be false, then it will not count to that view which is gone.

Example 1 : Frame Layout using layout gravity. Here we will put textview at different position in Frame Layout. Below is the code and final output :

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

```
    <TextView android:text="LeftTop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="RightTop"
        android:layout_gravity="top|right" />
    <TextView android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="CentreTop"
        android:layout_gravity="top|center_horizontal" />
    <TextView android:text="Left"
        android:layout_gravity="left|center_vertical"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="Right"
        android:layout_gravity="right|center_vertical" />
    <TextView android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="Centre"
        android:layout_gravity="center" />
    <TextView android:text="LeftBottom"
        android:layout_gravity="left|bottom"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="RightBottom"
        android:layout_gravity="right|bottom" />
    <TextView android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="CenterBottom"
        android:layout_gravity="center|bottom" />
</FrameLayout>
```

Step 3 : Let the MainActivity.java has default Android code or add the below code :

```
packagevrs.com.A12;
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);
    }
}
```

Output

Example :

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">
   <ImageView
      android:id="@+id/imgvw1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:scaleType="centerCrop"
      android:src="@drawable/garden" />
   <TextView
      android:id="@+id/txtvw1"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_marginTop="40dp"
      android:background="#4C374A"
      android:padding="10dp"
      android:text="My Garden"
      android:textColor="#FFFFFF"
      android:textSize="20sp" />
   <TextView
      android:id="@+id/txtvw2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="right|bottom"
      android:background="#AA000000"
      android:padding="10dp"
      android:textColor="#FFFFFF"
      android:textSize="18sp" />
</FrameLayout>
```

Here we used **ImageView** to show the image **(garden)** from drawable folder in framelayout. So add your image to **drawable** folder and replace **@drawable/garden** path with your image path.

open main activity file **MainActivity.java** from **\java\com.vrs.framelayout** path and write the code like as shown below.

MainActivity.java

```
package com.vrs.linearlayout;
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);
    }
}
```

Output

Methods of FrameLayout :

To control the framelayout from JAVA class, android gives us the following in-built methods to use.

1. **generateLayoutParams(AttributeSet attrs)** : It Returns a new set of layout parameters based on the supplied attributes set.
2. **getAccessibilityClassName()** : Return the class name of this object to be used for accessibility purposes.
3. **getMeasureAllChildren()** : • Determines whether all children, or just those in the VISIBLE or INVISIBLE state, are considered when measuring.
   - It will return **boolean** value. (True or False)
   - If it returns false then compiler will consider only VISIBLE or INVISIBLE state of children of framelayout.
   - For true, it will consider VISIBLE, INVISIBLE and GONE state of children of framelayout.
4. **setForegroundGravity(int foregroundGravity)** : Describes how the foreground is positioned.
5. **setMeasureAllChildren(boolean measureAll)** : Sets whether to consider all children, or just those in the VISIBLE or INVISIBLE state, when measuring.
6. **shouldDelayChildPressedState()** : Return true if the pressed state should be delayed for children or descendants of this ViewGroup.
7. **generateDefaultLayoutParams()** : Returns a set of layout parameters with a width of ViewGroup.LayoutParams.MATCH_PARENT, and a height ViewGroup.LayoutParams.MATCH_PARENT.
8. **generateLayoutParams(ViewGroup.LayoutParams lp)** : Returns a safe set of layout parameters based on the supplied layout params.
9. **onLayout(boolean changed, int left, int top, int right, int bottom)** : Called from layout when this view should assign a size and position to each of its children.
10. **onMeasure(int widthMeasureSpec, int heightMeasureSpec)** : Measure the view and its content to determine the measured width and the measured height. This method is invoked by **measure(int, int)** and should be overridden by sub classes to provide accurate and efficient measurement of their contents.

3.3.4 Android Table Layout

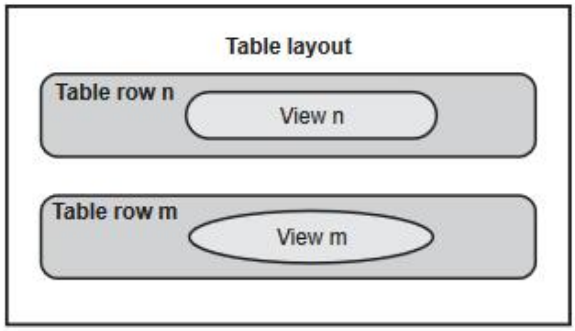


Fig. 3.3.7 Table layout

- The TableLayout is used for arranging the enclosed controls into rows and columns.
- Each new row in the TableLayout is defined through a TableRow object.

- A row can have zero or more controls, where each control is called a cell.
- The number of columns in a TableLayout is determined by the maximum number of cells in any row.
- The width of a column is equal to the widest cell in that column. All elements are aligned in a column; that is, the width of all the controls increases if the width of any control in the column is increased.
- Table Layout containers do not display a border line for their columns, rows or cells. A Table will have as many columns as the row with the most cells.
- **TableLayout** is a **ViewGroup** subclass which is used to display the child View elements in rows and columns.
- TableLayout will position its children elements into rows and columns and it won't display any border lines for rows, columns or cells.
- The TableLayout in android will work same as HTML table and table will have as many columns as the row with the most cells. The TableLayout can be explained as **<table>** and TableRow is like **<tr>** element.

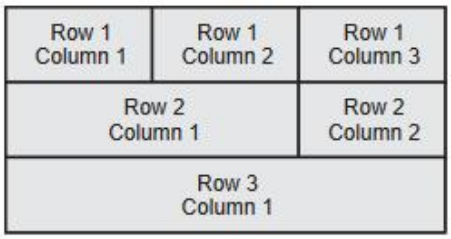


Fig. 3.3.8 Row and Column in Table Layout Android

Important Points About Table Layout In Android :

- For building a row in a table we will use the <TableRow> element. Table row objects are the child views of a table layout.
- Each row of the table has zero or more cells and each cell can hold only one view object like ImageView, TextView or any other view.
- Total width of a table is defined by its parent container
- Column can be both stretchable and shrinkable. If shrinkable then the width of column can be shrunk to fit the table into its parent object and if stretchable then it can expand in width to fit any extra space available.
- We cannot specify the width of the children's of the Table layout. Here, width always match parent width. However, the height attribute can be defined by a child; default value of height attribute is wrap content.

Basic Table Layout code in XML :

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:collapseColumns="0"> <!-- collapse the first column of the table row-->

   <!-- first row of the table layout-->
   <TableRow
      android:id="@+id/row1"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content">

      <!-- Add elements/columns in the first row-->
```

```
    </TableRow>
</TableLayout>
```

