

# Mobile Computing

using Android

and

iPhone

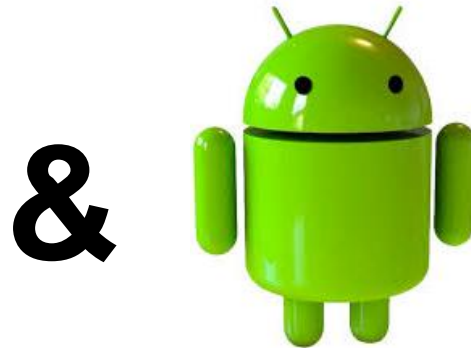


iPhone

# Android

CH - 1

**Introduction to Android**



**Android Application Design**

# Syllabus :

- The Open Handset Alliance
- The Android Platform, Android SDK
- Building a sample Android application
- Anatomy of an Android applications
- Android terminologies
- Application Context, Activities, Services, Intents

# Syllabus :

- Receiving and Broadcasting Intents
- Android Manifest File and its common settings
- Using Intent Filter, Permissions
- Managing Application resources in a hierarchy
- Working with different types of resources

# Android Versions...



**Cupcake**  
Android 1.5



**Donut**  
Android 1.6



**Eclair**  
Android 2.0/2.1



**Froyo**  
Android 2.2.x



**Gingerbread**  
Android 2.3.x



**Honeycomb**  
Android 3.x



**Ice Cream Sandwich**  
Android 4.0.x



**Jelly Bean**  
Android 4.1.x



**KitKat**  
Android 4.4.x



**Lollipop**  
Android 5.0



**Marshmallow**  
android 6.0



**Nougat**  
android 7.0

# Introduction Of Android

- Android is a mobile operating system (OS) based on the Linux kernel and currently developed by Google. With a user interface based on direct manipulation.
- Android delivers a complete set of software for mobile devices, an operating system, middleware and key mobile applications.

# Introduction Of Android

- Android is designed primarily for touch-screen mobile devices such as smartphones and tablet computers, with specialized user interfaces for televisions (Android TV), cars (Android Auto), and wrist watches (Android Wear).

# Introduction Of Android

- The OS uses touch inputs that loosely correspond to real-world actions, like swiping, tapping and reverse pinching to manipulate on-screen objects, and a virtual keyboard.
- Despite being primarily designed for touch-screen input, it also has been used in game consoles, digital cameras, and other electronics.



# Introduction Of Android

- Enter Android, which is a potential (संभवित) game-changer for the mobile development community. An innovative and open platform, Android is well positioned to address the growing needs of the mobile marketplace.

# A Brief History of Mobile Software Development

- To understand what makes Android so compelling (આકર્ષક), we must examine how mobile development has evolved and how Android differs from competing (હરીફાઈ) platforms.
- Before Android, mobile developers faced many roadblocks when it came to writing applications.

# A Brief History of Mobile Software Development

- Building the better application, the unique application, the competing application, the hybrid application and incorporating (समाविष्ट) many common tasks such as messaging and calling in a familiar (परिचित) way were often unrealistic (अवास्तविक) goals.

# A Brief History of Mobile Software Development

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

# A Brief History of Mobile Software Development

- “The Brick”
  - The Motorola DynaTAC 8000X was the first commercially available cell phone.



# A Brief History of Mobile Software Development

- “The Brick”
  - First marketed in 1983, it was 13 × 1.75 × 3.5 inches in dimension, weighed about 2.5 pounds (1.13 Kg), and allowed you to talk for a little more than half an hour. It retailed for \$3,995, plus hefty (ભારે) monthly service fees and per-minute charges.

# A Brief History of Mobile Software Development

- “The Brick”
  - We called it “The Brick,” and the nickname stuck (અટકી જવું) for many of those early mobile phones we alternatively loved and hated.

# A Brief History of Mobile Software Development

- “The Brick”
  - Early mobile phones were not particularly full featured. 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.



# A Brief History of Mobile Software Development

- The first-generation mobile phones were designed and developed by the handset manufacturers.
- Competition was fierce (ଓଘ) and trade secrets were closely guarded (સાવચેતીપૂર્વક). Manufacturers didn't want to expose the internal workings of their handsets, so they usually developed the phone software in-house.

# A Brief History of Mobile Software Development

- Nokia was famous for putting the 1970s video game Snake on some of its earliest monochrome phones.
- These early phones were flawed (ખામીયુક્ત), but they did something important—they changed the way people thought about communication.

