Android

BccFalna.com
97994-55505

Kuldeep

With this eBook you can Learn Core Concepts of an Android App Development with Deep Details in easy to understand Hindi Language.

I have included so many Examples and Code Fragements in this ebook to easily understand various kinds of App Development Concepts with Detaild Program Flow Discussion to understand the Internal Working of the App Step by Step.

So, learn Core Android App Development Basics and start moving in the way of Professional Android App Development for full of Joy and Healthy Programming Career.

I am sure, you will not regret with this eBook. It's a Value for Money EBook that would help you to Learn and Understand Underlying Basics and Core concepts of Android App development.

Android

In Hindi

Kuldeep Chand

BetaLab Computer Center
Falna

Android in HINDI

Lead Editors: **Kuldeep Chand**

Distributed to the book trade worldwide by Betalab Computer Center, Behind of Vidhya Jyoti School, Falna Station Dist. Pali (Raj.) Pin 306116

e-mail bccfalna@gmail.com,

or

visit http://www.bccfalna.com

For information on translations, please contact BetaLab Computer Center, Behind of Vidhya Jyoti School, Falna Station Dist. Pali (Raj.) Pin 306116

Phone **097994-55505**

This book is dedicated to those

who really wants to be

a

PROFESSIONAL DEVELOPER

INDEX OF CONTENTS

Index of Content

THE INTRODUCTION

Android – The Introduction

वर्तमान समय में Android Platform केवल Mobile Phones या Tablets में ही Use हो, ऐसा नहीं है। वास्तव में Android Platform Windows, Linux, Unix, MacOS की तरह ही Mobile जैसी छोटी Devices के लिए Develop किया गया एक Operating System है, जिसे किसी भी Digital Device में Platform की तरह Use किया जा सकता है।

इसीलिए Android Platform आधारित न केवल Mobile Phones व Tablet PCs आ रहे हैं, बल्कि Google ने इसे Set-Top Box व TVs में भी Add करना शुरू कर दिया है, जिसे Google TV के नाम से जाना जाता है। ठीक इसी तरह से Android Platform को विभिन्न प्रकार की अन्य Digital Devices जैसे कि Cars, Airplanes व Robots में भी मुख्य Platform की तरह Use किया जाने लगा है।

फिर भी Android Platform को मूल रूप से ऐसी Small Screen वाली Devices के लिए ही Develop किया गया था, जिनमें उसी तरह का Keyboard जैसा कोई Hardware Input Device नहीं होता, जैसा Computer System के साथ होता है, बल्कि Data Input करने के लिए इन Devices में इनकी Touch Screens को ही उपयोग में लेते हुए उस पर ही Virtual Keyboard बना दिया जाता है।

इसलिए ऐसा माना जा सकता है कि Android Platform को भविष्य में भी मूल रूप से Small Screen Devices जैसे कि Smart-Phones के लिए ही Develop किया जाने वाला है और ये स्थिति एक Android Developer के रूप में हमारे लिए Advantage भी है और कमी भी।

वर्तमान समय के Android आधारित Smart-Phones काफी आकर्षक होते हैं लेकिन 1990 के दशक के मध्य में Mobile Devices पर Internet Services Provide करने के लिए **Handheld Device Markup Language (HDML)** का प्रयोग करना पडता था और इस HDML द्वारा Provide की जाने वाली Internet Services की Capabilities भी काफी Limited हुआ करती थीं। जबकि वर्तमान समय के Phones काफी आसानी से Internet Services से सम्बंधित Smart व Advance Capabilities से युक्त होते हैं और इसका मुख्य कारण है, Apple का **iPhone**.

Apple Company द्वारा **iPhone** को एक ऐसी Smart Device के रूप में Develop किया गया था, जिस पर Internet द्वारा Provide की जाने वाली Services को उनकी पूर्ण क्षमता के साथ ठीक उसी तरह से Use किया जा सकता है, जिस तरह से उन्हें एक Computer System पर Use किया जाता है।

हालांकि iPhone से पहले भी कई कम्पनियों के Smart Phones Devices Market में उपलब्ध थे, लेकिन किसी भी Device को एक Smart Device बनाने हेतु सबसे महत्वपूर्ण Role हमेंशा उस Device के Operating System द्वारा Play किया जाता है और iPhone से पहले जितने भी Smart Devices उपलब्ध थे, उनके Operating Systems इतने Advance नहीं थे कि Internet द्वारा Provide किए जाने वाले सभी Features को उनकी पूर्ण क्षमता के साथ Run कर सकें।

इसलिए उस समय के Smart Devices के Operating System की इस कमी को सबसे पहले Apple ने पहचाना और iOS नाम का वह Operating System Develop किया जिसने Apple के **iPhone** को एक Computer की तरह ही Smart Internet Service Device बना दिया।

लेकिन क्योंकि iOS को मूल रूप से Apple Company द्वारा Develop किया गया था और Apple Company अपनी सभी Devices (Computer, Mobile, Tablet PC, Watch, Glass, etc…) स्वयं बनाता है, इसलिए इन Devices के लिए उसके द्वारा Develop किया गया iOS उसका स्वयं का Intellectual Property है, जिसे कोई भी अन्य Company अपने Device में Use नहीं कर सकता था।

परिणामस्वरूप iOS की वजह से **iPhone** व **iPad** के रूप में Mobile व Tablet PCs के Market में Apple का एकाधिकार हो गया था, जिसकी वजह से Apple अपने Products (iPhone, iPad, Watch, Glass, etc…) की कीमतें काफी ज्यादा रखने लगा और महंगे होने की वजह से ये Devices आम लोगों की पहुंच में नहीं आ पाईं। इसलिए iOS की तरह ही एक ऐसे Platform की जरूरत महसूस की गई, जो कि Internet Features युक्त Smart Devices के Market से Apple के एकाधिकार को समाप्त कर सके और iPhone/iPad की तरह ही Smart Devices को आम लोगों के लिए उपलब्ध हो सके।

इसी जरूरत के परिणाम के रूप में Android OS हमारे सामने है, जो कि पूरी तरह से iOS के समान ही Features Provide कर सकता है लेकिन ये एक पूरी तरह से Open Source व Free Platform है, जिसकी वजह से इस Platform को Use करते हुए कोई भी कम्पनी Android Platform पर आधारित नए Device बना सकता है।

वास्तव में Android Platform को Apple के iPhone के Market में आने के कई सालों पहले 2003 से ही **Android Inc.** नाम की एक कम्पनी द्वारा Develop किया जा रहा था, जो कि ***Andy Rubin, Rich Miner, Nick Sears*** व ***Chris White*** नाम के चार लोगों की Joint Development Company थी और ये लोग Mobile Phone के लिए नहीं बल्कि मूलत: Digital Cameras के लिए एक Advance Operating System Develop कर रहे थे, लेकिन कुछ समय काम करने के बाद उन्हें महसूस हुआ कि Digital Cameras का भविष्य अच्छा नहीं है और Digital Cameras, वास्तव में Mobile Phone का एक Feature मात्र रह जाऐंगे। इसलिए इन Developers ने Digital Cameras के स्थान पर Smart Mobile Phones के लिए Operating System Develop करने पर अपना ध्यान केंद्रित कर दिया और जो Research Work इन्होंने Digital Cameras के लिए Advance Operating System बनाने हेतु किया था, वो सारा Research Work इनके Mobile Phone का Operating System Develop करने में काम आ गया।

हालांकि इन लोगों द्वारा Android Platform के Development को पूरी तरह से Secret Project की तरह ही Develop किया जा रहा था, लेकिन 2005 में इन Developers की कम्पनी को आर्थिक तंगी का सामना करना पड़ा जिसकी वजह से इनका Secret Project, Secret नहीं रह गया और Google की नजर में आ गया। गूगल को इस Android Project का भविष्य उज्जवल दिखाई दिया क्योंकि Google, Mobile Market में Enter करना चाहता था, जिसके लिए उसे एक ऐसे ही Mobile Operating System की तलाश थी। परिणामस्वरूप गूगल ने पूरे **Android Inc.** को उनके Developers सहित 50 Million Dollars में अधिग्रहित कर लिया।

गूगल ने कई तरीकों से अपने Android OS के साथ Mobile Market में प्रवेश करने की कोशिश की, लेकिन किसी भी तरह से वह Mobile Market में ज्यादा स्थान नहीं बना पाया, जबकि इसी दौरान Apple के iPhone ने Launch होते ही जबरदस्त तरीके से Mobile Market को Capture कर लिया, जिसका मूल कारण ये था कि Apple अपने Devices स्वयं ही बनाता था जबकि Android केवल एक Operating System था जिसके लिए अन्य कम्पनियों को Mobile Phone बनाने पड़ते थे और मात्र इस Android OS के लिए कोई भी कम्पनी Google को Pay करना नहीं चाहता था।

Mobile Market में प्रवेश करने के लिए गूगल ने कई तरीकों से और कई कम्पनियों के साथ विभिन्न प्रकार के करार किए थे और iPhone के Launch के बाद Smart Mobile Device की जबरदस्त सफलता को देखते हुए वे सभी कम्पनियां आपस में मिलीं और Mobile जैसी Wireless Devices हेतु

Standard Operating System बनाने के लिए एक Open Standard Define करने हेतु प्रेरित हुए ताकि भविष्य में जितने भी Devices बनें, वे एक दूसरे के साथ आसानी से Communicate कर सकें और जिस समय इस Open Standard के लिए विभिन्न कम्पनियां सम्मिलित हुईं, उस समय तक गूगल Wireless Devices से सम्बंधित विभिन्न प्रकार के ढेर सारे Patent Register करवा चुका था इसलिए गूगल का Android Operating System एक Standard के रूप में Prototype की तरह Use किए जाने हेतु सबसे उपयुक्त, Advance व Developed OS था।

परिणामस्वरूप Android OS के लिए Register किए गए विभिन्न Patents एक Open Source Standard की तरह उपलब्ध हो गए और जैसे ही Android OS को Open Source Standard की तरह मान्यता प्राप्त हुई, दुनियांभर की विभिन्न कम्पनियों ने Android OS का उपयोग करते हुए Mobile Devices बनाने शुरू कर दिए और HTC सबसे पहली कम्पनी था जिसने Android के Open Standard के आधार पर सबसे पहले Android Mobile Phone को Apple के iPhone की टक्कर में Commercially Launch किया था।

इस प्रकार से Apple का iPhone, Android OS के Open Source Standard Platform बनने की वजह बना और Apple के iOS के Alternative के रूप में विभिन्न Smart Device बनाने वाली कम्पनियों द्वारा Use किया जाने लगा। परिणामस्वरूप iOS आधारित iPhone/iPad केवल Apple ही बनाता है लेकिन Android आधारित Devices (*Mobile Phones, Tablets, TVs, Dish, Watch, Glass, etc…*) न केवल **Samsung, Sony, HTC, MicroMax, Karbonn** जैसी Reputed Companies बनाती हैं, बल्कि कई China Companies भी सस्ते Smart Devices बनाती हैं, जो कि Android OS के Free उपलब्ध होने से पहले अपने स्वयं के Platform को Use करते हुए सस्ते Smart Devices बनाते थे।

iPhones/iPad Launch होने के साथ ही बडी ही तेजी से Popular हो गए थे क्योंकि ये Smart Devices के रूप में Internet को उसकी पूर्ण क्षमता के साथ उपयोग में लेने की सुविधा Provide करते थे। इसीलिए जब एक बार Android को Open Source Standard की तरह मान्यता प्राप्त हो गई, तो फिर Google द्वारा इसे मूल रूप से Apple के **iOS Platform** के Alternative के रूप में ही Develop किया जाने लगा, ताकि Smart Devices के Market में Apple की Monopoly को समाप्त किया जा सके व Apple iPhone द्वारा Provide किए जाने वाले सम्पूर्ण Features युक्त Smart Devices को आम लोगों तक काफी कम कीमत पर पहुंचाया जा सके ताकि वे भी Smart Devices का फायदा उठा सकें।

इसलिए जब हम Android Platform के लिए कोई App Design कर रहे होते हैं, तो हम वास्तव में काफी High Quality के Applications Develop कर रहे होते हैं जो कि Android के Open Source Standards पर आधारित होते हैं और हम ये Android Application, User को **iPhone/iPad** की तुलना में काफी कम कीमत पर अथवा मुफ्त उपलब्ध करवा रहे होते हैं क्योंकि Android आधारित Devices काफी सस्ते होते हैं।

साथ ही **Android Devices** (*Smart Phone, Tablet PCs, Watch, Glass, TV, etc...*) Apple द्वारा Provide किए जाने वाले iOS Devices की तुलना में काफी कम कीमत पर उपलब्ध होते हैं, इसलिए हमारे Android Apps की पहुंच iPhone/iPad Users की तुलना में कई गुना ज्यादा Users तक होती है। परिणामस्वरूप हम हमारे Android App की कीमत काफी कम या Free रखते भी अन्य तरीकों से काफी Earning कर सकते हैं।

लेकिन Android आधारित Devices के लिए Program Develop करना थोडा मुश्किल होता है, क्योंकि इन Devices की Screen सभी Dimensions में काफी छोटी होती है, इनके Keypad काफी छोटे होते हैं जो कि Hardware Keypad के स्थान पर Software Keypad होते हैं। इन Devices से सम्बंधित Pointing Devices भी काफी छोटे होते हैं और जिन लोगों के हाथ की अंगुलियां बडी होती हैं, उनके लिए ये छोटे Pointing Device, छोटी Screen व छोटे Touch Keypad Buttons काफी परेशानी पैदा करते हैं।

इतना ही नहीं, इन Devices के CPU व Memory की Speed तथा Storage भी Desktop व Server Computers की तुलना में काफी कम होते हैं, जिसकी वजह से इन Devices में बडे Application Software ठीक से Perform नहीं करते।

लेकिन Mobile Phone के Applications की सबसे बडी विशेषता भी यही होती है, कि वे एक ऐसी Device में होते हैं, जिन्हें User अपनी सुविधानुसार कभी भी और कहीं भी उपयोग में लेते हुए अपनी किसी Specific जरूरत को पूरा कर सकता है।

हालांकि Mobile Phones के कम व Slow CPU, RAM तथा कम Internal व External Storage एक Developer के रूप में हमारे लिए काफी Challenges पैदा करते हैं, क्योंकि **Desktop Applications** Develop करते समय ये Limitations वास्तव में कोई Limitation नहीं होते और वर्तमान समय के Modern Computers के CPU, RAM व Storage कोई विशेष परेशानी पैदा नहीं करते, बल्कि हम जरूरत के अनुसार बडी ही आसानी से इन्हें Change या Upgrade भी कर सकते हैं, लेकिन Mobile Devices के साथ ऐसा नहीं किया जा सकता क्योंकि Mobile Devices में CPU, RAM व Internal Storage को PCB पर ही Directly Sold कर दिया गया होता है, इसलिए Mobile Devices को आसानी से Upgrade करना सम्भव नहीं होता बल्कि Device को Upgrade करने का सीधा सा मतलब यही होता है कि Device को Change किया गया है।