Attributes of table layout :

Attributes	Where it is used	Why it is used
android:stretchColumns	TableLayout	When a column width is less and you need to expand(or stretch) it, you use this attribute.
android:shrinkColumns	TableLayout	When you do not need the extra space in a column, you can use this attribute to shrink and remove the space.
android:collapseColumns	TableLayout	It hides the column of the given index in the TableLayout.
android:layout_span	Any View inside the TableRow	If a view takes only one column width but when you want your view to take more than one column space, then you can use this attribute.
android:layout_column	Any view inside the TableRow	When you want your view present in the first TableRow to appear below the other TableRow's view, you can use this attribute.

1. **id:** id attribute is used to uniquely identify a Table Layout.

```
<TableLayout
android:id="@+id/simpleTableLayout"
android:layout_width="match_parent"
android:layout_height="match_parent/>
```

2. **stretchColumns:** • The default width of a column is set equal to the width of the widest column, but we can stretch the column(s) to take up available free space using the android:stretchColumns attribute in the TableLayout.

- The value assigned to this attribute can be a single column number or a comma-delimited list of column numbers. The specified columns are stretched to take up any available space on the row.
- Stretch column attribute is used in Table Layout to change the default width of a column which is set equal to the width of the widest column but we can also stretch the columns to take up available free space by using this attribute.
- The value that assigned to this attribute can be a single column number or a comma delimited list of column numbers (1, 2, 3…n).
- If the value is 1 then the second column is stretched to take up any available space in the row, because of the column numbers are started from 0.
- If the value is 0,1 then both the first and second columns of table are stretched to take up the available space in the row.
- If the value is '*' then all the columns are stretched to take up the available space.

Examples:

- **android:stretchColumns="1"** : The second column (because the column numbers are zero-based) is stretched to take up any available space in the row.
- **android:stretchColumns="0,1"** : Both the first and second columns are stretched to take up the available space in the row.

- **android:stretchColumns="*"** : All columns are stretched to take up the available space.