# A Brief History of Mobile Software Development

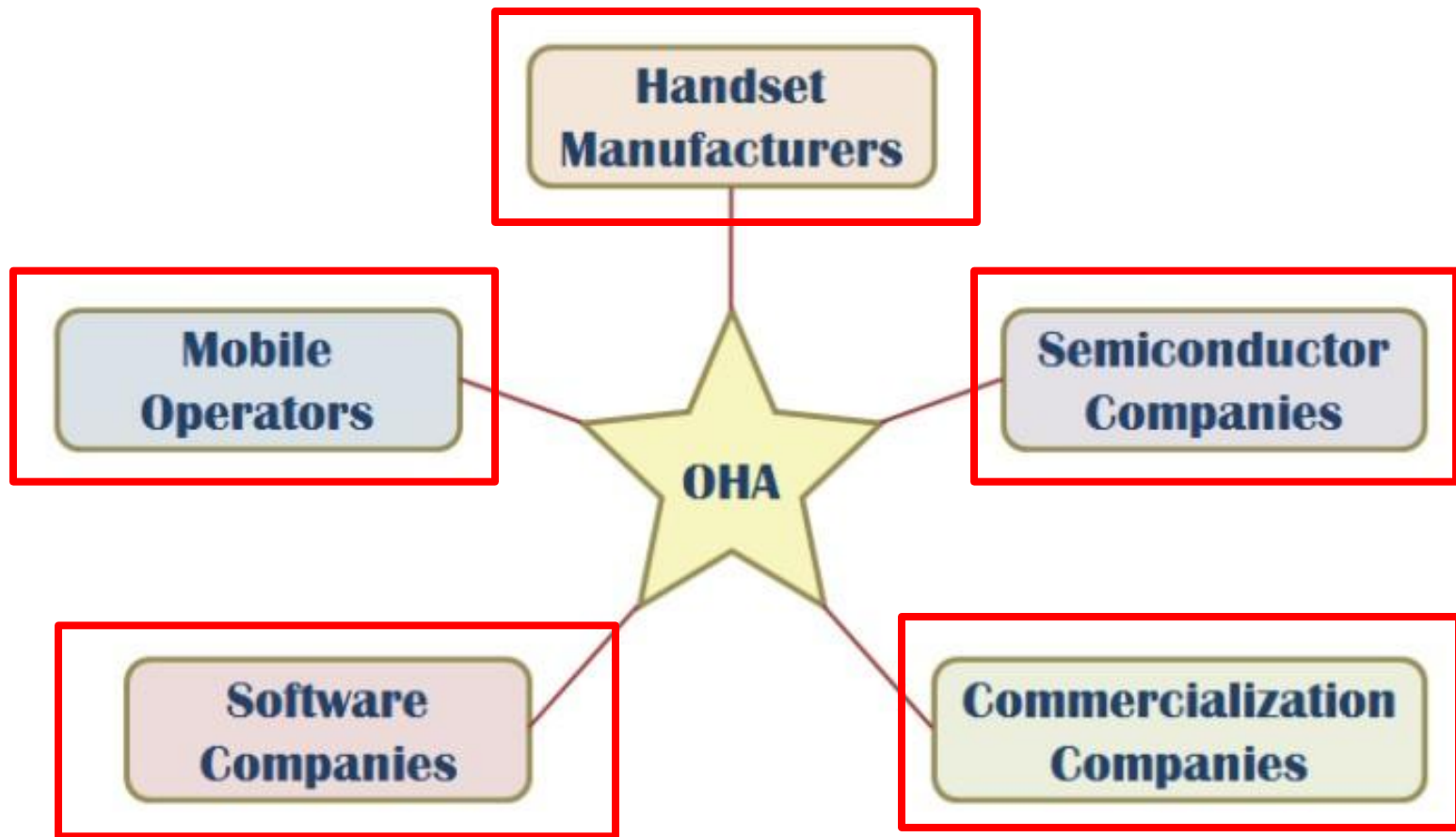
- As mobile phone prices dropped, batteries improved and reception areas grew, more and more people began carrying these handy devices.
- Customers began pushing for more features and more games. But there was a problem.

