Android Application Development

Module I:
Mobile and Information Architecture
Introduction to Mobile: A brief history of Mobile, The Mobile Eco system, Why Mobile? Types of Mobile Applications. **Mobile Information Architecture:** Mobile Design, Mobile 2.0, Mobile Web Development, Small Computing Device Requirements.
Module-II:
Introduction to Android: History of Mobile Software Development, The Open Handset Alliance, Android platform differences.
Android Installation: The Android Platform, Android SDK, Eclipse Installation, Android Installation, Building a Sample Android Application.
Module-III:
Android Application Design Essentials: Anatomy of an Android applications, Android terminologies, Application Context, Activities, Services, Intents, Receiving and Broadcasting Intents.
Android File Settings: Android Manifest File and its common settings, Using Intent Filter, Permissions, Managing Application resources in a hierarchy, working with different types of resources.
Module-IV:
Android User Interface Design: Essentials User Interface Screen elements, Designing User Interfaces with Layouts.
Animation Techniques: Drawing and Working with Animation- Drawing on the screen–Working with Text-Working with Bitmaps-Working with shapes-Working with animation.
Module-V:
Android APIs-I: Using Common Android APIs Using Android Data and Storage APIs, Managing data using SQLite, Sharing Data between Applications with Content Providers.
Android APIs-II : Using Android Networking APIs, Using Android Web APIs, Using Android Telephony APIs, Deploying Android Application to the World.
Text Books:
Lauren Darcey and Shane Conder,**“ Android Wireless Application Development”,** Pearson Education, 2nd ed. (2011).
•James Keogh, **“J2ME: The Complete Reference”,** Tata Mc Graw Hill.

1

MODULE-I

A brief History of Mobile

Alexander Graham Bell invented the first phone and was granted a patent on his new product in March of 1876 launched by the BellSouth phone company

Telephone
❖Greatest inventions of mankind.
❖It revolutionized communications
❖Enabling us to reach across great distances
❖Share thoughts, ideas, and dreams with our fellow man
❖Making the world a much smaller place in the process.

The Traditional Telephone

2

The Modern Mobile

The Modern Mobile phone is a distant **cousin** to the telephone, it is

1. A communication *and information device.*
2. It is nearly always connected to the Internet
3. Send and receive voice and text messages
4. Purchase goods and services without opening your wallet
5. It can locate which street corner you are standing
6. We use to publish information and knowledge
7. Capable of doing nearly everything you can do with a desktop computer

Modern Mobile Phone

The Evolution of Devices

Every story has a **beginning**, and mobile development is no different

Mobile technology has gone through many **Different Evolutions**.

In the industry, refer to these evolutions as **"Generations"** or simply **"G"** which refers to **the maturity and capabilities of the actual cellular networks**

The cellular network is only one element of the overall **Mobile Ecosystem**

Segment the history of mobile into **five distinct eras** of devices

1. The **Brick** Era
2. The **Candy Bar** Era
3. The **Feature Phone** Era
4. The **Smartphone** Era
5. The **Touch** Era

1. The Brick Era

The **first era** I call the Brick Era (**1973–1988).**
A **Corded Receiver** connected to a portable radio the size and weight of a car Battery.
Brick Era phones required **huge batteries** to get the power needed to reach the nearest cellular network site
Brick Era phones proved useful only to those who truly needed **Constant Communication**, such as **Stockbrokers** or those who worked in the field, such as **Salespeople or Real Estate Agents**; because they were so enormous and so expensive

Motorola DynaTAC introduced in 1983.
The Motorola DynaTAC 8000X *was the first mobile phone to receive FCC acceptance, in 1983.*
DynaTAC *was actually an abbreviation of* ***Dynamic Adaptive Total Area Coverage.***
Motorola discontinued the DynaTAC as late as 1994.

5

2. The Candy Bar Era

•The second era, the Candy Bar Era **(1988–1998)**

•Used **Long, Thin, Rectangular** form factor of the majority of mobile devices

•The network shifted **2G technology**, in Finland in 1991.

•Increased usage and decreased the power demands of the device

•Candy bar phones associated with **2G GSM** (Global System for Mobile communications) networks—included **SMS** (Short Message Service) capabilities.

•During the mid-1990s, a mobile device **future blossomed** in Northern Europe.

•Text message is limited to **140 characters**

The First Flip Phone (1996)

The StarTAC, created by Motorola in 1996, was the phone that started the whole revolution of flip phones

6

3. The Feature Phone Era

The third era, the Feature Phone Era (**1998–2008**)

These Mobile phones had

make voice calls, send text messages, play the Snake game,

listening to music , taking photos and

introduced the use of the Internet

GSM network providers added **GPRS** (General Packet Radio Service)

This network is often referred to as **2.5G**, or **halfway** between 2G and 3G

Network providers offering **CDMA** and other **TDMA**-based networks

The introduction of the **Motorola V3**, more commonly known as the **RAZR**

Selling over **100 million units**, to become the **second-best-selling mobile** phone of all time

PC World magazine ranked it **#12** in their "**50 Greatest Gadgets of the Past 50 Years**".
Due to **Poor Mobile Web Browser**, **high prices**, **poor marketing**, and **inconsistent rendering**, no one was using it.

4. The Smartphone Era

The Smartphone Era occurred at **third and fifth** eras and **spans** from **2002 to the present**

Smart phones have all the capabilities like **making a phone call**, **sending an SMS**, **taking a picture**, and **accessing the mobile web**,

Most smart phones are **distinctive** in that they use a **common OS**, **a larger screen size**, a **QWERTY keyboard or stylus for input**, and **Wi-Fi.**

Nokia 9000 series of "Communicator" smart phones

Symbian initially a joint venture of Nokia, Motorola, Ericsson, and Psion.

Later Nokia created the **Symbian OS**, containing common libraries, tools, and frameworks

The **Symbian OS** is used for a variety of mobile devices, **Nokia S60 , the 6260 and N95**

Microsoft attempted **Windows CE platform**, which would later become **Windows Mobile**

Palm OS based PDA with a phone module to create **PDA-style smart phones**.

Research in Motion (RIM) applied its two-way paging that create the first **BlackBerry**, which would be used to **"push" email.**

5. The Touch Era

"Change occurs because there's a gap between **what is and what should be "**-**Craig McCaw**
"Every once in a while a **Revolutionary Product** comes along that **Changes Everything**."-**Steve Jobs**

First Touch Screen Phone (1992)

The IBM Simon was the first of its kind when it came out in 1992.
This phone was the **1992 version of today's I-Phone**.
It was **touch screen**, **portable**, had a **calculator**, **email**, and could work on networks.

IBM Simon

I-Phone 1

Mobile devices of the **Touch Era** are a completely **new medium** capable of **offering real people** new and exciting ways to **interact and understand information**.

The devices of **tomorrow** will be able to **leverage location**, **movement**, and the **collective knowledge of mankind**, to provide people's **lives with greater meaning**

The Mobile Ecosystem

The mobile ecosystem or a system of layers where each layer is reliant on the others to create a seamless, end-to-end experience.

1. Operators

The **Base Layer** in the mobile ecosystem is the **operator.**
Operators can be referred to as

- **Mobile Network Operators (MNOs)**
- **Mobile Service Providers**,
- **Wireless Carriers or Carriers**
- **Mobile Phone Operators**
- **Cellular Companies**

Operators Responsibilities:

- They Install **Cellular Towers**,
- Operate the **Cellular Network**,
- Make **Services Available** for **Mobile Subscribers**,
- **Maintain Relationships** With the Subscribers,
- Handling **Billing and Support**,
- Offering **Subsidized Device Sales**
- Network of **Retail Stores**.

Services
Applications
Application frameworks
Operating systems
Platforms
Devices
Aggregators
Networks
Operators

The Layers of The Mobile Ecosystem

World's Largest Mobile Operators

Rank	Operator	Markets	Technology	Subscribers (in millions)
1.	China Mobile	China (including Hong Kong) and Pakistan	GSM, GPRS, EDGE, TD-SCDMA	436.12
2.	Vodafone	United Kingdom, Germany, Italy, France, Spain, Romania, Greece, Portugal, Netherlands, Czech Republic, Hungary, Ireland, Albania, Malta, Northern Cyprus, Faroe Islands, India, United States, South Africa, Australia, New Zealand, Turkey, Egypt, Ghana, Fiji, Lesotho, and Mozambique	GSM, GPRS, EDGE, UMTS, HSDPA	260.5
3.	Telefónica	Spain, Argentina, Brazil, Chile, Colombia, Ecuador, El Salvador, Guatemala, Mexico, Nicaragua, Panama, Peru, Uruguay, Venezuela, Ireland, Germany, United Kingdom, Czech Republic, Morocco, and Slovakia	CDMA, CDMA2000 1x, EV-DO, GSM, GPRS, EDGE, UMTS, HSDPA	188.9
4.	América Móvil	United States, Argentina, Chile, Colombia, Paraguay, Uruguay, Mexico, Puerto Rico, Ecuador, Jamaica, Peru, Brazil, Dominican Republic, Guatemala, Honduras, Nicaragua, Ecuador, and El Salvador	CDMA, CDMA2000 1x, EV-DO, GSM, GPRS, EDGE, UMTS, HSDPA	172.5
5.	Telenor	Norway, Sweden, Denmark, Hungary, Montenegro, Serbia, Russia, Ukraine, Thailand, Bangladesh, Pakistan, and Malaysia	GSM, GPRS, EDGE, UMTS, HSDPA	143.0
6.	China Unicom	China	GSM, GPRS	127.6
7.	T-Mobile	Germany, United States, United Kingdom, Poland, Czech Republic, Netherlands, Hungary, Austria, Croatia, Slovakia, Macedonia, Montenegro, Puerto Rico, and U.S. Virgin Islands	GSM, GPRS, EDGE, UMTS, HSDPA	126.6
8.	TeliaSonera	Norway, Sweden, Denmark, Finland, Estonia, Latvia, Lithuania, Spain, and Central Asia	GSM, GPRS, EDGE, UMTS, HSDPA	115.0

2. Networks

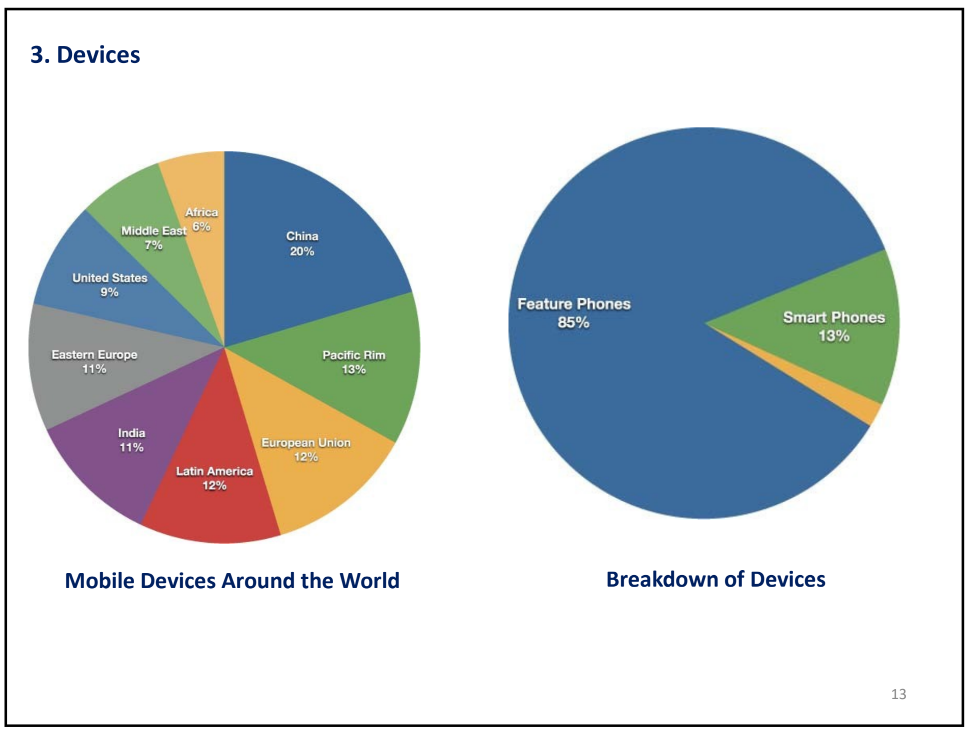
Operators operate **wireless networks** and cellular technology is just a **radio** that receives a **signal from an antenna**.
The type of radio and **antenna determines** the **capability of the network** and **the services** you can enable on it.

GSM Mobile Network Evolutions

2G	Second generation of mobile phone standards and technology	Theoretical max data speed
GSM	Global System for Mobile communications	12.2 KB/sec
GPRS	General Packet Radio Service	Max 60 KB/sec
EDGE	Enhanced Data rates for GSM Evolution	59.2 KB/sec
HSCSD	High-Speed Circuit-Switched Data	57.6 KB/sec

3G	Third generation of mobile phone standards and technology	Theoretical max data speed
W-CDMA	Wideband Code Division Multiple Access	14.4 MB/sec
UMTS	Universal Mobile Telecommunications System	3.6 MB/sec
UMTS-TDD	UMTS + Time Division Duplexing	16 MB/sec
TD-CDMA	Time Divided Code Division Multiple Access	16 MB/sec
HSPA	High-Speed Packet Access	14.4 MB/sec
HSDPA	High-Speed Downlink Packet Access	14.4 MB/sec
HSUPA	High-Speed Uplink Packet Access	5.76 MB/sec

3. Devices



13

4. Platforms

A mobile platform's primary duty is to **provide access to the devices**.
To **run software and services** on devices we need ***Platform*** *or a* ***Core Programming Language***.
Platforms are split into three categories:

1. **Licensed**
2. **Proprietary**
3. **Open Source**

1. Licensed Platforms

Licensed Platforms are sold to **device makers** for **nonexclusive** distribution on devices.
They create a common platform of development **Application Programming Interfaces (APIs)**.

Categories of Licensed Platforms:

A. Java Micro Edition (Java ME)

J2ME/ Java ME is the most **predominant software platform**
It provides a collection of Java APIs for the development of software

B. Binary Runtime Environment for Wireless (BREW)

BREW is a licensed platform created by **Qualcomm** for mobile devices for the U.S. market.
It is an **interface-independent platform** that runs a variety of application frameworks, such as **C/C++, Java, and Flash Lite**.

14

C. Windows Mobile
Windows Mobile is a compact version of the Windows operating system, combined with a suite of basic applications for mobile devices that is based on the **Microsoft Win32 API.**
D. LiMo
LiMo is a **Linux-based mobile platform** created by the **LiMo Foundation**.
Although Linux is open source, LiMo is a licensed mobile platform used for mobile devices.
LiMo includes SDKs for creating **Java, native, or mobile web applications using the WebKit** browser framework.

2. Proprietary Platforms
Proprietary platforms are designed and developed for use on their devices.
They are **not available** for use by **competing device** makers.
A. Palm
Palm uses three different proprietary platforms.
- **Palm OS platform** based on the C/C++ for their Palm **Pilot line**, but is now used in low-end smart phones such as the **Centro line**.
- **Windows Mobile**-based platform for higher-end smartphones like the **Treo line**.
- **webOS** is based on the WebKit browser framework, and is used in the **Prē line**.

B. BlackBerry
Research in Motion (RIM) maintains their own proprietary Java-based platform, used exclusively by their BlackBerry devices.
C. iPhone
Apple uses a proprietary version of **Mac OS X** as a platform for their iPhone and iPod touch line of devices, which is based on **Unix.**

3. Open Source Platforms
Open source platforms are **freely** available for users to **download**, **alter,** and **edit**.
Open source mobile platforms are **newer** and slightly **controversial,** but they are **increasingly gaining traction** with device makers and developers.
Android is one of these platforms.
It is developed by the **Open Handset Alliance**, which is spearheaded by **Google**.
The **Alliance** seeks to develop an open source mobile platform based on the **Java**

5. Operating Systems

Operating systems often have core services or toolkits that enable applications to talk to each other and share data or services.

Although not all phones have operating systems, the following are some of the most common:

Symbian

Symbian OS is a open source operating system designed for mobile devices, with associated libraries, user interface frameworks, and reference implementations of common tools.

Windows Mobile

It is the mobile operating system that runs on top of the Windows Mobile platform.

Palm OS

Palm OS is the operating system used in Palm's lower-end Centro line of mobile phones.

Linux

It is used as an operating system to power Smart phones, including Motorola's RAZR2.

Mac OS X

A specialized version of Mac OS X is the operating system used in Apple's iPhone and iPod touch.

Android

Android runs its own open source operating system, which can be customized by operators and device manufacturers.

17

Applications

Application Frameworks are used to create applications, such as a **game, a web browser, a camera, or media player**

Frameworks are well **standardized,** but the devices are not.

The largest **challenge** of deploying applications is knowing the specific **Device Attributes and Capabilities**

For example, if you are creating an application using the **Java ME application framework**, you need to know

- ❖What Version of Java ME the Device Supports,
- ❖The Screen Dimensions,
- ❖The Processor Power,
- ❖The Graphics Capabilities,
- ❖The Number of Buttons it has, and
- ❖How the Buttons are Oriented

Mobile Applications Provide **Excellent User Experience**

Always Comes at a **Fantastic Development Cost** and Potentially Create a **Positive Return** on Investment

18

Services

The Last Layer in the Mobile Ecosystem is Services.
Services Include Tasks Such as
Accessing The Internet,
Sending Text Message
Being Able to get a Location etc…

Why Mobile?

The majority of mobile strategies start with a **well-thought-out plan** of how to use the medium **to meet the needs of users** or further **the goals of the business**

The **people of Companies, people at Conferences**, **Online Article Writers**, and **Mobile Experts** claim that "**Mobile is the Next Big Thing**"

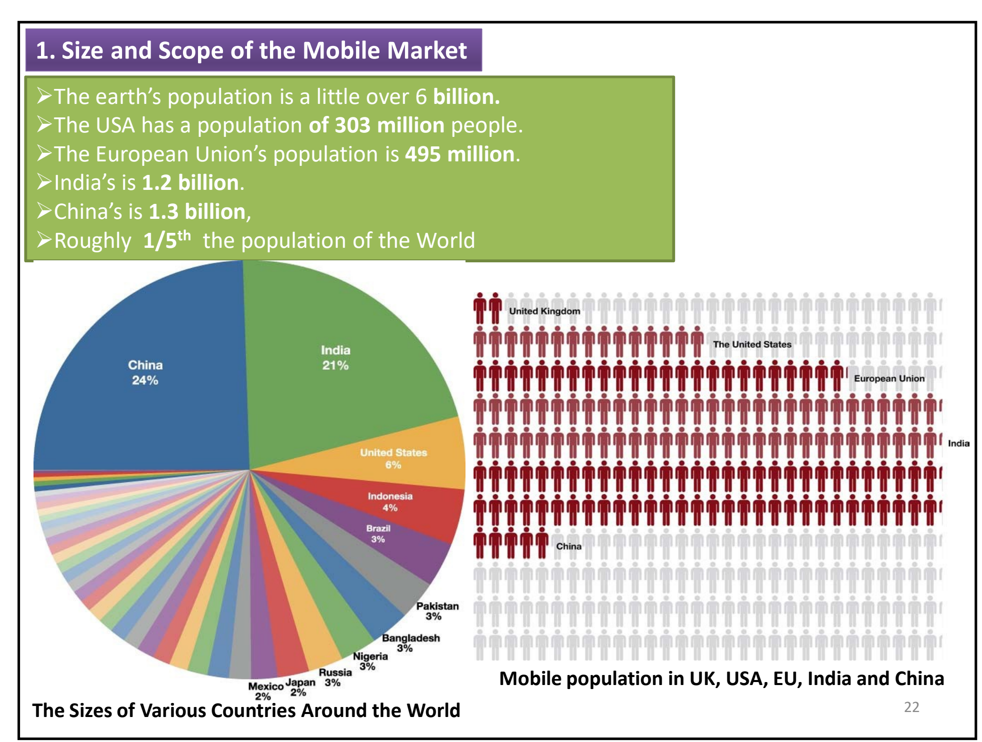
"Why Mobile?"

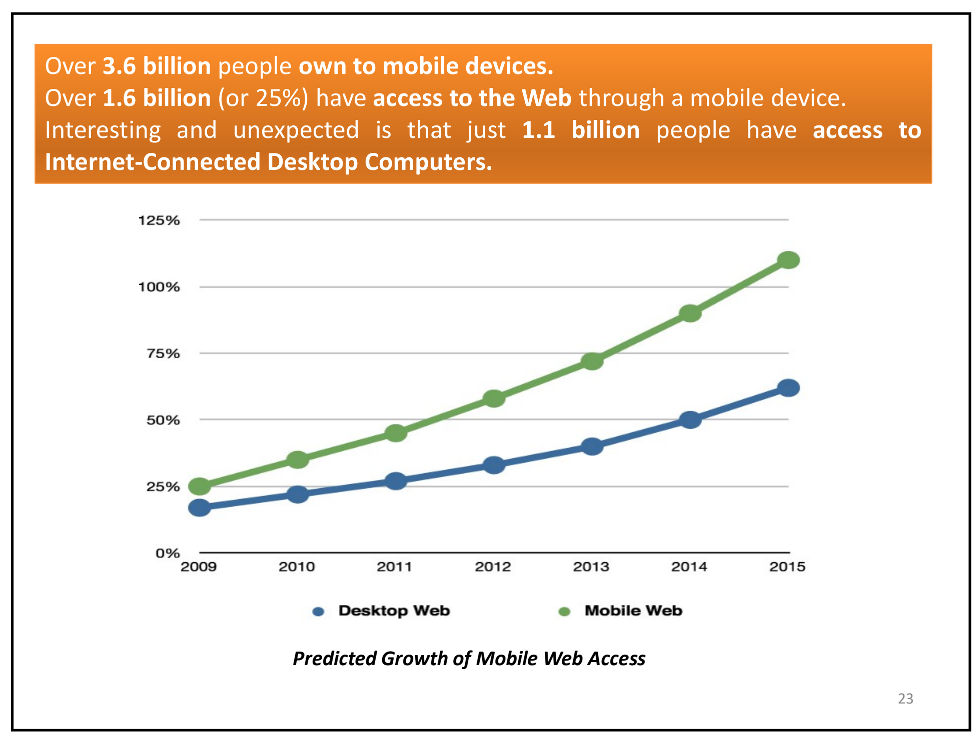
Mobile is not only a **New Medium** also a **New Business Model.**

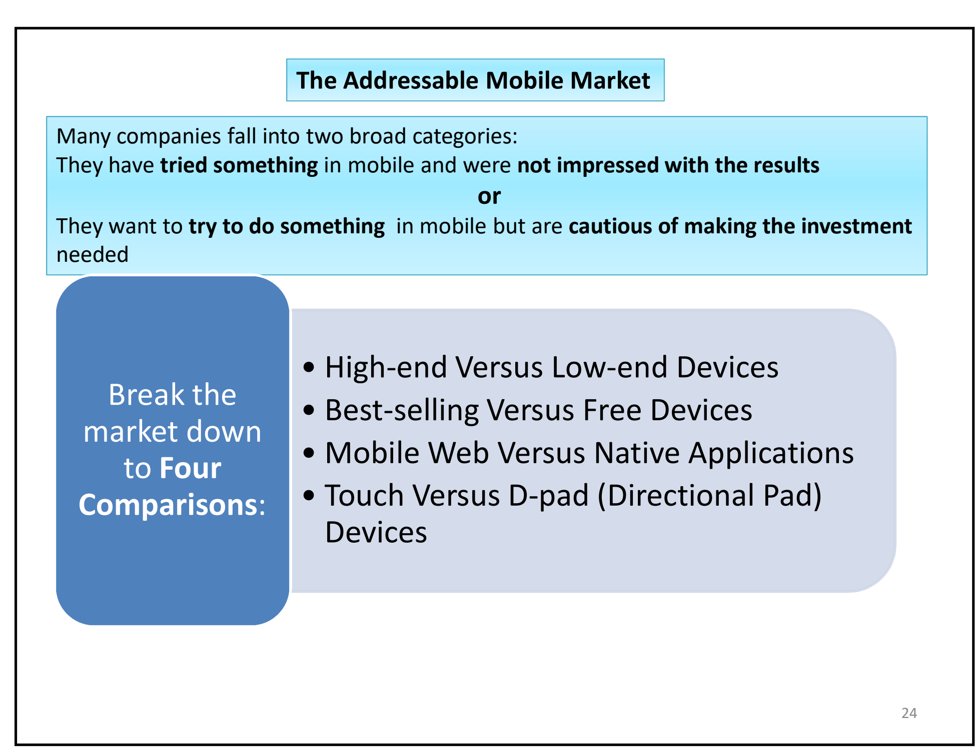
Building a **Successful Long-term Business** around the underserved needs of real people

It simply requires an honest look at **What Exists, What Users Want**

1. Size and Scope of the Mobile Market
2. The Addressable Mobile Market
   - High-End Versus Low-End Devices
   - Best-selling Versus Free
   - Mobile Web Versus Native Applications
   - Touch Versus D-Pad
1. Mobile As a Medium
   - The Printing Press
   - Recordings
   - Cinema
   - Radio
   - Television
   - The Internet
   - Mobile




Over 3.6 billion people own to mobile devices.
Over 1.6 billion (or 25%) have access to the Web through a mobile device.
Interesting and unexpected is that just 1.1 billion people have access to Internet-Connected Desktop Computers.
125%
100%
75%
50%
25%
0%
2009
2010
2011
2012
2013
2014
2015
Desktop Web
Mobile Web
Predicted Growth of Mobile Web Access
23


The Addressable Mobile Market
Many companies fall into two broad categories:
They have tried something in mobile and were not impressed with the results
or
They want to try to do something in mobile but are cautious of making the investment needed
Break the market down to Four Comparisons:
• High-end Versus Low-end Devices
• Best-selling Versus Free Devices
• Mobile Web Versus Native Applications
• Touch Versus D-pad (Directional Pad) Devices
24

Mobile As a Medium

After examining the **size and scope of the mobile market**, we need to understand what it **means to users** and ultimately **how it will benefit the business**

"Mobile is the Seventh Mass Media"- Tomi Ahonen and expert on next-generation wireless

Tomi Ahonen points out that **each of the mass media** has advantages and disadvantages, each playing a significant role in society, with **mobile being the latest in the series**.

Seven Mass Media:

- **The Printing Press**
- **Recordings**
- **Cinema**
- **Radio**
- **Television**
- **The Internet**
- **Mobile**

1. The Printing Press

The ***First Mass Medium*** was the printing press, one of the greatest inventions of mankind.

The time needed to ***Publish Information*** was dramatically reduced, and distributed farther and faster than **Handwritten Predecessors**.

Not to mention there was ***Less Damage*** from time or the elements.

The printing press has continually ***played a crucial role in history.***

Babasaheb Ambedkar Wrote In the **First Issue** of **Mooknayak–"Need of An Independent Newspaper"**

2. Recordings

The **Second Mass Medium** was the **Recorded Sound**, initially on ***Edison's Phonograph Cylinder*** and later on more durable materials like **Glass, Vinyl, Magnetic Tape, and CD.**

Recordings ***enabled people*** to ***Share Information Over Time*** and over ***Great Distances***.

Recorded music also played an important part in ***Influencing Society.***

Jazz gave new opportunities to ***Freed Slaves in America as Entertainers***.

After the ***End of Slavery***, ***African-American Jazz*** musicians ***Became Popular Figures*** in modern music

Elgar Recording in 1914

Edison's Phonograph Cylinder

Three African-American Jazz Composers: Davis, Ellington, Payton

3. Cinema

The ***Third Mass Medium*** was the **Cinema**.

Like recordings, Cinema as entertainment, but **Cinema Enabled a Visual Experience** to be shared over time and distance.

Suddenly, we were able to witness **distant or past events firsthand**, enabling the viewer to **Draw Conclusions** from **What we Saw and Heard.**

Worlds first movie **La Sortie de l'usine Lumière à Lyon ("the exit from the Lumière factory in Lyon") 46 seconds previewed in 1895.**

India's first feature film "**Raja Harishchandra"** by **Dadasaheb Phalke** was released in **1913**

https://upload.wikimedia.org/wikipedia/en/transcoded/6/67/L%27Arriv%C3%A9e_d%27un_train_en_gare_de_La_Ciotat%2C_Complete.webm/L%27Arriv%C3%A9e_d%27un_train_en_gare_de_La_Ciotat%2C_Complete.webm.360p.webm

4. Radio

The **Fourth Mass Medium was Radio,** an **extension of recordings**, but including **the *Live Broadcast of Material***.

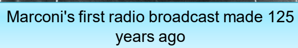
Information could be distributed **as it happened** and **as far as the Radio Signal Would Reach**.

Like Cinema, **Radio** could give listeners a **Powerful Personal Experience**.

Because recording technology was becoming smaller, events could be recorded where **Film Cameras Could not go**.

Example:

1. **Winston Churchill's** radio addresses, which brings **Hope and Confidence** to the people of **Great Britain** during **the Frequent Air Raids of World War II**.
2. **Edward R. Murrow's** radio **reports from the battlefield**, which brought the war into living rooms around the world


Marconi's first radio broadcast made 125 years ago

The First Radio Broadcast in India

India's first radio station was inaugurated in Mumbai on July 23, 1927

29

5. Television

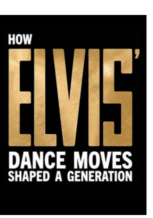
Television is the **Fifth Mass Medium and** a **Visual Extension of Radio**.