लेकिन दूसरे Point of View से देखें, तो जब हम Android आधारित Application Programs Develop करना चाहते हैं, तब Android SDK के रूप में हमें हमेंशा कुछ Standard Libraries के साथ Programming Language के रूप में **C++** अथवा **Java** जैसी Mature Languages को ही Use करना होता है और एक Standard Framework को Follow करना होता है, जिसकी वजह से एक Android Application Develop करना किसी Desktop या Web Application Develop करने की तुलना में काफी आसान भी हो जाता है।

यानी Android Applications Develop करने के लिए केवल **Core Java** का Basic Knowledge होना ही पर्याप्त है और यदि आपको Core Java का Basic Knowledge है, तो Android Applications Develop करने के लिए आपको केवल **Android Framework** को ही समझना होता है, जो कि पूरी तरह से Java Classes को एक व्यवस्थित तरीके से Use करना मात्र है।

अन्य शब्दों में कहें तो एक Java Programmer एक प्रकार से Android Programmer भी होता है और यदि आपने कभी भी कोई **Java Applet** Create किया है, तो Android App Development को समझना आपके लिए बिल्कुल भी मुश्किल नहीं होगा क्योंकि Java Applet की तरह ही एक Android App का भी निश्चित **Life-Cycle** होता है जिसके कुछ निश्चित **Methods** होते हैं और हमें केवल उन Methods को अपनी जरूरत के अनुसार Override करना होता है तथा Google द्वारा Provide किए गए **Android SDK** की Libraries को Use करते हुए ही Smart Device के विभिन्न Hardware जैसे कि *WiFi, Bluetooth, Screen, Keypad, Sensors* आदि को Access व Manipulate करना होता है।

Android App – The Basic Architecture

जब हम Desktop Application Develop करते हैं, तब हमारे Application पर हमारा पूरा Control होता है, जिसके अन्तर्गत हम हमारी जरूरत के अनुसार कभी भी अपने Application के Main Window को Launch कर सकते हैं व उसमें किसी Specific Child Window जैसे कि Dialog Box को Open कर सकते हैं।

यानी एक Desktop Application Develop करते समय हमारे Application के विभिन्न Aspects पर हमारा पूरा Control होता है, जिसकी विभिन्न जरूरतों को Operating System द्वारा पूरा किया जाता है।

लेकिन हमारे Application का किसी भी अन्य Application के साथ Directly कोई भी Interaction नहीं होता, हालांकि अन्य Applications भी उसी Operating System पर Run हो रहे होते हैं और वही Operating System उन अन्य Applications की विभिन्न Requirements को भी पूरा कर रहा होता है।

जबकि यदि हमें हमारे Application Program द्वारा किसी अन्य Application के साथ Interact करना हो, तो ये सुविधा हमें एक Separate API (Application Programming Interface) जैसे कि **Java Database Connectivity (JDBC)** या किसी अन्य Framework द्वारा प्राप्त होती है।

Android Applications भी कुछ इसी तरह से काम करता है, लेकिन Android Applications को थोडा अलग तरीके से Package किया जाता है, ताकि जिस Phone पर इन Applications को Run किया जाए, उन Phone Devices पर किसी भी प्रकार का Software Crash जैसा Issue पैदा न हो।

यानी प्रत्येक Android Application Crash-Registrant हो, इसलिए उन्हें एक Specific तरीके से Package किया जाता है, जिसके निम्नानुसार विभिन्न Important Parts होते हैं:

Views

User Screen पर जो कुछ भी देखता है, उसे ही User Interface (UI) कहते हैं और User के सामने कौनसा User Interface दिखाई देगा, इसे Set करने की जिम्मेदारी Activity की होती है।

किसी भी Android App में User Interface मूल रूप से दो Sub-Components के बने होते हैं, जिन्हें **Views** व **Layouts** या **ViewGroups** के नाम से जाना जाता है।

किसी Android App के User Interface (UI) Elements जैसे कि **Button, Label, Text Box, Radio Button, Checkbox** आदि को **Views** कहते हैं जो कि Android App का User Interface Create करते हैं, ताकि User उस Android App से Interact कर सके।

जबकि इन **View Elements** को Hold करने के लिए Container की तरह काम करने वाले अन्य Container Views को **Layouts** अथवा **ViewGroups** कहते हैं, क्योंकि ये Containers वास्तव में Multiple UI Views को Hold करते हुए हमारे Android App के User Interface का Layout Define करते हैं और क्योंकि एक User Interface, Multiple UI View Elements का Group होता है इसलिए इन Container Views को **ViewGroup** भी कहा जाता है जो कि *Group of Views* को Represent करते हैं।

चूंकि **Layout**, अन्य UI View Elements का Container होते हैं, इसलिए ये मूल रूप से किसी Android App के Screen पर दिखाई देने वाले View Elements के Pattern को Specify करते हुए इस बात को तय करते हैं कि विभिन्न User Interface Elements Screen पर Visually (*Height, Width, Position, etc…*) किस तरह से दिखाई देंगे।

उदाहरण के लिए **LinearLayout** Use करने पर इसके अन्दर Render होने वाले विभिन्न User Interface Elements **Vertically** (एक UI Element के ऊपर या नीचे दूसरा) अथवा **Horizontally** (एक UI Element के पहले या बाद में दूसरा) Stack होते हैं जबकि **RelativeLayout** Use करने पर इसके अन्दर Render होने वाले विभिन्न User Interface Elements अपने Parent अथवा Sibling UI Element के संदर्भ में Render होते हैं।

Android Application के User Interface को XML Markups के माध्यम से ठीक उसी तरह से Define किया जाता है, जिस तरह से हम किसी Webpage पर किसी Registration Form को HTML के **<form>** Markup द्वारा Define करते हैं।

यानी जिस तरह से एक Web Browser का **HTML Parser** हमारे HTML Webpage के <form> Element में Specify किए गए *<button>*, *<input>*, *<label>* आदि ***HTML Markups*** को *Button, TextBox* **व** *Label* जैसे **GUI Controls** के रूप में Render करता है, ठीक उसी तरह से एक Android App के User Interface को हम जिन ***XML Markups*** का प्रयोग करते हुए Define करते हैं, Android App का **XML Parser** उन XML Markups के आधार पर Android Device की Screen पर **User Interface Controls** को Display कर देता है।

उदाहरण के लिए जब भी हम **RelativeLayout** पर आधारित कोई नई **Activity** Create करते हैं, उस Activity के साथ ही Automatically एक **Layout View** भी Create हो जाता है और उस Automatically Create होने वाले **View** में Default रूप से निम्न XML Markups होते हैं-

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
  xmlns:android=http://schemas.android.com/apk/res/android
  xmlns:tools=http://schemas.android.com/tools
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  android:paddingBottom="@dimen/activity_vertical_margin"
  tools:context=".MainActivity" >

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

</RelativeLayout>
```

Create होने वाली View File मूलत: एक XML File होती है इसलिए किसी भी XML File की तरह ही इसकी शुरूआत भी निम्नानुसार **<?xml />** Element से होती है:

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

जो Android Framework के XML Parser को इस बात की जानकारी देता है कि Current File एक XML File है, जो कि Android App के **Visual Part** यानी **GUI Part** को Control करता है।

साथ ही किसी भी Framework की तरह ही Android Framework में भी Files का Naming Convention महत्वूपर्ण Role Play करता है। इसलिए इस File का नाम इसके साथ Create होने वाली **Activity** File के नाम से सम्बंधित होता है।

उदाहरण के लिए जब **MainActivity**.java नाम का Activity File Create होता है, तो उसके साथ Create होने वाले Layout View File का नाम **activity_main**.xml होता है। इसी तरह से यदि हम हमारे Current Android App में SecondActivity नाम की एक और Activity Create करें, तो Create होने वाली **SecondActivity**.java File के साथ जो Layout View File Create होता है, उसका नाम **activity_second**.xml होता है।

हालांकि यदि हम चाहें तो Android Framework के इस Naming Convention को Follow न करते हुए इन नामों को अपनी इच्छानुसार बदल भी सकते हैं, लेकिन बेहतर यही होता है कि हम इन नामों को न बदलते हुए Android Framework के Naming Convention को ज्यों का त्यों Follow करें क्योंकि जब हम इन Naming Conventions को ज्यों का त्यों Follow करते हैं, तो Android Development के लिए हम जिस किसी भी IDE को Use कर रहे होते हैं, वह IDE हमारे लिए काफी Android Codes Automatically Generate कर देता है, जिससे हमारे App Development की Speed काफी बढ़ जाती है।

चूंकि Android App का Layout व GUI Part, XML Markup आधारित Language द्वारा Create किया जाता है, जिसे Android Framework का XML Parser Parse करता है, इसलिए हम जब कोई नया Activity Create करते हैं, उसके साथ Create होने वाले Layout View में उपरोक्तानुसार Specified **<RelativeLayout … />** व **<TextView … />** नाम के दो Elements होते हैं और जैसाकि हमने पहले बताया कि Android App के User Interface मूलत: दो प्रकार के Sub-Components **Views** व **Layout या GroupViews** के बने होते हैं।

अत: इस Automatically Generate होने वाले XML Code में **<RelativeLayout … />** Element एक प्रकार का Layout अथवा GroupViews है, जो कि **<TextView … />** जैसे कई View Elements को एक Group के रूप में Contain करते हुए Create होने वाले Android App के User Interface को Organize करता है।

अन्य शब्दों में कहें तो Layout/GroupViews Element स्वयं Visually दिखाई नहीं देता, लेकिन Android App का User Interface कैसा दिखाई देगा और User Interface के विभिन्न UI Elements (*Button, Textbox, Checkbox, Radio Button, Images, etc…*) Screen पर कहां Place होंगे और कैसे दिखाई देंगे, इस बात को तय करके User Interface को Organize करने का काम Layout/GroupViews Element द्वारा ही तय होता है।

इसी वजह से उपरोक्त XML Code में **<TextView … /> Element** को **<RelativeLayout … /> Element** के बीच Enclose किया गया है क्योंकि <TextView … /> Element एक View Element है, जिसे हमेंशा किसी न किसी Layout/GroupViews Element के अन्दर Contained होना होता है और <RelativeLayout … /> Element किसी भी Android App का Default Container Layout होता है, जिसके अन्तर्गत प्रत्येक View Element (*Button, Textbox, Radio Button, Checkbox, etc…*) की Screen Positioning (Mobile/Tablet की Screen पर Control Situation), उसके Parent या Sibling UI Element की Screen Positioning से सम्बंधित होती है। इसलिए Current UI Element के Parent/Sibling की Screen Position Change करने पर उसका प्रभाव Current UI Control की Screen Position पर भी पड़ता है।

Android Framework हमें कई प्रकार के Layout Views Provide करता है, जहां <RelativeLayout … />, Android Framework द्वारा Automatically Generated Code का Default Layout View होता है और हम हमारे Android App के लिए अपनी जरूरत के अनुसार चाहे जो भी Layout View Use करें, सभी में सबसे पहले हमें निम्नानुसार दो XML Namespaces को Specify करना जरूरी होता है, जो कि Android के View Parser को इस बात की जानकारी देते हैं, कि उन्हें किसी Mobile/Tablet Screen पर Android App के GUI को किस प्रकार से Render करना है:

```
xmlns:android=http://schemas.android.com/apk/res/android
xmlns:tools=http://schemas.android.com/tools
```

चूंकि जब Android Framework का UI Part पूरी तरह से XML File द्वारा Control होता है, तब एक XML File में Specify किए जाने वाले विभिन्न Elements की कुछ न कुछ Descriptions होती हैं, जिनके आधार पर XML Parser उस XML File की Parsing करना है, यानी XML File को Run करता है और Description के आधार पर ही Output Generate करता है। इसी Description को **XML Namespace** के नाम से **xmlns** Attribute द्वारा Specify किया जाता है।

Namespace के Concept को ठीक से समझने के लिए हम एक C/C++ Program की Analogy लेते हैं जहां हम जानते हैं कि किसी C/C++ Program File में हम Header Files को इसीलिए Include करते हैं, ताकि उसमें Defined Functions, Class, Constants आदि को अपने Current C/C++ Program में Use कर सकें।

उदाहरण के लिए C/C++ में **<stdio.h>** नाम की Header File में **scanf()** नाम के Function को परिभाषित (Define) किया गया है, जो कि Keyboard से अपने वाले Input Data को किस तरह से Accept करना है, इस बात को Describe करता है।

इसलिए जब तक हम हमारे C/C++ Program में <stdio.h> Header File को Include न करें, तब तक हम हमारे Program में Keyboard से Input नहीं ले सकते क्योंकि Input कैसे लेना है, इस बात की जानकारी केवल ***scanf()*** Function के अन्तर्गत Specify किया गया है और ये Specification, <stdio.h> नाम की Header File में लिखा गया है।

परिणामस्वरूप जब हम ऐसे C/C++ Program को Compile करते हैं, जिसमें User Input Accept करने के लिए scanf() Function को तो Use किया गया है, लेकिन <stdio.h> Header File को Include नहीं किया गया है, तो Compilation के दौरान C/C++ का Compiler, Program में Use

किए गए scanf() Function का Description खोजता है, ताकि उसे पता चल सके कि उस इस Function के माध्यम से क्या और कैसे करना है।

किन क्योंकि हमने हमारे C/C++ Program में <stdio.h> नाम की Header File को Include ही नहीं किया है, इसलिए Compiler को पता ही नहीं चलता कि scanf() नाम के Code को Execute कैसे करना है, फलस्वरूप Program Compile नहीं होता और Compiler हमें "**Missing Symbol : scanf**" का Error देता है।

ठीक इसी तरह से उपरोक्त Code में **xmlns** Attribute के माध्यम से हम Android Framework के XML Parser को बताते हैं कि Current XML File में लिखे गए XML Codes को कैसे Parse करना है।

अत: यदि हम **xmlns** Attribute को Specify न करें, और <TextView … /> UI Element को Specify करते हुए Activity Screen पर कोई Text Display करना चाहें, तो उस स्थिति में Android Platform के XML Parser को पता ही नहीं होगा कि <TextView … /> Element मिलने पर उसे करना क्या है, क्योंकि <TextView … /> XML Code मिलने पर उसे एक Text Message को Display करना है, इस बात का Description उस **URI** में है, जिसे xmlns Attribute में Value के रूप में Set किया जाता है। इसलिए जब हम निम्नानुसार xmlns Specify करते हैं:

```
xmlns:android=http://schemas.android.com/apk/res/android
xmlns:tools=http://schemas.android.com/tools
```

तो हम Android Platform के Layout/View XML Parser को ये बता रहे होते हैं कि Screen पर दिखाई देने वाले विभिन्न Visual User Interface Controls को Mobile/Tablet Screen पर कहां और कैसे Render करना है।

क्योंकि Layout/View XML File की Parsing के दौरान **<TextView … />** XML Code मिलने पर एक **Label Control** बनना चाहिए और **<Button … />** XML Code मिलने पर Screen पर एक **Button** दिखाई देना चाहिए, XML Parser को इस बात का Description, xmlns Attribute में Value के रूप में Specify किए गए URI द्वारा ही चलता है और हम अपने View (User Interface) को Create करने से सम्बंधित जितने भी Symbols (*<RelativeLayout … />*, *<TextView … />*, *<Button … />*, आदि) Use करते हैं, उन सभी का Description इसी URI से सम्बंधित होता है।

यदि बिल्कुल ही सरल शब्दों में कहें, तो ये **xmlns** Attribute में Value के रूप में Specify किए गए जाने वाले ये URIs एक प्रकार से C/C++ की Header Files के समान ही होते हैं अत: यदि xmlns में इन्हें Specify नहीं किया, तो User Interface नहीं बनेगा, ठीक उसी तरह से, जिस तरह से यदि **<stdio.h>** Header File को अपने C/C++ Program में Include नहीं किया, तो Keyboard से Input Accept नहीं होगा।

XML Specification हमें एक ही XML File में एक से ज्यादा URI से सम्बंधित Symbols यानी XML Elements को एक ही XML File में ठीक उसी तरह से Use करने की सुविधा Provide करता है, जिस तरह से हम किसी C/C++ Program में एक से ज्यादा Header Files को Include करते हुए उनकी Functionalities को समान Program File में प्राप्त कर लेते हैं। लेकिन किसी XML File में एक से ज्यादा URI से सम्बंधित Symbols को Use करने के लिए हमें xmlns के Prefix Concept का प्रयोग करना पडता है।

उदाहरण के लिए उपरोक्त XML Code के **xmlns:android** Attribute में **xmlns** Namespace है जबकि Colon के बाद Specify किया गया **android** शब्द Prefix है। इसी तरह से **xmlns:tools** Attribute में **xmlns** Namespace है जबकि Colon के बाद Specify किया गया **tools** शब्द Prefix है।

इस प्रकार से Prefix युक्त URI Specify करके हम दो अलग Namespaces से सम्बंधित Descriptions पर आधारित Symbols (XML Codes or XML Elements) को समान XML File में Use कर सकते हैं और दोनों Namespaces से सम्बंधित Symbols द्वारा Provide की जाने वाली Functionalities को समान XML File में प्राप्त कर सकते हैं।

Prefix युक्त Namespaces को Specify करने का एक फायदा ये भी होता है कि यदि दोनों Namespaces में समान Symbol (XML Element) को Define किया गया हो, तब भी दोनों के बीच Name Conflict नहीं होता।

उदाहरण के लिए यदि **xml:android** Attribute से Associated URI की Definition में <red> नाम का एक Symbol यानी XML Element है, जिसे Use करने पर XML Parser एक Red Color की Line Draw करता है, जबकि **xml:tools** Attribute से Associated URI की Definition में भी <red> नाम का एक Symbol यानी XML Element है, लेकिन इसे Use करने पर XML Parser एक Error Message Display करता है।

हम समझ सकते हैं कि दोनों ही URI Definitions में <red> Symbol Exist है, लेकिन XML File में जब हम **<android:red ... />** XML Element Use करते हैं, तो Parsing के बाद Output में एक Red Line Render होता है, जबकि **<tools:red ... />** XML Element Use करते हैं, तो Parsing के बाद Output में एक Error Message Render होता है और हम इन Prefixes को तभी Use कर सकते हैं, जबकि हमने Prefix युक्त Namespaces को Specify किया हो। यदि हमने इन्हें Prefixed तरीके से Specify न किया हो, तो उस स्थिति में एक से अधिक Namespaces में समान Symbol होने पर XML Parsing के दौरान Name Conflict होगा।

चूंकि Android Platform व इसका Development Environment दोनों अभी पूरी तरह से Stable नहीं हैं और Google इनमें समय-समय पर उपयुक्त Modifications करके Different Defaults Set करता रहता है। इसीलिए Android Studio 2.2 तक जब भी कोई नया Android Project Create करते थे, Default रूप से RelativeLayout आधारित Layout ही Create होता था। लेकिन Android Studio 2.2 के बाद के Versions वाले Android Studio में हम जब भी कोई नया Project Create करते हैं, तो Default Layout के रूप में **RelativeLayout** के स्थान पर अब **ConstraintLayout** Create होता है जिसका XML Codes निम्नानुसार होता है-

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

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

क्योंकि Google ने RelativeLayout को ही और बेहतर करके इसके Improved Version को ConstraintLayout की तरह Android Studio के विभिन्न Projects के लिए Default की तरह Set कर दिया है।

लेकिन Android Studio हमें ये सुविधा देता है कि हम इस Default ConstraintLayout को अपनी जरूरत के अनुसार फिर से RelativeLayout में Convert कर सकते हैं और इसके लिए हमें केवल मात्र इतना ही करना है कि हम उपरोक्त XML Code में Highlighted ConstraintLayout Code को निम्नानुसार RelativeLayout से Replace कर दें-

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</ RelativeLayout>
```

जैसे ही हम उपरोक्तानुसार ConstraintLayout को RelativeLayout से Change करते हैं, हमारा Layout एक Relative Layout बन जाता है।

हालांकि इस Section में हमने जो भी Discussion किया वो RelativeLayout के XML Codes पर आधारित था लेकिन इस बात से कोई फर्क नहीं पड़ता क्योंकि Layout Change करने के बावजूद हमने Namespace व Prefix से सम्बंधित जो भी Discussion किया, वो सबकुछ ज्यों का त्यों है और क्योंकि अब Google, Android Apps के लिए Default Layout के रूप में

ConstraintLayout को ही Recommend करता है इसलिए अब आप जब भी कोई नया Android App Create करें, जहां तक सम्भव हो, उसे ConstraintLayout के आधार पर ही Develop करें।

चूंकि Android अभी पूरी तरह से विकसित Platform नहीं है बल्कि जरूरत के अनुसार निरंतर विकासशील है इसलिए Google समय-समय पर Android Studio के विभिन्न प्रकार के Defaults को भविष्य में भी Change करेगा। इसलिए अब ConstraintLayout ही हमेंशा के लिए Android Studio का Default Layout रहेगा, ऐसा मान लेना ठीक नहीं है बल्कि हो सकता है कि Google इस ConstraintLayout को भी और Improve करके कुछ और नए तरह का Layout Define कर दे जो कि आने वाले भविष्य में Android Studio का Default Layout बन जाए।

अत: Google जब भी कोई नया Recommendation दे, हमेंशा उसी को Follow करना अपने Android App को Long Term तक Maximum Android Devices के लिए Useful व Supported बनाए रखने का एकमात्र तरीका है।

Activity

Activity एक **User Interface (UI)** Concept होता है जो सामान्यत: किसी Android Application की एक **Single Screen** को Represent करता है। इस User Interface Screen पर जरूरत के अनुसार एक या अधिक **Views** हो सकते हैं लेकिन Views का होना Compulsory नहीं होता। यानी एक ऐसा Android App भी हो सकता है कि जिसकी Activity में कोई भी View न हो।

किसी Activity को हम हमारे Android App का एक ऐसा Part मान सकते हैं, जो User को कोई Action Perform करने में मदद करता है। सरल शब्दों में कहें तो Activity, User Interface के रूप में वह माध्यम होता है, जो User Interactions के कारण Perform होने वाले विभिन्न Actions को Handle करता है।

उदाहरण के लिए जब User किसी Android App की किसी Screen पर दिखाई देने वाले किसी **Button** को Tap करता है, तो उस Tapping Action के Response में Perform होने वाले Task को इसी **Activity** Part में Specify व Implement किया जाता है।

मानलो कि हम चाहते हैं कि जब User किसी Button को Click करे, तो वह Android App **Close** हो जाए।

इस जरूरत को पूरा करने के लिए **Button Click *Action*** के ***Response*** में Android App को Close करने से सम्बंधित Programming Codes (Program Logics) को **Activity** के अन्तर्गत ही लिखा जाता है, जो कि एक **Java Class** होती है।

क्योंकि Android App की प्रत्येक **Activity** एक **Java Class** File ही होती है जिसके Program Codes द्वारा इस Java Class File से Linked **Layout View** File के User Interface द्वारा जो भी Action Perform किया जाता है, उसे Linked Activity Class File द्वारा ही Handle किया जाता है।

किसी Activity के अन्तर्गत Data View करना, Data Create करना व Data Edit करने से सम्बंधित जरूरतों को पूरा किया जाता है। सरल शब्दों में कहें तो **CRUID** (**C**reate, **R**ead, **U**pdate, **I**nsert, **D**elete) Operations को Android App के Activity Part में Perform किया जाता है।

Activity किसी भी Android App का एक Compulsory Part होता है और किसी भी Android App को Open करने पर जो सबसे पहला Screen Display होता है, वह Activity ही होता है। साथ ही हर Activity में User Interface को या तो Hard Code के रूप में जरूर Specify किया गया होता है अथवा View नाम की एक Separate XML File के रूप में जरूर Create किया गया होता है, जो कि Activity के साथ Associate रहता है।

सरल शब्दों में कहें तो कोई भी Activity बिना User Interface के नहीं हो सकता। फिर भले ही User Interface, **View** नाम की एक Separate XML File के रूप में हो अथवा User Interface को Activity Class में ही Hard Code किया गया हो।

साथ ही जैसाकि हमने पहले कहा कि किसी भी Android App में एक या अधिक Activity हो सकते है, लेकिन सभी Activities में एक ऐसा Activity होता है, जो कि उस Android App का Entry Point होता है और एक Android App में ऐसा केवल एक ही Activity हो सकता है, जिसे सामान्यत: **Launcher Activity** अथवा **Main Activity** के नाम से Refer किया जाता है। यही वह Activity होता है, जो उस Android App के Start होते ही सबसे पहले Screen पर दिखाई देता है।

किसी भी Android App की प्रत्येक Activity का एक निश्चित **Life Cycle** भी होता है, जिसके विषय में हम आगे विस्तार से जानेंगे।

Activity मूलत: एक Java Class File होता है। इसलिए यदि हम चाहें तो User Interface Create करने से सम्बंधित विभिन्न Program Logics को इसी Class File में Hard Code के रूप में लिख सकते हैं।

लेकिन सामान्यत: एक Android App को **MVC Pattern** पर आधारित Application के रूप में Develop किया जाता है, जिसके अन्तर्गत **Activity** एक **Controller** का Role Play करता है, जबकि **User Interface** को **View** के रूप में Define किया जाता है।

परिणामस्वरूप MVC Pattern के कारण हम **Separation of Concerns** का Rule Follow कर पाते हैं जिसकी वजह से App के Program Logics को User Interface से सम्बंधित Visual Logics से Separate रखा जाना सम्भव हो जाता है और अपने App को Develop, Modify, Extend व Manage करना आसान हो जाता है।

Fragment

जब किसी Android App की Screen इतनी बड़ी हो कि उसकी सभी Functionalities को एक Single Activity में Manage करना कठिन हो, तो इस स्थिति में उस बड़ी Activity को कई छोटी–छोटी Sub-Activities में Divide कर दिया जाता है। इन्हीं छोटी–छोटी Sub-Activities को **Fragments** कहते हैं और एक Activity समान समय पर एक या अधिक Fragments को Screen पर Display कर सकता है।

जब Screen छोटी होती है, तब एक Activity में सामान्यत: केवल एक ही Fragment होता है और वही Fragment बड़ी Screen पर भी समान रूप से दिखाई देता है।

Fragments की सबसे बडी विषेशता यही है कि एक बार Create कर लेने के बाद हम जितनी चाहें उतनी Activities में इन्हें बार-बार Reuse कर सकते हैं। साथ ही Activity की तरह ही प्रत्येक Fragment का स्वयं का भी एक अलग Lifecycle होता है और हर Fragment स्वयं स्वतंत्र रूप से Separate Input Events Receive कर सकता है और उसे Response कर सकता है, जिनका Main Activity से कोई Direct संबंध नहीं होता।

किसी Fragment को हमेंशा किसी न किसी Activity में Embedded होना जरूरी होता है और जिस Activity में कोई Fragment Embedded होता है, उसकी Lifecycle का Direct Effect, Fragment की Lifecycle पर पड़ता है। यानी Parent Activity की Lifecycle का End हो जाए, तो Fragment की Lifecycle चाहे Run ही क्यों न हो रही हो, उसका भी End हो जाता है। इसी तरह से जब Parent Activity Pause होता है, तो उसके सभी Embedded Fragments भी Pause हो जाते हैं।

हालांकि जब Activity Resumed यानी Running Stat में होता है, तो उस स्थिति में उसके सभी Fragments को हम स्वतंत्र रूप से Access व Manipulate कर सकते हैं। यानी हमारी जरूरत के अनुसार एक या अधिक Fragments को अपनी किसी भी Activity में स्वतंत्र रूप से Add/Remove कर सकते हैं।

जब हम किसी Fragment को अपनी Activity Layout के एक Part के रूप में Add करते हैं, तो वह Fragment हमारी Activity के View Hierarchy के Layout यानी ViewGroup के अन्दर ही रहता है और वहीं पर अपना स्वयं का अलग Fragment Layout Define करता है।

किसी Fragment को अपनी Activity Layout में Insert करने के लिए हमें हमारे Activity की Layout File में **<fragment> Element** को Existing ViewGroup के अन्दर ही Use करना होता है अथवा अपने Application की Code File में Code के माध्यम से भी इसे Add किया जा सकता है, जिसके बारे में हम आगे जानेंगे।

हालांकि हम हमारी जरूरत व इच्छानुसार किसी Fragment को अपनी Activity में Add कर सकते हैं, लेकिन किसी भी Android App में इसे Compulsory रूप से Use करना जरूरी नहीं होता और बिना Fragment का प्रयोग किए हुए भी हम हमारा पूरा Android App Create कर सकते हैं। साथ ही जब हम किसी Fragment को Use करते हैं, तब ये भी जरूरी नहीं होता कि हमारे Fragment का UI Part भी Create किया जाए। यदि हम चाहें, तो Fragment को बिना UI के एक Invisible Background Worker के रूप में भी Create कर सकते हैं, जो कि हमारी Main Activity के लिए जरूरी किसी Functionality को Invisibly Fulfill करता हो।

Intents

Intents वास्तव में System Messages होते हैं, जो कि Android Platform को Device के अन्दर Running रखते हैं और उस Android Platform पर Run होने वाले विभिन्न Applications को विभिन्न प्रकार के Events के लिए Notify करते हैं।

यानी जिस प्रकार से हमारे Computer System में Operating System विभिन्न प्रकार के Hardware व Software Events को Trigger करके Applications को कोई Specific Task Complete करने के लिए Notify करता है, ठीक उसी तरह से Android Device में किसी भी प्रकार के **Hardware State Change** (*जैसे कि SD Card को Device में Insert या Remove करने*) अथवा **Software State Change** (*जैसे कि SMS के रूप में नए Incoming Data के आने अथवा*

जाने या फिर Application के Menu Option को Touch करने, आदि) के रूप में विभिन्न प्रकार के Events के Trigger होने पर भी Android System, Android App को किसी Specific Task के Complete होने के लिए Notify करता है।