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

  <TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/simpleTableLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="1"> <!-- stretch the second column of the layout-->

    <!-- first row of the table layout-->
    <TableRow

      android:id="@+id/firstRow"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content">

      <!-- first element of the row-->
      <TextView

        android:id="@+id/simpleTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#b0b0b0"
        android:padding="18dip"
        android:text="Text 1"
        android:textColor="#000"
        android:textSize="12dp" />

      <TextView

        android:id="@+id/simpleTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#FF0000"
        android:padding="18dip"
        android:text="Text 2"
        android:textColor="#000"
        android:textSize="14dp" />

    </TableRow>
  </TableLayout>
```

Output

1. whenandroid:stretchColumns="1"

2. whenandroid:stretchColumns="2"

3. whenandroid:stretchColumns="*"

1. shrinkColumns :

- We can shrink or reduce the width of the column(s) using the android:shrinkColumns attribute in the TableLayout.

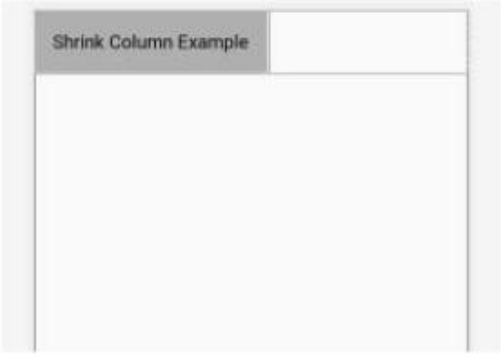
- Shrink column attribute is used to shrink or reduce the width of the column's. We can specify either a single column or a comma delimited list of column numbers for this attribute. The content in the specified columns word-wraps to reduce their width.
- If the value is 0 then the first column's width shrinks or reduces by word wrapping its content.
- If the value is 0,1 then both first and second columns are shrinks or reduced by word wrapping its content.
- If the value is '*' then the content of all columns is word wrapped to shrink their widths.

Examples :

- **android:shrinkColumns="0":** The first column's width shrinks or reduces by wordwrapping its content.
- **android:shrinkColumns="*" :** The content of all columns is word-wrapped to shrink their widths.

Example :

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:shrinkColumns="0"> <!-- shrink the first column of the layout-->

   <!-- first row of the table layout-->
   <TableRow
      android:id="@+id/firstRow"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content">

      <!-- first element of the first row-->
      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:background="#b0b0b0"
         android:padding="18dip"
         android:text="Shrink Column Example"
         android:textColor="#000"
         android:textSize="18dp" />

   </TableRow>
</TableLayout>
```



2. collapseColumns :

- We can make the column(s) collapse or become invisible through the android:collapseColumns attribute in the TableLayout.
- We can specify one or more comma-delimited columns for this attribute. These columns are part of the table information but are invisible.
- We can also make column(s) visible and invisible through coding by passing the Boolean values false and true , respectively, to the setColumnCollapsed() method in the TableLayout.

For example:

- **android:collapseColumns="0" :**
  - The first column appears collapsed; that is, it is part of the table but is invisible. It can be made visible through coding by using the setColumnCollapsed() method.
  - Collapse columns attribute is used to collapse or invisible the column's of a table layout. These columns are the part of the table information but are invisible.
  - If the values is 0 then the first column appears collapsed, i.e it is the part of table but it is invisible.

Example :

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:collapseColumns="0"> <!-- collapse the first column of the table row-->

   <!-- first row of the table layout-->
   <TableRow
      android:id="@+id/simpleTableLayout"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content">

      <!-- first element of the row that is the part of table but it is invisible-->
      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:background="#b0b0b0"
         android:padding="18dip"
         android:text="Columns 1"
         android:textColor="#000"
         android:textSize="18dp" />

      <!-- second element of the row that is shown in the screenshot-->
      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:background="#b0b0b0"
         android:padding="18dip"
         android:text="Columns 2"
         android:textColor="#000"
         android:textSize="18dp" />
   </TableRow>
</TableLayout>
```

Android TableLayout Example1

activity_main.xml

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

```
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_marginTop="100dp"
android:paddingLeft="10dp"
android:paddingRight="10dp" >
<TableRow android:background="#0079D6" android:padding="5dp">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="UserId" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="User Name" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Location" />
</TableRow>
<TableRow android:background="#DAE8FC" android:padding="5dp">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="1" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Abhi" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Pune" />
</TableRow>
<TableRow android:background="#DAE8FC" android:padding="5dp">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="2" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
```

```
            android:text="Rohit" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Nasik" />
    </TableRow>
    <TableRow android:background="#DAE8FC" android:padding="5dp">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="3" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Jaya" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Nasik" />
    </TableRow>
</TableLayout>
```

Output

MainActivity.java

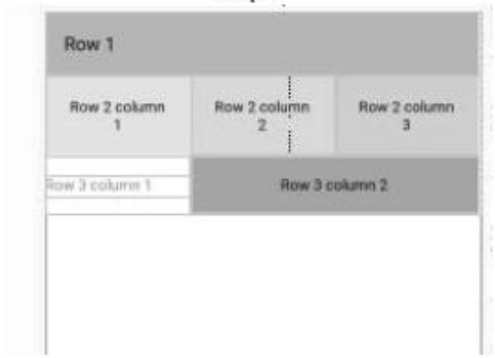
```
package com.vrs.linearlayout;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