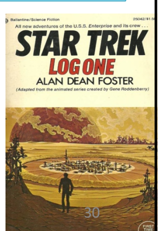
Television transformed into a more **Iconic , most influential** and **most disruptive Medium**.

The television became an **Alternative** to previous media like **cinema and Radio**.

The influence of television has had on culture, how events

- **Landing on the Moon**
- The **conflicts in Southeast Asia**
- The **Beatles performing on the *Ed Sullivan Show***
- ***Elvis Presley's dance moves*** *and*
- *The **weekly adventures of*** **the** ***USS Enterprise transformed culture***.





30

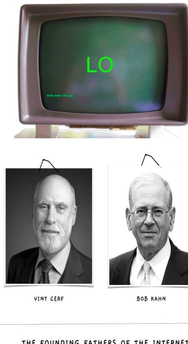
6. The Internet

Nothing Happen for a long time after the invention of the television.

Until we started **plugging our computers into the phone jack to hear that weird tone** means **connecting to the Internet and the WWW.**

On **October 29th, 1969** the **First Message** on the internet was sent **from UCLA** to **Stanford University**: it was just two letters **"lo."**

In late 2008, Web 2.0 showed us that it could be used in meaningful ways i.e., Users can **Work Together**, *add Information*, and **run Programs** all from the Web



The **Web** is **threatening** the **Printing Press** and **Newspaper** empires, Because

- ✓The **iTunes Music Store** is available freely on the Web .
- ✓**Purchase, Download and** S**tream** movies.
- ✓**Podcasts and streaming audio** are transforming what "radio" means.
- ✓All major TV networks are either **selling** or **streaming** their content online.

31

7. Mobile

The **mobile industry** actually started around the **same time as the Web**.

Mobile is **quite unique**, the **only mass medium** that can do **everything** the previous **Six Media** can do.

The **first mobile phone** in **India** was used in **1995** by the India from the **Nokia company** by paying a price of **45K**

India First Mobile Phone Call - C.M of Bengal, Joytib Basu had called the **C.M of Delhi, Sukhi Ram** on **1995/July/30**.

At that time there was a **charge of Rs 16 for a minute** call and the one who would **receive** the call had to **pay Rs 8 per minute**

The following are the unique and competitive **benefits**

- ✓ **Read and Publish**
- ✓ **Play Recordings**
- ✓ **Watch Movies**
- ✓ **Listen to Radio**
- ✓ **Watch Television**
- ✓ **Use the Internet**

32

Mobile's Unique Benefits

Tomi Ahonen points out, **Mobile** has **Five Unique Benefits** that none of the Media does.

- ***The First Truly Personal Mass Media***
  To interact with information in a **Personal and Intimate** way (
- ***The first Always-on Mass Media***
  The capability to **send and receive information at all times**, even when idle,
  Enabling the device to **Predict Tasks**
- ***The first Always-carried Mass Media***
  Seven out of ten people sleep with their phone within arm's reach.
- ***The only mass media with a Built-in Payment Channel***
  Every phone has a **built-in means of purchasing content,** goods etc..
- ***At the point of* Creative Impulse**
  - Create content and Distribute it
  - Uploading Pictures to Social Networks
  - **Information and Experiences** can be shared with audiences as they happen and from multiple points of view

33

Types of Mobile Applications

The ***mobile medium type*** *is the type of* ***Application Framework or Mobile Technology*** *that* presents **Content or Information** to the user.

It is a technical approach regarding which **Type of Medium** to use.

This decision is determined by the impact it will have on the **User Experience.**

The **Technical Capabilities** and **Capacity** of the publisher also factor into which approach to take.

Multiple Mobile Application Medium Types

34

Types of Mobile Applications

1. SMS
2. Mobile Websites
3. Mobile Web Widgets
4. Mobile Web Applications
5. Native Applications
6. Games

1. SMS

The most basic mobile application you can create is an SMS application.

Pros

- They **Work on Any Mobile Device** nearly instantaneously.
- They're **Useful for Sending Timely Alerts** to the user.
- They can be **Incorporated Into any Web or Mobile Application**.
- They can be **Simple to Set Up and Manage**.

Cons

- They're **Limited to 160 Characters**.
- They provide a **Limited Text-based Experience**.
- They can be **Very Expensive**.

35

2. Mobile Websites

Mobile websites are characterized by simple **"drill-down" architecture**, or the simple presentation of **Navigation Links** that take to a **Page a Level Deeper**.

- Mobile websites often have a **Simple Design , Informational** and few **Interactive Elements**
- Mobile websites are **Easy to Create** but they **fail** to display across **Multiple Mobile Browsers**
- Mobile websites have made up with the early **WML-based sites.**
- The mobile browsers for **iPhone and Android** are introduced the quality of mobile websites began to **Improve Dramatically** and **Usage Improved.**

Pros

- They are easy to **Create, Maintain, and Publish**.
- They can use the **same tools and techniques** like desktop sites.
- Nearly **all mobile devices** can view mobile websites

Cons

- **Difficult to support across multiple devices**.
- Offer users a **Limited Experience**.
- Mobile websites are simply desktop content reformatted for mobile devices.
- They can **Load Pages Slowly**, due to **Network Latency.**

An Example of a Mobile Website

36

3. Mobile Web Widgets

Mobile web user experience was severely **Underutilized** and **Failed** to gain traction
So Several **Operators, Device Makers, and Publishers** began creating **Widget Platforms** to counter the **Mobile Web's Weaknesses**

A Web Widget is component of a user interface that operates in a particular way (OR)
A mobile web widget is a **Standalone Chunk** of **HTML-based Code** that is executed by the end user in a particular way

Example: Opera Widgets, Nokia Web RunTime (WRT), Yahoo! Blueprint, and Adobe Flash Lite

web widget

Pros

- Easy to **Create,** using basic HTML, CSS, and JavaScript .
- Simple to **Deploy** across multiple handsets.
- Offer an i**Mproved** user experience and a **Richer Design**, tapping into device features and offline use.

Cons

- Require a **Compatible Widget Platform** to be installed on the device.
- They cannot run in any mobile web browser.
- They require learning additional proprietary, non-web-standard techniques.

37

4. Mobile Web Applications

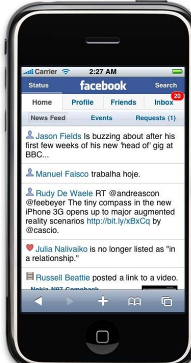
Mobile Web Applications are **Mobile Applications** that do **not need to be installed** or **compiled** on the target device.
Developed using **XHTML, CSS, and JavaScript**
They provide an **Application-like Experience** to the user
Allow users to **Interact With Content in Real Time**, where a **Click or Touch Performs** an action within the current view

Pros

- Easy to **Create** using basic HTML, CSS, and JavaScript knowledge.
- Simple to **Deploy** across multiple handsets.
- Offer a better **User Experience** and a rich design, tapping into device **features and offline** use.
- Content is **Accessible** on any mobile web browser.

Cons

- The **Optimal Experience** might not be available on all handsets.
- They can be **Challenging to Support** across multiple devices.
- They **Don't Always Support Native Application Features**, like **Offline Mode, Location Lookup, File System Access, Camera**, and so on.



The Facebook Mobile Web App

38

5. Native Applications / Platform Applications

1. **Platform Applications** which are developed and compiled for each mobile platform
2. The most common of all platforms is **Java ME** (formerly **J2ME**).
3. A device written as a **Java ME MIDlet** should work on the majority of feature phones.
4. **Platform Application-Devices to Target**, **Testing and Certification** and **Distribute** the application to users
5. Apps are **Certified, Sold, and Distributed** either through an **operator portal** or an **app store.**
6. Apps tap into the majority of the **Device Features**, **Working Online or Offline**, **Accessing the Location** and the **File System**

Pros

- Offer a **Best-in-class User Experience**, a **Rich Design** and **Offline Use**.
- Simple to **Develop** for a Single Platform.
- **Charge** for Applications.

Cons

- Cannot be **Easily Ported** to other mobile platforms.
- **Developing, Testing, and Supporting** multiple device platforms is costly.
- **Require Certification** and **Distribution** from a third party and have no control.
- **Require to Share Revenue** with the one or more third parties.

39

6. Games

Games are just like **Native Applications** that use the similar platform SDKs to create immersive experiences.
Games are different from native applications for two reasons:

1. **Cannot be easily duplicated** with web technologies
2. **Porting** them to multiple mobile platforms is a bit easier

Pros

- Provide a simple and easy to create an immersive experience.
- Ported to **Multiple Devices** relatively easily.

Cons

- They can be costly to develop as an original game title.
- They cannot easily be ported to the mobile web.

An Example Game for the Iphone

40

Table: Mobile Application Media Matrix

	Device support	Complexity	User experience	Language	Offline support	Device features
SMS	All	Simple	Limited	N/A	No	None
Mobile websites	All	Simple	Limited	HTML	No	None
Mobile web widgets	Some	Medium	Great	HTML	Limited	Limited
Mobile web applications	Some	Medium	Great	HTML, CSS, JavaScript	Limited	Limited
Native applications	All	Complex	Excellent	Various	Yes	Yes
Games	All	Complex	Excellent	Various	Yes	Yes

41

Mobile Information Architecture (MIA)

Information Architecture (IA), is a **Well-engineered Product** with **Good Visual Design**
It can still **fail** because of poor information architecture.
The **Successful Mobile Products** always have a **Well Thought-out Information Architecture**
The mobile information architecture defines **Not Just How Your Information Will be Structured** but also **How People Will Interact With it**.
A good **Mobile Information Architecture** is based on the various **User Contexts**

What Is Information Architecture?

The **Structural Design** of **Shared Information Environments**
The combination of **Organizations, Labeling, Search, and Navigation Systems** within websites and intranets
The **Art and Science** of **Shaping Information Products and Experiences** to support usability and findability.
An **Emerging Discipline and Community** of practice focused on bringing **Principles of Design and Architecture** to the **Digital Landscape**

42

Information Architecture

The **Organization of Data** within an informational space i.e, how the user will get to information or perform tasks within a website or application.

Interaction Design

The **Design of How the User can Participate** with the information present, either in a direct or indirect way, meaning how the user will interact with the website of application to create a more meaningful experience and accomplish her goals.

Information Design

The **Visual Layout of Information** or how the user will assess meaning and direction given the information presented to him.

Navigation Design

The words used to describe information spaces; the labels or triggers used to tell the users what something is and to establish the expectation of what they will find.

Interface Design

The **Design of the Visual Paradigms** used to create action or understanding.

The Elements of User Experience

Jesse James Garrett
jjg@jjg.net
30 March 2000

A basic duality: The Web was originally conceived as a hypertextual information space; but the development of increasingly sophisticated front- and back-end technologies has fostered its use as a remote software interface. This dual nature has led to much confusion, as user experience practitioners have attempted to adapt their terminology to cases beyond the scope of its original application. The goal of this document is to define some of these terms within their appropriate contexts, and to clarify the underlying relationships among these various elements.

Web as software interface

Visual Design: graphic treatment of interface elements (the "look" in "look-and-feel")

Interface Design: as in traditional HCI: design of interface elements to facilitate user interaction with functionality
Information Design: in the Tuftean sense: designing the presentation of information to facilitate understanding

Interaction Design: development of application flows to facilitate user tasks, defining how the user interacts with site functionality

Functional Specifications: "feature set": detailed descriptions of functionality the site must include in order to meet user needs

User Needs: externally derived goals for the site; identified through user research, ethno/techno/psychographics, etc.
Site Objectives: business, creative, or other internally derived goals for the site

task-oriented

Web as hypertext system

Visual Design: visual treatment of text, graphic page elements and navigational components

Navigation Design: design of interface elements to facilitate the user's movement through the information architecture
Information Design: in the Tuftean sense: designing the presentation of information to facilitate understanding

Information Architecture: structural design of the information space to facilitate intuitive access to content

Content Requirements: definition of content elements required in the site in order to meet user needs

User Needs: externally derived goals for the site; identified through user research, ethno/techno/psychographics, etc.
Site Objectives: business, creative, or other internally derived goals for the site

information-oriented

Content-heavy Site that works well on the desktop, and is designed to present the **Maximum Amount of Information** above the “fold” or where the screen cuts off the content.
However, in the **mobile browser**, the **text is far too small** to be useful.
Comparing the New York Times website in desktop and mobile browsers
45

The many mobile experiences of the New York Times
46

Mobile Information Architecture

1. Keeping It Simple
- i. Support Your Defined Goals
- ii. Clear, Simple Labels

2. Site Maps
- i. Limit Opportunities for Mistakes
- ii. Confirm the Path by Teasing Content

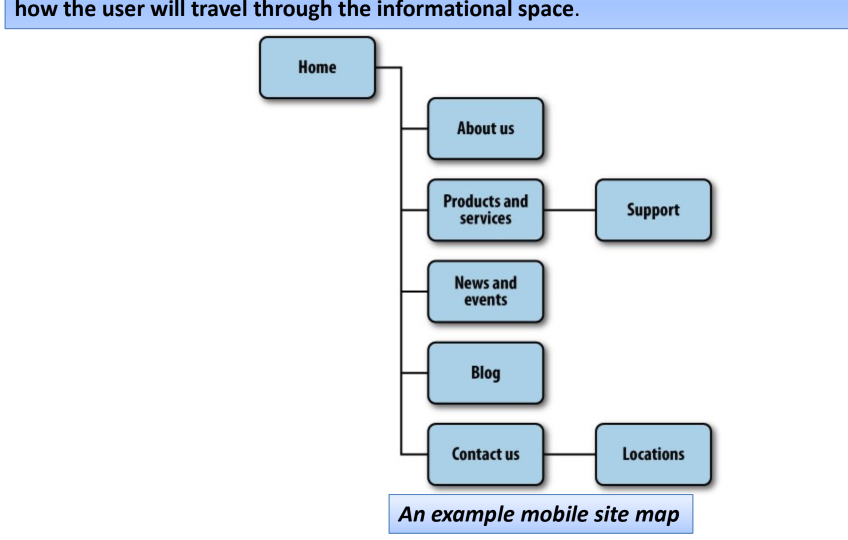
3. Clickstreams
4. Wireframes
5. Prototyping
- i. Paper Prototypes
- ii. Context prototype
- iii. HTML prototypes

1. Keeping It Simple
- i. Support Your Defined Goals
- ii. Clear, Simple Labels

➢**Good Trigger Labels** describe each **Link / Action** are crucial in Mobile.
➢Words like **"Products"** or "**Services"** aren't good trigger labels
➢Keep all your labels **Short and Descriptive**, and never try to use the words to evoke action

Example:
If the user is just trying to get music, don't call it **"My Music," "My MP3s,"** or something made up that only strokes our corporate egos, such as **"AudioJams™"**—just call it **"Music."**

2. Site Maps
Site Maps are a classic information architecture deliverable.
They **visually represent** the **relationship of content to other content** and provide a map for **how the user will travel through the informational space**.



An example mobile site map

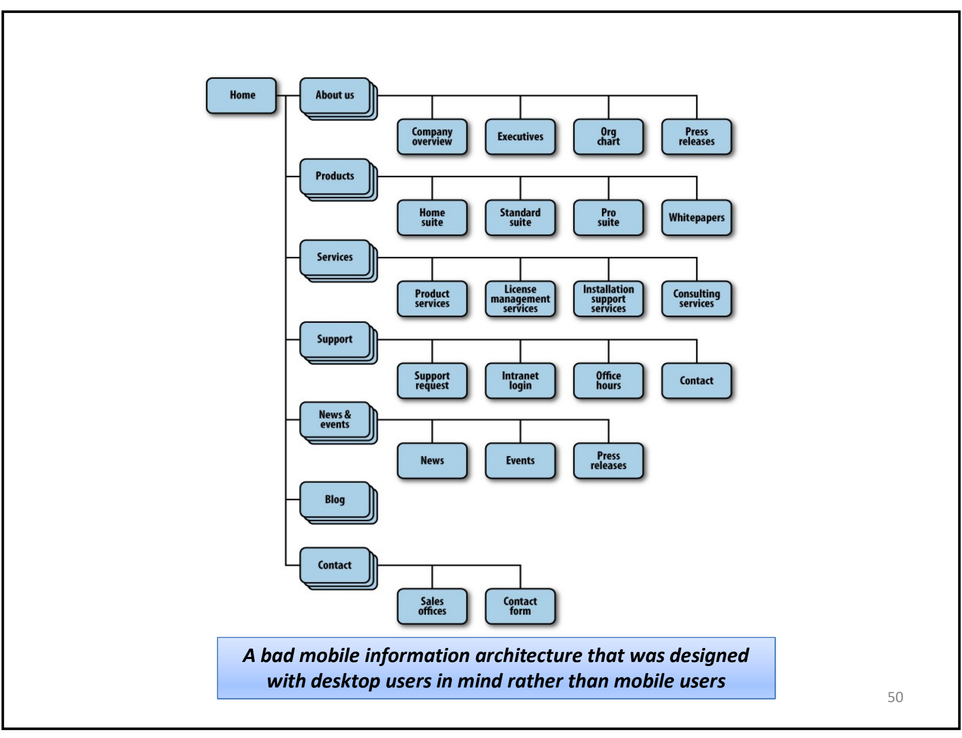
i. Limit Opportunities for Mistakes

Imagine a **road with a fork** in it. We can go either left or right.
The **risk** that we will make the **wrong choice** is only **50%** i.e., that we have a better than good chance that we will get to where we want to go.
But imagine **three roads**. Now our chances have dropped to **33%**.
Four roads drops your chances to **25%**, **Five roads** takes you down to **20%**.
Now a 20 percent chance isn't great, but it isn't too bad, either.

Now think of your **own website**.
How many primary navigation areas do you have? **Seven? Eight? Ten? Fifteen?**
What **risk** is there to the users for making a wrong choice?

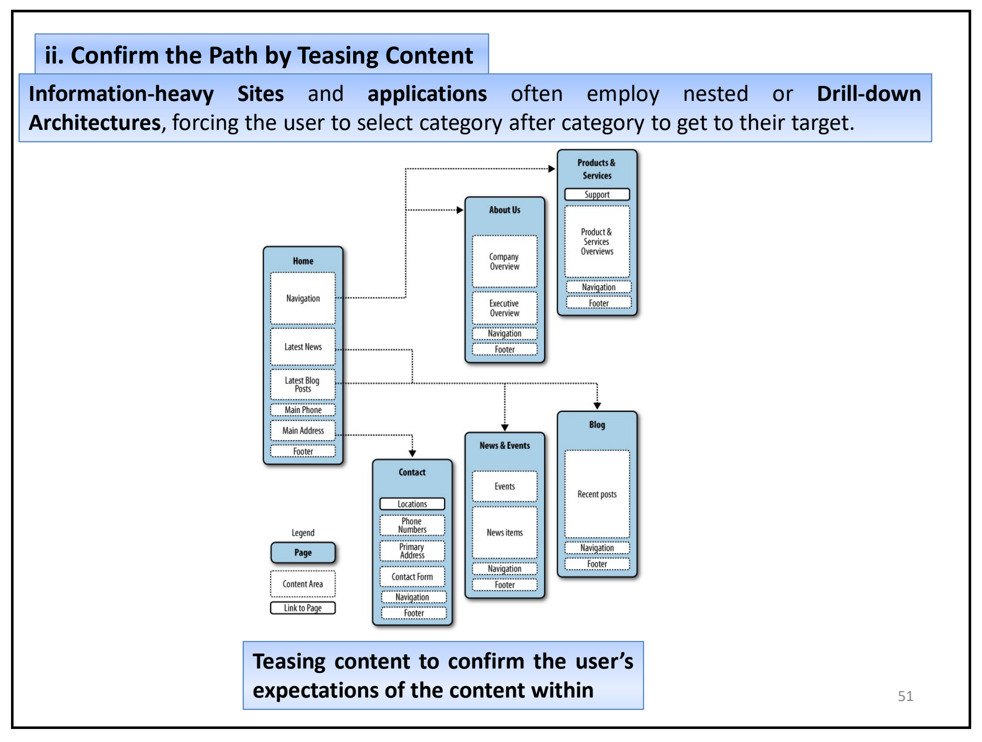
In the **Mobile Context**, tasks are **Short** and users have **Limited Time** to perform them.
And with **Mobile Websites**, users can't access to a **Reliable Broadband Connection** that allows them to quickly go back to the previous page.
The users more often than not have to **pay for each page view** in data charges.

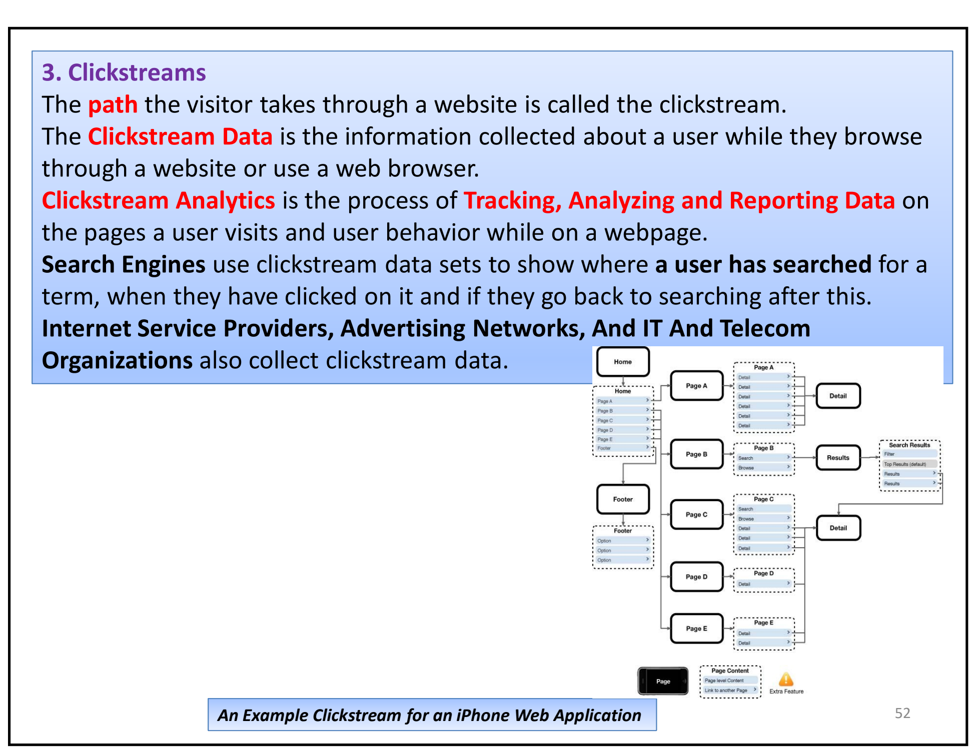
49

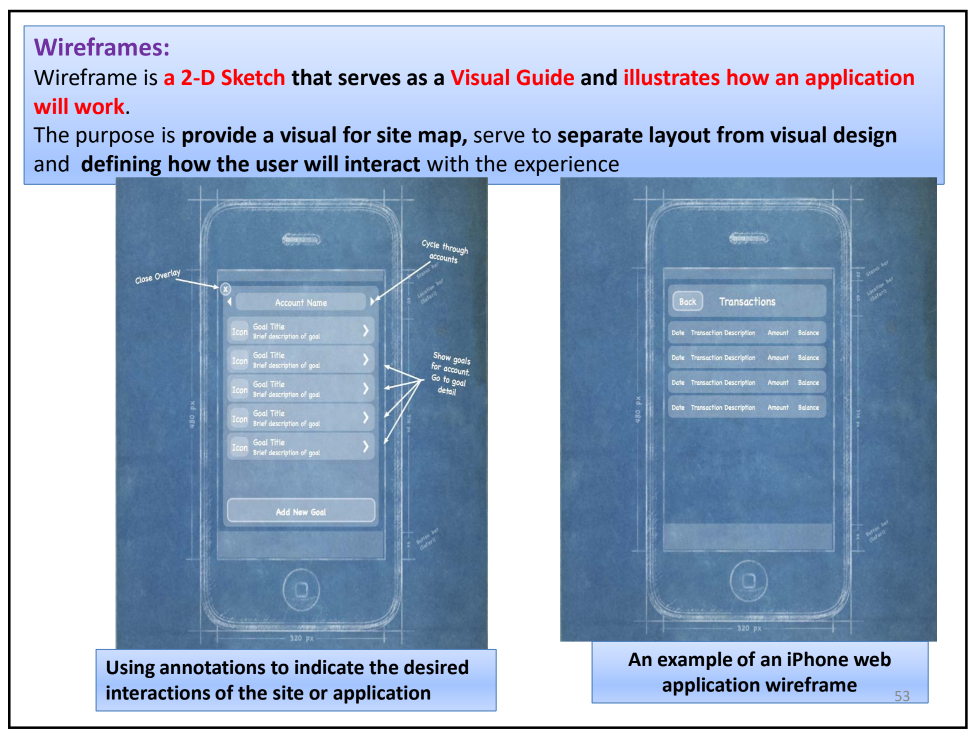


A bad mobile information architecture that was designed with desktop users in mind rather than mobile users

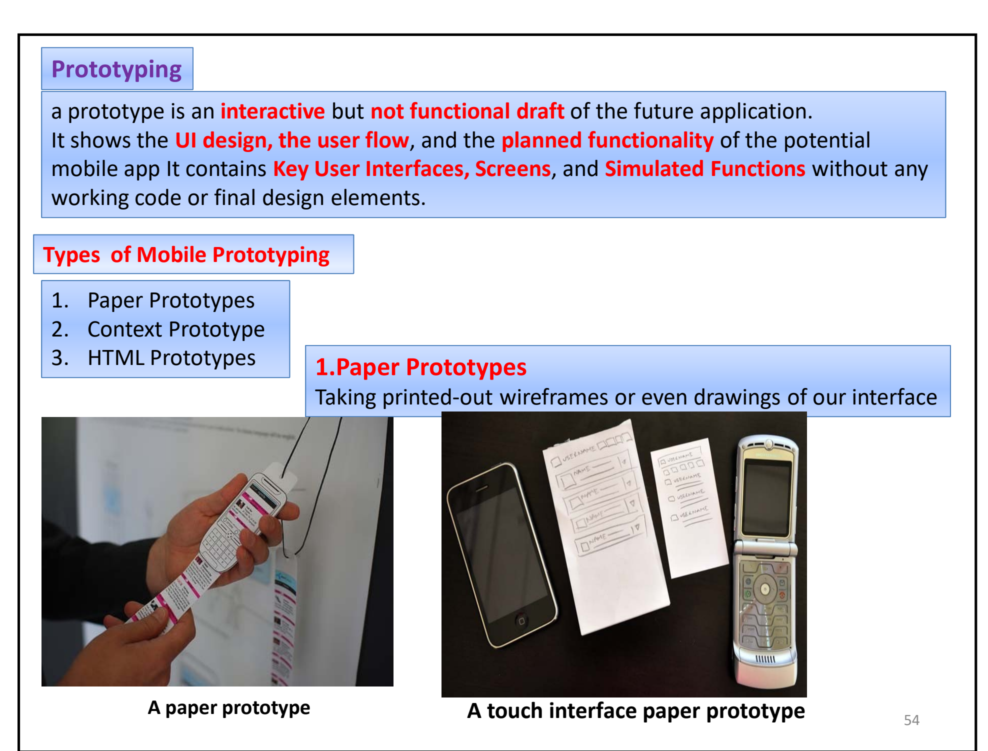
50


ii. Confirm the Path by Teasing Content
Information-heavy Sites and **applications** often employ nested or **Drill-down Architectures**, forcing the user to select category after category to get to their target.
Products & Services
Support
Product & Services Overviews
Navigation
Footer
About Us
Company Overview
Executive Overview
Navigation
Footer
Home
Navigation
Latest News
Latest Blog Posts
Main Phone
Main Address
Footer
Contact
Locations
Phone Numbers
Primary Address
Contact Form
Navigation
Footer
News & Events
Events
News items
Navigation
Footer
Blog
Recent posts
Navigation
Footer
Legend
Page
Content Area
Link to Page
Teasing content to confirm the user's expectations of the content within
51


3. Clickstreams
The path the visitor takes through a website is called the clickstream.
The Clickstream Data is the information collected about a user while they browse through a website or use a web browser.
Clickstream Analytics is the process of Tracking, Analyzing and Reporting Data on the pages a user visits and user behavior while on a webpage.
Search Engines use clickstream data sets to show where a user has searched for a term, when they have clicked on it and if they go back to searching after this.
Internet Service Providers, Advertising Networks, And IT And Telecom Organizations also collect clickstream data.
Home
Page A
Page B
Page C
Page D
Page E
Detail
Results
Search Results
Footer
Page
Page Content
Extra Feature
An Example Clickstream for an iPhone Web Application
52



Wireframes:
Wireframe is a 2-D Sketch that serves as a Visual Guide and illustrates how an application will work.
The purpose is provide a visual for site map, serve to separate layout from visual design and defining how the user will interact with the experience
Close Overlay
Cycle through accounts
Account Name
Goal Title
Add New Goal
Show goals for account Go to goal detail
Back
Transactions
Using annotations to indicate the desired interactions of the site or application
An example of an iPhone web application wireframe
53



Prototyping
a prototype is an interactive but not functional draft of the future application.
It shows the UI design, the user flow, and the planned functionality of the potential mobile app It contains Key User Interfaces, Screens, and Simulated Functions without any working code or final design elements.
Types of Mobile Prototyping
1. Paper Prototypes
2. Context Prototype
3. HTML Prototypes
1.Paper Prototypes
Taking printed-out wireframes or even drawings of our interface
A paper prototype
A touch interface paper prototype
54

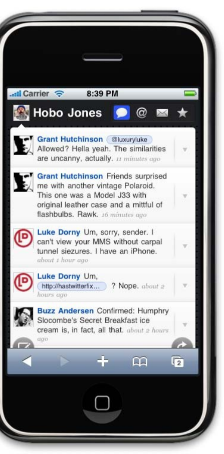
2. Context Prototype

Take a **higher-end device** that enables you to **load full-screen images** on it.
Take **wireframes or sketches** and load them onto the device, sized to fill the device screen

Context Prototype, or taking images loaded onto a device and testing them in the mobile context

3. HTML Prototypes

- This is a prototype that actually load onto a device **and produce the nearest experience** to the final product, but with **static dummy content and data.**
- A lightweight, semi functional static prototype is created using XHTML, CSS, and JavaScript.
- With a static XHTML prototype we use all the device metaphors of navigation and see how much content will really be displayed on screen.