# A Brief History of Mobile Software Development

- What better way to provide these services than the Internet?
  - Wireless Application Protocol (WAP).
  - Wireless Markup Language (WML).
  - Short Message Service (SMS)
  - Multimedia Messaging Service (MMS)
  - Enhanced Messaging Service (EMS)

# The Open Handset Alliance

- OHA is a group of 84 companies such as Google, Samsung, HTC, Sony, KDDI, Garmin, Teleca, Ebay, Intel etc.



# The Open Handset Alliance

- Forming the Open Handset Alliance :
  - 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(બેફત) access to applications and services.

# The Open Handset Alliance

- Its vast (ବିଶାଳ) resources, Google has taken a broad approach, examining the wireless infrastructure from the wireless spectrum (used to classify something) policies to the handset manufacturers' requirements, application developer needs, and mobile operator desires.

# The Open Handset Alliance

- ❑ 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) was formed in November 2007 to answer that very question.



# The Open Handset Alliance

- ❑ The OHA is a business alliance comprised (સમાવેલ છે) 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.

# The Open Handset Alliance

- ❑ Andy Rubin has been credited as the father of the Android platform, His company, Android Inc., was acquired by Google in 2005
- ❑ OHA members, including Google, began developing a non-proprietary open standard platform based upon technology developed at Android Inc.

# The Open Handset Alliance

- That would aim to alleviate the aforementioned problems hindering the mobile community.
- The result is the Android project. To this day, most Android platform development is completed by Rubin's team at Google, where he acts as VP of Engineering and manages the Android platform roadmap.

# The Open Handset Alliance

- ❑ Google's involvement in the Android project has been so extensive that the line between who takes responsibility for the Android platform (the OHA or Google) has blurred.
- ❑ Google hosts the Android open source project and provides online Android documentation, tools, forums, and the Software Development Kit (SDK) for developers.

# The Open Handset Alliance

- ❑ All major Android news originates at Google.
- ❑ The company has also hosted a number of events at conferences and the Android Developer Challenge (ADC), a contest to encourage developers to write killer Android applications for 10 million dollars in prizes to spur development on the platform.

# The Open Handset Alliance

- The winners of 10 million dollars in prizes and their apps are listed on the Android website.

# The Android Platform :

- 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.

# The Android Platform :

## ■ **Open:**

- The Android platform is provided through open source licensing.

Developers have unprecedented access to the handset features when developing applications.



# The Android Platform :

## ■ 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
- Since the Android 1.0 SDK was released, Android platform development has continued at a fast and furious pace.

# Android: A Next-Generation Platform

- For quite some time, there was a new Android SDK out every couple of months! In typical tech-sector jargon, each Android SDK has had a project name. In Android's case, the SDKs are named alphabetically after sweets. The latest version of Android is codenamed Gingerbread..

# VERSIONS OF ANDROID

<b>Version</b>	<b>Code name</b>
1.5	Cupcake
1.6	Donut
2.1	Eclair
2.2	Froyo
2.3	Gingerbread
3.1 and 3.3	Honeycomb
4.0	Ice Cream Sandwich
4.1, 4.2 and 4.3	Jelly Bean
4.4	KitKat
5.0	Lollipop

# VERSIONS OF ANDROID

- **Android Alfa & Apple Pie**
  - Released on September 23 , 2008
  - First Version of Android.
  - The focus of Android Alfa is testing incorporating usability.
  - Android Alfa will generally have many more problems on speed and performance.



# VERSIONS OF ANDROID

- Android Beta & Banna Bread
  - First full version of android.
  - Released on February 09, 2009.
  - Wi-Fi and Bluetooth support.
  - Quite slow in operating.
  - Copy and paste feature in the web browser is not present.



# VERSIONS OF ANDROID

## ■ Android Cupcake 1.5

- ❑ Released on April 30, 2009.
- ❑ Added auto-rotation option.
- ❑ Copy and Paste feature added in the web browser.
- ❑ *Increased speed and performance but not up required level.*