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

Android TableLayout Example2

```
  <TableLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:shrinkColumns="*"  android:stretchColumns="*" android:background="#ffffff">

            <!-- Row 1 with single column -->
            <TableRow
              android:layout_height="wrap_content"
              android:layout_width="fill_parent"
```

```
        android:gravity="center_horizontal">

        <TextView
            android:layout_width="match_parent" android:layout_height="wrap_content"
            android:textSize="18dp" android:text="Row 1"  android:layout_span="3"
            android:padding="18dip" android:background="#b0b0b0"
            android:textColor="#000"/>

    </TableRow>

     <!-- Row 2 with 3 columns -->

    <TableRow
        android:id="@+id/tableRow1"
        android:layout_height="wrap_content"
        android:layout_width="match_parent">
        <TextView
            android:id="@+id/TextView04" android:text="Row 2 column 1"
            android:layout_weight="1" android:background="#dcdcdc"
            android:textColor="#000000"
            android:padding="20dip" android:gravity="center"/>
        <TextView
            android:id="@+id/TextView04" android:text="Row 2 column 2"
             android:layout_weight="1" android:background="#d3d3d3"
            android:textColor="#000000"
            android:padding="20dip" android:gravity="center"/>
        <TextView
            android:id="@+id/TextView04" android:text="Row 2 column 3"
             android:layout_weight="1" android:background="#cac9c9"
            android:textColor="#000000"
            android:padding="20dip" android:gravity="center"/>
    </TableRow>
    <!-- Row 3 with 2 columns -->
    <TableRow
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:gravity="center_horizontal">
        <TextView
            android:id="@+id/TextView04" android:text="Row 3 column 1"
            android:layout_weight="1  android:background="#b0b0b0"
            android:textColor="#000000"
            android:padding="18dip" android:gravity="center"/>
                    <TextView
            android:id="@+id/TextView04" android:text="Row 3 column 2"
            android:layout_weight="1" android:background="#a09f9f"
            android:textColor="#000000"
            android:padding="18dip" android:gravity="center"/>
    </TableRow>
</TableLayout>
```

Output



3.3.5 Android Relative Layout

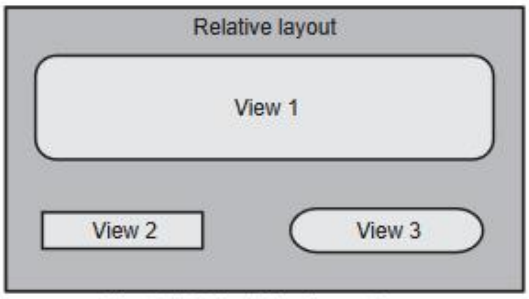
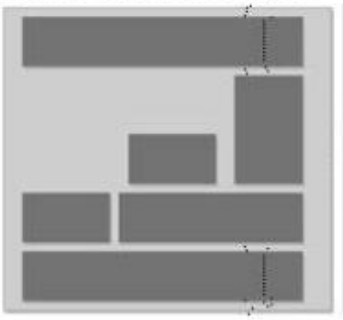


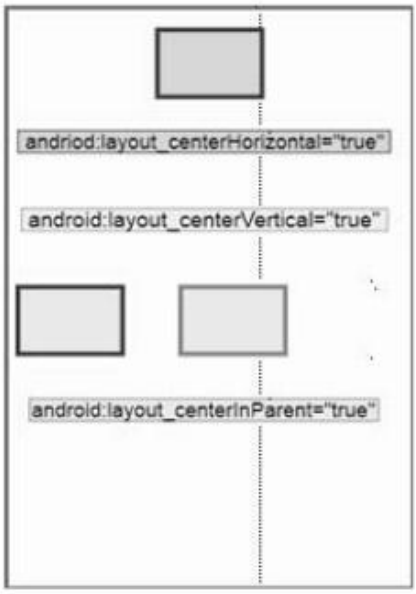
Fig. 3.3.9 Relative layout

- In RelativeLayout, each child element is laid out in relation to other child elements; that is, the location of a child element is specified in terms of the desired distance from the existing children.
- In android, **RelativeLayout** is a **ViewGroup** which is used to specify the position of child **View** instances relative to each other (Child **A** to the left of Child **B**) or relative to the parent (Aligned to the top of parent).
- In android, **RelativeLayout** is very useful to design user interface because by using relative layout we can eliminate the nested view groups and keep our layout hierarchy flat, which improves performance of application.
- Following is the pictorial representation of relative layout in android applications.