An XHTML prototype

Benefits of Prototyping:

- Improves UX
- Helps With Focus and Collaboration
- Saves Time and Effort

Mobile Design

Create a great experience design with : **Context, Information Architecture, and Visual Design**

The visual design is the direct representation of everything under and the first impression the user will have.

A **Great Design** gives the user **High Expectations** of your site or application where as a **Poor Design** leads to **Lower Expectations**.

Lowest Common Denominator

- To reach the widest possible number of platforms, create a product that works on the most common architectural components on all platforms (Figure).
- **Computers-** Dozen of different platforms
- **Mobile Development-** Hundreds of different devices
- Typically, mobile design starts with the **Lowest Common Denominator**

NOKIA
Yahoo! Front Page
YAHOO!
oneSearch
Mail
Messenger
Upgrade to Yahoo! Go
More for your mobile
Options Edit Back
vodafone

A Lowest Common Denominator Design

57

1. Interpreting Design

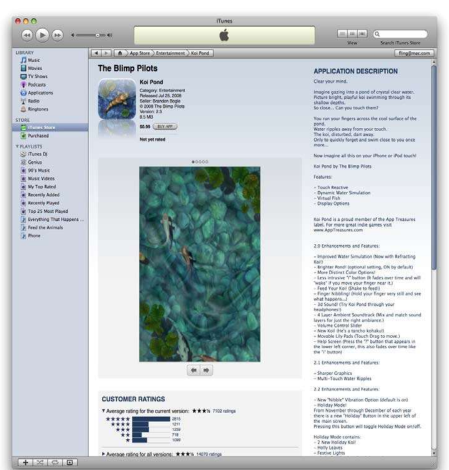
- Creating **A VISION** for how to communicate information or ideas and then duplicating that on the printed page.
- Every **EXPERIENCE** Is unique, which is depends on the user's screen size, web browser, text settings, the processor speed and connection to the Internet.
- There are too many variables to try to **"CONTROL"** the design completely.
- In mobile design, interpret about **good design** and translate it to the **new medium** that is both **technologically precise** and incredibly **demanding**
- **P**rovide the design with the **flexibility to present** information on a number of different devices.

58

2. The Mobile Design Tent-Pole

- In Hollywood, executives like to use the term **"Tent-pole"** to describe their movies and television shows.
- Tent-Pole means **BUSINESS**, and **CREATIVE**
- In mobile design, the de facto strategy is to create **Tent-pole Products**
- The products that support the **Largest Number of Devices** that **no one will ever use**.
- They are **creatively old**, **lack of inspiration**, and simply exist with no meaningful purpose to the user.
- To have a successful mobile design, **Made Emotional Connection** that **Serves Many Audiences**, **Many Cultures**, and **Many Places** and **Design Experiences**.
- Too often designers simply repeat the visual trends copying the inspiration of others

the best-selling games and applications for the iPhone are the ones with the best designs



Users are able to determine the quality of the app, largely influenced by the design, before they make a purchase

3. Designing for the Best Possible Experience

- ❖ When the first iPhone came out, it provides the best possible experience and that is where consumers will go.
- ❖ Since starting, the iPhone destroy every record in mobile devices, becoming
  - ❖ One of the **best-selling phones** ever
  - ❖ One of the **most used mobile browsers** in the world
  - ❖ **2/3 of mobile browsing** in the U.S. comes from an iPhone or an iPod touch.
  - ❖ More than a **billion mobile applications** have been sold for these devices in a year.

The Elements of Mobile Design

- Context
- Message
- Look and Feel
- Layout
- Color
- Typography
- Graphics

A good design requires **Three Abilities**:

- **Natural Gift** to see visually how something should look that produces a **Desired Emotion** with the target audience.
- **Ability** to manifest that **Vision** into something for others to see, use, or participate in.
- **Knowledge** how to **Utilize** the medium to achieve your design goals.

1. Context

Context is core to the mobile experience.
As the designer, user can figure out how to address context using your app.

- Who are the users?
- What is happening?
- When will they interact?
- Are they at home and have large amounts of time?
- Are they at work where they have short periods of time?
- Will they have idle periods of time while waiting for a train, for example?
- Where are the users?
- Are they in a public space or a private space?
- Are they inside or outside?
- Is it day or is it night?
- Why will they use your app?
- What value will they gain from your content or services in their present situation?

How are they using their mobile device?
Is it held in their hand or in their pocket?
How are they holding it?
Open or closed?
Portrait or Landscape?

63

2. Message

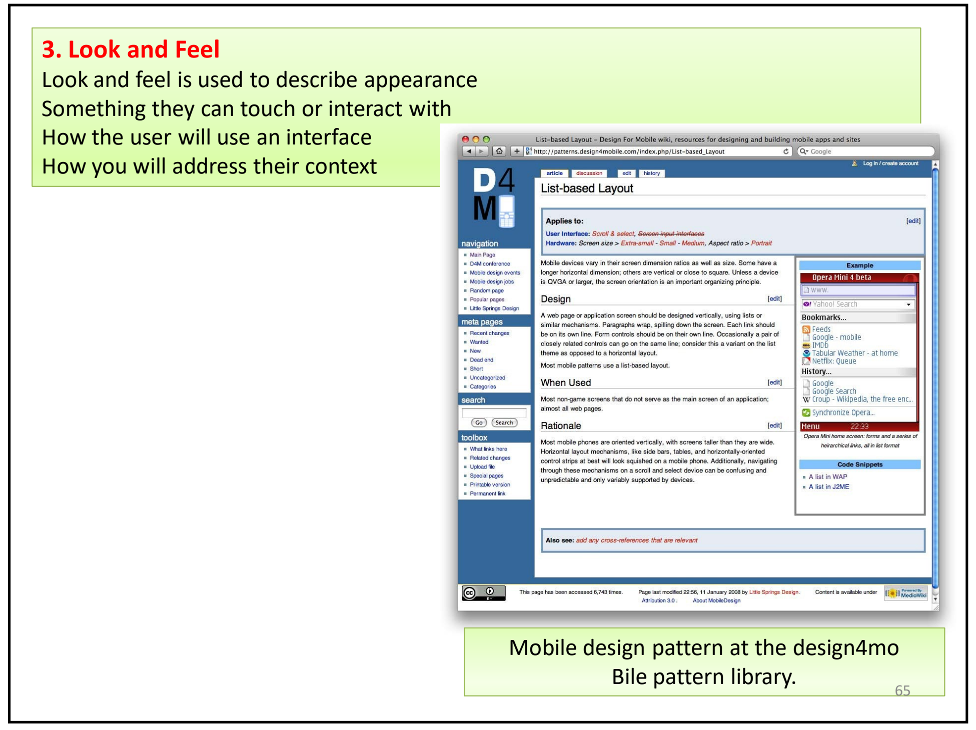
Message is the **Overall Mental Impression** designer create explicitly through visual design

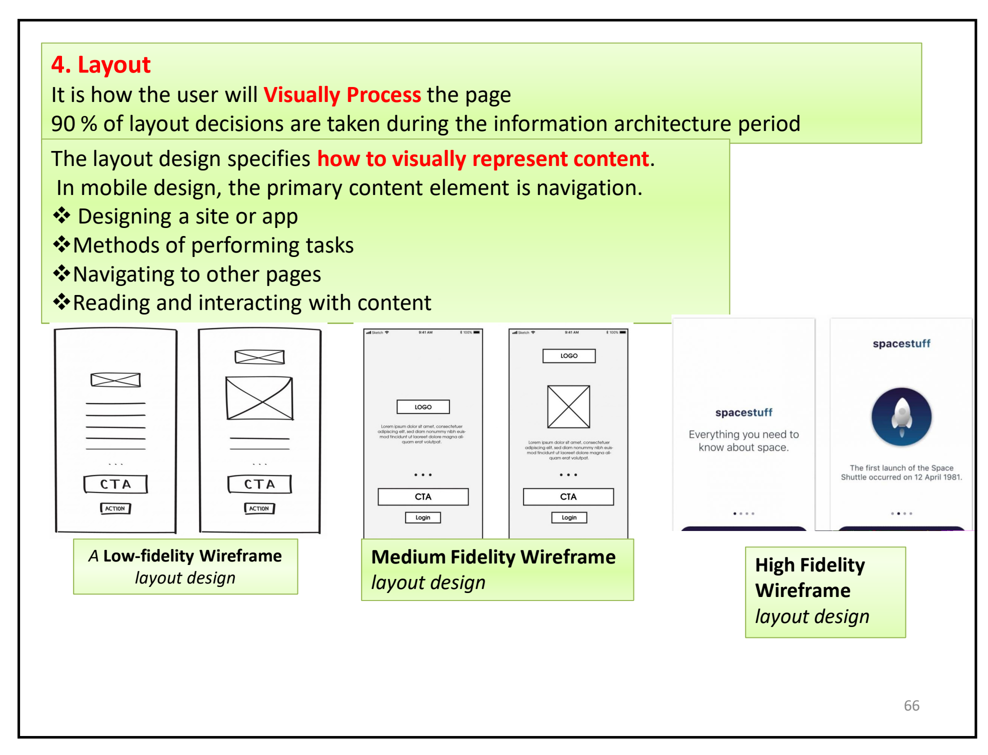
The approach to the design will define that message and create expectations.
Sparse and simple design with **lots of whitespace -** the user to expect a focus on content
H**eavy design** with dark colors and lots of graphics will tell the user to expect something more immersive.

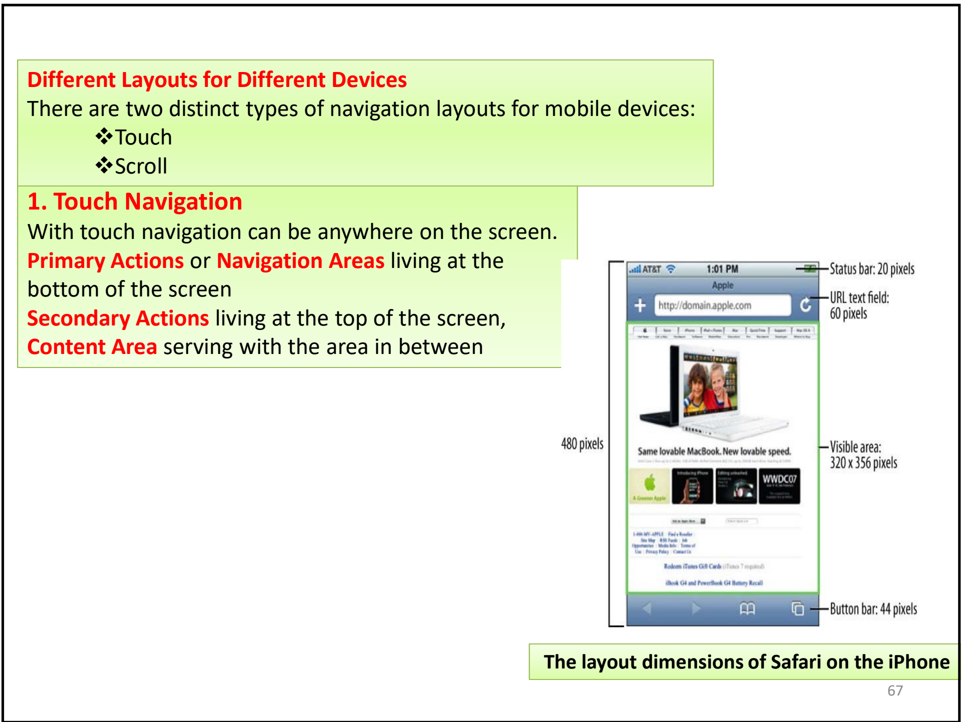
Yahoo! *ESPN* *Disney* *Wikipedia* *Amazon*

- ***Yahoo!:*** Hard, Clean, and Sharp
- ***ESPN:*** Bold, Disorderly, and Content-heavy
- ***Disney:*** Bold, Busy, and Disorienting.
- ***Wikipedia:*** Clean, Minimal, and Text-heavy
- **Amazon:** Minimal But Messy, Product-heavy, and Disorienting

64


3. Look and Feel
Look and feel is used to describe appearance
Something they can touch or interact with
How the user will use an interface
How you will address their context
List-based Layout
Mobile design pattern at the design4mo
Bile pattern library.
65


4. Layout
It is how the user will Visually Process the page
90 % of layout decisions are taken during the information architecture period
The layout design specifies how to visually represent content.
In mobile design, the primary content element is navigation.
❖ Designing a site or app
❖Methods of performing tasks
❖Navigating to other pages
❖Reading and interacting with content
CTA
ACTION
LOGO
Login
spacestuff
Everything you need to know about space.
The first launch of the Space Shuttle occurred on 12 April 1981.
A Low-fidelity Wireframe layout design
Medium Fidelity Wireframe layout design
High Fidelity Wireframe layout design
66



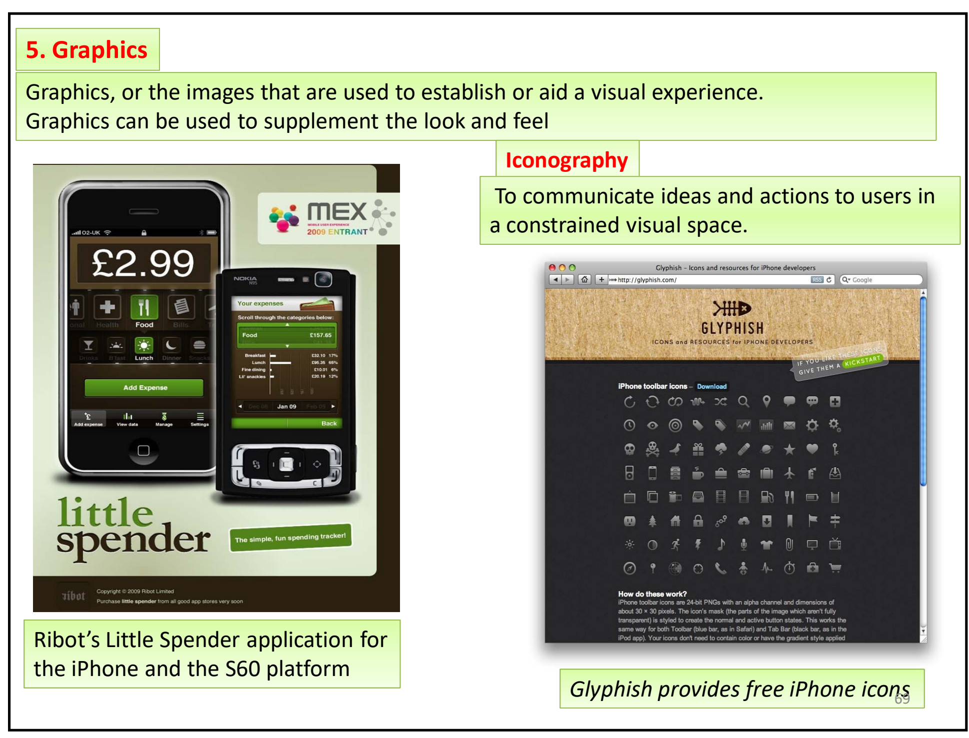
2. Scroll Navigation

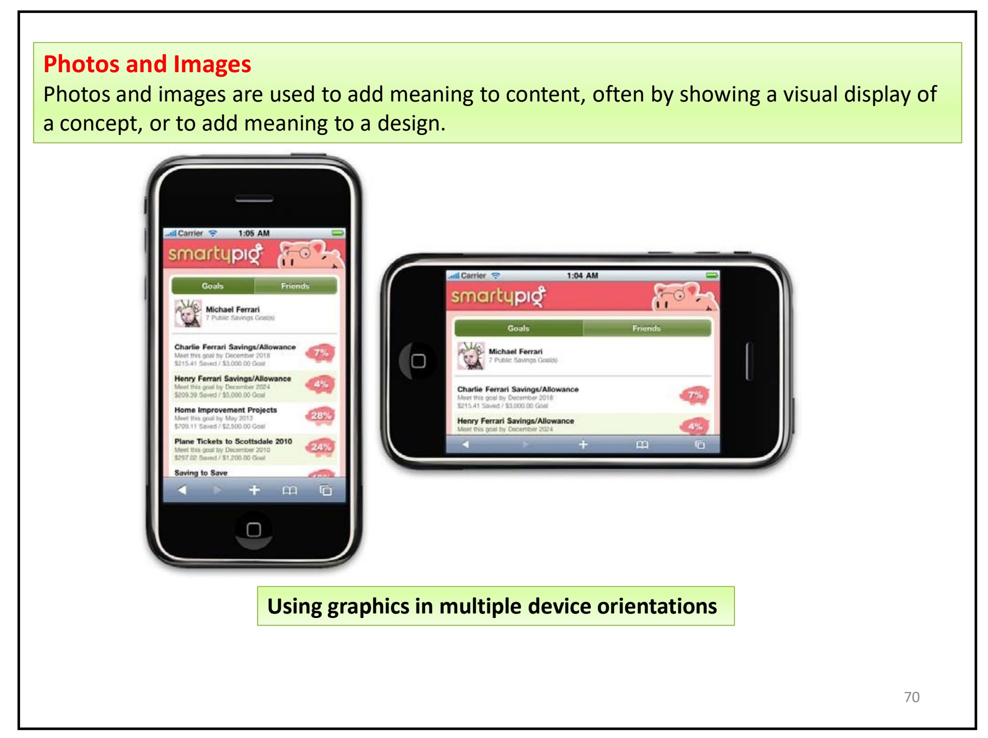
D-pad is used to go left, right, up, or down. In this, the primary and the secondary actions should live at the top of the screen.
Scroll navigation

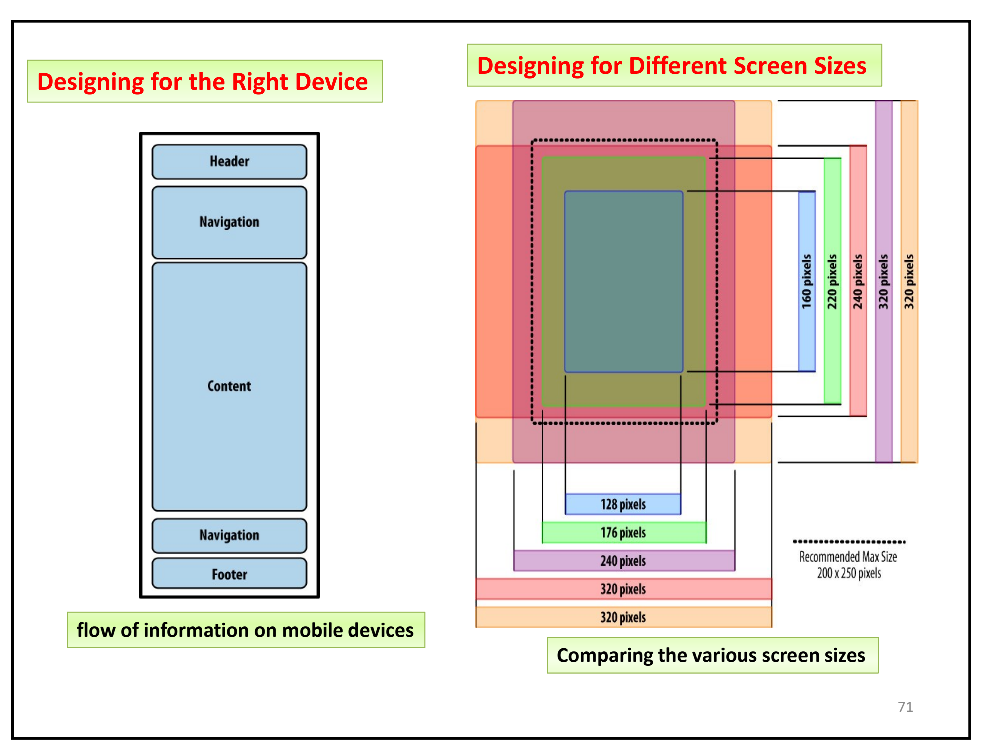
Fixed versus fluid Layout

Design will scale as the device orientation changes i.e., device is rotated from portrait mode to landscape and vice versa.
Fixed -A set number of pixels wide
Fluid -having the ability to scale to the full width of the screen regardless of the device orientation.

68


5. Graphics
Graphics, or the images that are used to establish or aid a visual experience.
Graphics can be used to supplement the look and feel
Iconography
To communicate ideas and actions to users in a constrained visual space.
£2.99
Add Expense
MEX
little spender
The simple, fun spending tracker!
GLYPHISH
iPhone toolbar icons
Ribot's Little Spender application for the iPhone and the S60 platform
Glyphish provides free iPhone icons
69


Photos and Images
Photos and images are used to add meaning to content, often by showing a visual display of a concept, or to add meaning to a design.
smartypig
Goals
Friends
Michael Ferrari
Charlie Ferrari Savings/Allowance
Henry Ferrari Savings/Allowance
Home Improvement Projects
Plane Tickets to Scottsdale 2010
Saving to Save
Using graphics in multiple device orientations
70



Mobile Design Tools

Mobile Framework	Design Tool	Interface Toolkits
Java ME	Photoshop, NetBeans	JavaFX, Capuchin
BREW	Photoshop, Flash	BREW UI Toolkit, uiOne, Flash
Flash Lite	Flash	Flash Lite
iPhone	Photoshop, Interface Builder	iPhone SDK
Android	Photoshop, XML-based themes	Android SDK
Palm webOS	Photoshop, HTML, CSS, and JavaScript	Mojo SDK
Mobile web	Photoshop, HTML, CSS, and JavaScript	W3C Mobile Web Best Practices
Mobile widgets	Photoshop, HTML, CSS, and JavaScript	Opera Widget SDK, Nokia Web Runtime
Mobile web apps	Photoshop, HTML, CSS, and JavaScript	iUI, jQTouch, W3C Mobile Web App Best Practices

72

Small Computing Device Requirements

Hardware Requirements

- Display 96 X 54 pixels should support the bitmapped pixels.
- Keypad, keyboard, or touch screen for input
- ROM - 128KB to run the mobile information device (MID)
- 8KB for permanent storage like application data.
- Ram - 32kb to run JVM
- Device must provide the two-way network connectivity (SIM)

Software Requirements

- Native Operating System (Supported System)
- It must include the Exception Handling, Scheduling, Process Interrupts, ability to run the JVM.
- The file system is not required to run the JVM.
- Able to read and write permanent data onto the nonvolatile memory.

Module-II

Introduction to Android:

History of Mobile Software Development, The Open Handset Alliance, Android platform differences.

Android Installation:

The Android Platform, Android SDK, Eclipse Installation, Android Installation, Building a Sample Android Application.

Introduction to Android

A Brief History of Mobile Software Development

To understand what makes Android so compelling, we must examine how mobile development has evolved and how Android differs from competing platforms.

Way Back When

Remember way back when a phone was just a phone? When we relied on fixed landlines? When we ran for the phone instead of pulling it out of our pocket? When we lost our friends at a crowded ballgame and waited around for hours hoping to reunite? When we forgot the grocery list (Figure 1.1) and had to find a payphone or drive back home again?

Figure 1.1 Mobile phones have become a crucial shopping accessory.

Those days are long gone. Today, commonplace problems like these are easily solved with a one-button speed dial or a simple text message like "WRU?" or "20?" or "Milk and?"

Our mobile phones keep us safe and connected. Nowadays, we roam around freely, relying on our phones not only to keep in touch with friends, family, and coworkers, but also to tell us where to go, what to do, and how to do it. Even the most domestic of events seem to revolve around my mobile phone.

Consider the following true, but slightly enhanced for effect, story:

- *Once upon a time, on a warm summer evening, I was happily minding my own business cooking dinner in my new house in rural New Hampshire when a bat swooped over my head, scaring me to death.*
- *The first thing I did—while ducking—was pull out my cell and send a text message to my husband, who was across the country at the time: "There's a bat in the house!"*
- *My husband did not immediately respond (a divorce-worthy incident, I thought at the time), so I called my Dad and asked him for suggestions on how to get rid of the bat.*
- *He just laughed.*
- *Annoyed, I snapped a picture of the bat with my phone and sent it to my husband and my blog, simultaneously guilt-tripping him and informing the world of my treacherous domestic wildlife encounter.*
- *Finally, I Googled "get rid of a bat" and followed the helpful do-it-yourself instructions provided on the Web for people in my situation. I also learned that late August is when baby bats often leave the roost for the first time and learn to fly. Newly aware that I had a baby bat on my hands, I calmly got a broom and managed to herd the bat out of the house.*

- *Problem solved—and I did it all with the help of my trusty cell phone, the old LG VX9800.*

My point here? Mobile phones can solve just about *anything*—and we rely on them for *everything* these days.

You notice that I used half a dozen different mobile applications over the course of this story. Each application was developed by a different company and had a different user interface. Some were well designed; others not so much. I paid for some of the applications, and others came on my phone.

As a user, I found the experience functional, but not terribly inspiring. As a mobile developer, I wished for an opportunity to create a more seamless and powerful application that could handle all I'd done and more. I wanted to build a better bat trap, if you will.

Before Android, mobile developers faced many roadblocks when it came to writing applications. Building the better application, the unique application, the competing application, the hybrid application, and incorporating many common tasks such as messaging and calling in a familiar way were often unrealistic goals.

To understand why, let's take a brief look at the history of mobile software development.

"The Brick"

The Motorola DynaTAC 8000X was the first commercially available cell phone. First marketed in 1983, it was 13 x 1.75 x 3.5 inches in dimension, weighed about 2.5 pounds, and allowed you to talk for a little more than half an hour. It retailed for $3,995, plus hefty monthly service fees and per-minute charges.

We called it "The Brick," and the nickname stuck for many of those early mobile phones we alternatively loved and hated. About the size of a brick, with a battery power just long enough for half a conversation, these early mobile handsets were mostly seen in the hands of traveling business execs, security personnel, and the wealthy. First-generation mobile phones were just too expensive. The service charges alone would bankrupt the average person, especially when roaming.

Early mobile phones were not particularly full featured. (Although, even the Motorola DynaTAC, shown in Figure 1.2, had many of the buttons we've come to know well, such as the SEND, END, and CLR buttons.) These early phones did little more than make and receive calls and, if you were lucky, there was a simple contacts application that wasn't impossible to use.

Figure 1.2 The first commercially available mobile phone: the Motorala DynaTAC.

These first-generation mobile phones were designed and developed by the handset manufacturers. Competition was fierce and trade secrets were closely guarded. Manufacturers didn't want to expose the internal workings of their handsets, so they usually developed the phone software in-house. As a developer, if you weren't part of this inner circle, you had no opportunity to write applications for the phones.

It was during this period that we saw the first "time-waster" games begin to appear. Nokia was famous for putting the 1970s video game *Snake* on some of its earliest monochrome phones. Other manufacturers followed, adding games like Pong, Tetris, and Tic-Tac-Toe.

These early phones were flawed, but they did something important—they changed the way people thought about communication. As mobile phone prices dropped, batteries improved,

and reception areas grew, more and more people began carrying these handy devices. Soon mobile phones were more than just a novelty.

Customers began pushing for more features and more games. But, there was a problem. The handset manufacturers didn't have the motivation or the resources to build every application users wanted. They needed some way to provide a portal for entertainment and information services without allowing direct access to the handset.

And what better way to provide these services than the Internet?

Wireless Application Protocol (WAP)

It turned out allowing direct phone access to the Internet didn't scale well for mobile.

By this time, professional Web sites were full color and chock full of text, images, and other sorts of media. These sites relied on JavaScript, Flash, and other technologies to enhance the user experience and were often designed with a target resolution of 800×600 pixels and higher.

When the first clamshell phone, the Motorola StarTAC, was released in 1996, it merely had a LCD 10-digit segmented display. (Later models would add a dot-matrix type display.) Meanwhile, Nokia released one of the first slider phones, the 8110—fondly referred to as "The Matrix Phone," as the phone was heavily used in films. The 8110 could display four lines of text with 13 characters per line. Figure 1.3 shows some of the common phone form factors.

Figure 1.3 Various mobile phone form factors: the candy bar, the slider, and the clamshell.

With their postage stamp-sized low-resolution screens and limited storage and processing power, these phones couldn't handle the data-intensive operations required by traditional Web browsers. The bandwidth requirements for data transmission were also costly to the user.

The Wireless Application Protocol (WAP) standard emerged to address these concerns. Simply put, WAP was a stripped-down version of HTTP, which is the backbone protocol of the Internet. Unlike traditional Web browsers, WAP browsers were designed to run within the memory and bandwidth constraints of the phone. Third-party WAP sites served up pages written in a markup language called Wireless Markup Language (WML). These pages were then displayed on the phone's WAP browser. Users navigated as they would on the Web, but the pages were much simpler in design.