अन्य शब्दों में कहें तो जिन्हें अन्य Operating Systems में **Messages** या **Events** के नाम से जाना जाता है, उन्हें ही Android Platform के अन्तर्गत **Intents** कहते हैं और हम न केवल किसी Intent को Respond कर सकते हैं, बल्कि अपनी जरूरत के अनुसार नया Intent Create भी कर सकते हैं और उन Intents के Response में किसी Specific Activity को Launch भी कर सकते हैं अथवा इस बात को तय कर सकते हैं कि वह Situation कब आएगी, जब कोई Specific Intent Trigger होगा।

उदाहरण के लिए यदि हमें किसी Intent को उस समय Trigger करना हो जबकि वह किसी Specific Location के 200 Meter के दायरे में आए, तो इस प्रकार के Intent को Schedule करने के लिए हमें हमारे स्वयं के Intent Create करने की जरूरत पडती है।

यदि एक Android Developer के रूप में Intent के महत्व को सरल शब्दों में समझने की कोशिश करें, तो Intent हमें किसी Specific Hardware/Software Event Trigger होने के Response में एक Activity से दूसरी Activity पर Move करने की सुविधा देता है।

उदाहरण के लिए मानलो कि हमारे Android App में Main Activity पर एक Button है, जिसे Click करने पर एक नया Alert Box Display होता है। इस स्थिति में एक Button के Click होने पर Response रूप में Display होने वाला Alert Box एक प्रकार का **Intent** होता है।

Intents का प्रयोग करते हुए हम एक Activity के Data को दूसरी Activity बीच Pass करने का काम करते हैं। हम अपनी किसी Specific जरूरत को पूरा करने के लिए एक Intent के माध्यम से ही सम्पूर्ण Current Activity को भी एक Object की तरह दूसरी Activity पर Pass कर सकते हैं। इतना ही नहीं हम चाहें तो Intents का प्रयोग करते हुए किसी नए Android Application को भी Open कर सकते हैं अथवा किसी Service को Launch/Activate कर सकते हैं, जो कि Background में Run होते हुए Current Android Application की किसी Requirement को Invisibly पूरा करने लगता है।

सरल शब्दों में कहें तो Intent ही वह तरीका हैं, जिनके माध्यम से Current Android Application की Current Activity के किसी Trigger होने वाले Event के Response में Current Application अथवा किसी अन्य Android Application की किसी दूसरी Activity को Launch कर सकते हैं। जबकि सामान्यत: किसी Intent को Current Activity के User Interface में स्थित किसी UI Control को Click करने पर Trigger होने वाले Click Event के Response में ही Fire किया जाता है।

Intents का प्रयोग सामान्यत: कोई SMS Send करने, किसी Service को Start करने, किसी Activity को Launch करने, किसी Webpage को Display करने, Contact List को Display करने, कोई Phone Number Dial करने अथवा किसी Phone Call को Receive करने जैसी जरूरतों को पूरा करने के लिए किया जाता है।

Intents को हमेंशा Android App द्वारा ही Initiate किया जाता हो, ऐसा जरूरी नहीं है, बल्कि इन्हें Android Operating System द्वारा भी Initiate किया जाता है ताकि वह हमारे Android Application को कोई Specific Notification दे सके।

उदाहरण के लिए जब आप किसी Android App को Use कर रहे होते हैं, उसी समय कोई SMS आता है तो आपको उस नए आने वाले SMS का जो Notification प्राप्त होता है, वह एक Intent के माध्यम से ही प्राप्त होता है, लेकिन इस Intent को किसी Application द्वारा नहीं बल्कि Android OS द्वारा Initiate किया गया होता है।

Intents, Implicit व Explicit यानी Automatic व Manual दोनों तरीकों से Initiate हो सकते हैं। उदाहरण के लिए जब आप अपने Android Device के Mail App में आए हुए किसी Email में Exist किसी URL को देखने के लिए उसे Tap करते हैं, तो आपके Intention को समझते हुए उस URL से Associated Webpage को दिखाने के लिए आपके Android Device का Operating System स्वयं Intent के माध्यम से Web Browser Application को Launch कर देता है क्योंकि Android OS को पता है कि एक URL को Web Browser द्वारा ही देखा जा सकता है। इस स्थिति में Create होने वाला Intent, Android System द्वारा **Automatically (Implicitly)** Create होता है।

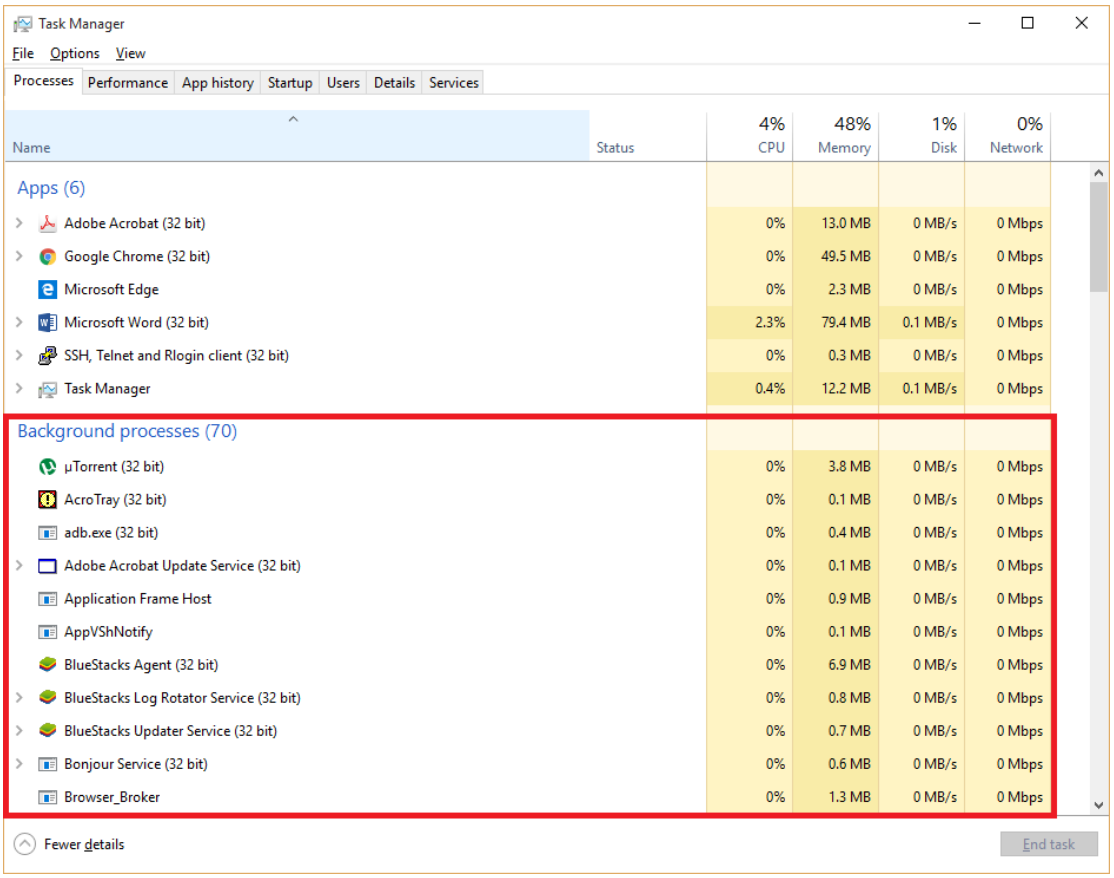
जबकि अपने Android App के किसी Button को Click करने पर Web Browser Application को Launch के लिए, हमें हमारे **Intent** को Manually Create करके Execute करना पडता है और इस Situation में हमारा Intent, Trigger होने वाले **Action (Event)** और उसके Response में Execute होने वाले **Action Handler** या **Event Handler** के साथ Loosely Coupled रहता है।

Activities, Views व Intents मात्र इन तीनों का प्रयोग करते हुए भी हम एक Completely Working Android App Create कर सकते हैं और ढेरों ऐसे Android Apps पहले से उपलब्ध हैं, जिन्हें मात्र इन तीना चीजों का प्रयोग करते हुए ही Develop किया गया है।

Services

Activities का Lifetime काफी कम होता है और वे कभी भी Shut-Down हो सकते हैं अथवा जरूरत के अनुसार उन्हें Close किया जा सकता है। जबकि Services वे Background Programs होते हैं, जो User की जानकारी के बिना Background में Continually Run होते रहते हैं।

Services को हम Background Processes भी कह सकते हैं और निम्न चित्रानुसार Windows OS के **Task Manager** में दिखाई देने वाले Background Processes की तरह मान सकते हैं, जिनका कोई User Interface नहीं होता, बल्कि वे Background में Invisibly Run हो रहे होते हैं और विभिन्न प्रकार की System व Application Related Requirements को Fulfill कर रहे होते हैं:



उदाहरण के लिए जब हम हमारे Android Device पर कोई Music सुन रहे होते हैं, उसी दौरान हम अपने WhatsApp Application पर किसी को Reply भी कर रहे होते हैं अथवा Hangone App या Facebook App पर किसी के साथ Chat भी कर रहे हैं, लेकिन हमारा Music, WhatsApp, Facebook या Hangout App को Use करते समय भी Continuously Run हो रहा होता है, क्योंकि वह हमारी जानकारी के बिना Background में एक **Service** की तरह Run हो रहा होता है।

Services की यही विशेषता है कि उनका कोई User Interface होना जरूरी नहीं होता और बिना User Interface के भी वे Background में Invisibly Run होते हुए हमारी किसी Requirement को पूरा कर रहे होते हैं।

Services, Android OS का एक ऐसा तरीका हैं, जिनके माध्यम से हम किसी Operation को Background में Invisibly Running रख सकते है। इसलिए जब हमें किसी Task को लम्बे समय तक Running रखना होता है, जैसे कि किसी बड़ी File को Upload/Download करना, तब इस प्रकार की जरूरत को पूरा करने के लिए हम Service Create करते हैं।

चूंकि Services के लिए कोई User Interface होना जरूरी नहीं होता, इसलिए Services के Start, Activate, Pause, Resume या End आदि से सम्बंधित Stat Related Information को Notifications के माध्यम से Handle किया जाता है।

Services Create करने के हमेंशा दो तरीके होते हैं:

1. पहले तरीके के अन्तर्गत किसी Service को Activity के माध्यम से Create किया जाता है। इस प्रकार की Services, उस Activity के Stop होने के साथ ही समाप्त हो जाती हैं, जिसके माध्यम से इन्हें Create किया गया होता है।

2. जबकि दूसरे तरीके के अन्तर्गत इन Activities को स्वतंत्र रूप से Create किया गया होता है, जिसकी वजह से ये किसी Application की किसी Activity के साथ Bound नहीं होते, परिणामस्वरूप किसी Application के Start या Stop होने से इनके Run होने पर कोई प्रभाव नहीं पड़ता और एक बार Start होने के बाद ये Background में Invisibly Continually Run होते रहते हैं।

इसी बात को यदि हम दूसरे तरीके से कहें तो एक **Android System**, **Local Service** व **Remote Service** के रूप में दो प्रकार की Services Define कर सकता है। Local Service के रूप में जो Service Create की जाती है, उसे Current Device का केवल वही Applications Use कर सकता है, जिसने Service को Create किया होता है, जबकि Remote Service के रूप में जिस Service को Create किया गया होता है, उसे उस Device के विभिन्न Applications अपनी जरूरत के अनुसार Use कर सकते हैं।

अत: Local Service को हम किसी Activity द्वारा किसी Application के माध्यम से Create की गई Service कह सकते हैं, जबकि Remote Service को हम एक स्वतंत्र रूप से Create की गई Service कह सकते हैं।

Services को हम एक निश्चित समयान्तराल पर किसी Specific Task के Complete होने के लिए बार-बार Execute होने हेतु Schedule भी कर सकते हैं और ये Scheduling करने के लिए हम Cron-Jobs Setup कर सकते हैं, जिनके बारे में हम आगे विस्तार से जानेंगे।

Content Providers

ऐसे Data, जिन्हें Device के Multiple Applications द्वारा Access किया जाना हो, को इन Applications हेतु Access करवाने के लिए जिस तरीके का प्रयोग किया जाता है, उसे **Content Provider** कहते हैं।

यानी Android Development Model हमें **Content Providers** के माध्यम से Data को इस प्रकार से Develop करने की सुविधा Provide करता है, ताकि उन्हें Device के अन्य Applications भी उपयोग में ले सकें। साथ ही हमारे Data पर इन Content Providers का पूरा Control भी होता है, जो इस बात को निश्चित करते हैं कि विभिन्न Applications हमारे Data को किस तरह से Access व Manipulate कर सकेंगे।

यहां Content Providers द्वारा Provide किए जाने वाले Content को हम किसी भी प्रकार का ऐसा Content मान सकते हैं, जिसे हमारे Application में Use किया जा सकता हो। उदाहरण के लिए RSS Feed, Local SQL Database, किसी Text File में Stored Data आदि, सभी **Content** के Example हैं, क्योंकि हम इन्हें हमारे Android Device के किसी भी Application में Android Architecture में Defined **Content Providers** के माध्यम से Use कर सकते हैं।

उदाहरण के लिए हमारे Android Device का **Contacts** App, एक Content Provider का Best Example है क्योंकि हम हमारे किसी भी **SMS** या **Phone Dialer** Android App में इस Contact App के माध्यम से Stored **Contact List** को Access कर सकते हैं।

इसी प्रकार से यदि हम हमारे Android App के Data को Store करने के लिए **SQLite Database** Use करना चाहते हैं, तो SQLite Database को Access व Manipulate करने के लिए Android Architecture के **Content Providers** को Use कर सकते हैं, जो कि हमें एक Developer के रूप में किसी भी प्रकार के Content को Access/Manipulate करने हेतु आसान Encapsulated Access Provide करता है।

इस प्रकार से **Content Providers** वास्तव में हमें हमारे Android Device में Stored Data को Android Applications के माध्यम से Share करने की सुविधा Provide करता है, जो कि Android Architecture द्वारा Define किया गया एक Standard Mechanism है, जिसमें Data Sharing हेतु Underlying Storage, Structure व उनके Implementation को Expose करना जरूरी नहीं होता।

AndroidManifest.xml

प्रत्येक Android App में **AndroidManifest.xml** नाम की एक Configuration File भी होती है, जिसमें Android App से सम्बंधित सभी Activities को Define व Specify किया गया होता है साथ ही Main Activity अथवा Android App की **Launcher Activity** को भी इसी *AndroidManifext.xml* File में Mark किया जाता है। यानी इसी File में इस बात को तय किया जाता है कि हमारे Android App की विभिन्न Activities में से कौन सी Activity Screen पर सबसे पहले Display होगी।