Common Attributes of RelativeLayout



1. **Center relative to Parent View.** The attributes used to set the location of the control relative to a container are
   - **android:layout_alignParentTop** - The top of the control is set to align with the top of the container.
   - **android:layout_alignParentBottom** - The bottom of the control is set to align with the bottom of the container.
   - **android:layout_alignParentLeft** - The left side of the control is set to align with the left side of the container.
   - **android:layout_alignParentRight** - The right side of the control is set to align with the right side of the container.
   - **android:layout_centerHorizontal** - The control is placed horizontally at the center of the container.
   - **android:layout_centerVertical** - The control is placed vertically at the center of the container.
   - **android:layout_centerInParent** - The control is placed horizontally and vertically at the center of the container



When you want to place your Views in the center relative to the parent, you can use the following 3 attributes :

1. **android:layout_centerHorizontal="true"** : This places the view horizontally in the center of the parent. As our parent view covers the whole screen of mobile therefore the view gets placed in the middle of the mobile screen horizontally.

```
<!-- centerHorizontal example -->
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Text1 center Horizontal"
   android:layout_centerHorizontal="true"
   android:textSize="20sp"
   android:textColor="#000"
/>
```

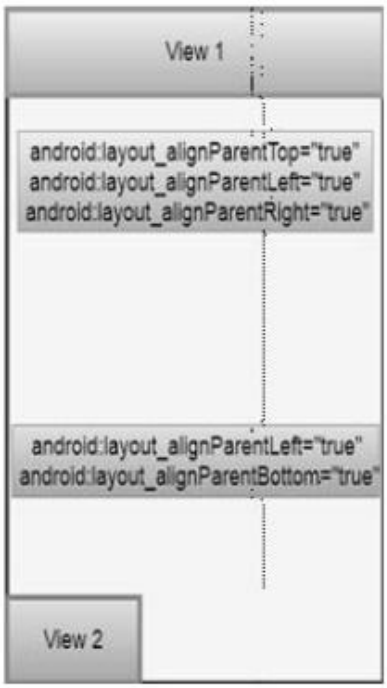
2. **android:layout_centerVertical="true"** : This places the view vertically in the center of the parent. Since the parent view covers the whole screen of mobile hence the view gets placed in the middle of the mobile screen vertically.


```
<!-- centerVertical example -->
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Text1 center vertical"
   android:layout_centerVertical="true"
   android:textSize="20sp"
   android:textColor="#000"
/>
```

3. **android:layout_centerInParent="true"** : This attribute will place the view in the center of the parent. Since the parent in our example covers the whole screen of mobile, so the view gets placed in the middle of the mobile screen, both horizontally and vertically.

```
<!-- centerInParent example -->
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Text1 center in parent"
   android:layout_centerInParent="true"
   android:textSize="20sp"
   android:textColor="#000"
/>
```

2. **Align by the parent view** : These type of attributes make the view act like a chewing gum as it can be fixed to any side of the parent view using these attributes.



Again, for the example, we are considering our parent view to be a RelativeLayout with height and width set as match_parent, therefore it will cover the whole screen of mobile. So the complete screen is our parent view.

1. **android:layout_alignParentTop="true"** : If you write this attribute for a View, then that view will stick to the top of its parent. Since the parent covers the whole screen of mobile therefore, the view will appear sticking to the top-left of the mobile screen.

```
<!-- alignParentTop example -->
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Text1 align parent top"
   android:layout_alignParentTop="true"
   android:layout_margin="20dp"
   android:textSize="20sp"
   android:textColor="#000"/>
```

2. **android:layout_alignParentBottom="true"** : If you write this attribute for a View, then that view will stick to the bottom of its parent. Since the our parent covers the whole screen of mobile therefore, the view will appear sticking to the bottom of the mobile screen.

```
<!-- textView is alignParentBottom -->
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:textAppearance="?android:attr/textAppearanceLarge"
   android:id="@+id/textView"
   android:text="Text Here is AlignParentBottom with bottom margin of 120dp"
   android:layout_alignParentBottom="true"
   android:layout_alignParentLeft="true"
   android:layout_alignParentStart="true"
   android:layout_marginBottom="120dp" />
```

3. **android:layout_alignParentLeft="true"** : If you write this attribute for a View, then that view will stick to the left of its parent. Since the parent in our example covers the whole screen of mobile therefore, the view will appear sticking to the left of the mobile screen.