The WAP solution was great for handset manufacturers. The pressure was off—they could write one WAP browser to ship with the handset and rely on developers to come up with the content users wanted.

The WAP solution was great for mobile operators. They could provide a custom WAP portal, directing their subscribers to the content they wanted to provide, and rake in the data charges associated with browsing, which were often high.

Developers and content providers didn't deliver. For the first time, developers had a chance to develop content for phone users, and some did so, with limited success.

Most of the early WAP sites were extensions of popular branded Web sites, such as CNN.com and ESPN.com, looking for new ways to extend their readership. Suddenly phone users accessed the news, stock market quotes, and sports scores on their phones.

Commercializing WAP applications was difficult, and there was no built-in billing mechanism. Some of the most popular commercial WAP applications that emerged during this time were simple wallpaper and ringtone catalogues, allowing users to personalize their phones for the first time. For example, the users browsed a WAP site and requested a specific item. They filled out a simple order form with their phone number and their handset model. It

was up to the content provider to deliver an image or audio file compatible with the given phone. Payment and verification were handled through various premium-priced delivery mechanisms such as Short Message Service (SMS), Enhanced Messaging Service (EMS), Multimedia Messaging Service (MMS), and WAP Push.

WAP browsers, especially in the early days, were slow and frustrating. Typing long URLs with the numeric keypad was onerous. WAP pages were often difficult to navigate. Most WAP sites were written once for all phones and did not account for individual phone specifications. It didn't matter if the end-user's phone had a big color screen or a postage stamp-sized monochrome one; the developer couldn't tailor the user's experience. The result was a mediocre and not very compelling experience for everyone involved.

Content providers often didn't bother with a WAP site and instead just advertised SMS short codes on TV and in magazines. In this case, the user sent a premium SMS message with a request for a specific wallpaper or ringtone, and the content provider sent it back. Mobile operators generally liked these delivery mechanisms because they received a large portion of each messaging fee.

WAP fell short of commercial expectations. In some markets, such as Japan, it flourished, whereas in others, like the United States, it failed to take off. Handset screens were too small for surfing. Reading a sentence fragment at a time, and then waiting seconds for the next segment to download, ruined the user experience, especially because every second of downloading was often charged to the user. Critics began to call WAP "Wait and Pay."

Finally, the mobile operators who provided the WAP portal (the default home page loaded when you started your WAP browser) often restricted which WAP sites were accessible. The portal allowed the operator to restrict the number of sites users could browse and to funnel subscribers to the operator's preferred content providers and exclude competing sites. This kind of walled garden approach further discouraged third-party developers, who already faced difficulties in monetizing applications, from writing applications.

Proprietary Mobile Platforms

It came as no surprise when users wanted more—they will always want more.

Writing robust applications such as graphic-intensive video games with WAP was nearly impossible. The 18-year-old to 25-year-old sweet-spot demographic—the kids with the disposable income most likely to personalize their phones with wallpapers and ringtones—looked at their portable gaming systems and asked for a device that was both a phone and a gaming device or a phone and a music player. They argued that if devices such as Nintendo's Game Boy could provide hours of entertainment with only five buttons, why not just add phone capabilities? Others looked to their digital cameras, Palms, Blackberries, iPods, and even their laptops and asked the same question. The market seemed to be teetering on the edge of device convergence.

Memory was getting cheaper; batteries were getting better; and PDAs and other embedded devices were beginning to run compact versions of common operating systems such as Linux and Windows. The traditional desktop application developer was suddenly a player in the embedded device market, especially with Smartphone technologies such as Windows Mobile, which they found familiar.

Handset manufacturers realized that if they wanted to continue to sell traditional handsets, they needed to change their protectionist policies pertaining to handset design and expose their internal frameworks, at least, to some extent.

A variety of different proprietary platforms emerged—and developers are still actively creating applications for them. Some Smartphone devices ran Palm OS (now Garnet OS) and RIM Blackberry OS. Sun Microsystems took its popular Java platform and J2ME emerged (now known as Java Micro Edition [Java ME]). Chipset maker Qualcomm developed and licensed its Binary Runtime Environment for Wireless (BREW). Other platforms, such as

Symbian OS, were developed by handset manufacturers such as Nokia, Sony Ericsson, Motorola, and Samsung. The Apple iPhone OS (OS X iPhone) joined the ranks in 2008. Figure 1.4 shows several different phones, all of which have different development platforms.

Figure 1.4 Phones from various mobile device platforms.

Many of these platforms have associated developer programs. These programs keep the developer communities small, vetted, and under contractual agreements on what they can and cannot do. These programs are often required and developers must pay for them.

Each platform has benefits and drawbacks. Of course, developers love to debate over which platform is "the best." (Hint: It's usually the platform we're currently developing for.)

The truth is no one platform has emerged victorious. Some platforms are best suited for commercializing games and making millions—if your company has brand backing. Other platforms are more open and suitable for the hobbyist or vertical market applications. No mobile platform is best suited for all possible applications. As a result, the mobile phone has become increasingly fragmented, with all platforms sharing part of the pie.

For manufacturers and mobile operators, handset product lines became complicated fast. Platform market penetration varies greatly by region and user demographic. Instead of choosing just one platform, manufacturers and operators have been forced to sell phones for all the different platforms to compete. We've even seen some handsets supporting multiple platforms. (For instance, Symbian phones often also support J2ME.)

The mobile developer community has become as fragmented as the market. It's nearly impossible to keep track of all the changes in the market. Developer specialty niches have formed. The platform development requirements vary greatly. Mobile software developers work with distinctly different programming environments, different tools, and different programming languages. Porting among the platforms is often costly and not straightforward. Keeping track of handset configurations and testing requirements, signing and certification programs, carrier relationships, and application marketplaces have become complex spin-off businesses of their own.

It's a nightmare for the ACME Company wanting a mobile application. Should they develop a J2ME application? BREW? iPhone? Windows Mobile? Everyone has a different kind of phone. ACME is forced to choose one or, worse, all of the above. Some platforms allow for free applications, whereas others do not. Vertical market application opportunities are limited and expensive.

As a result, many wonderful applications have not reached their desired users, and many other great ideas have not been developed at all

The Open Handset Alliance

Enter search advertising giant Google. Now a household name, Google has shown an interest in spreading its brand and suite of tools to the wireless marketplace. The company's business model has been amazingly successful on the Internet, and technically speaking, wireless isn't that different.

Google Goes Wireless

The company's initial forays into mobile were beset with all the problems you would expect. The freedoms Internet users enjoyed were not shared by mobile phone subscribers. Internet users can choose from the wide variety of computer brands, operating systems, Internet service providers, and Web browser applications.

Nearly all Google services are free and ad driven. Many applications in the Google Labs suite would directly compete with the applications available on mobile phones. The applications

range from simple calendars and calculators to navigation with Google Maps and the latest tailored news from News Alerts—not to mention corporate acquisitions like Blogger and YouTube.

When this approach didn't yield the intended results, Google decided to a different approach—to revamp the entire system upon which wireless application development was based, hoping to provide a more open environment for users and developers: the Internet model. The Internet model allowes users to choose between freeware, shareware, and paid software. This enables free market competition among services.

Forming of the Open Handset Alliance

With its user-centric, democratic design philosophies, Google has led a movement to turn the existing closely guarded wireless market into one where phone users can move between carriers easily and have unfettered access to applications and services. With its vast resources, Google has taken a broad approach, examining the wireless infrastructure from the FCC wireless spectrum policies to the handset manufacturers' requirements, application developer needs, and mobile operator desires.

Next, Google joined with other like-minded members in the wireless community and posed the following question: What would it take to build a better mobile phone?

The Open Handset Alliance (OHA) (Figure 1.5) was formed in November 2007 to answer that very question. The OHA is a business alliance comprised of many of the largest and most successful mobile companies on the planet. Its members include chip makers, handset manufacturers, software developers, and service providers. The entire mobile supply chain is well represented.

open handset alliance

Figure 1.5 The Open Handset Alliance.

Working together, OHA members began developing a nonproprietary open standard platform that would aim to alleviate the aforementioned problems hindering the mobile community. They called it the Android project.

Google's involvement in the Android project has been extensive. The company hosts the open source project and provides online documentation, tools, forums, and the Software Development Kit (SDK). Google has also hosted a number of events at conferences and the Android Developer Challenge, a contest to encourage developers to write killer Android applications—for $10 million dollars in prizes.

Manufacturers: Designing the Android Handsets

More than half the members of the OHA are handset manufacturers, such as Samsung, Motorola, HTC, and LG, and semiconductor companies, such as Intel, Texas Instruments, NVIDIA, and Qualcomm. These companies are helping design the first generation of Android handsets.

The first shipping Android handset—the T-Mobile G1—was developed by handset manufacturer HTC with service provided by T-Mobile. It was released in October 2008. Many other Android handsets are slated for 2009 and early 2010.

Content Providers: Developing Android Applications

When users have Android handsets, they need those killer apps, right?

Google has led the pack, developing Android applications, many of which, like the email client and Web browser, are core features of the platform. OHA members, such as eBay, are also working on Android application integration with their online auctions.

The first Android Developer Challenge received 1,788 submissions—all newly developed Android games, productivity helpers, and a slew of Location-Based Services (LBS). We also saw humanitarian, social networking, and mash-up apps. Many of these applications have debuted with users through the Android Market—Google's software distribution mechanism for Android.

Mobile Operators: Delivering the Android Experience

After you have the phones, you have to get them out to the users. Mobile operators from Asia, North America, Europe, and Latin America have joined the OHA, ensuring a market for the Android movement. With almost half a billion subscribers, telephony giant China Mobile is a founding member of the alliance. Other operators have signed on as well.

Taking Advantage of All Android Has to Offer

Android's open platform has been embraced by much of the mobile development community—extending far beyond the members of the OHA.

As Android phones and applications become more readily available, many in the tech community anticipate other mobile operators and handset manufacturers will jump on the chance to sell Android phones to their subscribers, especially given the cost benefits compared to proprietary platforms. Already, North American operators, such as Verizon Wireless and AT&T, have shown an interest in Android, and T-Mobile already provides handsets.

If the open standard of the Android platform results in reduced operator costs in licensing and royalties, we could see a migration to open handsets from proprietary platforms such as BREW, Windows Mobile, and even the Apple iPhone. Android is well suited to fill this demand

Android Platform Differences

Android is hailed as "the first complete, open, and free mobile platform."

- **Complete:** The designers took a comprehensive approach when they developed the Android platform. They began with a secure operating system and built a robust software framework on top that allows for rich application development opportunities.
- **Open:** The Android platform is provided through open source licensing. Developers have unprecedented access to the handset features when developing applications.
- **Free:** Android applications are free to develop. There are no licensing or royalty fees to develop on the platform. No required membership fees. No required testing fees. No required signing or certification fees. Android applications can be distributed and commercialized in a variety of ways.

Android: A Next Generation Platform

Although Android has many innovative features not available in existing mobile platforms, its designers also leveraged many tried-and-true approaches proven to work in the wireless world. It's true that many of these features appear in existing proprietary platforms, but Android combines them in a free and open fashion, while simultaneously addressing many of the flaws on these competing platforms.

The Android mascot is a little green robot, shown in Figure 1.6. You'll see this little guy (girl?) often used to depict Android-related materials.

Figure 1.6 The Android mascot.

Android is the first in a new generation of mobile platforms, giving its platform developers a distinct edge on the competition. Android's designers examined the benefits and drawbacks of existing platforms and then incorporate their most successful features. At the same time, Android's designers avoided the mistakes others suffered in the past.

Free and Open Source

Android is an open source platform. Neither developers nor handset manufacturers pay royalties or license fees to develop for the platform.

The underlying operating system of Android is licensed under GNU General Public License Version 2 (GPLv2), a strong “copyleft” license where any third-party improvements must continue to fall under the open source licensing agreement terms. The Android framework is distributed under the Apache Software License (ASL/Apache2), which allows for the distribution of both open and closed source derivations of the source code. Commercial developers (handset manufacturers especially) can choose to enhance the platform without having to provide their improvements to the open source community. Instead, developers can profit from enhancements such as handset-specific improvements and redistribute their work under whatever licensing they want.

Android application developers have the ability to distribute their applications under whatever licensing scheme they prefer. Developers can write open source freeware or traditional licensed applications for profit and everything in between.

Familiar and Inexpensive Development Tools

Unlike some proprietary platforms that require developer registration fees, vetting, and expensive compilers, there are no upfront costs to developing Android applications.

Freely Available Software Development Kit

The Android SDK and tools are freely available. Developers can download the Android SDK from the Android Web site after agreeing to the terms of the Android Software Development Kit License Agreement.

Familiar Language, Familiar Development Environments

Developers have several choices when it comes to integrated development environments (IDEs). Many developers choose the popular and freely available Eclipse IDE to design and develop Android applications. Eclipse is the most popular IDE for Android development and there is an Android plug-in available for facilitating Android development. Android applications can be developed on the following operating systems:

- Windows XP or Vista
- Mac OS X 10.4.8 or later (x86 only)
- Linux (tested on Linux Ubuntu 6.06 LTS, Dapper Drake)

Reasonable Learning Curve for Developers

Android applications are written in a well-respected programming language: Java.

The Android application framework includes traditional programming constructs, such as threads and processes and specially designed data structures to encapsulate objects commonly used in mobile applications. Developers can rely on familiar class libraries, such as java.net and java.text. Specialty libraries for tasks like graphics and database management are implemented using well-defined open standards like OpenGL Embedded Systems (OpenGL ES) or SQLite.

Enabling Development of Powerful Applications

In the past, handset manufacturers often established special relationships with trusted third-party software developers (OEM/ODM relationships). This elite group of software developers wrote native applications, such as messaging and Web browsers, which shipped on the handset as part of the phone’s core feature set. To design these applications, the manufacturer would grant the developer privileged inside access and knowledge of a handset’s internal software framework and firmware.

On the Android platform, there is no distinction between native and third-party applications, enabling healthy competition among application developers. All Android applications use the same libraries. Android applications have unprecedented access to the underlying hardware, allowing developers to write much more powerful applications. Applications can be extended or replaced altogether. For example, Android developers are now free to design email clients tailored to specific email servers such as Microsoft Exchange or Lotus Notes.

Rich, Secure Application Integration

If you recall the bat story I previously shared, you'll note that I accessed a wide variety of phone applications in the course of a few moments: text messaging, phone dialer, camera, email, picture messaging, and the browser. Each was a separate application running on the phone—some built-in and some purchased. Each had its own unique user interface. None were truly integrated.

Not so with Android. One of the Android platform's most compelling and innovative features is well-designed application integration. Android provides all the tools necessary to build a better "bat trap," if you will, by allowing developers to write applications that leverage core functionality such as Web browsing, mapping, contact management, and messaging seamlessly. Applications can also become content providers and share their data among each other in a secure fashion.

Platforms like Symbian have suffered from setbacks due to malware. Android's vigorous application security model helps protect the user and the system from malicious software.

No Costly Obstacles to Publication

Android applications have none of the costly and time-intensive testing and certification programs required by other platforms such as BREW and Symbian.

A "Free Market" for Applications

Android developers are free to choose any kind of revenue model they want. They can develop freeware, shareware, or trial-ware applications, ad-driven, and paid applications. Android was designed to fundamentally change the rules about what kind of wireless applications could be developed. In the past, developers faced many restrictions that had little to do with the application functionality or features:

- Store limitations on the number of competing applications of a given type
- Store limitations on pricing, revenue models, and royalties
- Operator unwillingness to provide applications for smaller demographics

With Android, developers can write and successfully publish any kind of application they want. Developers can tailor applications to small demographics, instead of just large-scale money-making ones often insisted upon by mobile operators. Vertical market applications can be deployed to specific, targeted users.

Because developers have a variety of application distribution mechanisms to choose from, they can pick the methods that work for them instead of being forced to play by others' rules. Android developers can distribute their applications to users in a variety of ways.

- Google developed the Android Market (Figure 1.7), a generic Android application store with a revenue-sharing model.

Figure 1.7 The Android market.

- Handango.com added Android applications to its existing catalogue using their billing models and revenue sharing model.
- Developers can come up with their own delivery and payment mechanisms.

Mobile operators are still free to develop their own application stores and enforce their own rules, but it will no longer be the only opportunity developers have to distribute their applications.

Android might be the next generation in mobile platforms, but the technology is still in its early stages. Early Android developers have had to deal with the typical roadblocks associated with a new platform: frequently revised SDKs, lack of good documentation, and

market uncertainties. There are only a handful of Android handsets available to consumers at this time.

On the other hand, developers diving into Android development now benefit from the first-to-market competitive advantages we've seen on other platforms such as BREW and Symbian. Early developers who give feedback are more likely to have an impact on the long-term design of the Android platform and what features will come in the next version of the SDK. Finally, the Android forum community is lively and friendly. Incentive programs, such as the Android Developer Challenge, have encouraged many new developers to dig into the platform.

A New and Growing Platform

What's New in Android 1.5

The much-anticipated Android 1.5 SDK, released in late April 2009, provided a number of substantial improvements to both the underlying software libraries and the Android development tools and build environment. Also, the Android system received some much-needed UI "polish," both in terms of visual appeal and performance.

Although most of these upgrades and improvements were welcome and necessary, the new SDK version did cause some upheaval within the Android developer community. A number of published applications required retesting and resubmission to the Android Marketplace to conform to the new SDK requirements, which were quickly rolled out to all Android phones in the field as a firmware upgrade, rendering older applications obsolete

The Android Platform

Android is an operating system and a software platform upon which applications are developed. A core set of applications for everyday tasks, such as Web browsing and email, are included on Android handsets.

As a product of the Open Handset Alliance's vision for a robust and open source development environment for wireless, Android is an emerging mobile development platform. The platform was designed for the sole purpose of encouraging a free and open market that all mobile applications phone users might want to have and software developers might want to develop.

Android's Underlying Architecture

The Android platform is designed to be more fault-tolerant than many of its predecessors. The handset runs a Linux operating system, upon which Android applications are executed in a secure fashion. Each Android application runs in its own virtual machine (Figure 1.8). Android applications are managed code; therefore, they are much less likely to cause the phone to crash, leading to fewer instances of device corruption (also called "bricking" the phone, or rendering it useless).

Figure 1.8 Diagram of the Android platform architecture.

The Linux Operating System

The Linux 2.6 kernel (Figure 1.9) handles core system services and acts as a hardware abstraction layer (HAL) between the physical hardware of the handset and the Android software stack.

Figure 1.9 Tux, the Linux kernel mascot.

What's New in Android 1.5

For Android 1.5, the Linux kernel received an upgrade from version 2.6.25 to 2.6.27. Although this type of change might not have an obvious effect for the typical Android developer, it is important to note that the kernel can and will be upgraded frequently. These seemingly minor incremental updates often include major security, performance, and functional features.

Kernel changes often have an impact on the security of the underlying device operating system and provide features and improvements for OEM-level Android device manufacturers. When stable, these features can be exposed to developers as part of an Android SDK upgrade, in the form of new APIs and performance enhancements to existing features.

The Android 1.5 version provides substantial feature enhancements, many of which tie back to features of the upgraded Linux kernel. Although the kernel memory footprint is larger, overall system performance has improved and a number of bugs have been fixed.

Some of the core functions the kernel handles include

- Enforcement of application permissions and security
- Low-level memory management
- Process management and threading
- The network stack
- Display, keypad input, camera, WiFi, Flash memory, audio, and binder (IPC) driver access

Android Application Runtime Environment

Each Android application runs in a separate process, with its own instance of the Dalvik virtual machine (VM). Based on the Java VM, the Dalvik design has been optimized for mobile devices. The Dalvik VM has a small memory footprint and multiple instances of the Dalvik VM can run concurrently on the handset.

Security and Permissions

The integrity of the Android platform is maintained through a variety of security measures.

Applications as Operating System Users

When an application is installed, the operating system creates a new user profile associated with the application. Each application runs as a different user, with its own private files on the file system, a user ID, and a secure operating environment.

The application executes in its own process with its own instance of the Dalvik VM and under its own user ID on the operating system.

Explicitly Defined Application Permissions

To access shared resources on the system, Android applications register for the specific privileges they require. Some of these privileges enable the application to use phone functionality to make calls, access the network, and control the camera and other hardware sensors. Applications also require permission to access shared data containing private and personal information such as user preferences, user's location, and contact information.

Applications might also enforce their own permissions by declaring them for other applications to use. The application can declare any number of different permission types, such as read-only or read-write permissions, for finer control over the application.

Limited Ad-Hoc Permissions

Applications that act as content providers might want to provide some on-the-fly permissions to other applications for specific information they want to share openly. This is done using ad-hoc granting and revoking of access to specific resources using Uniform Resource Identifiers (URIs).

URIs index specific data assets on the system, such as images and text. Here is an example of a URI that provides the phone numbers of all contacts:

```
content://contacts/phones
```

To understand how this permission process works, let's look at an example.

Let's say we've got an application that keeps track of the user's public and private birthday wish lists. If this application wanted to share its data with other applications, it could grant URI permissions for the public wish list, allowing another application permission to access this list without explicitly having to ask for it.

Application Signing for Trust Relationships

All Android applications packages are signed with a certificate, so users know that the application is authentic. The private key for the certificate is held by the developer. This helps establish a trust relationship between the developer and the user. It also allows the developer to control which applications can grant access to one another on the system. No certificate authority is necessary; self-signed certificates are acceptable.

Developing Android Applications

The Android SDK provides an extensive set of application programming interfaces (APIs) that is both modern and robust. Android handset core system services are exposed and accessible to all applications. When granted the appropriate permissions, Android applications can share data among one another and access shared resources on the system securely.

Android Programming Language Choices

Android applications are written in Java (Figure 1.10). For now, the Java language is the developer's only choice on the Android platform. There has been some speculation that other programming languages, such as C++, might be added in future versions of Android.

Figure 1.10 Duke, the Java mascot.

No Distinctions Made Between Native and Third-Party Applications

Unlike other mobile development platforms, there is no distinction between native applications and developer-created applications on the Android platform. Provided the application is granted the appropriate permissions, all applications have the same access to core libraries and the underlying hardware interfaces.

Android handsets ship with a set of native applications such as a Web browser and contact manager. Third-party applications might integrate with these core applications and even extend them to provide a rich user experience.

Commonly Used Packages

With Android, mobile developers no longer have to reinvent the wheel. Instead, developers use familiar class libraries exposed through Android's Java packages to perform common tasks such as graphics, database access, network access, secure communications, and utilities (such as XML parsing).

The Android packages include support for

- Common user interface widgets (Buttons, Spin Controls, Text Input)
- User interface layout
- Secure networking and Web browsing features (SSL, WebKit)
- Structured storage and relational databases (SQLite)
- Powerful 2D and 3D graphics (SGL and OpenGL ES 1.0)
- Audio and visual media formats (MPEG4, MP3, Still Images)
- Access to optional hardware such as Location-Based Services (LBS), WiFi, and Bluetooth

Android Application Framework

The Android application framework provides everything necessary to implement your average application. The Android application lifecycle involves the following key components:

- Activities are functions the application performs.
- Groups of views define the application's layout.
- Intents inform the system about an application's plans.
- Services allow for background processing without user interaction.
- Notifications alert the user when something interesting happens.

Android Applications can interact with the operating system and underlying hardware using a collection of managers. Each manager is responsible for keeping the state of some underlying system service. For example, there is a LocationManager that facilitates interaction with the location-based services available on the handset.
The ViewManager and WindowManager manage user interface fundamentals.

Applications can interact with one another by using or acting as a ContentProvider. Built-in applications such as the Contact manager are content providers, allowing third-party applications to access contact data and use it in an infinite number of ways. The sky is the limit

Module-II:

Introduction to Android: History of Mobile Software Development, The Open Handset Alliance, Android platform differences.

Android Installation: The Android Platform, Android SDK, Eclipse Installation, Android Installation, Building a Sample Android Application.

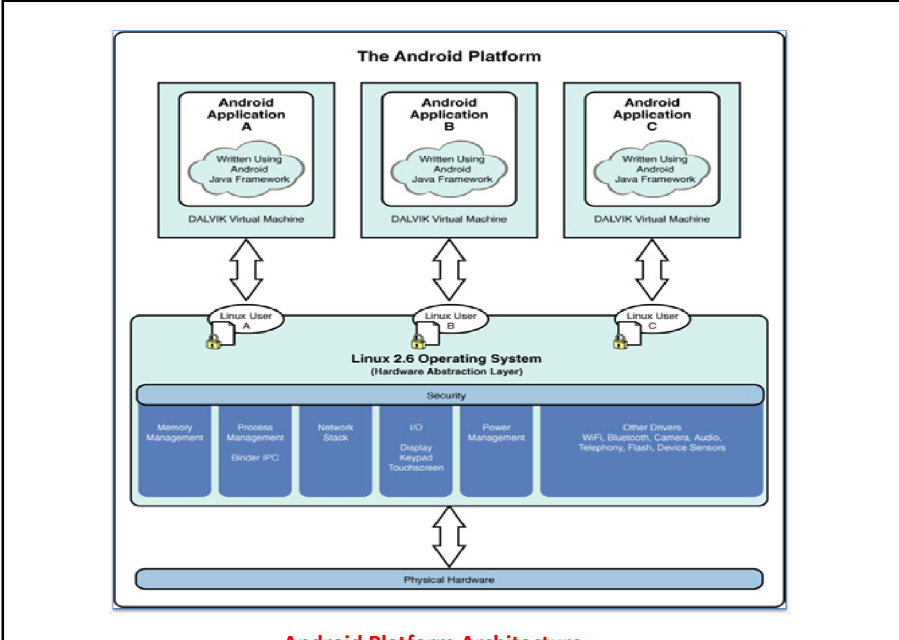
Android Installation

The Android Platform

- ❑Android is an OS and a software platform upon which applications are developed.
- ❑A core set of applications for everyday tasks, such as Web browsing and email, are included on Android handsets.
- ❑As a **product of the Open Handset Alliance's** vision for a robust and open source development environment for wireless, Android is an **emerging mobile development platform.**
- ❑The platform was designed for the sole purpose of encouraging a free and open market

Android's Architecture

- ➢ The Android platform is designed to be more **fault-tolerant** than many of its predecessors.
- ➢ The handset runs a **Linux operating system**, upon which Android applications are executed in a secure fashion.
- ➢ Each Android application runs in its own **virtual machine**.
- ➢ Android applications are managed code i.e., they are much less likely to cause the phone to **crash.**



Android Platform Architecture

Key components of the Android Platform Architecture

A. The Linux Operating System

B. Android Application Runtime Environment

C. Security and Permissions

- Applications as Operating System Users
- Define Application Permissions Explicitly
- Limited Ad-Hoc Permissions
- Application Signing for Trust Relationships

D. Developing Android Applications

- Android Programming Language Choices
- No Distinctions Made Between Native and Third-Party Applications
- Commonly Used Packages
- Android Application Framework

A. The Linux Operating System

The Linux kernel handles **Core System Services** and acts as a **Hardware Abstraction Layer** (HAL) between the **Physical Hardware** of the handset and the **Android Software Stack**.

Functions of the Linux Kernel :

1. Enforcement of application permissions and security
2. Low-level Memory Management
3. Process Management and Threading
4. The Network Stack
5. Display, Keypad Input, Camera, Wifi, Flash Memory, Audio, and Binder (IPC) Driver Access

B. Android Application Runtime Environment

- The **Dalvik VM** has a small unit of memory and **Multiple Instances** can run concurrently.
- Each Android application runs in a separate process, with its own instance of the **Dalvik Virtual Machine (VM).**
- Based on the **Java VM**, the **Dalvik** design has been **optimized** for mobile devices.

C. Security and Permissions

1. Applications as Operating System Users

- When an application is **installed**, the OS **creates User Profile** associated with the application.
- Each application runs as a **different user**, with **own private files** , a user ID, and A **Secure Operating Environment**.
- The application executes in its **Own Process** with its own instance of the Dalvik VM and under its own user ID on the operating system.

2. Define Application Permissions Explicitly

- To **access shared resources** applications register for the **Specific Privileges** they require.
- These **privileges enable** the application to use **phone functionality**, to **make calls**, **access the network**, and **control the camera** and other hardware sensors.
- Applications also **require permission** to access **private** and **personal** information such as **user preferences**, **user's location**, and **contact information**.
- Applications might also enforce their **own permissions** by declaring them for other applications to use.
- The application can declare **read-only or read-write permissions** for finer control over the application.

3. Limited Ad-Hoc Permissions

- Applications that act as **Content Providers** might want to provide some on-the-fly permissions to other applications for specific information they want to share openly.
- This is done using ad-hoc granting and revoking of access to specific resources using **Uniform Resource Identifiers** (URIs).
- URIs index specific data assets on the system, such as images and text.

Example of a URI that provides the phone numbers of all contacts:

content://contacts/phones

4.Application Signing for Trust Relationships

- All applications pkgs are signed with a **certificate**, so that the application is authentic.
- The **Private Key** for the certificate is held by the **Developer**.
- It establish a **Trust Relationship** between the developer and the user.
- It also allows the developer to **Control** which applications can grant access to one another on the system.
- No certificate authority is necessary; self-signed certificates are acceptable.

D. Developing Android Applications

The Android SDK provides a **set of Application Programming Interfaces** (APIs).
Android handset **Core System Services** are exposed and accessible to all applications. .
When granted the **Appropriate Permissions**, applications can share data among one another and access shared resources on the system securely.

1. Android Programming Language Choices

- Present Applications are written in **Java, Kotlin** languages
- Other programming languages, such as **C++** in future versions of Android.