सामान्यत: हर नया Android Developer एक मुख्य गलती ये करता है कि वह जब भी अपने Android App में कोई नई Activity Add करता है, उसे *AndroidManifext.xml* File में Specify करना भूल जाता है और जब तक किसी Activity को इस File में Specify नहीं किया जाता, तब तक हमारा Android App उसे नहीं पहचानता। इसलिए जब भी कभी नया Activity Create करें, सबसे पहले उसे इस Manifest File में Specify करें।

Android App – The Architecture Extensions

उपरोक्तानुसार विभिन्न Core Elements के अलावा कुछ और Elements भी होते हैं, जो Android Application के Development व उपयोग में अपना महत्वपूर्ण Role Play करते हैं, जो कि निम्नानुसार हैं:

Storage

हम हमारे Application से सम्बंधित ऐसे Elements जो कि Change नहीं होते जैसे कि Icons, Help Files आदि को अपने Application की Data Files के रूप में Package कर सकते हैं। साथ ही जिस Device पर हमारा Android Application Run होना होता है, उस Device पर Database या Files के लिए कुछ Extra Space भी Reserved कर सकते हैं। इस Reserved Space पर उस Database या File को Store किया जाता है, जिसमें वे Data होते हैं, जिन्हें User स्वयं हमारे Android Application को Use करते समय उसमें Input करता है अथवा यदि User SD Card के रूप में Extra Storage Use करता है, तो उस Extra Storage में Application से सम्बंधित Data Files को जरूरत के अनुसार Read व Write किया जा सकता है।

Network

Android Devices सामान्यत: Internet Ready होते हैं। यानी इन Devices पर Use होने वाले ज्यादातर Applications किसी न किसी तरह से Internet से Connected रहते हुए ही Operate व Update होते हैं। इसलिए हम हमारे Android Application में जरूरत के अनुसार **Java Sockets** का प्रयोग करते हुए अथवा **WebKit-Based** Web Browser Widget को Embed करते हुए अपने Application में Internet Feature को Use कर सकते हैं।

Multimedia

Android Devices सामान्यत: Multimedia Ready भी होते हैं। यानी ये Devices **Audio/Video** को Record व Playback करने की क्षमता से युक्त होते हैं। हालांकि Different Devices में इन Multimedia Devices की Capabilities व Qualities में अन्तर हो सकता है, लेकिन हम हमारे Application द्वारा इस बात का पता लगाने के लिए Device की Query कर सकते हैं कि उसमें Audio, Video, Camera आदि से सम्बंधित क्या-क्या Capabilities Exist हैं।

Location Services

Android Devices सामान्यत: Location Providers जैसे कि **GPS** व **Cell Triangulation** जैसी Services को Access करने में सक्षम होते हैं, जिसकी वजह से ये Device इस बात को Identify कर सकते हैं कि वे पूरी दुनियां में धरती के किस हिस्से पर हैं और इस बात की जानकारी को हम हमारे Application में उपयोग में ले सकते हैं।

साथ ही हम Location Data का उपयोग अन्य तरीकों से करते हुए Location का Map भी Display कर सकते हैं, जिसकी वजह से Devices के Movements को Track करते हुए उस स्थिति में Device की Location का पता लगा सकते हैं, जबकि वह चोरी हो गया हो अथवा खो गया हो।

Phone Services

चूंकि Android Devices मूलत: Phone ही होते हैं, इसलिए हमारा Application इन Devices पर New Phone Calls को Initiate कर सकता है, SMS Messages Send/Receive कर सकता है और वो सबकुछ कर सकता है, जो एक Modern Telephone के साथ किया जा सकता है।

हालांकि उपरोक्तानुसार Specified Android App Architecture को ठीक से समझ कर हम आसानी से एक Basic Android App Create कर सकते हैं, लेकिन एक High Quality के Android Application को Develop करने के लिए हमें सम्पूर्ण Android Architecture को ठीक से समझना बहुत जरूरी है जिसमें बहुत सारे Elements होते हैं और हर Element किसी विशिष्ट प्रकार की जरूरत को पूरा करता है।

लेकिन सम्पूर्ण Android Architecture अपने आप में काफी Technical व Theoretical है, इसलिए इस Architecture के विभिन्न Elements को विस्तार से समझने से पहले हम एक Basic Android App Create करना सीखेंगे, ताकि जब हम Android Architecture के विभिन्न Elements को Discuss करें, तो उन्हें बेहतर तरीके से समझने के लिए छोटे-छोटे Android Apps Create करते हुए Practical Approach Follow कर सकें क्योंकि Practical Approach को Follow करने से ही App Development का Confidence आता है।

यहां एक और बात समझना जरूरी है कि Android App वास्तव में Android Framework पर आधारित है जो कि Windows की तरह ही एक सम्पूर्ण Operating System है और जिस तरह से Windows Operating System के लिए Application Develop करने हेतु हमें Visual Studio जैसे

किसी IDE को Use करना होता है, ताकि हमारा Development Speed Fast हो सके उसी तरह से Android App Develop करने का काम भी हम किसी IDE का प्रयोग करते हुए ही करेंगे।

क्यों?

हालांकि हम चाहें तो बिना कोई IDE Use किए हुए भी मात्र Notepad जैसे Text Editor का प्रयोग करते हुए Android Application Develop कर सकते हैं, लेकिन जब हम इस प्रकार के Manual Approach का प्रयोग करते हुए अपना Application Create करते हैं, तब अपने Android App को **Compile, Run, Build, Debug, Test** व **Deploy** करने से सम्बंधित सारे काम हमें Command Line Tools के माध्यम से Manually करने पड़ते हैं, जिससे हमारे App के Develop होने की Speed काफी कम हो जाती है।

जबकि यदि हम अपने Android App को किसी IDE का प्रयोग करते हुए Develop करते हैं, तो उस स्थिति में अपने App को Compile, Run, Build, Debug, Test व Deploy करने से सम्बंधित विभिन्न कामों को ये IDE स्वयं अपने स्तर पर Automatically कर देते हैं। परिणामस्वरूप हम हमारा पूरा ध्यान अपने App को Develop करने पर लगा सकते हैं, जिससे हमारे App Development की Speed काफी तेज हो जाती है।

एक और महत्वपूर्ण बात ये है कि हम चाहे जिस किसी भी Operating System के लिए Application Develop करें, हर Operating System का काम करने का अपना अलग तरीका होता है जिसके माध्यम से वह Underlying Hardware के साथ Interact करता है और Device के विभिन्न Hardware को Access व Manipulate करने के लिए जिन Functions की जरूरत होती है, सामान्यत: उन्हें Assembly या C/C++ जैसी Hardware Level की Programming Languages में ही Define किया गया होता है।

लेकिन क्योंकि इन Hardware Level की Programming Languages में Applications Develop करना काफी जटिल काम होता है, इसलिए Operating System Provide करने वाली Companies अपने Operating System के लिए Applications Develop करने हेतु एक High Level API भी Provide करते हैं और APIs के साथ इन APIs को Use करते हुए Develop किए जाने वाले Application Software को आसानी से Develop, Compile, Debug, Test, Deploy आदि करने हेतु कुछ Tools भी Provide करते हैं, जिन्हें सामान्यत: **SDK** (*Software Development Kit*) के नाम से जाना जाता है।

Windows Operating System के लिए Application Software Develop करने हेतु Microsoft Company जो API Provide करता है, उसे Win32 API के नाम से जाना जाता है। Java Programming Language का प्रयोग करते हुए Application Software Develop करने हेतु Oracle Company द्वारा हमें APIs के साथ Tools का जो Set प्राप्त होता है, उसे JDK के नाम से जाना जाता है। ठीक इसी तरह से Android Platform के लिए Apps Develop करने हेतु Google भी हमें APIs के साथ कुछ Tools Provide करता है, जिन्हें **Android SDK** के नाम से जाना जाता है।

Operating System के लिए Applications को आसानी से Develop किया जा सके, इसलिए Microsoft हमें Visual Studio के रूप में IDE Provide करता है। इसी तरह से Oracle, Java आधारित Application Development के लिए NetBeans नाम का IDE Provide करता है। ठीक इसी तरह से Android आधारित Apps Develop करने के लिए Google हमें Android Studio नाम का IDE Provide करता है।

लेकिन सामान्यत: Android Apps Development का काम भी Windows/Linux Operating System पर आधारित Desktop Computers पर किया जाता है, जो कि Android APIs से पूरी तरह से अनजान होते हैं। इसलिए Android Apps Develop करने हेतु हमें इन Desktop Computers पर पूरा Android Application Development Environment Setup करना पडता है और इस Environment को Setup करने से सम्बंधित सभी जरूरी Tools, Google द्वारा Provide किया जाता है।

Android Applications को Windows/Linux आधारित Computer System पर Develop करने हेतु पूरा Environment Setup करने के बाद Windows/Linux Computer System पर न केवल Android API Setup होता है, बल्कि एक Android Emulator भी Setup होता है, जिस पर हम हमारे Develop किए जा रहे Android Application को Live Simulate करते हुए उसकी Working को Test व Debug कर सकते हैं और सम्पूर्ण Android App Development Environment को Setup व Configure करने से सम्बंधित विस्तृत वर्णन Next Chapter में किया गया है।

DEVELOPMENT ENVIRONMENT

Android – Development Environment

एक Android Application व Device के Basic Architecture के बारे में जानने के बाद जब हम आगे बढते हैं, तो सबसे पहले हमें Android Application को Develop करने से सम्बंधित Environment को Setup करना होता है, क्योंकि बिना ये Environment Setup किए हुए, Android Application को सुविधाजनक तरीके से Develop नहीं किया जा सकता। तो चलिए, Android Application Development से सम्बंधित सम्पूर्ण Setup को Step-by-Step विस्तार से समझने की कोशिश करते हैं।

Setup Java SDK

जैसाकि हमने पिछले Chapter में भी कहा कि **Android Applications** वास्तव में Java Applications ही होते हैं, जिन्हें Java Programming Language आधारित Source Codes के माध्यम से Define किया जाता है और फिर Android Platform आधारित Compiler के माध्यम से Dalvik Bytecode के रूप में Compile करके एक **.apk** File की तरह Android Package Create कर लिया जाता है, जो कि Android Platform पर Run होने वाला **Executable** होता है।

इसलिए Android Application Develop करते समय सबसे पहले Step के रूप में हमें हमारे Computer System पर Java SDK को Install व Setup करना होता है, ताकि हम हमारे Android Application के लिए Java Classes Create कर सकें, जिन्हें Android Platform द्वारा **.apk Package** File के रूप में Dalvik Bytecode में Compile करने के लिए Use किया जा सके।

चूंकि Java SDK, Oracle Company द्वारा Free Provide किया जाता है। इसलिए सबसे पहले Oracle की Website से **Java SE Development Kit (JDK)** के Latest Version को **Download** करके अपने Computer System पर Install करना होता है।

यहां ये बात ध्यान रखना जरूरी है कि हमें पूरा Java SDK Install करना होता है केवल JRE नहीं, क्योंकि Android Application, **Java Compiler (javac)** व **Core Classes** का प्रयोग करते हुए ही Android Applications को Dalvik Bytecode में Compile करता है।

अपने Computer पर Installed **Operating System** (*Windows, Linux, MacOS, etc…*) व Operating System के **Architecture** (*32-Bit, 64-Bit, etc…*) के आधार पर Appropriate Java Version को **Download** करने के बाद उसे Install करना होता है, जिसे Install करने से सम्बंधित Detailed Information को इसी Oracle की Website से प्राप्त किया जा सकता है।

चूंकि, Android Application पूरी तरह से Java Source Codes पर आधारित होते हैं, इसलिए Android Application तभी Develop किया जा सकता है, जबकि आपको Java Programming का अच्छा ज्ञान हो। इसलिए यदि आपको Java Programming का ठीक-ठीक ज्ञान न हो, तो बेहतर यही होगा कि आप पहले Java की Basic Programming सीखें जिसमें हमारी अन्य पुस्तक "**Java in Hindi**" आपकी काफी मदद कर सकती है।

हालांकि आपको कम से कम निम्न Java Concepts का Basic ज्ञान होना जरूरी होता है, तभी आप एक Android Application Develop कर सकते हैं:

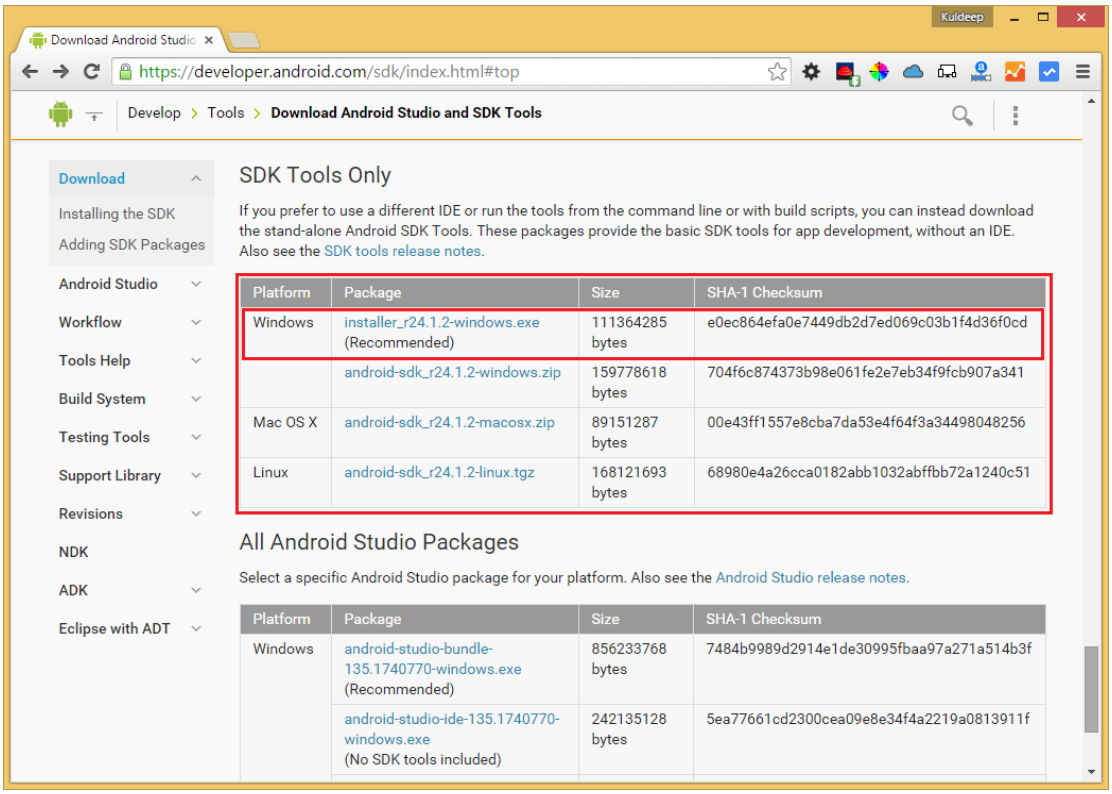
- *Language fundamentals (flow control, etc.)*
- *Classes and objects*
- *Methods and data members*
- *Public, private, and protected*
- *Static and instance scope*
- *Exceptions*
- *Threads and concurrency control*
- *Collections*
- *Generics*
- *File I/O*
- *Reflection*
- *Interfaces*