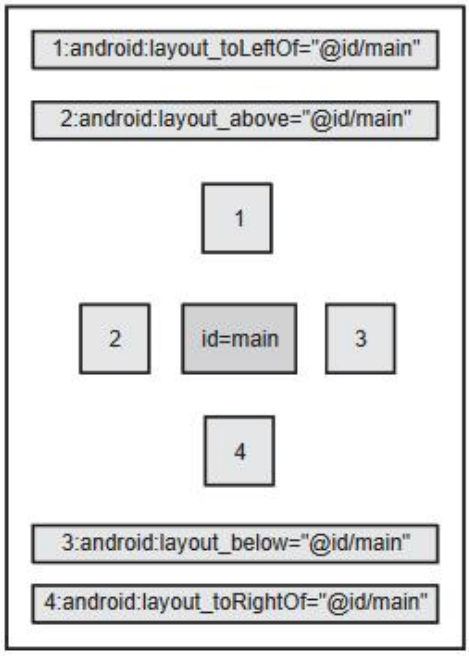
```
<!-- align parent left in Android -->
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:textAppearance="?android:attr/textAppearanceLarge"
   android:id="@+id/textView"
   android:text="Text in Parent Left"
   android:layout_alignBottom="@+id/imageView"
   android:layout_alignParentLeft="true"
    />
```

4. **android:layout_alignParentRight="true"** : If alignParentRight property is true, then it makes the right edge of this view match the right edge of the parent. The value of align parent right is either true or false. Example: android:layout_alignParentRight="true".

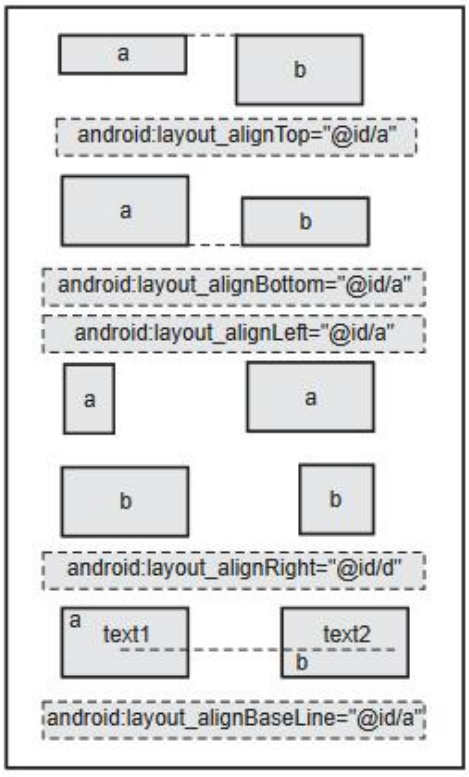
```
<!-- alignRightParent Example -->
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:textAppearance="?android:attr/textAppearanceLarge"
  android:id="@+id/textView"
   android:text="Text in Parent Right"
   android:layout_alignBottom="@+id/imageView"
   android:layout_alignParentRight="true"
 />
```

3. Place new View relative to existing sibling View

- In a RelativeLayout you can keep(position) the new views relative to other existing views. Following attributes can be used for doing so.
- Suppose there is one view in the center and its id is given as android:id="@+id/main" Therefore, the other new views can be placed relative to this view as following :



1. **android:layout_toLeftOf="@id/main"** : This tells the new view that you have to be on the left side of the view whose id is **main**.
2. **android:layout_toRightOf="@id/main"** : This tells the new view that you have to be on the right side of the view whose id is **main**.
3. **android:layout_above="@id/main"** : This tells the new view that you have to be above the view whose id is **main**.
4. **android:layout_below="@id/main"** : This tells the new view that you have to be below the view whose id is **main.**



Note : When you assign an id to the View, you write **android:id="@+id/main"** i.e you write a + sign after the @ symbol, indicating you are assigning an id to that view. But when you use that id for other purpose, like above, you are adding a new view relative to an existing view having the specfied value of id, hence we do not have to mention the + sign. We simply refer it as **android:layout_below="@id/main"** i.e without the + sign.

Align new View relative to existing sibling View

If you want to align the new view relative to any existing view, then you can use the following attributes.

1. **android:layout_alignTop="@id/a"** : This aligns the top margin of the new view with the top margin of the view having id as a.
2. **android:layout_alignBottom="@id/a"** : This aligns the bottom margin of the new view with the bottom margin of the view having id as a.