2.No Distinctions Made Between Native and Third-Party Applications

There is no distinction between **Native Applications** and **Developer Created Applications.**
All applications have the same **Access** to **Core Libraries** and **Hardware Interfaces,** with **Appropriate Permissions**
Handsets are with a set of native applications such as a **Web browser** and **contact manager**.
Third-party Applications might integrate with these core applications and even extend them to provide a rich user experience.

3. Commonly Used Packages

The Android packages include support for

- ❖**Common user interface widgets** (Buttons, Spin Controls, Text Input)
- ❖**User interface layout**
- ❖**Secure networking and Web browsing features** (SSL, WebKit)
- ❖**Structured storage and relational databases** (SQLite)
- ❖**Powerful 2D and 3D graphics** (SGL and OpenGL ES 1.0)
- ❖**Audio and visual media formats** (MPEG4, MP3, Still Images)
- ❖Access to optional hardware such as **Location-Based Services** (LBS), **WiFi**, and **Bluetooth**

4. Android Application Framework

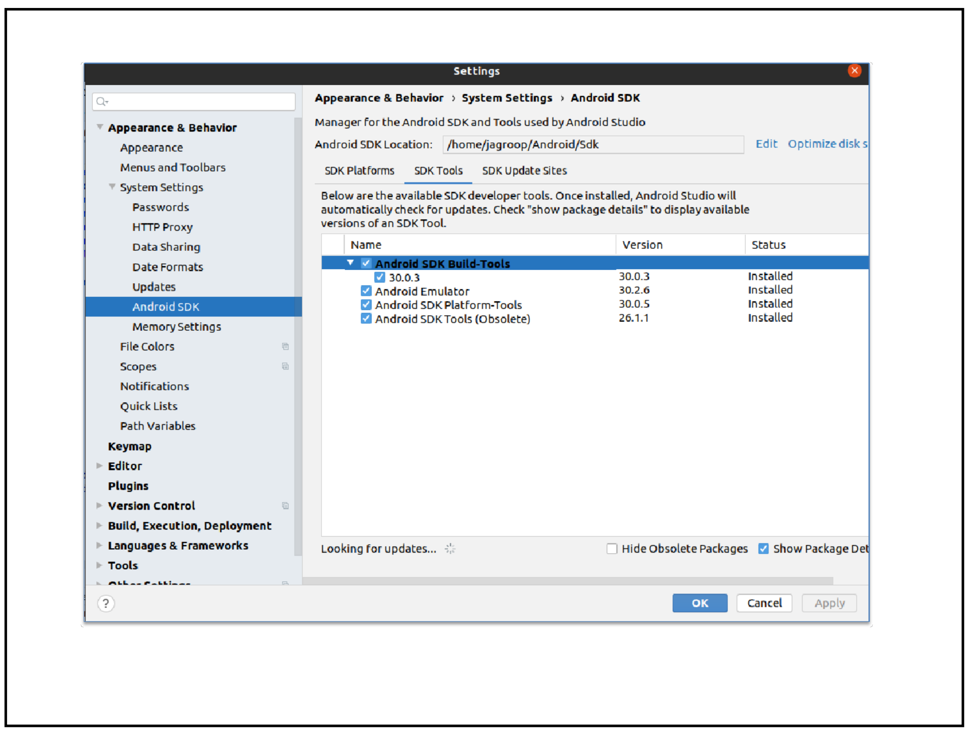
The Android application framework provides everything necessary to implement an application.

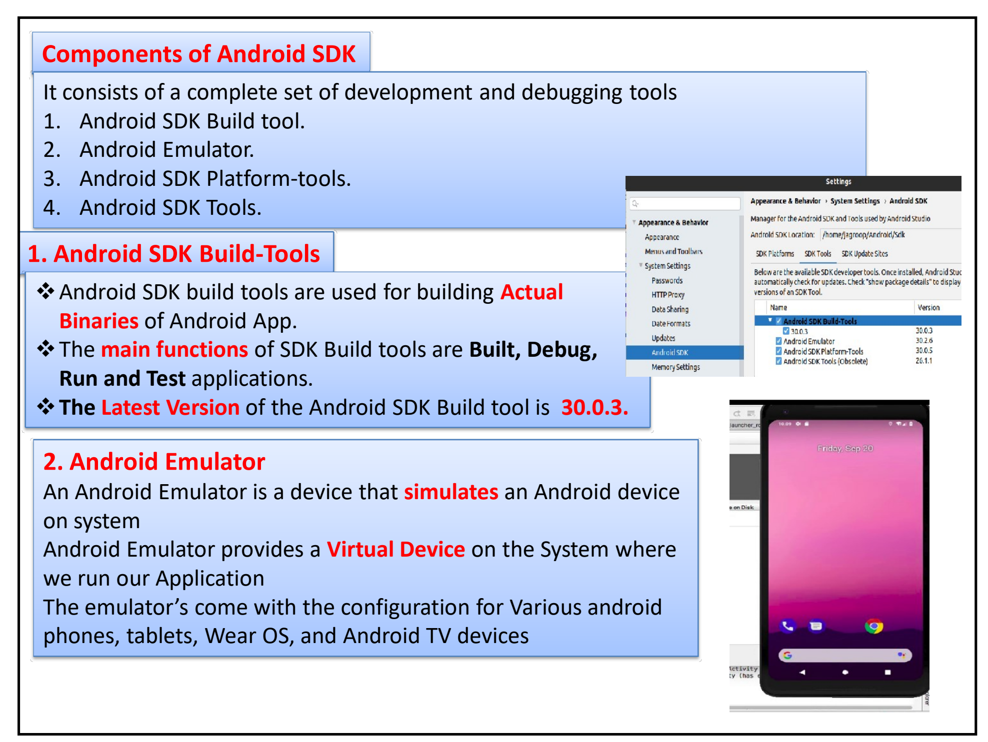
The components Android application lifecycle :

- ❖**Activities** are functions the application performs.
- ❖**Groups** of views define the application's layout.
- ❖**Intents** inform the system about an application's plans.
- ❖**Services** allow for background processing without user interaction.
- ❖**Notifications** alert the user when something interesting happens.

Android SDK

- ❖**Android SDK** or **Android Software Development Kit** which is developed by Google for Android Platform.
- ❖**SDK** is a collection of **Libraries** and **Development Tools** that are essential for Developing Android Applications.
- ❖Whenever **Google** released a **new version or update**, a corresponding SDK also released.
- ❖Android SDK consists of tools which are very essential for the development of Android Application.
- ❖These tools provide a smooth flow of the **Development Process** from developing and debugging.
- ❖Android SDK is **compatible** with all operating systems such as Windows, Linux, macOS, etc.
- ❖ First Android SDK released in 23 September 2008.
- ❖The first Android mobile was publicly released with Android 1.0 of the T-Mobile G1 (aka HTC Dream) in October 2008.


Settings
Appearance & Behavior › System Settings › Android SDK
Manager for the Android SDK and Tools used by Android Studio
Android SDK Location: /home/jagroop/Android/Sdk
Edit
SDK Platforms
SDK Tools
SDK Update Sites
Below are the available SDK developer tools. Once installed, Android Studio will automatically check for updates. Check "show package details" to display available versions of an SDK Tool.
Name
Version
Status
Android SDK Build-Tools
30.0.3 30.0.3 Installed
Android Emulator 30.2.6 Installed
Android SDK Platform-Tools 30.0.5 Installed
Android SDK Tools (Obsolete) 26.1.1 Installed
Appearance & Behavior
Appearance
Menus and Toolbars
System Settings
Passwords
HTTP Proxy
Data Sharing
Date Formats
Updates
Android SDK
Memory Settings
File Colors
Scopes
Notifications
Quick Lists
Path Variables
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Languages & Frameworks
Tools
Looking for updates...
Hide Obsolete Packages
OK
Cancel
Apply


Components of Android SDK
It consists of a complete set of development and debugging tools
1. Android SDK Build tool.
2. Android Emulator.
3. Android SDK Platform-tools.
4. Android SDK Tools.
1. Android SDK Build-Tools
❖ Android SDK build tools are used for building Actual Binaries of Android App.
❖ The main functions of SDK Build tools are Built, Debug, Run and Test applications.
❖ The Latest Version of the Android SDK Build tool is 30.0.3.
2. Android Emulator
An Android Emulator is a device that simulates an Android device on system
Android Emulator provides a Virtual Device on the System where we run our Application
The emulator's come with the configuration for Various android phones, tablets, Wear OS, and Android TV devices

3. Android SDK Platform-Tools

1. **Android Debug Bridge (ADB)**

   It is a **Command Line Tool** that helps to **Communicate** a smartphone, tablet, smartwatch, set-top box, or any other device that can run the Android .

   It allows us to perform **Installing** and **Debugging App** etc.

   ADB contains three components:

   i. **Client-**which sends commands. The client runs on your development machine. Invoke a client from a command-line terminal by issuing an **ADB command**.

   ii. **A daemon-** which **runs commands** on a device. The daemon runs as a **background** process on each device.

   iii. **A server,** which **manages communication** between the **client and the daemon**. The server runs as a background process on your development machine.

2. **Fastboot** allows to **flash a device** with a new system image.

3. **Systrace** tools help to **collect and inspect timing information** and **App Debugging**.

4. Android SDK Tools

SDK tools are generally platform independent and are required 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 &Ddescription
1	**Android** 9441452588 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**-PatchThis 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	**AdbAndroid** Debug Bridge (adb) is a versatile command line tool that lets you communicate with an emulator instance or connected Android-powered device.

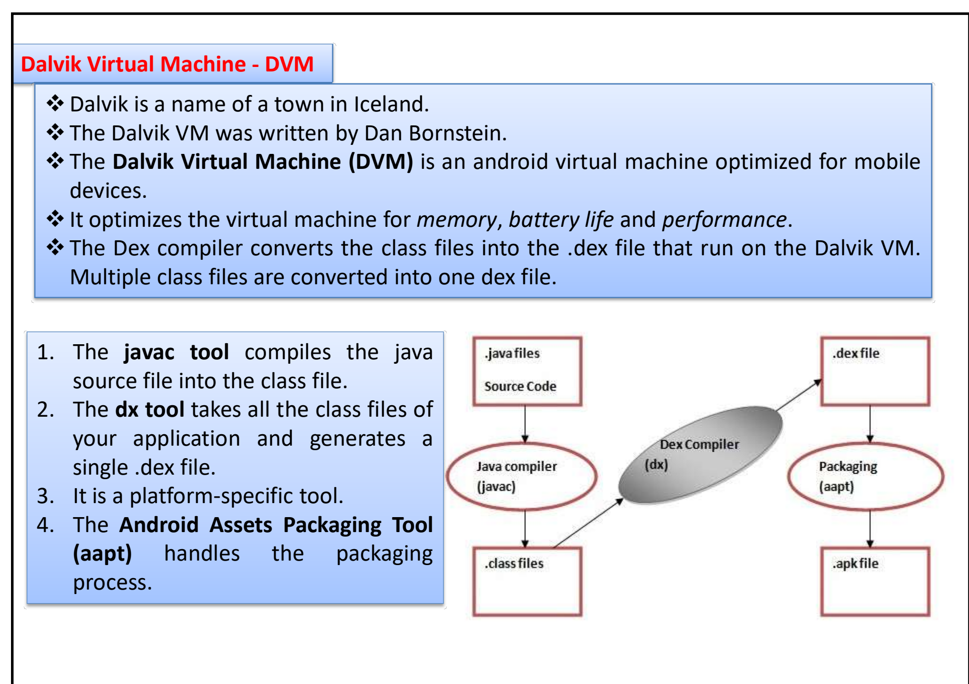
Android versions

Code Name	Version	API Level	Release Date
No codename	1.0	1	September 23, 2008
No codename	1.1	2	February 9, 2009
Cupcake	1.5	3	April 27, 2009
Donut	1.6	4	September 15, 2009
Eclair	2.0 - 2.1	5 - 7	October 26, 2009
Froyo	2.2 - 2.2.3	8	May 20, 2010
Gingerbread	2.3 - 2.3.7	9 - 10	December 6, 2010
Honeycomb	3.0 - 3.2.6	11 - 13	February 22, 2011
Ice Cream Sandwich	4.0 - 4.0.4	14 - 15	October 18, 2011
Jelly Bean	4.1 - 4.3.1	16 - 18	July 9, 2012
KitKat	4.4 - 4.4.4	19 - 20	October 31, 2013
Lollipop	5.0 - 5.1.1	21- 22	November 12, 2014
Marshmallow	6.0 - 6.0.1	23	October 5, 2015
Nougat	7.0	24	August 22, 2016
Nougat	7.1.0 - 7.1.2	25	October 4, 2016
Oreo	8.0	26	August 21, 2017
Oreo	8.1	27	December 5, 2017
Pie	9.0	28	August 6, 2018
Android 10	10.0	29	September 3, 2019
Android 11	11	30	September 8, 2020
Android 12	12	31	Oct-2021
Android 13	13	32	August 2022

Dalvik Virtual Machine - DVM

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

1. The **javac tool** compiles the java source file into the class file.
2. The **dx tool** takes all the class files of your application and generates a single .dex file.
3. It is a platform-specific tool.
4. The **Android Assets Packaging Tool (aapt)** handles the packaging process.



Install and Setup Eclipse IDE For Android App Development

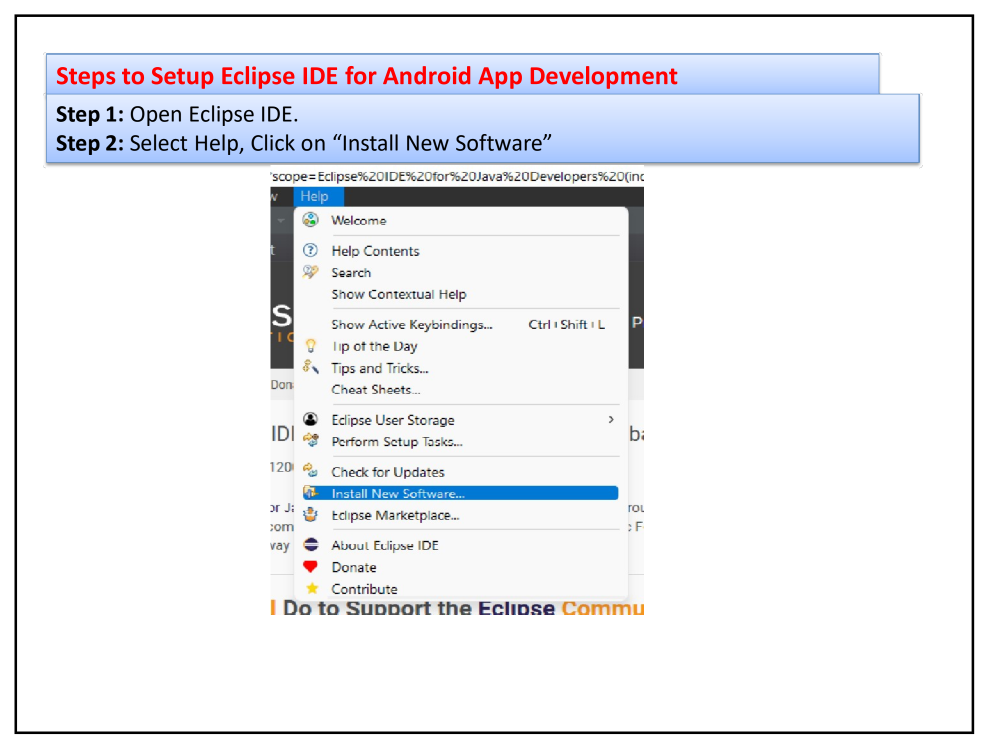
- Android Application Development can be done using **Android Studio** as well as **Eclipse IDE**.
- We can create android applications in Eclipse IDE using the **ADT plugin**.
- Eclipse is preferred for creating **small android applications**.
- Eclipse IDE is an **open-source software** used by developers,
- We will be using Eclipse IDE to set up Android App Development.
- First, we need to install Eclipse IDE, and then we will be setting it up for Android App Development.

Steps to Install Eclipse IDE

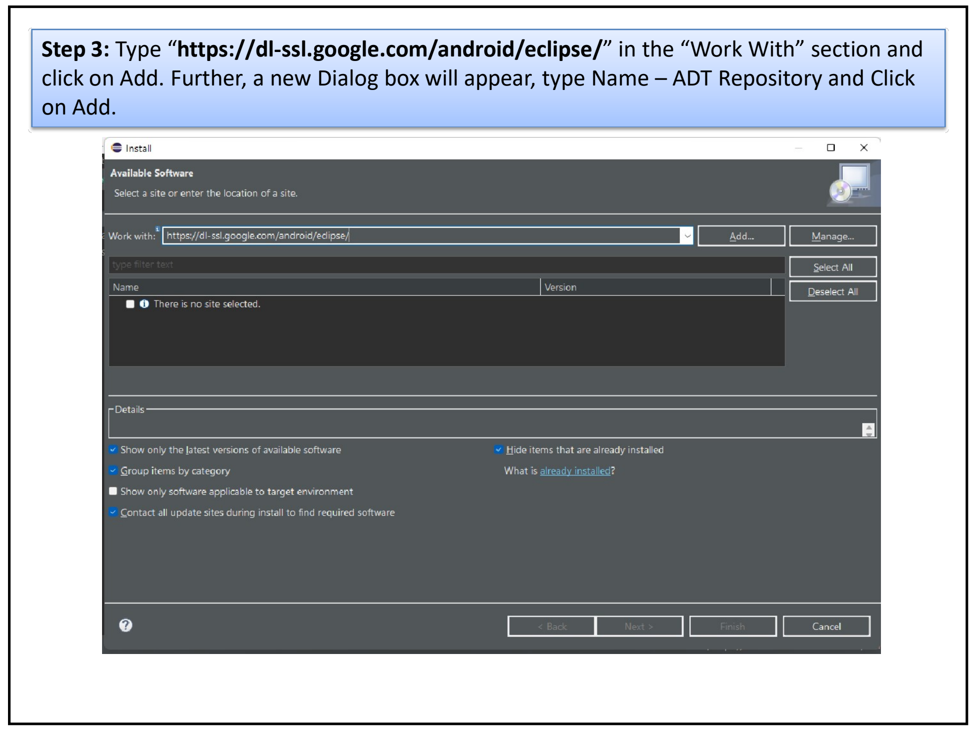
- To install Eclipse IDE, click on Download Eclipse
- Download JDK (Java Development Kit) and Android Studio as well.
- In File Explorer, go to Downloads – "Eclipse IDE" will be downloaded.
- Open Eclipse IDE, choose Eclipse IDE for Java Developers, and Install.
- Eclipse IDE environment is ready, now it's time to set up Android Development.

Steps to Setup Eclipse IDE for Android App Development

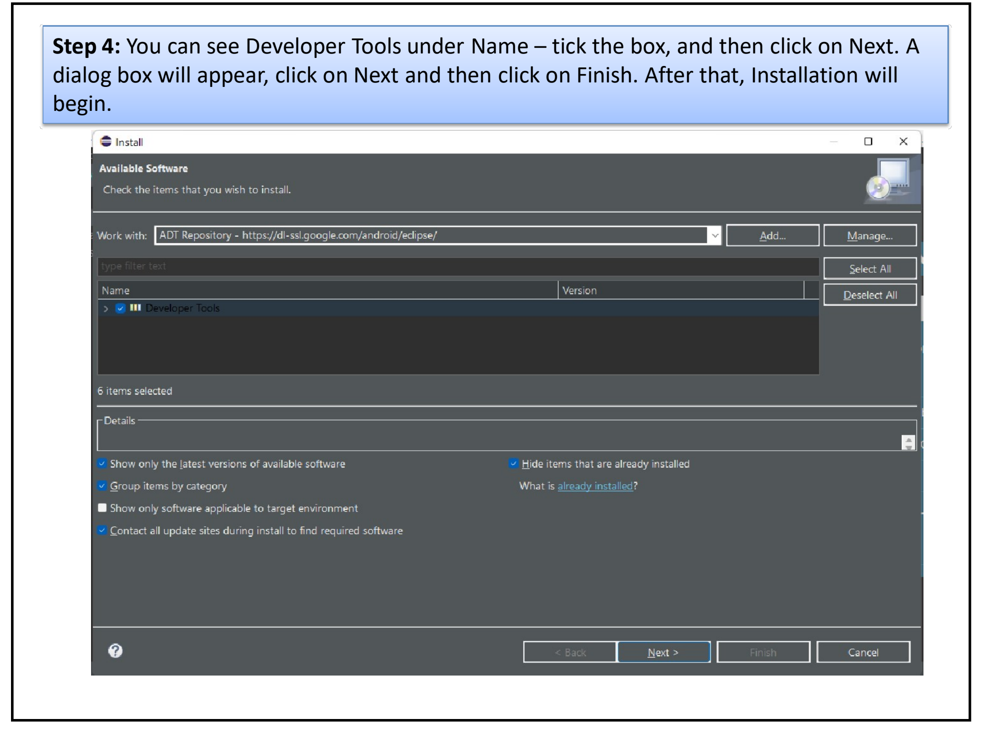
Step 1: Open Eclipse IDE.
Step 2: Select Help, Click on "Install New Software"



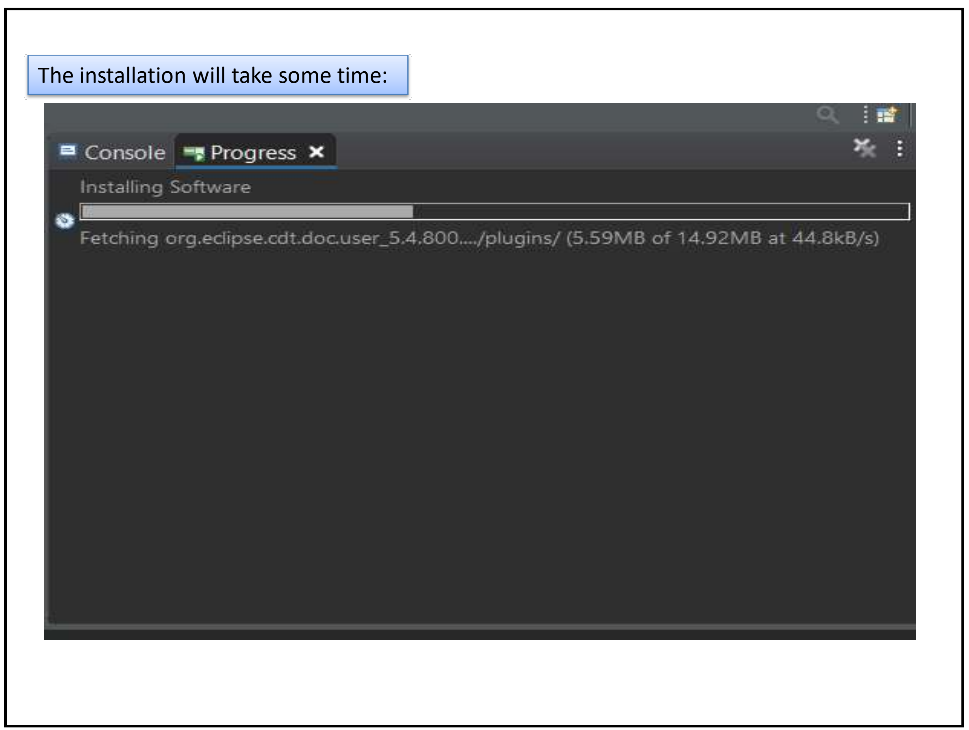
Step 3: Type “**https://dl-ssl.google.com/android/eclipse/**” in the “Work With” section and click on Add. Further, a new Dialog box will appear, type Name – ADT Repository and Click on Add.



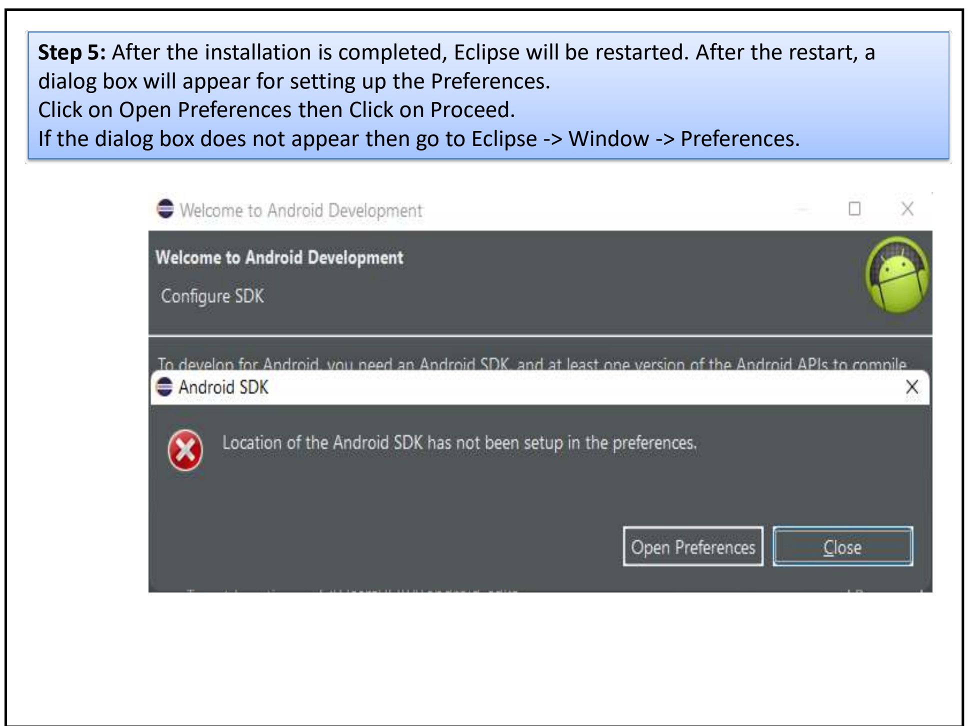
Step 4: You can see Developer Tools under Name – tick the box, and then click on Next. A dialog box will appear, click on Next and then click on Finish. After that, Installation will begin.



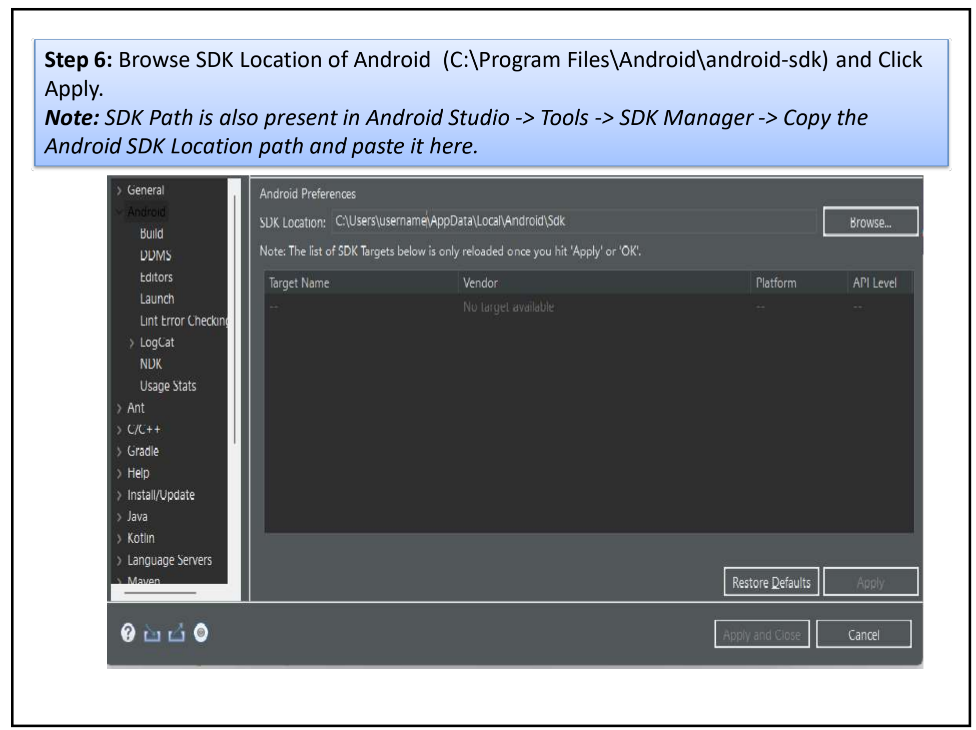
The installation will take some time:



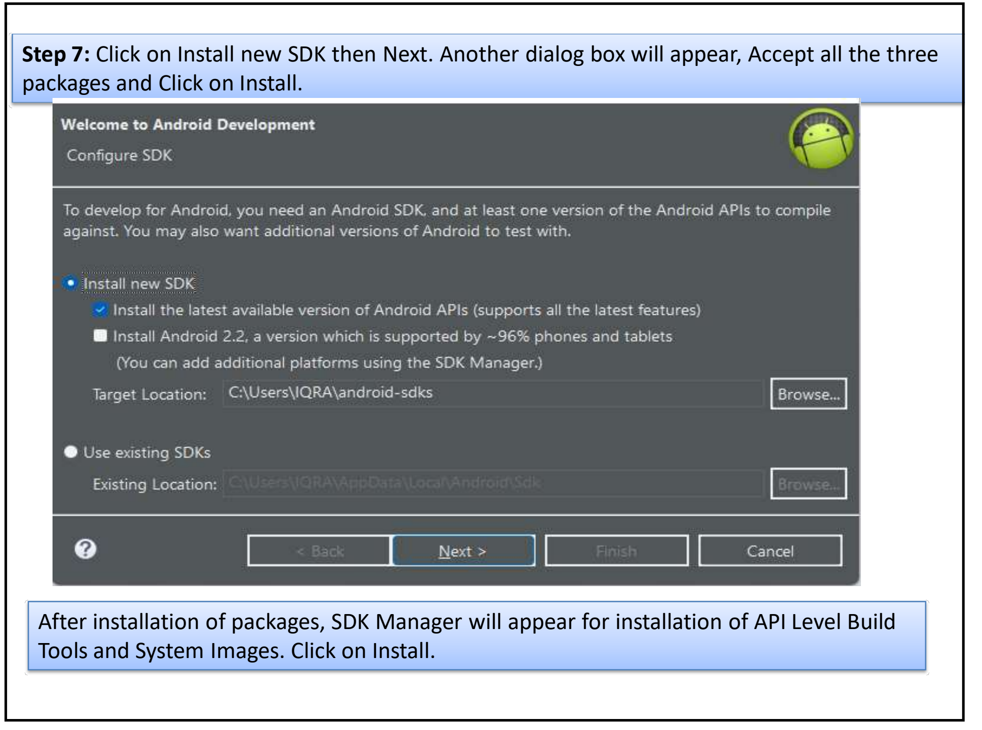
Step 5: After the installation is completed, Eclipse will be restarted. After the restart, a dialog box will appear for setting up the Preferences.
Click on Open Preferences then Click on Proceed.
If the dialog box does not appear then go to Eclipse -> Window -> Preferences.