Downloading and Installing Android SDK Tools

Java SDK को Install करने के बाद हमें Android SDK को Install करना होता है, जो कि हमें हमारे Android Application को अपने Local Computer पर Develop, Design, Test व Debug करने से सम्बंधित सभी जरूरी Functionalities Provide करता है।

Android SDK का Installation वास्तव में दो हिस्सों में Divide होता है, जहां पहले हिस्से के अन्तर्गत हमें Android Platform के **Base Tools** को Install करना होता है और दूसरे हिस्से के अन्तर्गत Android Platform से सम्बंधित अन्य Add-On को।

सबसे पहले हमें http://developer.android.com/sdk/index.html से निम्न चित्रानुसार "SDK Tools Only" Section से अपने Computer System पर Installed Operating System के अनुसार Appropriate "**SDK Tools**" को Download करना होता है:

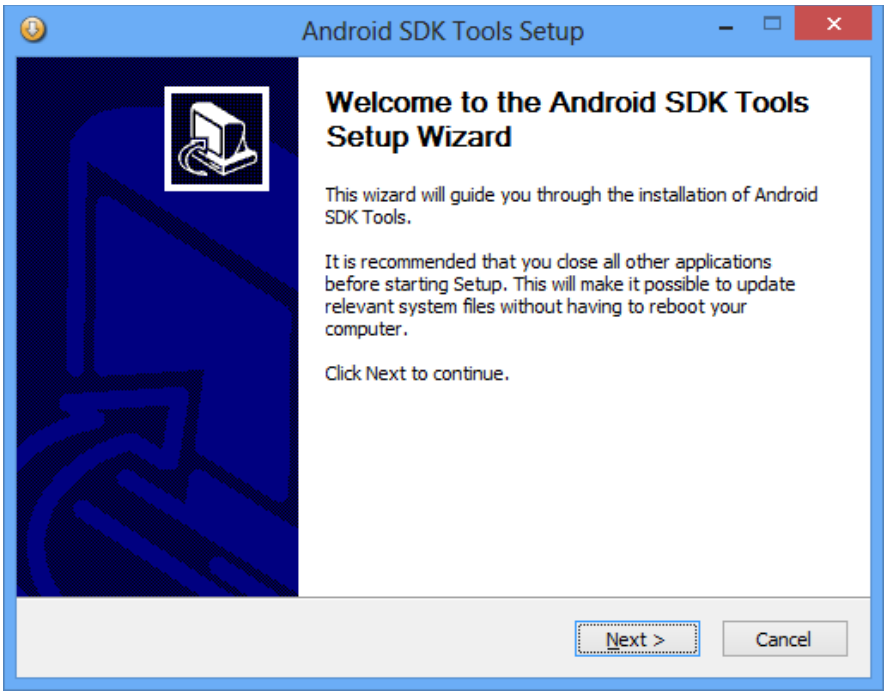


चूंकि हमारे Computer System पर Windows Operating System Installed है, इसलिए हमने उपरोक्त चित्रानुसार Windows Operating System के लिए Recommend Executable File को Download किया है। हालांकि हम इसी Webpage पर दिखाई देने वाले "**Android Studio Bundle Package**" को भी Download कर सकते हैं, जो कि वास्तव में Google Provided IDE के साथ सभी जरूरी SDK Tools युक्त पूर्ण Package होता है और इस IDE का प्रयोग करते हुए अपना Android Development कर सकते हैं।

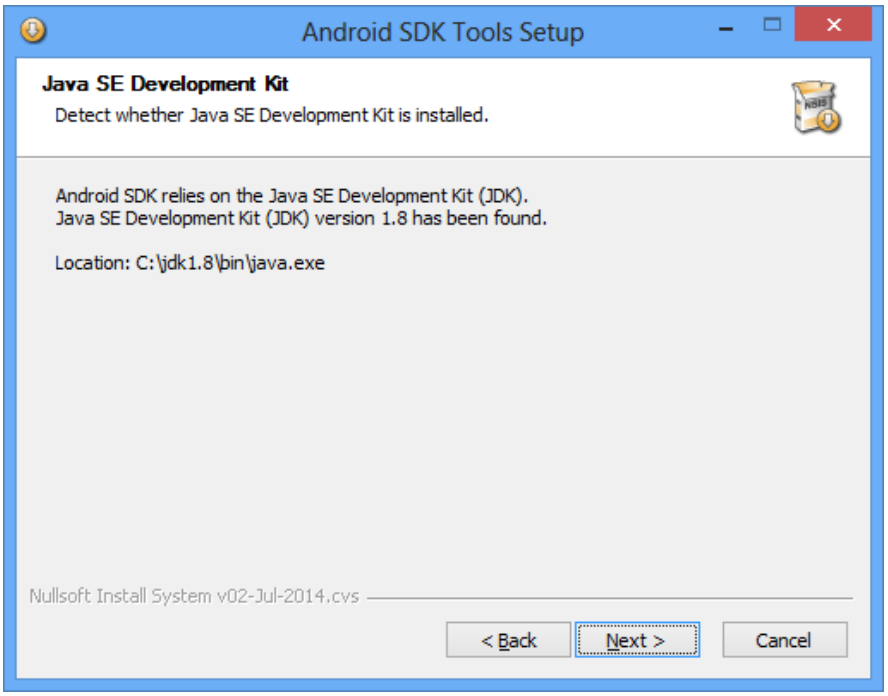
लेकिन **SDK Tools** को Manually Install and Setup करने के बारे में जानना हमारे लिए काफी उपयोगी साबित होता है, क्योंकि इस Manual तरीके से Android SDK को Setup करके हम किसी भी IDE या Text Editor को Android Development के लिए स्वतंत्र रूप से Use करने के बारे में बेहतर तरीके से समझ लेते हैं व अपने Application को Different IDEs व Text Editors का प्रयोग करते हुए Develop, Compile, Test व Debug करने की क्षमता प्राप्त करते हैं, जो कि हमें किसी Specific IDE या Editor से Bound नहीं करता।

साथ ही Manual Configuration व Installation सीख लेने के बाद हम बडी ही आसानी से किसी भी IDE को आसानी से उपयोग में लेते हुए Android Development कर सकते हैं। लेकिन यदि हम सीधे ही किसी IDE के माध्यम से Development शुरू करते हैं, तो कई बार हमें IDE Configuration व Setup के दौरान बहुत सारी परेशानियों का सामना करना पडता है, क्योंकि हमें पता ही नहीं होता कि IDE Internally किस प्रकार से Android SDK, Java SDK, Text Editor आदि को Combined रूप में रूप करते हुए Application Development के लिए उपयुक्त Environment Create करता है।

चूंकि हम **Android SDK** को Windows Operating System के लिए Manually Install करना चाहते हैं, इसलिए जैसे ही हम Windows के लिए Download किए गए Android Installer को Execute करते हैं, हमारे सामने निम्न चित्रानुसार Dialog Box Display होता है:

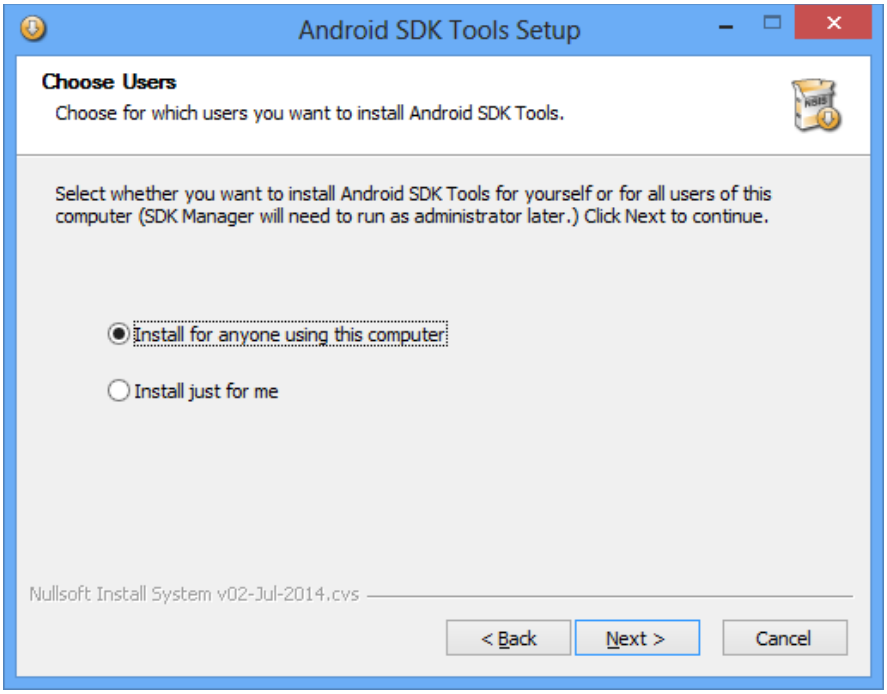


इस Dialog Box पर दिखाई देने वाले "**Next >**" Button को Click करना होता है, जिसके परिणामस्वरूप निम्नानुसार अगला Wizard Screen Display होता है, जो कि हमारे Computer System पर Installed Java Interpreter के Path को Retrieve करता है:

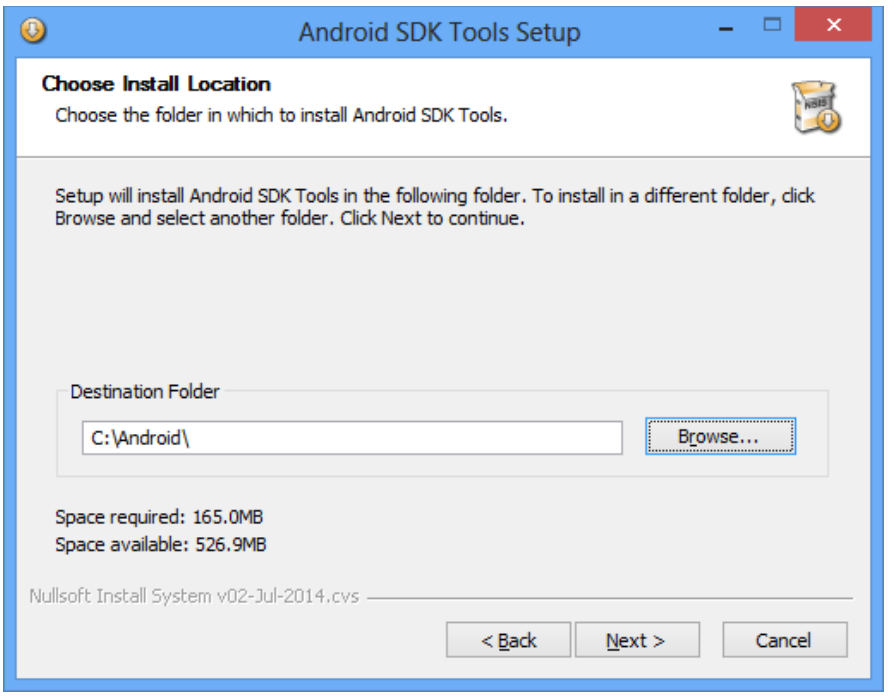


यदि हमने पहले से ही Java SDK को Install न किया हो, तो **Android SDK Tools Setup Wizard** का इसी Screen पर अन्त हो जाता है, क्योंकि Android SDK Tools का Installation करने से पहले Computer System पर Java SDK का Install होना जरूरी होता है।

अब हमें इस Wizard Screen पर दिखाई देने वाले "**Next >**" Button को फिर से Click करना होता है, जिसके परिणामस्वरूप निम्नानुसार अगला Wizard Screen Display होता है:

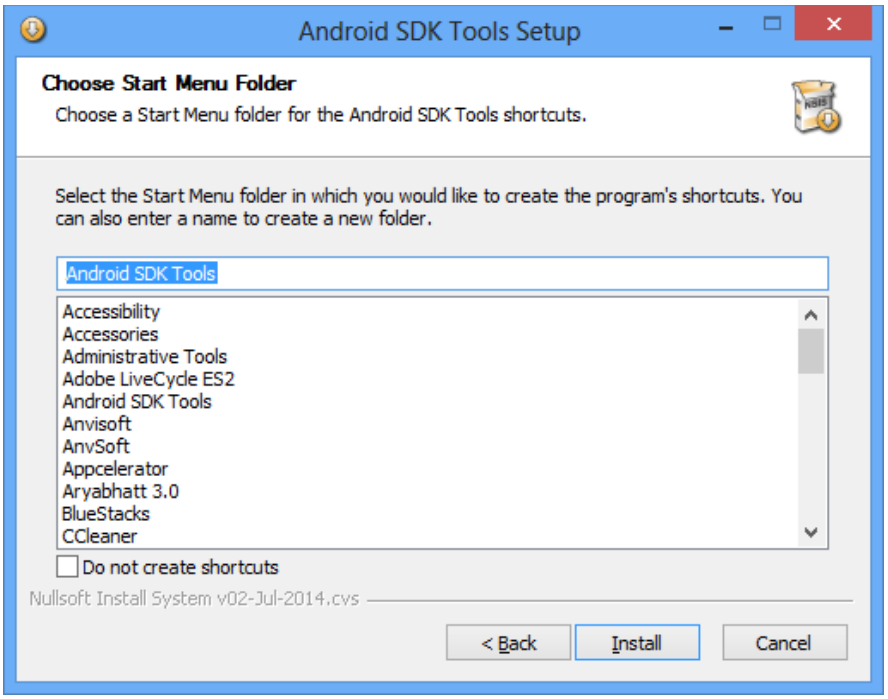


इस Screen पर Appropriate Radio Button को Select करके फिर से "**Next >**" Button को Click करना होता है, जिसके परिणामस्वरूप निम्नानुसार अगला Wizard Screen Display होता है:

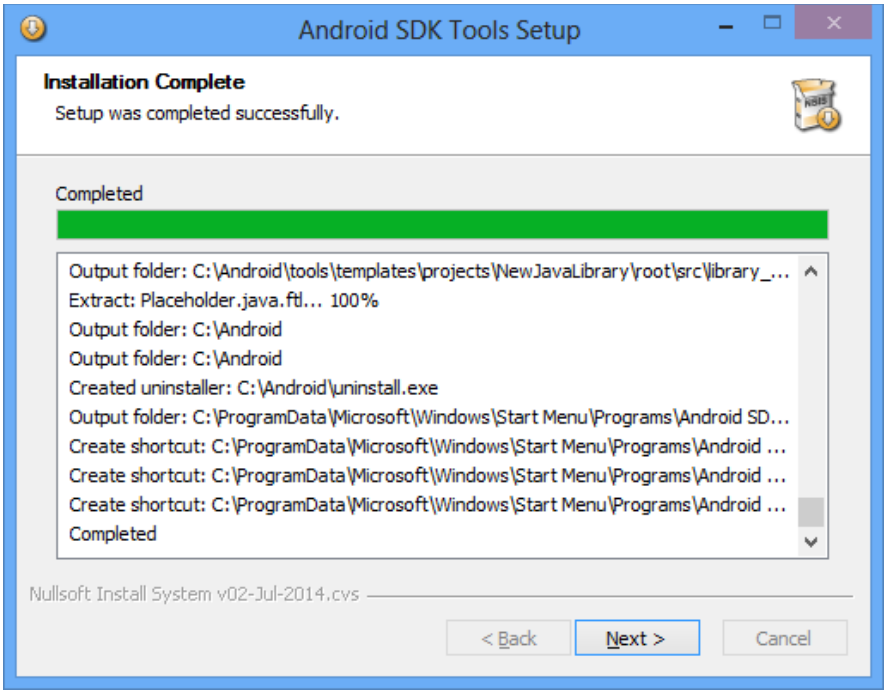


इस Screen पर हमें "**Browse…**" Button को Click करके वह "Destination Folder" Specify करना होता है, जहां हम हमारे Android SDK Tools को Install करना चाहते हैं।