3. **android:layout_alignLeft="@id/a"** : This aligns the left margin of the new view with the left margin of the view having id as a.
4. **android:layout_alignRight="@id/a"** : This aligns the right margin of the new view with the right margin of the view having id as a.
5. **android:layout_alignBaseLine="@id/a"** : This aligns the text1 of the new view with the text2 of the view having id as a.

4. above :

- **android:layout_above -** The control is placed above the referenced control.

Position the bottom edge of the view above the given anchor view ID and must be a reference of the another resource in the form of id. Example, android:layout_above="@+id/textView" .

For example, suppose a view with id textview2 is what we want to place above another view with id textview.

```
<!-- textView2 is placed above textView-->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:text="Text2"
    android:id="@+id/textView2"
    android:layout_above="@+id/textView"
    android:layout_marginBottom="100dp"
    android:layout_centerHorizontal="true"/>
```

- The attributes to control the position of a control in relation to other controls are
- **android:layout_below** - The control is placed below the referenced control.
- **android:layout_toLeftOf** - The control is placed to the left of the referenced control.
- **android:layout_toRightOf** - The control is placed to the right of the referenced control.

5. alignBottom :

- **android:layout_alignBottom -** The bottom of the control is set to align with the bottom of the referenced control.

  alignBottom is used to makes the bottom edge of the view match the bottom edge of the given anchor view ID and it must be a reference to another resource, in the form of id. Example: android:layout_ alignBottom ="@+id/button1?

For example align a view with id textView2 Bottom of another view with id textView..

```
<!-- textView2 alignBottom of textView -->
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:textAppearance="?android:attr/textAppearanceLarge"
   android:layout_centerHorizontal="true"
   android:id="@+id/textView2"
   android:layout_alignBottom="@+id/textView"
   android:text="Text2 alignBottom of Text1"
   android:layout_marginBottom="90dp"
/>
```

6. alignLeft :

- **android:layout_alignLeft -** The left side of the control is set to align with the left side of the referenced control.

  alignLeft is used to make the left edge of the view match the left edge of the given anchor view ID and must be a reference to another resource, in the form of Example: android:layout_ alignLeft ="@+id/button1?.

  For example align a view with id textView2 left of another view with id textView.

```
<!-- textView2 alignLeft of textView -->
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:textAppearance="?android:attr/textAppearanceLarge"
   android:id="@+id/textView2"
   android:layout_alignLeft="@+id/textView"
   android:text="Text2 alignLeft of Text1"
```

```
android:layout_below="@+id/textView"
android:layout_marginTop="20dp"/>
```

7. alignRight :

- **android:layout_alignRight** - The right side of the control is set to align with the right side of the referenced control.

  alignRight property is used to make 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 like this example: android:layout_alignRight="@+id/button1?

  For example align a view with id textView2 right of another view with id textView.

```
<!-- textView2 alignRight of textView-->
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:textAppearance="?android:attr/textAppearanceLarge"
   android:id="@+id/textView2"
   android:layout_alignRight="@+id/textView"
   android:text="Text2 alignRight of Text1"
   android:layout_below="@+id/textView"
   android:layout_marginTop="20dp"/>
```

8. alignStart :

- alignStart property is used to 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 of like this example: android:layout_alignStart="@+id/button1?

  For example align a view with id textView2 start of another view with id textView.

```
<!-- Text2 alignStart-->
<TextView
   android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/textView2"
    android:text="Text2 align start of Text1"
    android:layout_alignStart="@+id/textView"
/>
```

9. alignTop :

- **android:layout_alignTop** - The top of the control is set to align with the top of the referenced control. alignTop property is used to 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 like this example: android:layout_alignTop="@+id/button1?.

For example align a view with id textView Top of another image with id imageView.

```
<!--text is align top on Image-->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/textView"
    android:layout_alignTop="@+id/imageView"
    android:text="Text Here is AlignTop on Image"
    />
```

10. **alignParentEnd :** If alignParentEnd property is true, then it makes the end edge of this view match the end edge of the parent. The value of align parent End is either true or false. Example: android:layout_alignParentEnd="true".