Step 6: Browse SDK Location of Android (C:\Program Files\Android\android-sdk) and Click Apply.
Note: *SDK Path is also present in Android Studio -> Tools -> SDK Manager -> Copy the Android SDK Location path and paste it here.*

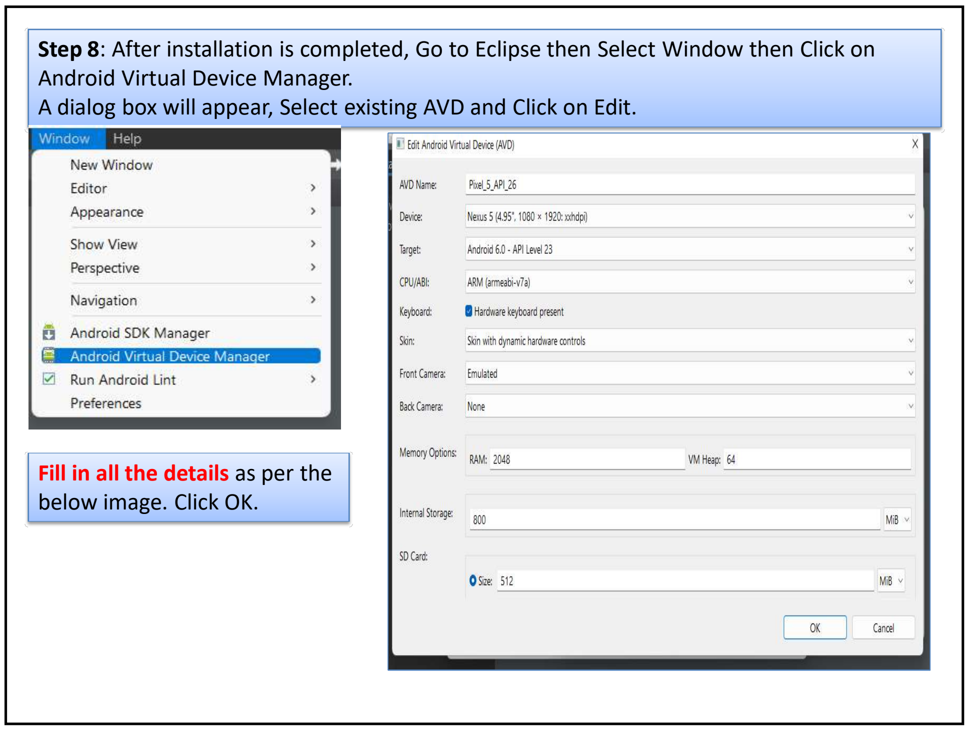


Step 7: Click on Install new SDK then Next. Another dialog box will appear, Accept all the three packages and Click on Install.



After installation of packages, SDK Manager will appear for installation of API Level Build Tools and System Images. Click on Install.

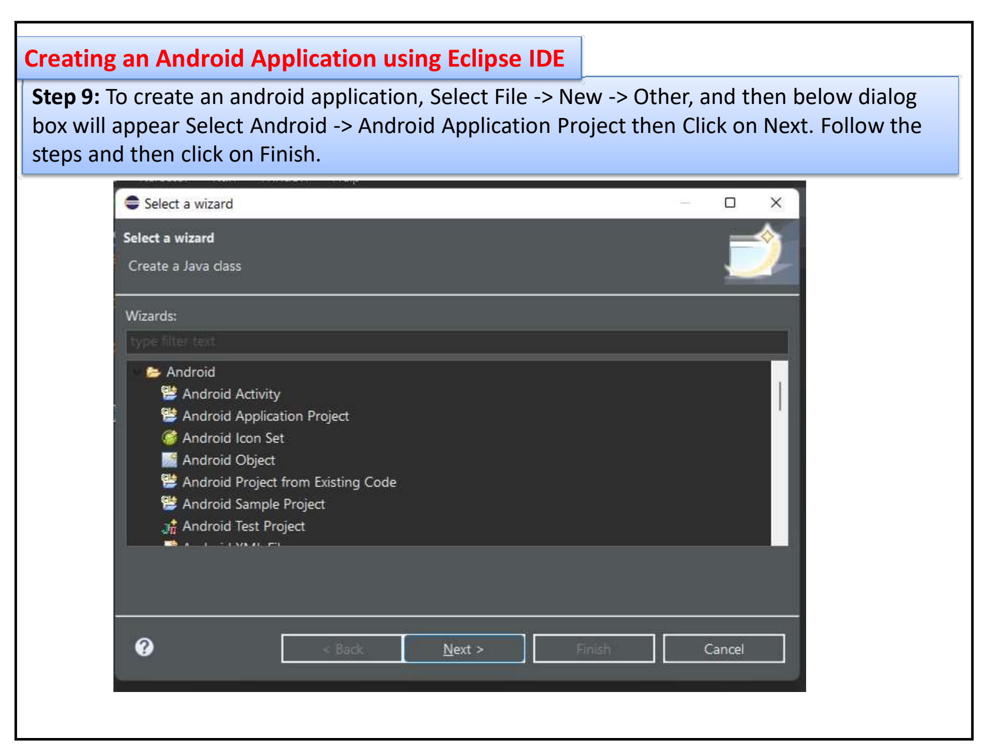
Step 8: After installation is completed, Go to Eclipse then Select Window then Click on Android Virtual Device Manager.
A dialog box will appear, Select existing AVD and Click on Edit.



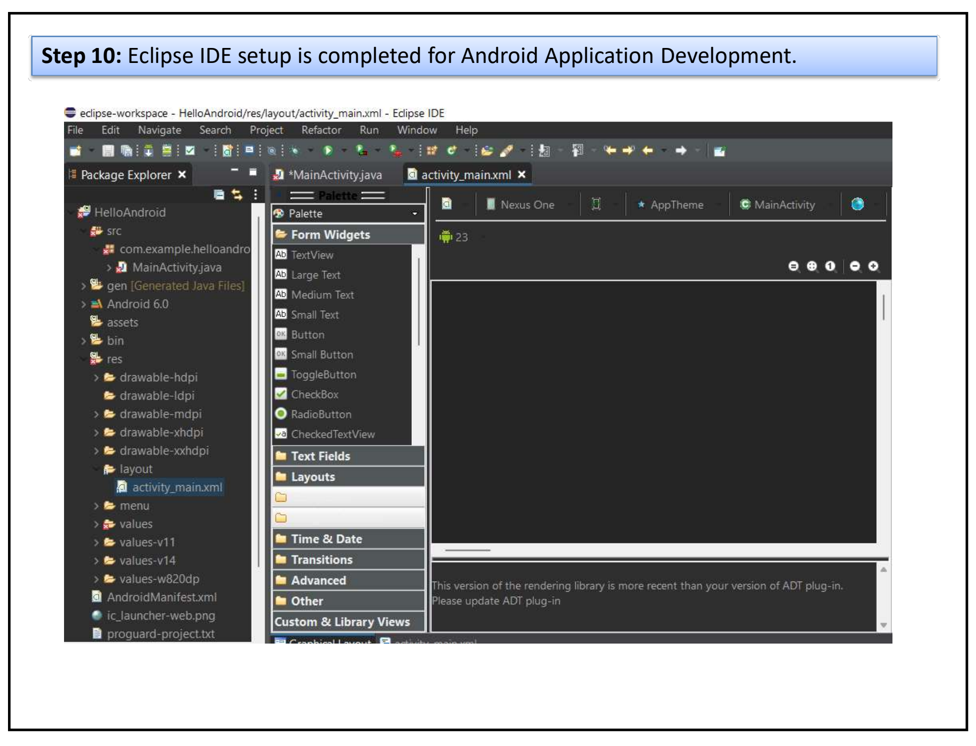
Fill in all the details as per the below image. Click OK.

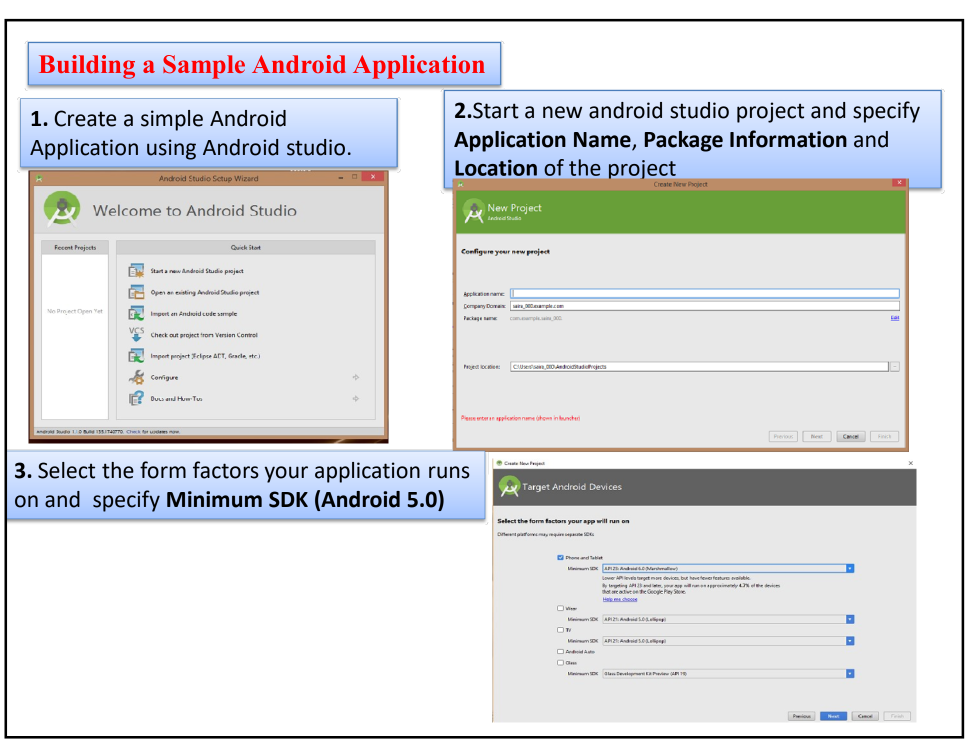
Creating an Android Application using Eclipse IDE

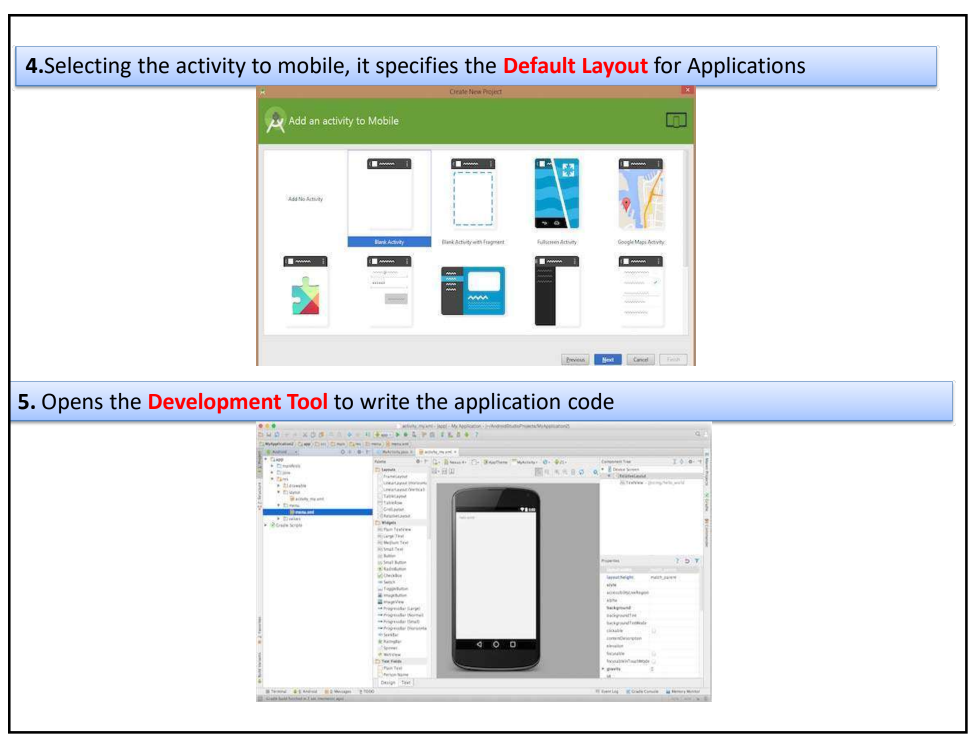
Step 9: To create an android application, Select File -> New -> Other, and then below dialog box will appear Select Android -> Android Application Project then Click on Next. Follow the steps and then click on Finish.

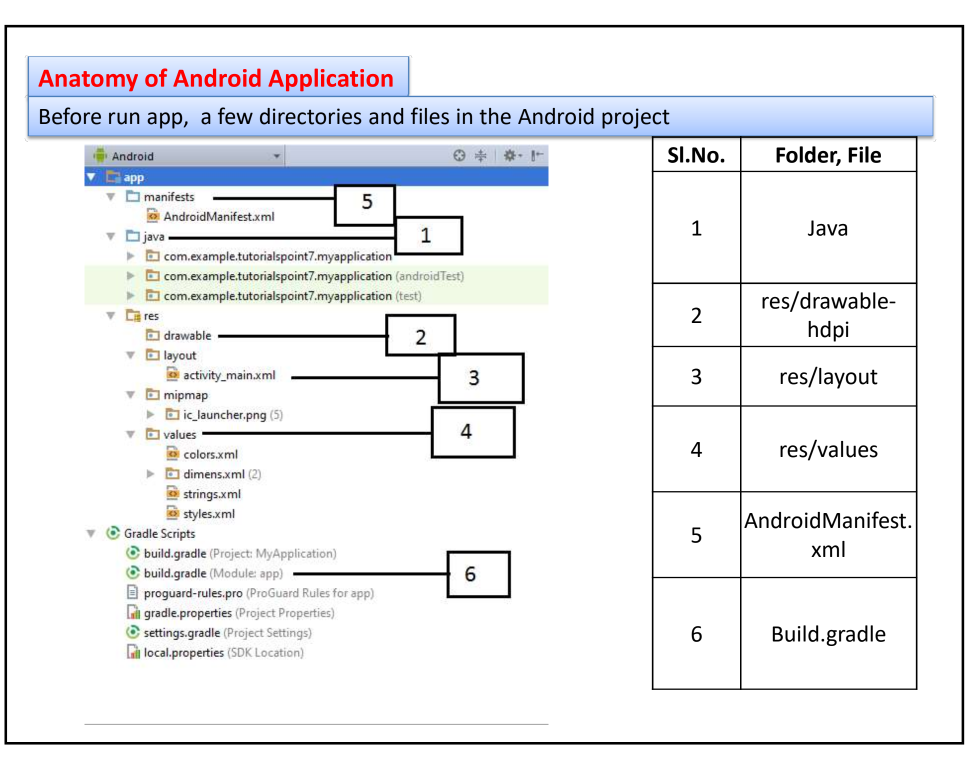


Step 10: Eclipse IDE setup is completed for Android Application Development.









Sl.No.	Folder, File
1	Java
2	res/drawable-hdpi
3	res/layout
4	res/values
5	AndroidManifest.xml
6	Build.gradle

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

1. The Main Activity File

The main activity code is a Java file **MainActivity.java**.
This is the actual application file which converted to a Dalvik executable and runs your application.

```
package com.example.helloworld;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
public class MainActivity extends AppCompatActivity
{
@Override
protected void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
 }
}
```

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

2. The Manifest File

Whatever component developed as a part of your application, declare all its components in a *manifest.xml* which resides at the root of the application project directory.
This file works as an interface between Android OS and your application.
For example, a default manifest file will look like as following file –

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.tutorialspoint7.myapplication">
<application android:allowBackup="true"
            android:icon="@mipmap/ic_launcher"
            android:label="@string/app_name"
            android:supportsRtl="true"
            android:theme="@style/AppTheme">
<activity android:name=".MainActivity">
     <intent-filter>
        <action android:name="android.intent.action.MAIN" />
       <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
 </activity>
</application>
 </manifest>
```

Here
<application>...</application> tags enclosed the components related to the application.
Attributes:
android:icon will point to the application icon available under *res/drawable-hdpi*.
The <activity> tag is used to specify an activity
android:name attribute specifies the fully qualified class name of the *Activity* subclass
android:label attributes specifies a string to use as the label for the activity.
android.intent.action.MAIN to indicate that this activity serves as the entry point for the application.
android.intent.category.LAUNCHER to indicate that the application can be launched from the device's launcher icon.
The ***@string*** refers to the *strings.xml* file
@string/app_name refers to the *app_name* string defined in the strings.xml file **(e.g. "HelloWorld")**

List of tags used in manifest file to specify different Android application components .
❖<activity>elements for activities
❖<service> elements for services
❖<receiver> elements for broadcast receivers
❖<provider> elements for content providers

3. The Strings File

The **strings.xml** file is located in the *res/values* folder and it contains all the text that application uses.
Example: The **names of buttons, labels, default text**
This file is responsible for their textual content.
Example:

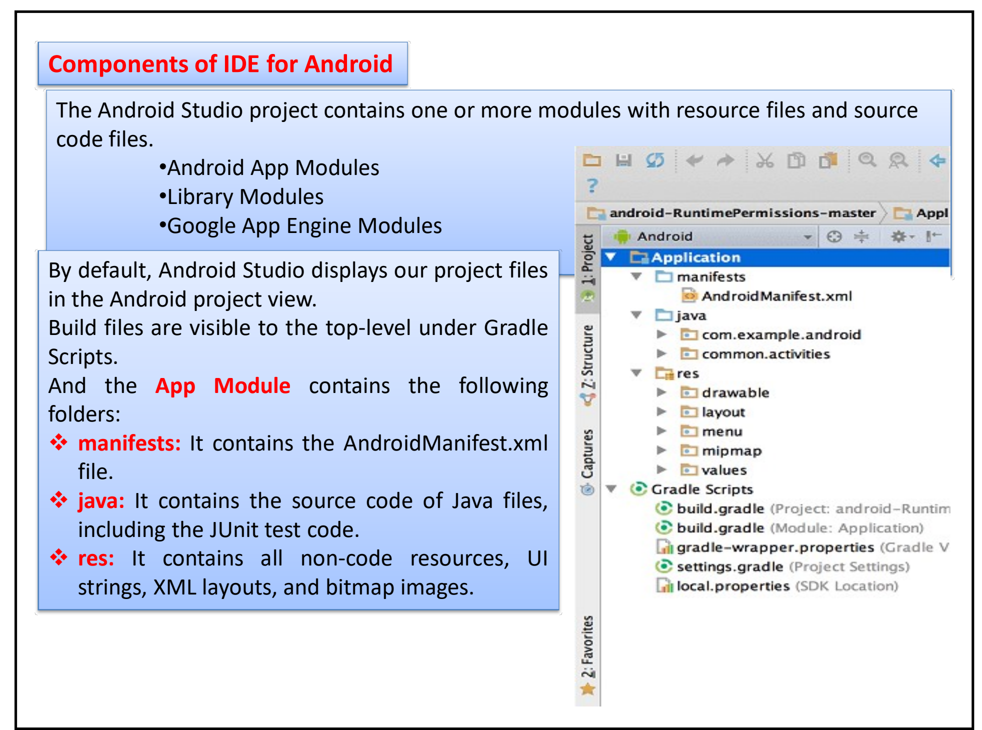
```
<resources>
    <string name="app_name">HelloWorld</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_main">MainActivity</string>
</resources>
```

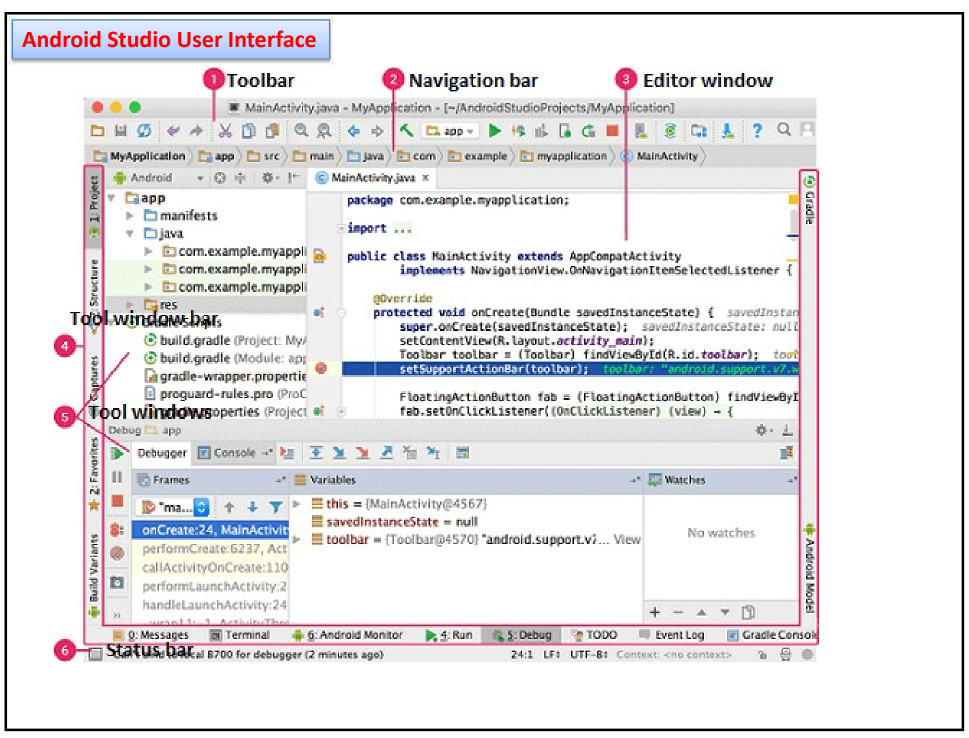
4. The Layout File

The **activity_main.xml** is a layout file available in *res/layout* directory.
It is referenced by our application when building its interface.
We will modify this file very frequently to change the layout of your application

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
<TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:padding="@dimen/padding_medium"
        android:text="@string/hello_world" tools:context=".MainActivity" />
</RelativeLayout>
```

5.Running the Application
To run the app from Android studio, open one of your project's activity files and click Run icon from the tool bar.
Android studio installs the app on your AVD and starts it and if everything is fine with your set-up and application, it will display following Emulator window –
Hello world!


Components of IDE for Android
The Android Studio project contains one or more modules with resource files and source code files.
•Android App Modules
•Library Modules
•Google App Engine Modules
By default, Android Studio displays our project files in the Android project view.
Build files are visible to the top-level under Gradle Scripts.
And the App Module contains the following folders:
❖ manifests: It contains the AndroidManifest.xml file.
❖ java: It contains the source code of Java files, including the JUnit test code.
❖ res: It contains all non-code resources, UI strings, XML layouts, and bitmap images.
android-RuntimePermissions-master
Appl
Android
Application
manifests
AndroidManifest.xml
java
com.example.android
common.activities
res
drawable
layout
menu
mipmap
values
Gradle Scripts
build.gradle (Project: android-Runtim
build.gradle (Module: Application)
gradle-wrapper.properties (Gradle V
settings.gradle (Project Settings)
local.properties (SDK Location)
1: Project
2: Structure
Captures
2: Favorites



1. **Toolbar**: provides us how to **running apps** and **launching Android tools**.
2. **Navigation Bar**: Helps in **navigating to project and open files for editing**. It gives a Compact View of structure visible in the Project window.
3. **Editor Window:** is a space where we can **create and modify our code**.
4. **Tool Window Bar:** Contains buttons to **expand and collapse individual tool windows**.
5. **Tool Windows:** Access specific tasks like **search, project management, version control etc**.
6. **Status Bar:** displays the **status of our project** and IDE, as well as any messages or warnings.

Android Studio Tool Window

Tool window	Windows /Linux	Mac
Project	Alt+1	Command+1
Version Control	Alt+9	Command+9
Run	Shift+F10	Control+R
Debug	Shift+F9	Control+D
Logcat	Alt+6	Command+6
Return to Editor	Esc	Esc
Hide all Tool Windows	Control+Shift+F12	Command+Shift+F12

Gradle Build System

✓Gradle build used as the foundation of the **Build System**.

✓It uses more Android-specific capabilities provided by the Android plugin for Gradle.

✓This build system runs independently from the command line and integrated tool from the Android Studio menu.

We can use build features for the following purpose:

- Configure, customize, and extend the build process.
- Create multiple APKs from our app, with different features using the same project and modules.
- Reuse resource and code across source sets.

Android Core Building Blocks

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

The **core building blocks / components** of Android :

- ❖Activities
- ❖ views
- ❖Intents
- ❖Services
- ❖content providers
- ❖fragments and AndroidManifest.xml.

1. Activity

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

2. View

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

3. Intent

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

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

Example:

```
Intent intent=new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://www.it.mrec.ac.in"));
startActivity(intent);
```

4. Service

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

There are two types of services **local** and **remote**.

- **Local Service** is accessed from within the application
- **Remote Service** is accessed remotely from other applications running on the same device.

5. Content Provider

Content Providers are used to share data between the applications.

6. Fragment

Fragments are like parts of activity.

An activity can display one or more fragments on the screen at the same time.

7. AndroidManifest.xml

It contains information about activities, content providers, permissions etc.

It is like the web.xml file in Java EE.

8. Android Virtual Device (AVD)

It is used to test the android application without the need for mobile or tablet etc.

It can be created in different configurations to emulate different types of real devices.

Android Emulator

❖ The **Android emulator** is an **Android Virtual Device (AVD),** which represents a specific Android device.

❖ We can use the Android emulator as a target device to execute and test our Android application on our PC.

❖ The Android emulator provides almost all the functionality of a real device.

❖ We can get the incoming phone calls and text messages.

❖ It also gives the location of the device and simulates different network speeds.

❖ Android emulator simulates rotation and other hardware sensors.

❖ It accesses the Google Play store, and much more

❖ Testing Android applications on emulator are sometimes faster and easier than doing on a real device.

❖ For example, we can transfer data faster to the emulator than to a real device connected through USB.

❖ The Android emulator comes with predefined configurations for several Android phones, Wear OS, tablet, Android TV devices.

Requirement and Recommendations

The Android emulator takes additional requirements beyond the basic system requirement for Android Studio.