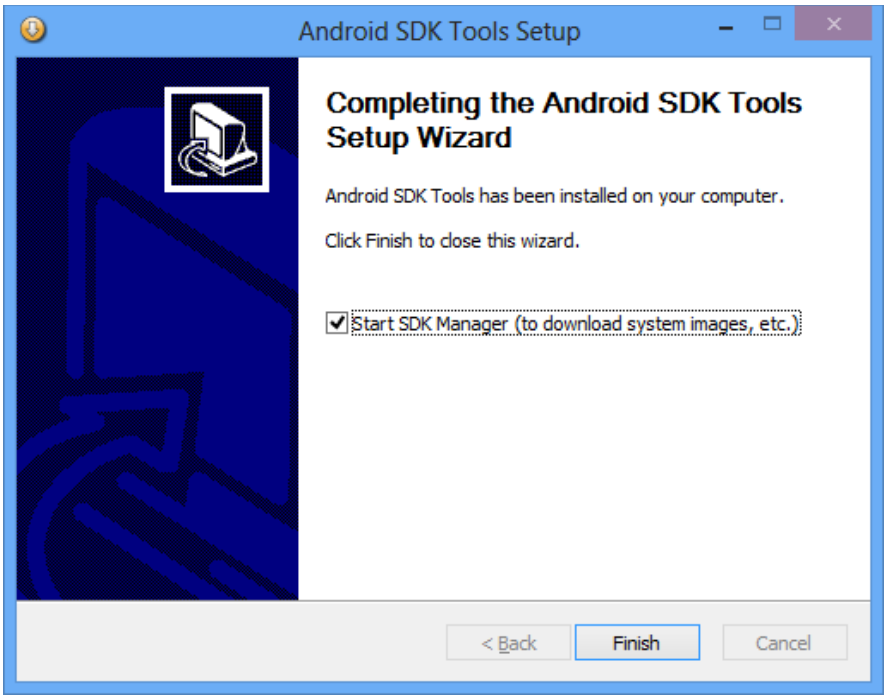
Destination Folder को Specify करने के बाद हमें फिर से "**Next >**" Button को Click करना होता है, जिसके परिणामस्वरूप निम्नानुसार अगला Wizard Screen Display होता है:



जहां हमें Finally "**Install**" Button को Click करना होता है, जिसके परिणामस्वरूप निम्न चित्रानुसार **Android SDK Tools** का Installation होने लगता है:



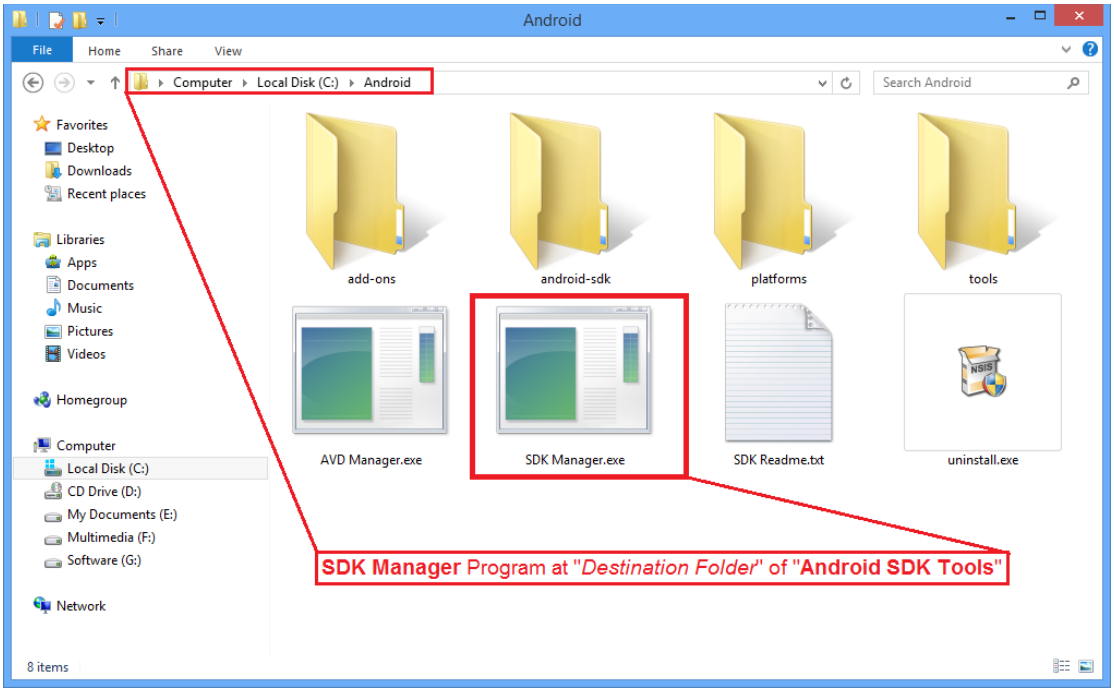
Android SDK Tools का Installation पूरी तरह से Complete होने के बाद हमें फिर से "**Next >**" Button को Click करना होता है, जिसके परिणामस्वरूप निम्नानुसार अगला Wizard Screen Display होता है:



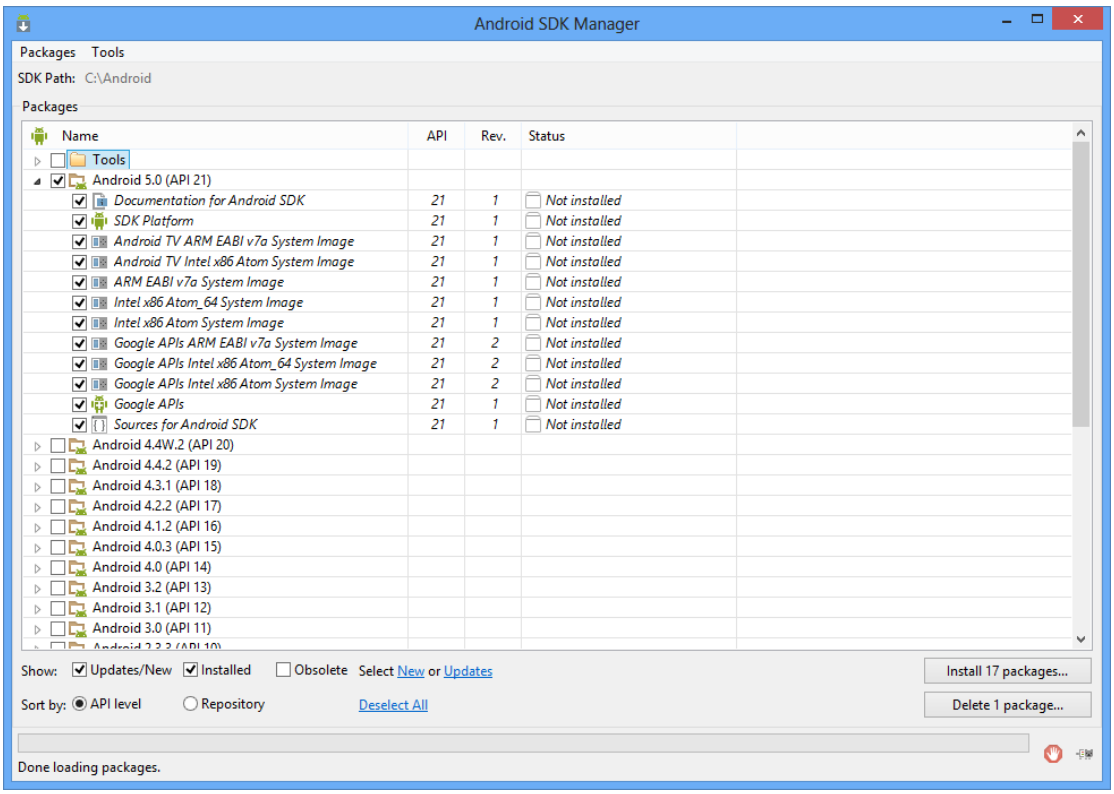
इस Screen से हमारे **Android SDK Installation** का दूसरा Part शुरू होता है। यानी इस Wizard Screen पर दिखाई देने वाले Checkbox "*Start SDK Manager (to download system images, etc.)*" को Checked रखते हुए "**Finish**" Button पर Click करने पर Android SDK Tools का

SDK Manager Program Execute होता है, जो कि हमारे Android Development के लिए जरूरी अन्य Add-Ons, Packages, Images आदि को Download करने के लिए उपयोगी होता है।

हालांकि यदि हम चाहें तो इस Screen पर दिखाई देने वाले Checkbox को Uncheck करते हुए भी "**Finish**" Button पर Click करके **Android SDK Tools** Setup Wizard का अन्त कर सकते हैं। क्योंकि **SDK Manager** को हम निम्न चित्रानुसार Android Installation के Destination Folder से भी Execute कर सकते हैं:



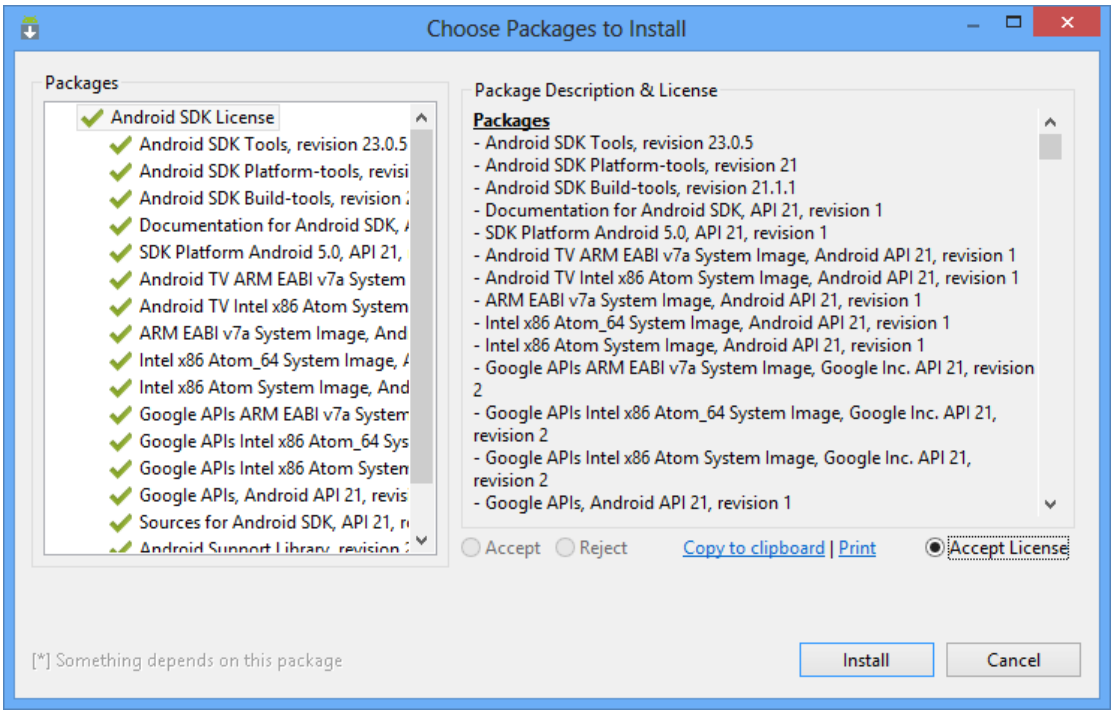
जैसे ही **Android SDK Manager** Program Execute होता है, निम्न चित्रानुसार Android Development के लिए जरूरी सभी Core Packages को Automatically Select कर लेता है:



यानी ये Manager उन Minimum Packages को Automatically Select कर लेता है, जिन्हें Install किए बिना Simplest Android Application भी Develop नहीं किया जा सकता।

यदि हम उपरोक्त चित्र में ही देखें, तो कुल 17 Packages Automatically Select हुए हैं, जिन्हें Compulsory रूप से Download करके Install करना जरूरी है और इन 17 Packages के Install हुए बिना Simple से Simple Android Application भी Develop नहीं किया जा सकता।

इसलिए इन Minimum Android Packages को Install करने के लिए हमें इस Android SDK Manager Window पर दिखाई देने वाले "**Install 17 packages...**" Button को Click करना होता है, जिसके परिणामस्वरूप निम्न चित्रानुसार अगला Screen Display होता है:



इस **Dialog Box** पर हमें "**Accept License**" Radio Button को Select करके "**Install**" Button को Click करते हुए अपने सभी Selected Packages को Download करके Install कर सकते हैं।

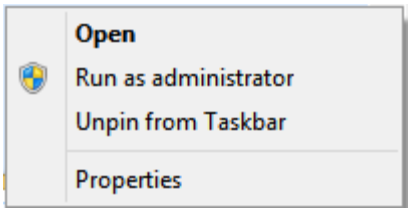
हमें हमेंशा **Latest Android SDK Tools, Android SDK Platform-Tools** व **Android SDK Build-Tools** को ही Download करना चाहिए, हालांकि हम किसी भी Version के API को Download कर सकते हैं, जिससे Supported Android Platform के लिए हम हमारा Application Develop करना चाहते हैं।

उदाहरण के लिए यदि हम **API 8** Download करते हैं, तो हम **Android 2.2** तक के Device के लिए अपना Application Develop कर सकते हैं। जबकि यदि हम **API 15** Download करते हैं, तो हमारा Android Application Run होने के लिए कम से कम वह Device Use करना पडेगा, जिस पर **Android 4.0** Installed हो।