# VERSIONS OF ANDROID

## ■ Android Donut 1.6

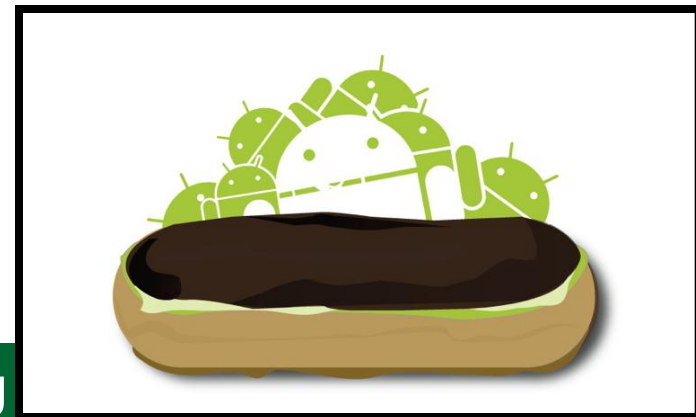
- ❑ Released on September 15, 2009.
- ❑ Voice search and Search box were added.
- ❑ Faster OS boot times and fast web browsing experience.
- ❑ *Typing is quite slower.*





# VERSIONS OF ANDROID

- **Android Eclair 2.0 & 2.1**
  - Released on October 26, 2009.
  - Bluetooth 2.1 support.
  - Improved typing speed on virtual keyboard, with smarter dictionary.
  - *No Adobe flash media support.*



# VERSIONS OF ANDROID

## ■ Android Froyo 2.2

- ❑ Released on May 20, 2010.
- ❑ Support for Adobe Flash 10.1
- ❑ Improved Application launcher with better browser
- ❑ *No internet calling.*



# VERSIONS OF ANDROID

- Android Gingerbread 2.3
  - ❑ Released on December 6, 2010.
  - ❑ Updated User Interface with high efficiency and speed
  - ❑ Internet calling
  - ❑ One touch word selection and copy/paste.



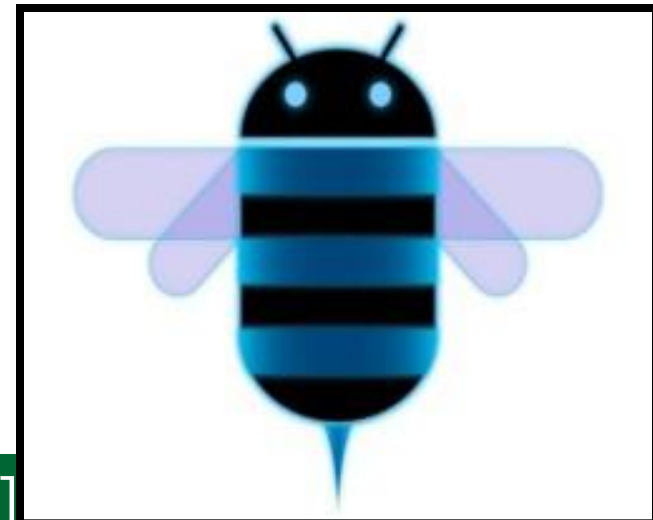
# VERSIONS OF ANDROID

- Android Gingerbread 2.3
  - New keyboard for faster word input.
  - More successful version of Android than previous versions.
  - Not supports multi-core processors.



# VERSIONS OF ANDROID

- Android Honeycomb 3.0 TO 3.2
  - Released on February 22, 2011.
  - Support for multi-core processors
  - Ability to encrypt all user data.
  - *This version of android is only available for tablets.*



# VERSIONS OF ANDROID

- Android Ice-Cream Sandwich(ICS) 4.0
  - Released on November 14, 2011.
  - Virtual button in the UI.
  - A new typeface family for the UI.
  - Ability to shut down apps that are using data in the background.



# VERSIONS OF ANDROID

- Android JellyBean 4.1 TO 4.3
  - Released on July 09, 2012  
TO July 24, 2013
  - Photo Sphere enhancements
  - 4K resolution support
  - Bluetooth Low Energy (BLE) support.



# VERSIONS OF ANDROID

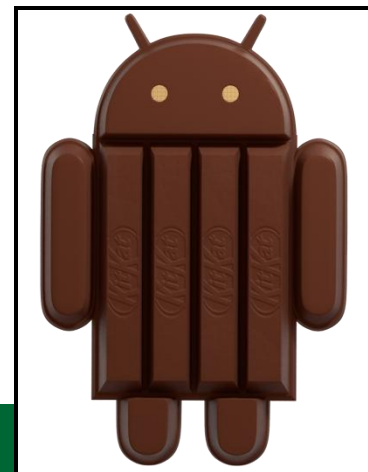
- Android JellyBean 4.1 TO 4.3
  - Hebrew & Arabic right-to-left (RTL) support
  - Camera app UI updated
  - Lock screen widgets
  - 360 degree images with Photo Sphere
  - Google Now
  - **Accessibility** : gesture mode , enable external keyboards.