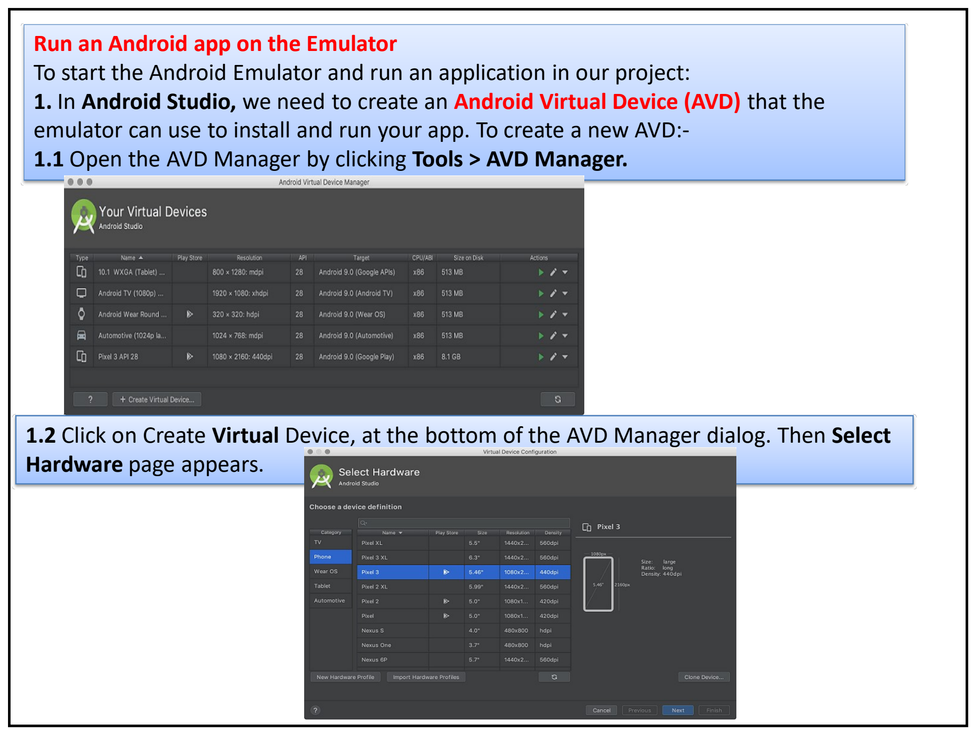
❖SDK Tools 26.1.1 or higher
❖64-bit processor
❖Windows: CPU with UG (unrestricted guest) support
❖HAXM 6.2.1 or later (recommended HAXM 7.2.0 or later)

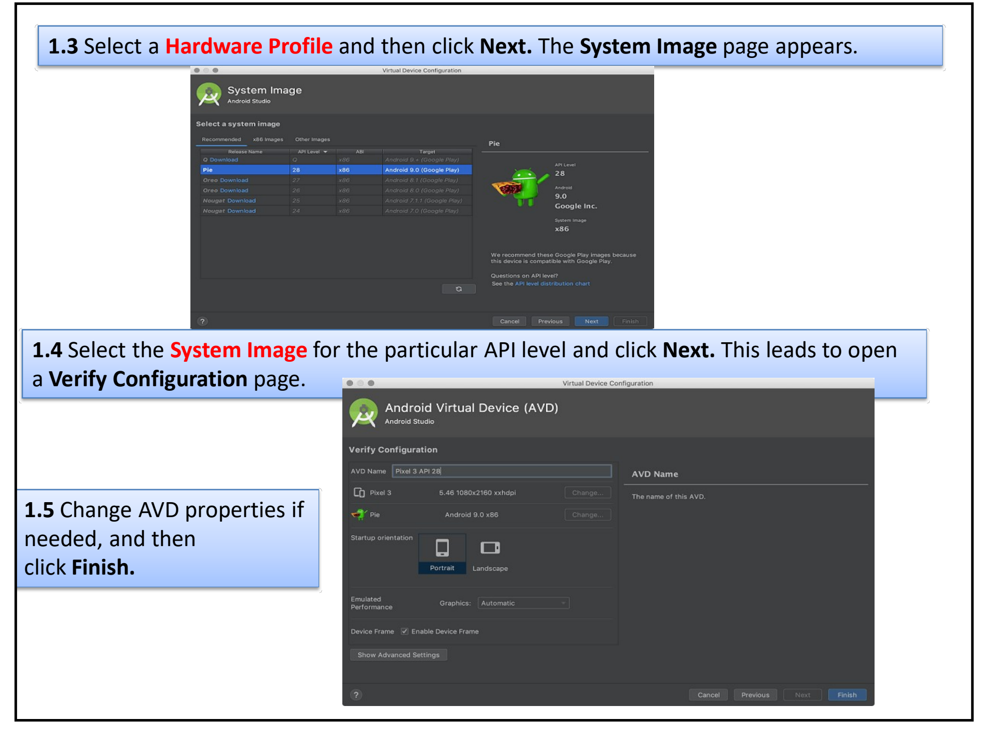
Install the Emulator

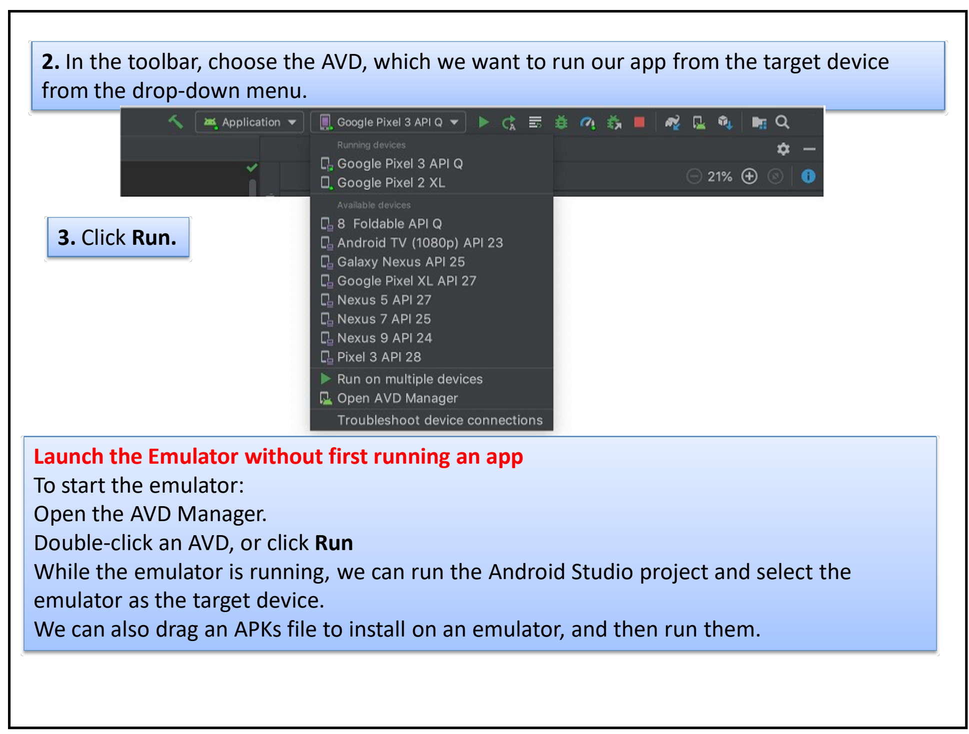
The Android emulator is installed while installing the Android Studio.

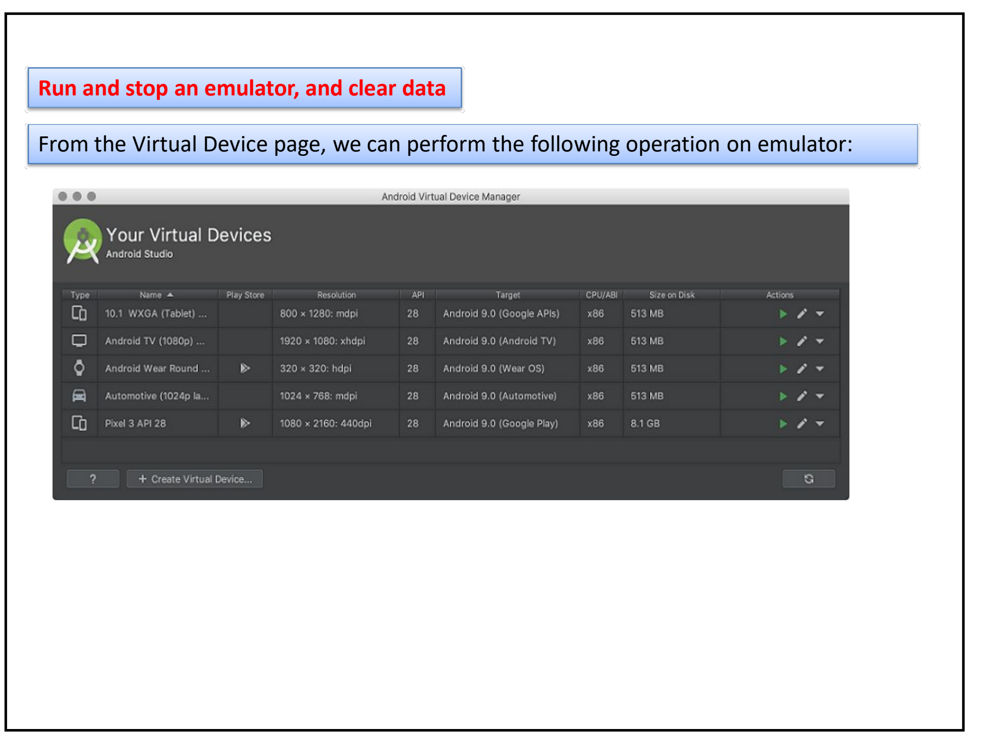
However some components of emulator may or may not be installed while installing Android Studio.

To install the emulator component, select the **Android Emulator** component in the **SDK Tools** tab of the **SDK Manager.**


Run an Android app on the Emulator
To start the Android Emulator and run an application in our project:
1. In Android Studio, we need to create an Android Virtual Device (AVD) that the emulator can use to install and run your app. To create a new AVD:-
1.1 Open the AVD Manager by clicking Tools > AVD Manager.
Your Virtual Devices
1.2 Click on Create Virtual Device, at the bottom of the AVD Manager dialog. Then Select Hardware page appears.
Select Hardware


1.3 Select a Hardware Profile and then click Next. The System Image page appears.
System Image
1.4 Select the System Image for the particular API level and click Next. This leads to open a Verify Configuration page.
Android Virtual Device (AVD)
1.5 Change AVD properties if needed, and then click Finish.


2. In the toolbar, choose the AVD, which we want to run our app from the target device from the drop-down menu.
Application
Google Pixel 3 API Q
Running devices
Google Pixel 3 API Q
Google Pixel 2 XL
Available devices
8 Foldable API Q
Android TV (1080p) API 23
Galaxy Nexus API 25
Google Pixel XL API 27
Nexus 5 API 27
Nexus 7 API 25
Nexus 9 API 24
Pixel 3 API 28
Run on multiple devices
Open AVD Manager
Troubleshoot device connections
21%
3. Click Run.
Launch the Emulator without first running an app
To start the emulator:
Open the AVD Manager.
Double-click an AVD, or click Run
While the emulator is running, we can run the Android Studio project and select the emulator as the target device.
We can also drag an APKs file to install on an emulator, and then run them.


Run and stop an emulator, and clear data
From the Virtual Device page, we can perform the following operation on emulator:
Android Virtual Device Manager
Your Virtual Devices
Android Studio
Type
Name
Play Store
Resolution
API
Target
CPU/ABI
Size on Disk
Actions
10.1 WXGA (Tablet) ...
800 × 1280: mdpi
28
Android 9.0 (Google APIs)
x86
513 MB
Android TV (1080p) ...
1920 × 1080: xhdpi
28
Android 9.0 (Android TV)
x86
513 MB
Android Wear Round ...
320 × 320: hdpi
28
Android 9.0 (Wear OS)
x86
513 MB
Automotive (1024p la...
1024 × 768: mdpi
28
Android 9.0 (Automotive)
x86
513 MB
Pixel 3 API 28
1080 × 2160: 440dpi
28
Android 9.0 (Google Play)
x86
8.1 GB
?
+ Create Virtual Device...

To run an Android emulator that uses an AVD, double-click the AVD, or click **Launch**

To stop the running emulator, right-click and select **Stop,** or click Menu ▼ and select Stop.

If we want to clear the data from an emulator and return it to the initial state when it was first defined, then right-click an AVD and select **Wipe Data.**

Or click menu ▼ and select **Wipe Data.**

Module-III

Android Application Design Essentials

- ❖Android Terminologies
- ❖Application Context
- ❖Activities
- ❖Services
- ❖Intents
- ❖Receiving and Broadcasting Intents

Android File Settings

- ❖Manifest File and its Common Settings
- ❖Intent Filter
- ❖Permissions
- ❖Managing Application Resources in a Hierarchy
- ❖Working with Different Types of Resources

Android Application Development Terminology

- Context
- Activity
- Intent
- Service
- Broadcast Receivers
- Content Providers
- Views
- Fragments
- Resources
- Manifest

1. Context

The context is the **Central Command Center** for an Android application.
All **Application-Specific Functionality** can be accessed through the context.
Context is the "**Base Class"** for Activity, Service, Application, etc
To get **information** of another part of the program (**Activity/Package/Application**).

1. Loading a Resource.
2. Launching a New Activity.
3. Creating Views.
4. Obtaining System Service.

Methods used to get context

1.getApplicationContext()
2.getContext()
3.getBaseContext()
4.this (when in the activity class)

2. Activity

An Android application is a **Collection of Tasks**, where each task is called an **Activity**.
Each Activity within an application has a **Unique Task** .
Unlike programming paradigms in which apps are launched with a **main() method**, the Android initiates code in an Activity instance by invoking **Specific Callback Methods**
An activity is implemented as a subclass of **Activity class** as

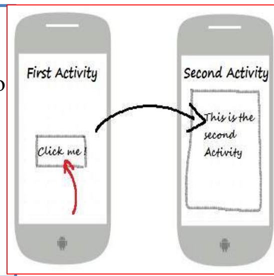
public class **MainActivity** extends **Activity**
{ //Implementation }

3. Intent

An intent is an abstract description of an operation to be performed
The Android follows **Asynchronous Messaging Mechanism** to match task requests with the appropriate Activity.
Each request is packaged as an **Intent** to *do* something.

Example:

a) Sending the User to Another App
b) Getting a Result from an Activity
c) Allowing Other Apps to Start Your Activity



4. Service

A Service is an **Application Component** that can perform **Long-running Operations** in the background. It does not provide a **User Interface**.
Once started, it might **Continue Running** even after the user switches to another application,

Example:

i. Handle Network Transactions
ii. Play Music
iii. Perform File I/O
iv. Interacting with a Content Provider

Types of Services

i. Foreground-Music Player and Downloading
ii. Background-Syncing and Storing data
iii. Bound

5. Broadcast Receivers

Handles communication between Android OS and Applications.

Broadcast Receivers respond to broadcast messages from **other applications** or from the **system**.

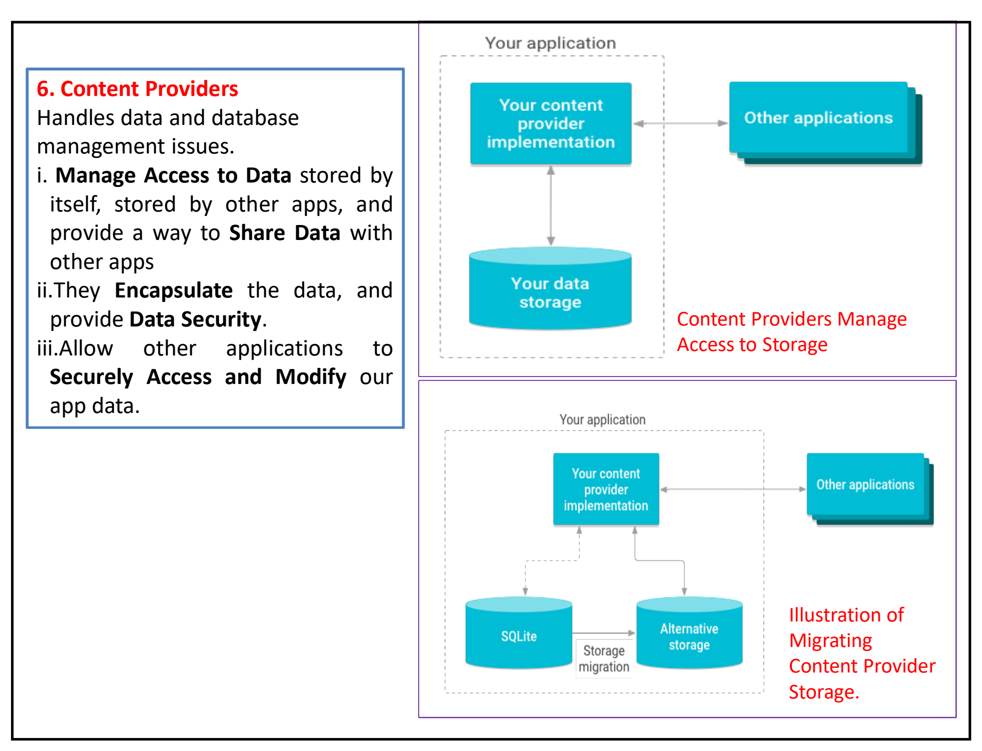
Two important steps to make BroadcastReceiver

i. Creating the Broadcast Receiver.

ii. Registering Broadcast Receiver

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

Charger Connected
Android System
Message Recieved
Broadcast Receiver



7. Views

View s are the basic building block for **User Interface Components**.

Views can be used to create a useful I/O fields. They are same as, input text field, image tag to show images, radio field in HTML.

i. Text View
ii. EditText
iii. Button
iv. ImageView
v. ImageButton
vi. CheckBox
vii. Radio button
viii. ListView
ix. Spinner

8. Fragment

A **Fragment** is a piece of an activity or **sub-activity** which enable more **modular activity** design. Fragments were added to the Android API in **Honeycomb Version** API 11.

Types of Fragments:

a. Single Frame Fragments: Using in **hand hold devices** like mobiles, here we can show only one fragment as a view**.**

b. List Fragments – Fragments having **Special List View** is called as list fragment

c. Fragments Transaction – Using with it we can **move one fragment to another** fragment.

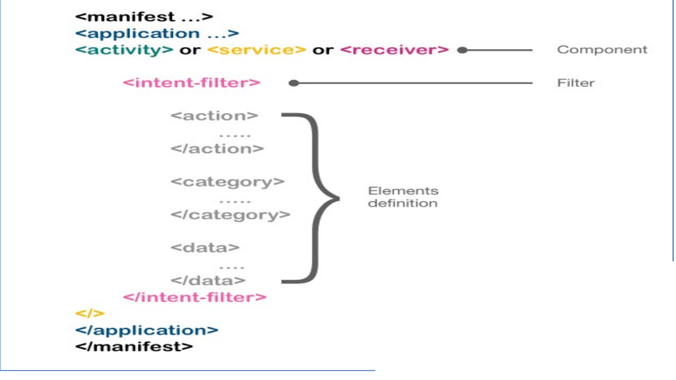
9. Manifest:

The manifest file describes essential information about your app to the **Android build tools**, the **Android operating system**, and **Google Play**

It contains information about activities, content providers, permissions etc.

The manifest file is required to declare the following:

- ❖The **components of the App-**activities, Services, Broadcast Receivers, & Content Providers
- ❖The **Permissions** to access protected parts of the system or other apps
- ❖The **Hardware and Software** features the app requires



10.Resources

Resources are the additional files and static content that your code uses, such as

i. Bitmaps
ii. Layout Definitions
iii. User Interface Strings
iv. Animation Instructions etc..

We placed each type of resource in a specific subdirectory of your project's **res/ directory**

```
MyProject/
    src/
        MyActivity.java
    res/
        drawable/
            graphic.png
        layout/
            main.xml
            info.xml
        mipmap/
            icon.png
        values/
            strings.xml
```

Application Context

The Application Context is the **Central Location** for all top-level application functionality.

It's used to get the **context associated with the Application**, which contains all of the activities running within it.

It can be thought of as **a layer** that sits behind the entire program.

So, as long as it does not kill the entire Application, this context stays alive.

The Context class can be used to:

a. Manage **Application-specific Configuration , Application-wide Operations** and **data.**
b. Access **settings** and **resources** shared across multiple Activity instances.

1. getApplicationContext(): To Retrieving the Application Context

To retrieve the Context for the **Current Process** we can use the getApplicationContext() method. It can be represented as:

Context context = getApplicationContext();

❖**Using the Application Context**

After retrieving a valid application Context, it can be used to access application-wide features and services.

2. getResources(): T**o** Retrieving Application Resources

To retrieve application resources we can use the getResources() by using its **Resource Identifier**.

Example: Retrieves a **String Instance** from the application resources by its resource ID

String greeting = getResources().getString(R.string.hello);

3. getSharedPreferences() : Accessing Application Preferences
The **SharedPreferences** class can be used to save simple application data, such as configuration settings.
Shared Preferences allow you to save and retrieve data in the form of (key,value).
Example:
SharedPreferences SP = getSharedPreferences(MyPREFERENCES, Context.MODE_PRIVATE);
Here the first parameter is the key and the second parameter is the MODE.
4. Accessing Other Application Functionality
The application Context provides access to a number of other **top-level application** features.

i. Launch Activity instances
ii. Retrieve assets packaged with the application
iii. Request a system service (for example, location service)
iv. Manage private application files, directories, and databases
v. Inspect and enforce application permissions

```
<application
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:theme="@style/AppTheme" >
 <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:theme="@style/MyCustomTheme">
```

Activity:
The Android Activity class **(android.app.Activity)** is core to any Android application.
User can Define and implement an Activity class for each screen in the application.
Example: A simple **game application** have the following five Activities, as shown in Figure :

1.A Startup or Splash screen: Serves as the **primary entry point** to the application and displays the **Name** and **Version** and transitions to the Main menu after a short interval.

2. A Main Menu screen: Acts as a **switch to drive the user to the core Activities** of the application. Here the users must choose what they want to do within the application.

3.A Game Play screen: This activity is where the core game play occurs.

4.A High Scores screen: This activity might display game scores or settings.

5.A Help/About screen: This activity might display the information the user might need to play the game.

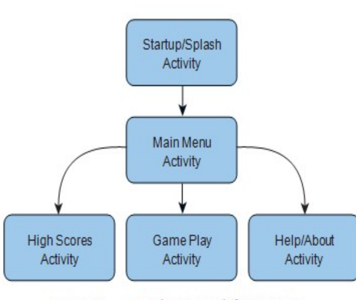


Figure 4.1 A simple game with five activities.

The Activity Stack

- The Android operating system **keeps track of all Activity objects** running by placing them on an **Activity stack**.
- When a **new Activity starts**, the Activity on the top of the stack **pauses**, and the new Activity pushes onto the top of the stack.
- When that **Activity finishes**, that Activity is **removed** from the activity stack, and the **previous** Activity in the stack resumes.
- Android applications are responsible for **managing** their **state** and **memory, resources,** and **data.**
- They must pause and **resume** seamlessly.

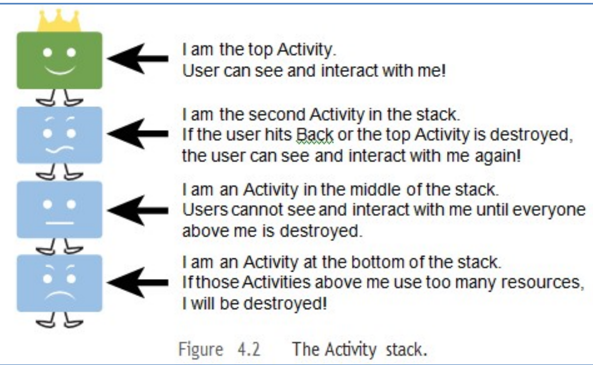


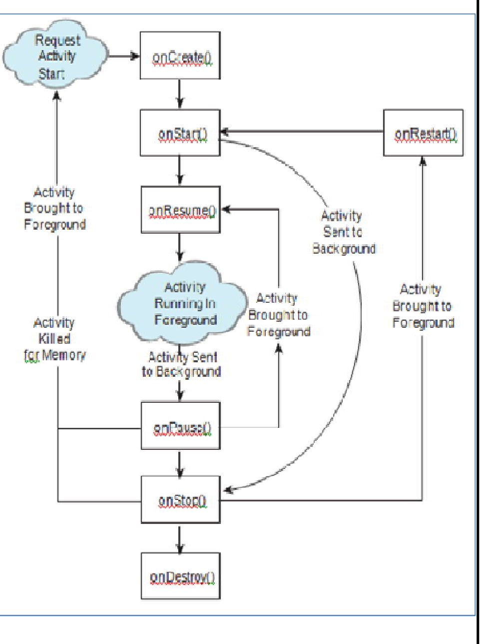
Figure 4.2 The Activity stack.

The Lifecycle of an Android Activity

- Activity Callbacks are used to **Manage** Application **State and Resources**
- Different **state changes** within the Activity are represented by a series of callbacks are shown in fig.

- **Method stubs** for the callbacks of the Activity class:

```
public class MyActivity extends Activity
{
protected void onCreate(Bundle
savedInstanceState);
protected void onStart();
protected void onRestart();
protected void onResume();
protected void onPause();
protected void onStop();
protected void onDestroy();
}
```



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

1. onCreate() : Initializing Static Activity Data

- When an Activity first starts, the onCreate() method is called.
- The onCreate() method has a **single parameter**, **a Bundle**, which is **null** if this is a newly started Activity.
- If this Activity was **killed** due to **low memory** and is restarted, the Bundle contains the **previous state information** and it can reinitiate.

2. onResume() :Initializing and Retrieving Activity Data

onResume() method is called when the Activity reaches the top of the activity stack and becomes the **foreground process**.

The **onResume()** method is the appropriate place to start audio, video, and animations

3. onPause(): Stopping, Saving, and Releasing Activity Data

- The onPause() method **alerts** the current Activity that it is being pushed down the activity stack when another Activity rises to the top of the activity stack.
- Save any uncommitted data when an application does not resume.
- The new foreground Activity is not started until the onPause() method returns.
- Activity should stop any audio, video, and animations it started in the onResume() method.

4. onDestroy(): Destroy Static Activity Data

When an Activity is being destroyed, the onDestroy() method is called.
The onDestroy() method is called for one of **two reasons**:

i. The Activity has completed its lifecycle voluntarily (or)
ii. The Activity is being killed by the Android OS because it needs the resources.

5. onStop(): Avoiding Activity Objects Being Killed

- The Android OS provides the ability to terminate any activity that has been paused, stopped, or destroyed in low memory situations.
- Means that any Activity not in the foreground is shutdown.
- If the Activity is killed after onPause(), the onStop() and onDestroy() methods not called.
- The more resources released by an Activity in the onPause() method, the less likely the Activity is to be killed while in the background.

6. onSaveInstanceState(): Saving Activity State into a Bundle

If an Activity is weak to killed by the Android OS due to low memory, the Activity can save state information to a Bundle object using the onSaveInstanceState() .
This call is not guaranteed under all circumstances, so use the onPause() method for essential data commits.

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

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

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

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

Services

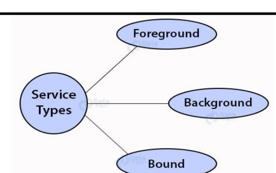
- ❖ A service is a **component/activity** that facilitates an application to run in the **background** in order to perform **Long Running Tasks.**
- ❖ The **aim** of a service is to ensure that the application remains **active** in the background so that the user can **operate multiple applications** at the same time.
- ❖ An **User Interface** is not required for services as it is operate long-running processes without any **user intervention**.
- ❖ A service can **run** continuously in the **background** even if the application is **closed** or the user **switches** to another application
- ❖ Application components can **bind** itself to service to carry out **Inter Process Communication**(IPC).

Difference between Services and Threads

Thread: Provided by the OS to allow the user to perform operations in the background.

Service: An Android component that performs a **Long-running Operation**

Types of Android Services



1. **Foreground Services**
2. **Background Services**
3. **Bound Services**

1. Foreground Services

❖Services that **notify the user** about its ongoing operations.

❖ Users can **interact** with the service by the **notifications** provided about the ongoing task.

Example: Downloading a file (User **keep track** of the **progress** and **pause** and **resume** the process).

2. Background Services

- ❖ Background services do **not require user intervention**.
- ❖ These services do **not notify** the user about ongoing background tasks and users **cannot access** them.

Example: Schedule syncing of data or storing of data

3. Bound Services:

- ❖ A service is **bound** when an application component binds to it by calling **bindService().**
- ❖ Offers a **client-server interface** that allows components to interact with the **service**, **send requests**, **receive results** and **IPC**
- ❖ A bound service **runs** only as long as another **application component** is **bound** to it.
- ❖ **Multiple** components can bind to the service **at once**
- ❖ The service is **destroyed** when all of them unbind.

onStartCommand() Method

The **onStartCommand()** must return an integer value that describes how the system should continue the service in the event that the system kills it.

The **onStartCommand()** return one of the following constants:

1. START_NOT_STICKY
2. START_STICKY
3. START_REDELIVER_INTENT

Starting a Service: To start a service from an activity or application component by passing an **Intent** to **startService()** or **startForegroundService()**. The Android system calls the service's **onStartCommand()** and passes it the Intent, which specifies which service to start.

Example:

An activity can start the service "**HelloService"** using an **intent** with **startService()**

```
Intent intent = new Intent(this, HelloService.class);
startService(intent);
```

Stopping a Service: A started service must manage its own lifecycle. That is, the system doesn't **stop or destroy** the service unless it must recover system memory and the service continues to run after **onStartCommand(). A service can be stopped only in one of the two cases**

Itself by calling **stopself(),** or

➢Another component can stop it by calling **stopservice().**

The Life Cycle of Android Services

Services have **2 paths** to complete its life cycle

1. **Started**
2. **Bounded**.

1. Started Service (Unbounded Service)

- ❖ A service will **initiate** when an application component calls the **startService()** .
- ❖ Once initiated, the service can run continuously in the background even if the component is **Destroyed** which was **responsible** for the start of the service.

Methods used to **stop** the running service:

a. **stopService()**

b. **stopSelf()**

2. Bounded Service

❖It can be treated as a **server** in a client-server interface.

❖Application components can **Send Requests** to the service and it can **fetch results**.

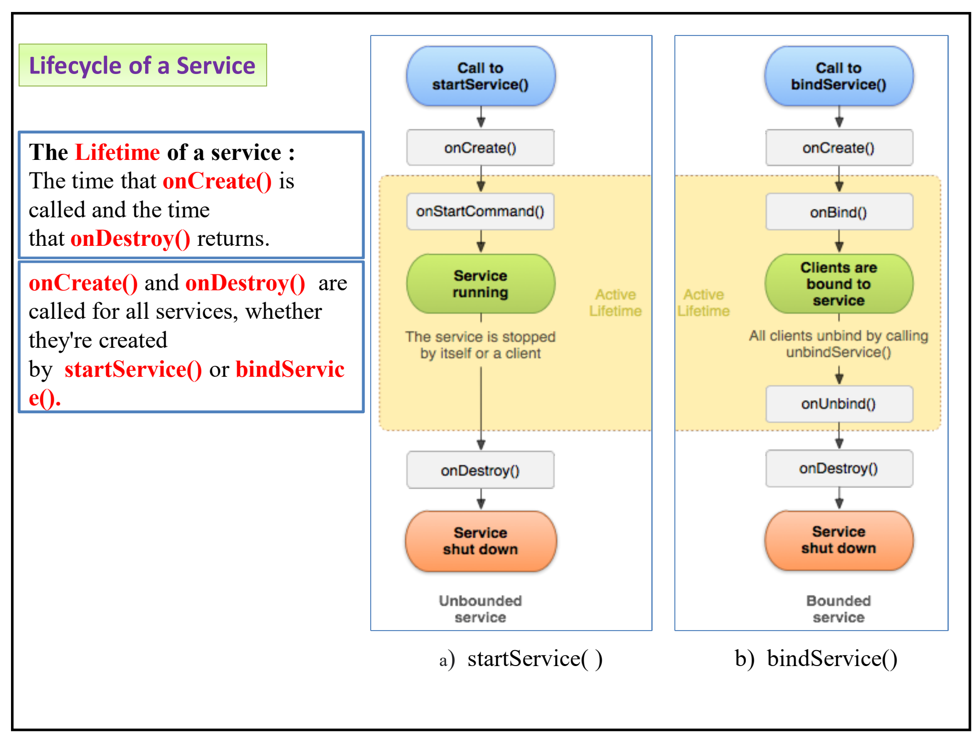
Methods:

1. bindService():

A service is bounded when an application component binds itself with a service

2. unbindService():

To stop the unning service, all the components must unbind themselves from the service.