For example textView is simply displayed on Image in the end.

```
<!-- Text displayed in the end of parent image-->
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:textAppearance="?android:attr/textAppearanceLarge"
   android:id="@+id/textView"
   android:text="Text in Parent End"
   android:layout_alignBottom="@+id/imageView"
   android:layout_alignParentEnd="true"
/>
```

11. **alignParentStart:** If alignParentStart is true, then it makes the start edge of this view match the start edge of the parent. The value of align parent start is either true or false. Example: android:layout_alignParentStart="true".
    For example textView is simply displayed on parent Image in the right side.

```
<!-- alignParentStart Example -->
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:textAppearance="?android:attr/textAppearanceLarge"
   android:id="@+id/textView"
   android:text="Text in Parent Start"
   android:layout_alignTop="@+id/imageView"
   android:layout_alignParentStart="true"
   />
```

12. For spacing, Android defines two attributes: android:layout_margin and android:padding .
    - **The android:layout_margin** attribute defines spacing for the container, while android:padding defines the spacing for the view. Let's begin with padding.
    - **android:padding** - Defines the spacing of the content on all four sides of the control. To define padding for each side individually, use android:paddingLeft ,android:paddingRight , android:paddingTop , and android:paddingBottom .
    - **android:paddingTop** - Defines the spacing between the content and the top of the control.
    - **android:paddingBottom** - Defines the spacing between the content and the bottom of the control.

- **android:paddingLeft** - Defines the spacing between the content and the left side of the control.
- **android:paddingRight** - Defines the spacing between the content and the right side of the control.

13. Here are the attributes that define the spacing between the control and the container:
    - **android:layout_margin** - Defines the spacing of the control in relation to the controls or the container on all four sides. To define spacing for each side individually, we use the android:layout_marginLeft , android:layout_marginRight , android:layout_marginTop , and android:layout_marginBottom options.
    - **android:layout_marginTop** - Defines the spacing between the top of the control and the related control or container.
    - **android:layout_marginBottom** - Defines the spacing between the bottom of the control and the related control or container.
    - **android:layout_marginRight** - Defines the spacing between the right side of the control and the related control or container.
    - **android:layout_marginLeft** - Defines the spacing between the left side of the control and the related control or container.

Example

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingLeft="10dp"
   android:paddingRight="10dp">
   <Button
      android:id="@+id/btn1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:text="Button1" />
   <Button
      android:id="@+id/btn2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentRight="true"
      android:layout_centerVertical="true"
      android:text="Button2" />
   <Button
      android:id="@+id/btn3"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:layout_centerVertical="true"
      android:text="Button3" />

   <Button
      android:id="@+id/btn4"
      android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:text="Button4" />
    <Button
        android:id="@+id/btn5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/btn2"
        android:layout_centerHorizontal="true"
        android:text="Button5" />
    <Button
        android:id="@+id/btn6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/btn4"
        android:layout_centerHorizontal="true"
        android:text="Button6" />
    <Button
        android:id="@+id/btn7"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toEndOf="@+id/btn1"
        android:layout_toRightOf="@+id/btn1"
        android:layout_alignParentRight="true"
        android:text="Button7" />
</RelativeLayout>
```

MainActivity.java

```
package com.vrs.relativelayout;
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);
    }
}
```

Output

Example :

activity_relative_layout_android_example.xml :

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

            <TextView
                android:id="@+id/text1"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerInParent="false"
    android:text="LOGIN"
    android:layout_marginTop="14dp"
    android:textAppearance="?android:attr/textAppearanceLarge"
    />

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/text1"
    android:layout_marginTop="20dp"
    android:text="Username :"
    android:textAppearance="?android:attr/textAppearanceLarge" />

<EditText
    android:id="@+id/editText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:layout_alignTop="@+id/textView1"
     android:layout_toRightOf="@+id/textView1"
    />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView1"
    android:layout_marginTop="20dp"

    android:text="Password :"
    android:textAppearance="?android:attr/textAppearanceLarge" />

<EditText
    android:id="@+id/editText2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
     android:layout_alignTop="@+id/textView2"
     android:layout_toRightOf="@+id/textView2"
    android:inputType="textPassword"
    />

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

Output

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignParentLeft="false"
android:layout_below="@+id/editText2"
android:layout_centerInParent="true"
android:text="Submit" />

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignParentBottom="true"
android:text="SIGNUP"
android:layout_centerHorizontal="true"/>

</RelativeLayout>
```

Difference between Linear And Relative Layout :

Relative Layout :

- Every element of relative layout arranges itself to the other element or a parent element.
- It is helpful while adding views one next to other etc
- In a relative layout you can give each child a Layout Property that specifies exactly where it should go in relative to the parent or relative to other children.
- Views can be layered on top of each other.

Linear Layout :

- In a linear layout, like the name suggests, all the elements are displayed in a linear fashion either vertically or horizontally.
- Either Horizontally or Vertically this behavior is set in android:orientation which is an property of the node Linear Layout.

`android:orientation="horizontal" or android:orientation="vertical"`

- Linear layouts put every child, one after the other, in a line, either horizontally or vertically.

❑❑❑

Notes