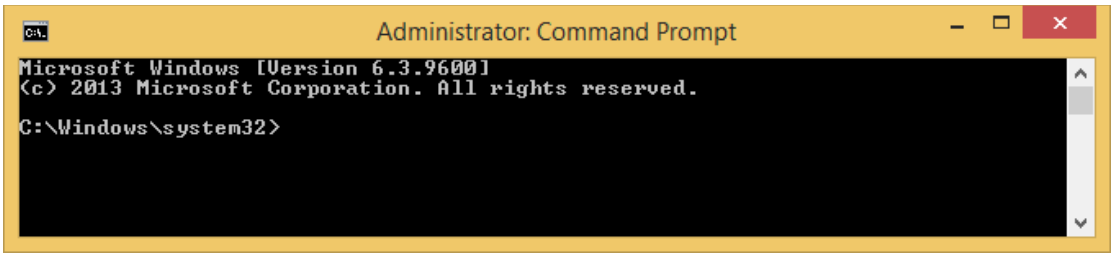
Path Configuration for Command Prompt

एक बार **Android SDK Tools** Install करने के बाद हमें "**Android SDK Tools**" Directory में Exist "**tools**", "**platform**" व "**platform-tools**" Folders को भी Path Variable में Add होना जरूरी होता है, ताकि Operating System इन Folders में Exist Tools व Commands को किसी भी Location से Directly Access कर सके।

इन तीनों Folders को **Path** Environment Variable में Set करने के लिए सबसे पहले हमें Command Window को "**Administrator**" Mode में Open करना होता है। यानी Command Window के Icon पर Right Click करना होता है जिसके परिणामस्वरूप निम्न चित्रानुसार एक Popup Menu Display होता है:



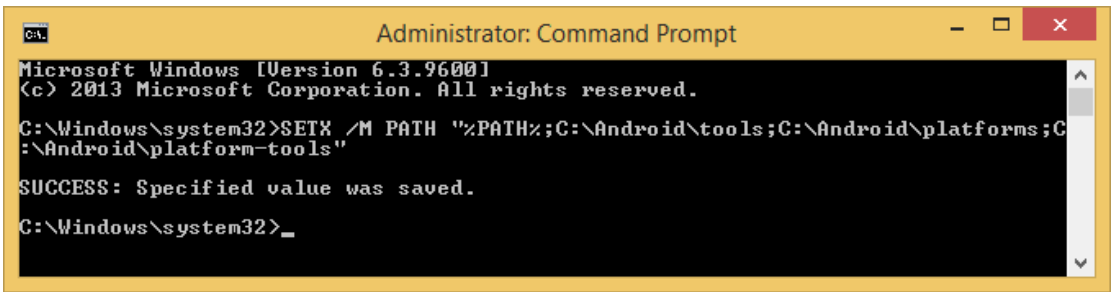
और इस Popup Menu में दिखाई देने वाले "*Run as administer*" Option को Click करना होता, जिसके परिणामस्वरूप निम्न चित्रानुसार एक Command Prompt Window Open होता है:



इस Command Prompt में हमें निम्नानुसार **SETX** Command Execute करने होते हैं:

```
SETX /M PATH "%PATH%;C:\Android\tools;C:\Android\platforms;C:\Android\platform-tools"
```

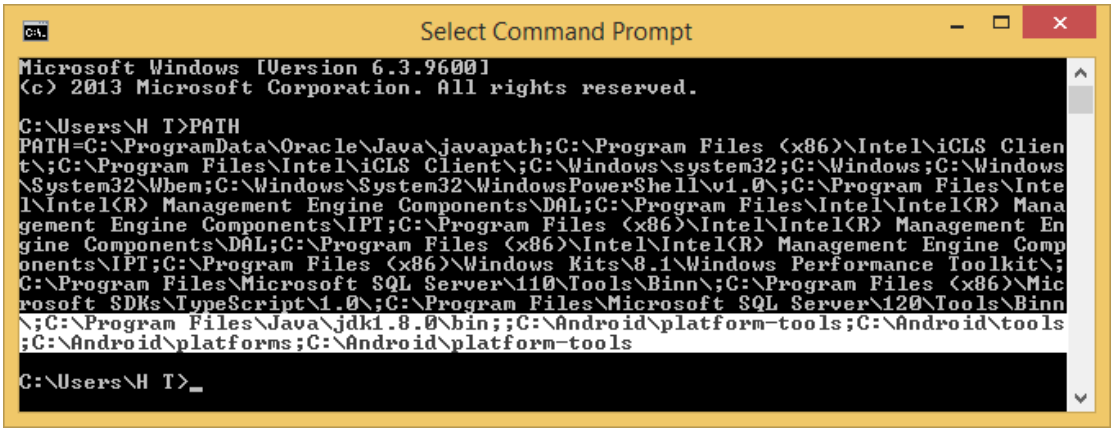
जैसे:



जैसे ही हम इन Commands को Execute करते हैं, हमारे Computer System के "**Path**" नाम के Global Environment Variable में "**tools**", "**platforms**" व "**platform-tools**" नाम के तीनों Folders Set हो जाते हैं, जिनमें Exist किसी भी Command Line Tools को अब किसी भी Drive से और किसी भी Directory से Directly Execute किया जा सकता है।

इस बात का पता लगाने के लिए कि ये तीनों Folders Global Path Environment Variable में Set हुए या नहीं, हमें एक नया Command Prompt Window Open करना होता है क्योंकि Current Command Prompt Window में किया गया Change हमें इसी Window में Reflect नहीं होता।

इसलिए जब हम नया Command Prompt Window Open करके उसमें Current Path को Echo करते हैं, तो हमें प्राप्त होने वाला Output निम्नानुसार होता है:



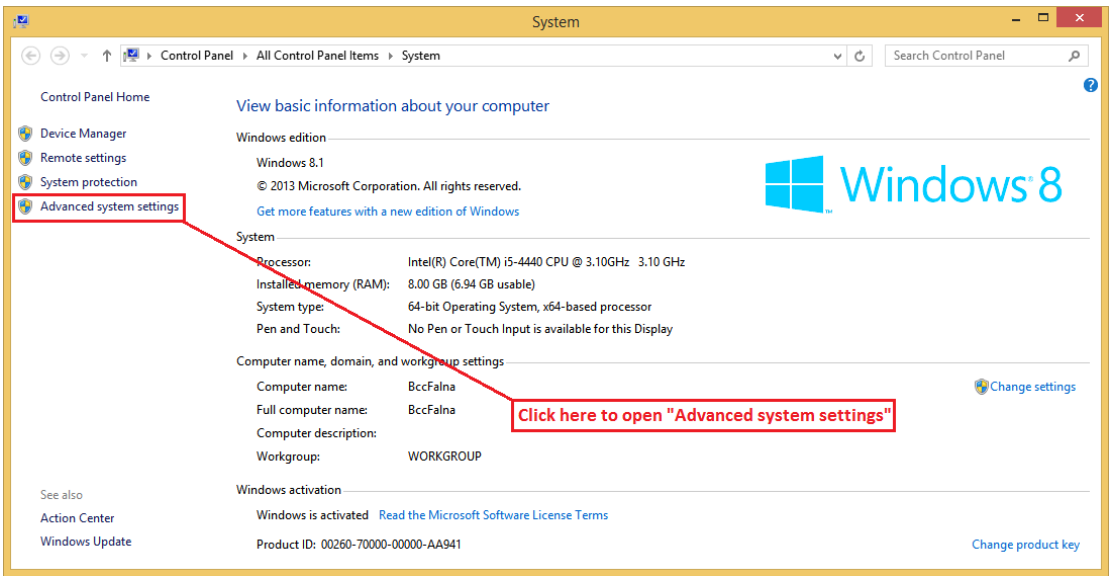
हम देख सकते हैं कि उपरोक्त चित्र की अन्तिम दो Highlighted Lines में वही Path दिखाई दे रहा है, जिसे हमने पिछले Command Window द्वारा SETX Command के माध्यम से Add किया था।

हमारे Android से सम्बंधित Tools, Path Variable में Set हुए या नहीं इस बात का पता लगाने का एक तरीका और है जिसके अन्तर्गत यदि हम Command Prompt Open करें और उसमें "**adb devices**" Command Enter करें, तो हमें निम्नानुसार Output प्राप्त होता है:

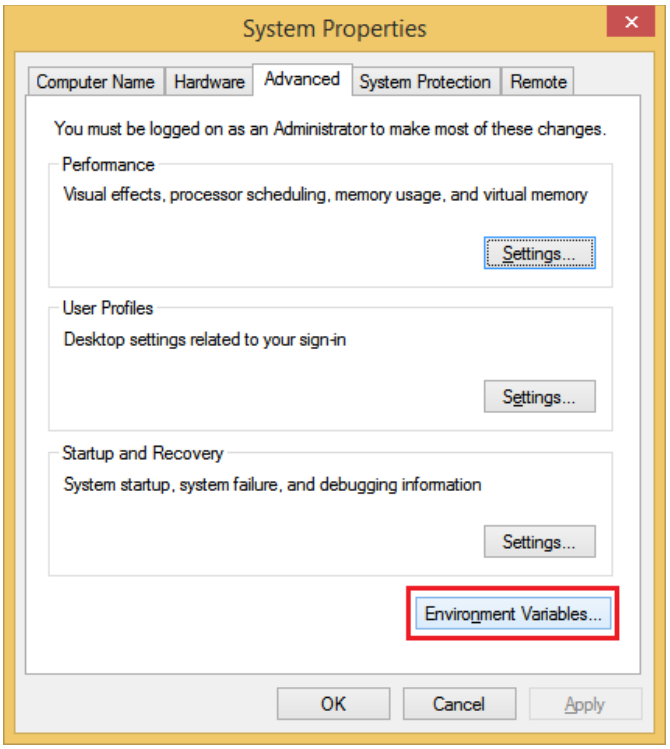
जो इसी बात का Indication है कि Currently हमारे Computer System से कोई भी Actual Android Device या Android Emulator Attached है, जिस पर Develop किए जाने वाले Android Application को Test किया जा सके।

यानी हमारा Android-Tools पूरी तरह से Path Environment Variable में Set है, इसी वजह से हम Android SDK के "**tools**" Directory में Exist **adb.exe** File को उपरोक्त चित्र में दर्शाए अनुसार Connected Devices की List प्राप्त करने हेतु Execute कर पा रहे हैं।

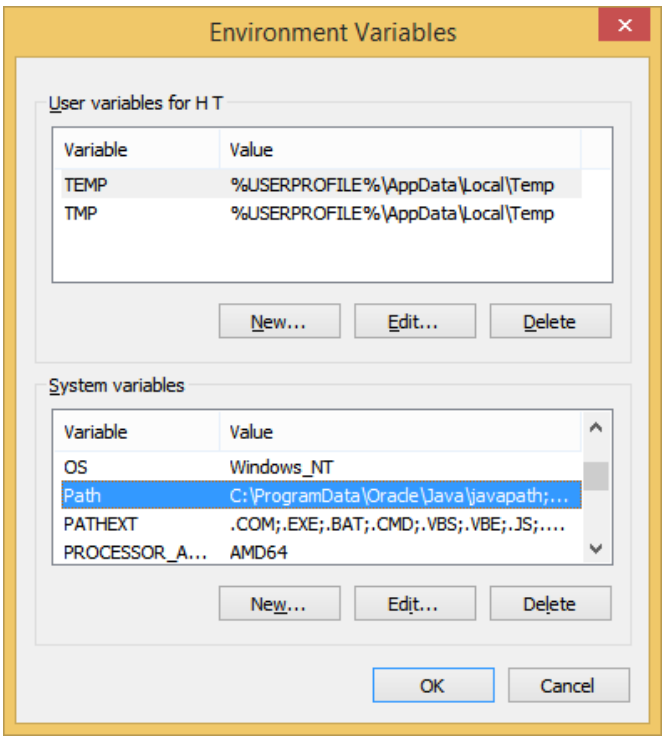
हालांकि यदि हम चाहें तो अपने "**Path**" नाम के Environment Variable को GUI तरीके से भी Set कर सकते हैं, जिसके अन्तर्गत हमें "**My Computer**" पर Right Click करना होता है और Display होने वाले Popup Menu से "**Properties**" Option को Select करना होता है। परिणामस्वरूप निम्न चित्रानुसार System Window Display होता है:



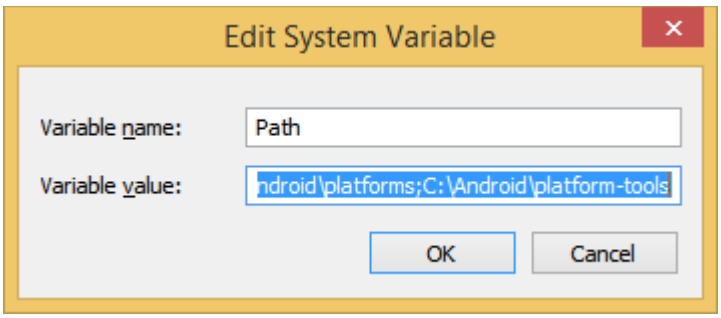
इस **System** Window में उपरोक्त चित्र में दर्शाए अनुसार "**Advanced system settings**" Option को Click करने पर निम्न चित्रानुसार एक "**System Properties**" Dialog Box Display होता है:



इस "**System Properties**" Dialog Box में दिखाई देने वाले "**Environment Variables...**" Button को Click करते ही हमारे सामने निम्न चित्रानुसार "**Environment Variables...**" नाम का Dialog Box Display होता है:



इस Dialog Box के "**System variables**" List Box में उपरोक्त चित्र में दर्शाए अनुसार "**Path**" Option को Select करके "**Edit…**" Button को Click करते ही हमारे सामने निम्न चित्रानुसार "**Edit System Variable**" नाम का Dialog Box Display होता है:



बस इसी Dialog Box के "**Variable value:**" Textbox में दिखाई देने वाली String के अन्त में एक Semicolon लगाकर हमें हमारे उस Folder का Path Specify करना होता है जिसे हम Configure करना चाहते हैं, क्योंकि प्रत्येक नए Path को हमेंशा एक Semicolon से ही Separated रखा जाता है।

Creating New Android Project with Command Line Tools

जब एक बार SDK Tools को Install कर लिया जाता है तथा उसके "**tools**", "**platforms**" व "**platform-tools**" Directories को उपरोक्तानुसार **SETX** Command का प्रयोग करते हुए Path नाम के Environment Variable में Set कर लिया जाता है, उसके बाद हम Android SDK के विभिन्न Android Tools का प्रयोग करते हुए अपना नया Android Project, Command Prompt के माध्यम से भी Create कर सकते हैं।

How to Get Complete PDF EBook

आप **Online Order** करके **Online** या **Offline** Payment करते हुए इस Complete EBook को तुरन्त Download कर सकते हैं।

Order करने और पुस्तक को Online/Offline Payment करते हुए खरीदने की पूरी प्रक्रिया की विस्तृत जानकारी प्राप्त करने के लिए आप **BccFalna.com** के निम्न Menu Options को Check Visit कर सकते हैं।

How to Make Order

How to Order?

How to Buy Online

How to Pay Online using PayUMoney

How to Pay Online using Instamojo

How to Pay Online using CCAvenue

How to Buy Offline

How to Pay Offline

Bank A/c Details

जबिक हमारे Old Buyers के **Reviews** भी देख सकते हैं ताकि आप इस बात का निर्णय ले सकें कि हमारे Buyers हमारे PDF EBooks से कितने Satisfied हैं और यदि आप एक से अधिक EBooks खरीदते हैं, तो **Extra Discount** की Details भी Menubar से प्राप्त कर सकते हैं।