Methods of Android Services

Methods	Description
onStartCommand()	**onStartCommand()** is called when a component (eg: activity) requests to **start a service**. Once the **service is started**, it can be **stopped** using **stopService()** or **stopSelf().**
onBind()	It is **invoked** when an application component calls the **bindService().** If the **binding of service** is not required then it returns **NULL.**
onUnbind()	The Android System **invokes onUnbind()** when all the clients get disconnected from a particular service interface.
onRebind()	Once **all clients are disconnected** from the particular interface of service and there is a need to connect the service with new clients, the system calls **onRebind().**
onCreate()	Whenever a service is created either using **onStartCommand()** or **onBind(),** the android system calls onCreate(). This method is necessary to perform a one-time set-up.
onDestroy()	When a service is **no longer in use**, the system invokes **onDestroy() .** Services must implement it in order to clean up resources like **registered listeners, threads, receivers, etc**.

Intents

An Intent is a **Messaging Object** which is used to **request an action** from an **App Component**. Intents facilitate communication between components in several ways.
There are three use cases:
1. Starting an Activity
2. Starting a Service
3. Delivering a Broadcast

1. Starting an Activity using Intent
An Activity represents a single screen in an app. An activity started using an Intent by using **startActivity()** method.
1. **startActivity():**
   To start a **new instance** of an Activity by **passing an Intent**. which describes the activity **to start** and carries any important data.
2. **startActivityForResult():**
   To receive a result from the activity.
3. **onActivityResult():**
   To receives the result as a **separate Intent object**.

2. Starting a Service
A Service is a component that performs operations in the background without a user interface.
A Service can be started with a **JobScheduler** (Android 5.0 (API level 21) and Later).
Also a service can be start by using **methods** of the Service class. (Earlier to Android 5.0 (API level 21)

A. startService():
A service can be start to perform a one-time operation (e,g,downloading a file) by passing an **Intent,** which describes the service to start and carries any needed data.

B. bindService():
If the service is designed with a **Client Server Interface**, bind to the service from another component by passing an Intent.

3. Delivering a Broadcast
A broadcast is a message that any app can receive.
The **system** delivers **Various Broadcasts** for system events.
Example:
- when the system boots up or
- when the device starts charging.

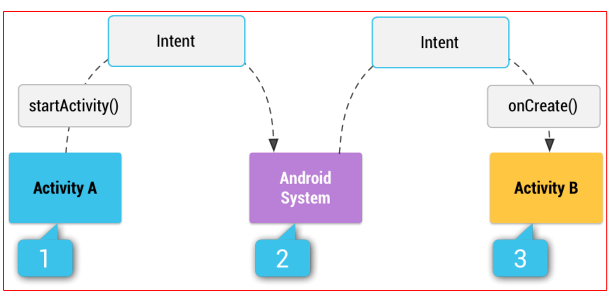
User can deliver a broadcast to other apps by passing an Intent to
- sendBroadcast() or
- sendOrderedBroadcast().

Intent Types:
There are two types of intents
1. **Explicit Intents**
2. **Implicit Intents**

1.Explicit Intents
Requires Specific Package /a fully-qualified Component class name.
Typically use an explicit intent to **start a component in your own app**.
Example:
Start a new activity within app in response to a user action (or)
Start a service to download a file in the background.

```
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

2. Implicit Intents
Specific component name not required , but declare a **General Action** to perform, which allows a component from another app to handle it.
Example:
If user wants to **show the user a location** on a map, use an **implicit intent** to request that another app shows a specified location on a map.



An Implicit Intent

[1] Activity A **creates an Intent** with an **action description** and passes it to startActivity().

[2] The Android System **searches all apps for an intent filter** that matches the intent.

When a match is found,

[3] the **system starts the matching activity** (Activity B) by invoking its onCreate() method and passing it the Intent.

Building an Intent

An Intent carries information that the Android system uses to determine which component to start and information that the recipient component uses in order to properly perform the action.
An Intent contains the following **information**
1. Component Name
2. Action
3. Data
4. Category
5. Extras
6. Flags

1.Component Name
The name of the component to start.
This field is a ComponentName object, which specify a fully qualified class name of the target component, including the package name of the app.
Example: com.example.ExampleActivity
Set the component name with setComponent(), setClass(), setClassName(), or with the Intent constructor.

2. Action
A string that specifies the generic action to perform (such as *view* or *pick*).
Common Actions for starting an activity:
❖**ACTION_VIEW:**
Use this action in an intent with startActivity() when some information that an activity can show to the user.
Example: View Photo in a gallery app (or) an address to view in a map app.
❖**ACTION_SEND:**
It is like a *share* intent, use this in an intent with startActivity() when you have some data that the user can share through another app.
Example: An email app or Social sharing app.
❖To Define our own actions, include app's package name as a prefix .
Example:static final String ACTION_TIMETRAVEL = "com.example.action.TIMETRAVEL";.
3. Data
The **URI** (a Uri object) that references the data to be acted on and/or the **MIME type** of that data. The type of data supplied is generally dictated by the intent's action.
The following methods are used to supply data to the intent:
1. **setData():**To set only the data URI type.
2. **setType():** To set only the MIME type, call.
3. **setDataAndType():** Set both explicitly .

4. Category
A string containing additional information about the kind of component that should handle the intent. Any number of category descriptions can be placed in an intent.
Some common Categories:
❖**CATEGORY_BROWSABLE:**
The target activity allows itself to be started by a web browser to display data referenced by a link, such as an image or an e-mail message.
❖**CATEGORY_LAUNCHER**
The activity is the initial activity of a task and is listed in the system's application launcher.
5. Extras
Key-value pairs to carry addl. info. required to accomplish the requested action.
Add extra data with various **putExtra()** methods, accepting two parameters:
the key name and the value.
Also create a **Bundle object** with all the extra data, then insert the Bundle in
the Intent with putExtras().
The **Intent class** specifies many EXTRA_* constants for standardized data types:
Example: static final String EXTRA_GIGAWATTS =
"com.example.EXTRA_GIGAWATTS";
6. Flags
Flags are defined in the Intent class that function as **metadata** for the intent.
The flags instruct the Android system **how to launch an activity** and **how to treat** it after it's launched.
Example: public Intent setFlags (int flags)

Receiving and Broadcasting Intents

❖ Broadcast intents are **Intent Objects** that are **broadcast via a call** to the **sendBroadcast(), sendStickyBroadcast()** or **sendOrderedBroadcast()** methods of the Activity class.
❖ Broadcast intents are also used to **notify** interested applications about key system events.
❖ A broadcast intent is a **Background Operation** that the user is not normally aware of
Example: The external power supply or headphones being connected or disconnected
❖**Identifying the Broadcast Event :**
The **Action String** identifies the broadcast event and must be unique and normally uses the **application's package name syntax**.
When a broadcast intent is created, it includes an **Action String** along with **Optional Data** and a **Category String**.
a. putExtra() : Data is added to a broadcast intent using **key-value pairs** in conjunction with this method of the intent object.
b. addCategory() : The **Optional Category String** assigned to a broadcast intent via a call
Example:

```
Intent intent = new Intent();
intent.setAction("com.example.Broadcast");
intent.putExtra("MyData", 1000);
sendBroadcast(intent);
```

Example:

```
package com.example.broadcastdetector;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
public class MyReceiver extends BroadcastReceiver {
        public MyReceiver()
{               }
@Override
public void onReceive(Context context, Intent intent) {
           // Implement code here to be performed when broadcast is detected
 } }
```

Starting Component of a Stopped Application

If an intent is to be allowed to start a component of a stopped application, the flag **FLAG_INCLUDE_STOPPED_PACKAGES** can be appended to the intent before it is sent.

Example:

```
Intent intent = new Intent();
intent.addFlags(Intent.FLAG_INCLUDE_STOPPED_PACKAGES);
intent.setAction("com.example.Broadcast");
 intent.putExtra("MyData", 1000);
sendBroadcast(intent);
```

Broadcast Receivers

1. Broadcast Receivers are used to **respond** to these system-wide events.
2. Broadcast Receivers allow us to **register** for the system and application events, and when that event happens, then the register receivers get notified.

There are mainly two types of Broadcast Receivers:

- **Static Broadcast Receivers**:
  These types are declared in the **manifest file** and **works** even if the **app is closed**.
- **Dynamic Broadcast Receivers**
  These types of receivers work only if the **app is active** or minimized.

- An application **listens** for specific broadcast intents by registering a broadcast receiver.
- These are **implemented** by extending the **BroadcastReceiver** class and overriding the **onReceive()** method.
- The broadcast receiver **registered** either within **code** or within a **manifest** file.
- The receiver must listen for certain broadcast intents, which are indicated by **intent filters**.
- When a **matching broadcast** is detected, the **onReceive()** receiver is called.
- A broadcast receiver does not need to be running all the time.
- Android launches the broadcast receiver automatically **after detecting a matching intent** before invoking the onReceive() function.

Note:

Since from **API Level 26**, the broadcast can only be caught by the **dynamic receiver**

Important System-wide Generated Intents

Intent Type	Description
android.action.BATTERY_LOW	Indicates low battery condition on the device.
android.intent.action.BOOT_COMPLETED	This is broadcast once after the system has finished booting
android.intent.action.CALL	To perform a call to someone specified by the data
android.intent.action.DATE_CHANGED	Indicates that the date has changed
android.intent.action.REBOOT	Indicates that the device has been a reboot
android.net.conn.CONNECTIVITY_CHANGE	The mobile network or wifi connection is changed(or reset)
android.intent.ACTION_AIRPLANE_MODE_CHANGED	This indicates that airplane mode has been switched on or off.

a. Registering a Broadcast Receiver

It is registered in a manifest file within a **<receiver>** tag added for the receiver.

Example (manifest file):

```
<?xml version="1.0" encoding="utf-8"?>
 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.broadcastdetector.broadcastdetector"  >
 <uses-sdk android:minSdkVersion="17" />
<application
 android:icon="@mipmap/ic_launcher"
android:label="@string/app_name" >
            <receiver
            android:name="MyReceiver" >
            </receiver>
 </application>
</manifest>
```

b. Unregistered a Broadcast Receiver

When a broadcast receiver is **no longer required**, it is unregistered via a call to the **unregisterReceiver()** by passing a **reference** to the receiver object as an argument.

Example :

```
unregisterReceiver(receiver);
```

Obtaining Results from a Broadcast

sendOrderedBroadcast():
When a broadcast intent is sent using the sendBroadcast() method, return results are accessed through this method.
When a broadcast intent is sent using this method, it is **delivered** in sequential order to each broadcast receiver with a registered interest.
It is **called** with a **number of arguments** to be notified when all other broadcast receivers have handled the intent.

Sticky Broadcast Intents

A normal broadcast reaches the receiver then terminates.
A sticky broadcast remains sticks around so that it can **notify other apps** if they need the same information
Example:
Consider that the battery is fully charged.
When you register a new app that needs to know the information, or when an inactive app is launched, the sticky broadcast will be sent to the new app's receiver.
A new sticky broadcast with updated information on the same topic will rewrite an earlier sticky broadcast.

Example: Broadcast Intent

Creating the broadcast Receiver and how to register them for a particular event and how to use them in the application.
Step 1: Create a New Project
Step 2: Working with the activity_main.xml file
Go to the **activity_main.xml** file and refer to the following code

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity">
</androidx.constraintlayout.widget.ConstraintLayout>
```

Step 3: Working with the MainActivity file

Go to the **MainActivity** file and refer to the following code.

```
import android.app.Activity;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
public class MainActivity extends AppCompatActivity {
        AirplaneModeChangeReceiver airplaneModeChangeReceiver = new
AirplaneModeChangeReceiver();
        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);         }
        @Override
        protected void onStart() {
                super.onStart();
                IntentFilter filter = new
IntentFilter(Intent.ACTION_AIRPLANE_MODE_CHANGED);
                registerReceiver(airplaneModeChangeReceiver, filter);       }
        @Override
        protected void onStop() {
                super.onStop();
                unregisterReceiver(airplaneModeChangeReceiver);
        }  }
```

Step 4: Create a new class:

Go to **app > java > your package name(in which the MainActicity is present) > right-click > New > Java File/Class** and name the files as **AirplaneModeChangeReceiver**.

Below is the code for the **AirplaneModeChangeReceiver** file.

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.provider.Settings;
import android.widget.Toast;
public class AirplaneModeChangeReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
                if (isAirplaneModeOn(context.getApplicationContext())) {
                Toast.makeText(context, "AirPlane mode is on", Toast.LENGTH_SHORT).show();
                }
                else
                {
                Toast.makeText(context, "AirPlane mode is off", Toast.LENGTH_SHORT).show();
                }
        }
        private static boolean isAirplaneModeOn(Context context) {
                return Settings.System.getInt(context.getContentResolver(),
Settings.Global.AIRPLANE_MODE_ON, 0) != 0;
        } }
```

AndroidManifest.xml file

❖Required XML file for all the android application and located inside the **Root** directory.

1. It contains information of the package, including components of the application such as
   1. **Activities**
   2. **Services**
   3. **Broadcast Receivers**
   4. **Content Providers** etc.

❖It is responsible to **Protect the Application** by providing the permissions.

❖It also declares the **Android API** that the application is going to use.

❖It lists the **Instrumentation Classes** which provides **Profiling** and other information

❖**Instrumentation Class** is **Removed** just before the application is published etc.

Elements of the AndroidManifest.xml file

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

2. <application>
It includes the namespace declaration which contains several sub elements that declares the application component such as activity etc.
The commonly used attributes are

- **android:icon** represents the icon for all the components.
- **android:label** works as the **default label** for all the components.
- **android:theme** represents a common theme for all the android activities.

3. <activity>
It represents an activity attributes such as label, name, theme, launchMode etc.

- **android:label** represents a label i.e. displayed on the screen.
- **android:name** represents a name for the activity class. It is required attribute.

4.<intent-filter>
Describes the **type of intent** to which activity, service or broadcast receiver can respond to.

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

6. <category>
It adds a category name to an intent-filter.

A simple AndroidManifest.xml file

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.javatpoint.hello"
   android:versionCode="1"
   android:versionName="1.0" >
    <uses-sdk
     android:minSdkVersion="8"
     android:targetSdkVersion="15" />
    <application
     android:icon="@drawable/ic_launcher"
     android:label="@string/app_name"
     android:theme="@style/AppTheme" >
     <activity
       android:name=".MainActivity"
       android:label="@string/title_activity_main" >
       <intent-filter>
         <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
       </intent-filter>
     </activity>
   </application>
  </manifest>
```

Intent Filters

Intent

The intent is a messaging object which tells **what kind of action to be performed**.

❖**Launching of the Activity**.
❖**Delivering a Broadcast**
❖**Starting a Service**
❖**Communication between the Components**.

Intent Types

i. Explicit Intent

Does the specific application action which is set by the
User knows about all the things like after clicking a button which activity will start
Explicit intents are used for communication inside the application

ii. Implicit Intent

Implicit intents do not name a specific component
Declare general action to perform, which allows a component from another app to handle.

Example:
when you tap the **share** button in any app you can see the Gmail, Bluetooth, and other sharing app options.

Intent Filter

❖Specifies the **types of intents** that an activity, service, or broadcast receiver can respond.
❖intent filter is used by **implicit intent** to serve the user request.
❖Intent filters are declared in the **Android manifest file**.
❖Intent filter must contain **<action>** tag

Example:

```
<activity android:name=".MainActivity">
   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
      <data android:mimeType="text/plain"/>
   </intent-filter>
</activity>
```

Intent filter are describe by its

1. **<action>**
2. **<category>**
3. **<data>**

1.<action>

Syntax:

```
<action android:name="string" />
```

Adds an action to an intent filter.
An <intent-filter> element must contain **one or more <action> elements**.
If there are **no <action> elements** in an intent filter, the filter doesn't accept any Intent objects.

Examples:

- ❖ **ACTION_VIEW:** An Activity can show to the users like **showing an image in a gallery app** (or) **an address to view in a map app**
- ❖ **ACTION_SEND:** You should use this in intent with **startActivity()** when you have some data that the user can **share through another app**, such as an **email app or social sharing app.**

2. <category>

Syntax:
<category android:name="string" />
Adds a category name to an intent filter.
A string containing additional information about **the kind of component that should handle the intent.**
Example:
CATEGORY_BROWSABLE: The target activity **allows itself** to be started by a web browser to display data referenced by a link.

3. <data>

Syntax:

```
<data android:scheme="string"
  android:host="string"
  android:port="string"
  android:path="string"
  android:pathPattern="string"
  android:pathPrefix="string"
  android:mimeType="string" />
```

Adds a data specification to an intent filter.
The specification can be a data type, a URI, or both a data type and a URI.

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

Call startActivity method to start an email activity and is shown below

Send email via:
Email
Gmail

Multiple Intent Filters in Manifest File

```
<activity android:name=".MainActivity">
  <!-- This activity is the main entry, should appear in app launcher -->
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
    <data android:mimeType="text/plain"/>
  </intent-filter>
</activity>
<activity android:name=".ResultActivity">
  <!-- This activity handles "SEND" actions with text data -->
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="text/plain"/>
  </intent-filter>
</activity>
```

The activity “**MainActivity**” will act as an **entry point** for our app and
The second activity “**ResultActivity**” is intended to help us to **share the text**.

Permissions on Android

App permissions help support user privacy by protecting access to the following

1. **Restricted Data:** System State and Users' Contact Information
2. **Restricted Actions:** Connecting to a Paired Device and Recording Audio

High-level Workflow for using Permissions

If an app require access to restricted data or restricted actions, determine whether you can get the information or perform the actions without needing to declare permissions.

Use Cases to be fulfill in the App without Declaring any permissions :

- ❖**Taking Photos**
- ❖**Pausing Media Playback**
- ❖**Displaying Relevant Ads.**

If an app must access restricted data or perform restricted actions to fulfill a use case, declare the appropriate permissions.

- ❖**Install-time Permissions:** Automatically granted when your app is installed
- ❖ **Runtime Permissions:** Requires app to request the permission at runtime.

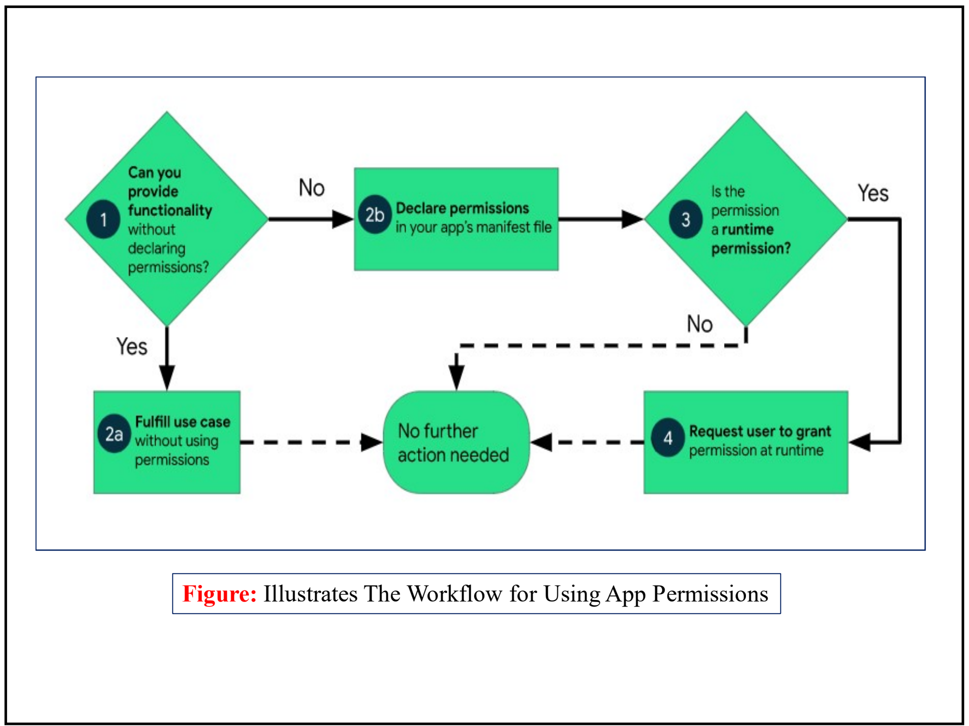
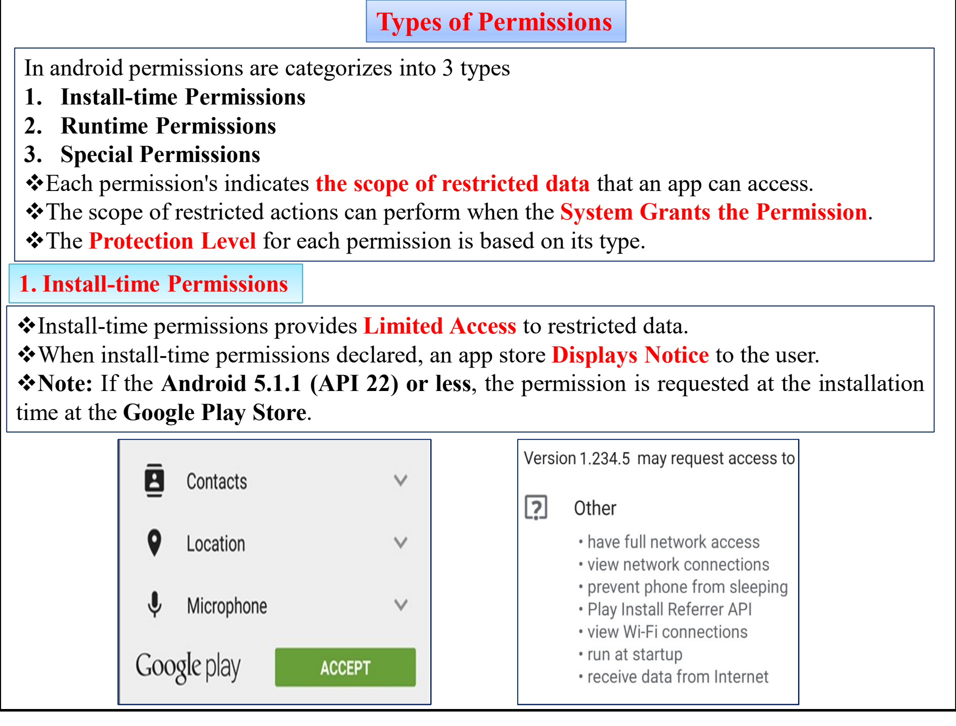


Figure: Illustrates The Workflow for Using App Permissions



Types of Install-time Permissions

1. **Normal Permissions**

   Allow access to data and actions that extend beyond your app's sandbox.

Note:
- ❖ **Risk** to the **User's Privacy** and the operation of other apps.
- ❖ The system assigns the **Normal Protection Level** to these permissions.

2. Signature Permissions

The system grants a signature permission to an app only when the app is signed by the **Same Certificate** as the app that defines the permission.

Example:
Autofill or VPN Services

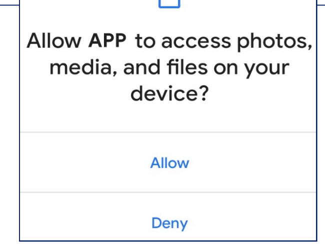
Note:
These apps require **Service-binding** signature permissions so that only the system can bind to the services.
The system assigns the **Signature Protection Level** to signature permissions.

2. Runtime Permissions

- ❖ Runtime permissions **(Dangerous Permissions)** give **additional access** to restricted data or perform restricted actions which affect the system and other apps.
- ❖ Before accessing the restricted data or perform restricted actions user need to request runtime permissions in the app.
- ❖ Check the permissions before each access.
- ❖ When an app requests a permission, the system **displays** a runtime permission **prompt**.
- ❖ Many runtime permissions access **Private User Data**
- ❖ The microphone and camera provide access to sensitive information.

Example:
- ❖ **Location and Contact Information.**

Note:
1. The system assigns the **Dangerous Protection Level** to runtime permissions.
2. If the **Android 6 (API 23) or higher**, the permission is requested at the run time during the running of the app.



3. Special Permissions

- ❖Special permissions correspond to **Particular App Operations**.
- ❖Only the **platform** and **OEMs** can define special permissions.
- ❖ The **Platform** and **OEMs** define special permissions when they want to protect access to particularly powerful actions (drawing over other apps).
- ❖ The **Special App Access** in system settings contains a set of user-toggleable operations.
- ❖Many of these operations are implemented as special permissions.
- ❖Each special permission has its own implementation details.

Note:

The system assigns the **Appop Protection Level** to special permissions.

Steps for Requesting Permissions at Run Time

Step 1:
Declare the **Permission in the Android Manifest File** in the AndroidManifest.xml file using the **<Uses-permission> Tag.**
<uses-permission android:name=”android.permission.PERMISSION_NAME”/>
Step 2:
Modify activity_main.xml file to **Add Two Buttons** to request permission on button click.
Step 3:
Check whether permission is already granted or not. If permission isn’t already granted, request the user for the permission.
Step 4:
Override **onRequestPermissionsResult()** method.

A. Check for Permissions
Beginning with Android 6.0 (API level 23), the user has the right to revoke permissions from any app at any time, even if the app targets a lower API level.
So to use the service, the app needs to check for permissions every time.
B. Request Permissions
When **PERMISSION_DENIED** is returned from the **checkSelfPermission()** method

Android Resource Types

1. Drawable Resources
2. Color State List Resources
3. Animation Resources
4. Layout Resources
5. Menu Resources
6. Style Resource
7. String Resources
8. Others

Resource Types

Layout | Animation | Menu | Style | String | Drawables | Color state list | Others

1. Drawable Resources

These resources define the **Graphics** of application with xml or bitmaps.
They are accessed from **R.drawable** class and saved in **res/drawable/** folder.

Different types of drawables are as follows:

i. Nine-Patch File

- This is a PNG file which has stretchable regions.
- With this image can be resized according to content.

ii. Layer List

- This is a drawable which manages an array of other Drawables
- They are drawn in array order.
- Element with largest index will be at top.

iii. Level List

- This is an XML file.
- It defines one drawable which manages number of alternate drawables.
- These alternatives are assigned with a maximum number.

iv. Bitmap File

This is a simple bitmap graphic file.

v. Clip Drawable: This XML file defines a drawable that clips another drawable.

vi. Shape Drawable: This XML file defines geometric shape.

vii. Transition Drawable:

- This Xml file deals with the transition.
- It cross-fades between two drawable resources.

2.Color State List Resources

i. A ColorStateList is an object which can be defined in XML.
ii. This can be applied as a color.
iii. Depending on the state of view object to which it is applied the color actually changes.
iv. Each color can be defined in a XML file under **<item /> tag.**
v. So the state list in an XML file can be described.
vi. When state changes, state list is traversed from top to bottom and the most suitable match is picked.

3.Animation Resource

There are two types of animations which an animation resource can refer to and they are:

1. Property Animation:

- An animator is used to set an object's property over a period of time.
- In short we modify the properties of object.

➢2. View Animation:

There are two types of animations which can be viewed:

i. Frame animation:

- A sequence of images is displayed in order.

ii. Tween animation:

- An animation is created by performing a series of transformations on a single image.

4. Layout Resource

A layout resource defines the **architecture** for the UI in an Activity or a component of a UI.

5.Menu

Android Menu resource is used to design and define the menu of application.

i. Options Menu
ii. Context Menu
iii. Submenu.

This can be inflated by MenuInflater.
A **MenuInflater** is an object that is able to create Menu from xml resources

6. String Resource

- Android String resource provides text strings for application.
- We have an option to format text and style it as well.

We have three types of resources:

i. String Array: It is an xml resource which provides an array of strings
ii. String: This is an xml resource which provides a single string
iii. Quantity Strings: It is an xml resource. It carries the strings for pluralization.

7. Style Resources

- Style resource is used to define the format and look of user interface.
- An individual view can have a specific style.
- An entire activity or an application can be stylized by manifest file.
- It is nothing but a resource which has to be referenced properly.

8. Other Resources

i. Color:

- It is an xml resource which has a hexadecimal number.
- This number corresponds to a particular color.

ii. Bool:

- This resource carries a color value.

iii. Dimension:

- It contains the value of dimension with the specific unit of measure.

iv. ID:

- This is also an xml resource.
- This is an unique identifier which identifies application resources and components.

v. Integer:

- It is an xml resource which carries an integer value.

vi. Typed Array:

- We can use this as array of drawables.

vii. Integer Array:

- It is an xml resource and it is an array of integers.