# VERSIONS OF ANDROID

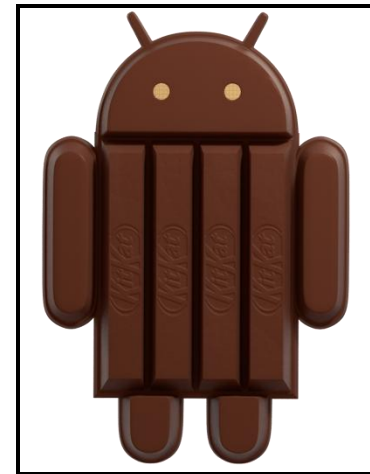
## ■ Android Kitkat 4.4 TO 4.4.4

- ❑ Released on October 31, 2013 TO June 23, 2014
- ❑ Screen Recording
- ❑ New Translucent system UI
- ❑ Enhanced notification access
- ❑ System-wide settings for closed captioning.



# VERSIONS OF ANDROID

- Android Kitkat 4.4 TO 4.4.4
  - Performance improvements
  - Enhance the camera on the Nexus 5
  - Fix Heartbleed / OpenSSL vulnerability (જાબભાઈ).



# VERSIONS OF ANDROID

- Lollipop 5.0 , 5.0.1 , 5.0.2 , 5.1& 5.1.1
  - Released on October 17, 2014 TO April 21, 2015
  - Speed improvement
  - Bug fixes
  - Multiple SIM cards support
  - Lock protection if lost or stolen
  - High Definition voice call



# VERSIONS OF ANDROID

- Lollipop 5.0 , 5.0.1 , 5.0.2 , 5.1& 5.1.1
  - Stability & performance enhancements
  - Performance improvements and bug fixes.
  - Quick settings shortcuts to join Wi-Fi networks or control Bluetooth devices.

# VERSIONS OF ANDROID

- Android Marshmallow 6.0 & 6.0.1
  - Released on October 05, 2015 & December 07, 2015
  - USB Type-C support
  - Fingerprint Authentication support
  - Better battery life with "deep sleep"
  - Permissions dashboard
  - Android Pay



# VERSIONS OF ANDROID

- Android Marshmallow 6.0& 6.0.1
  - MIDI support
  - Google New improvements



# VERSIONS OF ANDROID



## ■ Android Nougat 7.0

- ❑ Released on August 22, 2016.
- ❑ Unicode 9.0 emoji
- ❑ Better multitasking
- ❑ Multi-window mode (PIP, freeform window)
- ❑ Seamless system updates (with dual system partition)
- ❑ Better performance & code size thanks to new JIT Compiler

# VERSIONS OF ANDROID



## ■ Android Nougat 7.1

- ❑ Released on October 04, 2016.
- ❑ Daydream Virtual Reality mode
- ❑ Night Light
- ❑ Storage manager improvements
- ❑ Performance improvements for Touch & Display managements
- ❑ Options to enable fingerprint swipe down gesture
- ❑ Seam-less system updates



# Android : 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).

# Android : Free and Open Source

- A strong “copyleft” license where any third-party improvements must continue to fall under the open source licensing agreement terms.
- Android application developers have the ability to distribute their applications under.

# Android : Free and Open Source

- Whatever licensing scheme they prefer.
- Developers can write open source freeware or traditional licensed applications for profit and everything in between.

# Android : Familiar and Inexpensive Development Tools

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

# Freely Available Software Development

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

# Freely Available Software Development

- Familiar Language, Familiar Development Environments
  - Windows XP (32-bit) or Vista (32-bit or 64-bit)
  - Mac OS X 10.5.8 or later (x86 only)
  - Linux (tested on Linux Ubuntu 8.04 LTS, Hardy Heron)

# Freely Available Software Development

- Familiar Language, Familiar Development Environments
- Reasonable Learning Curve for Developers
- Enabling Development of Powerful Applications
- Rich, Secure Application Integration
- No Costly Obstacles (અવરોધો) to Publication

# Freely Available Software Development

- A “Free Market” for Applications
  - Store limitations on the number of competing applications of a given type.
  - Store limitations on pricing, revenue models, and royalties.
  - Operator unwillingness (અનિચ્છા) to provide applications for smaller demographics (વસ્તી વિષયક વિષય).



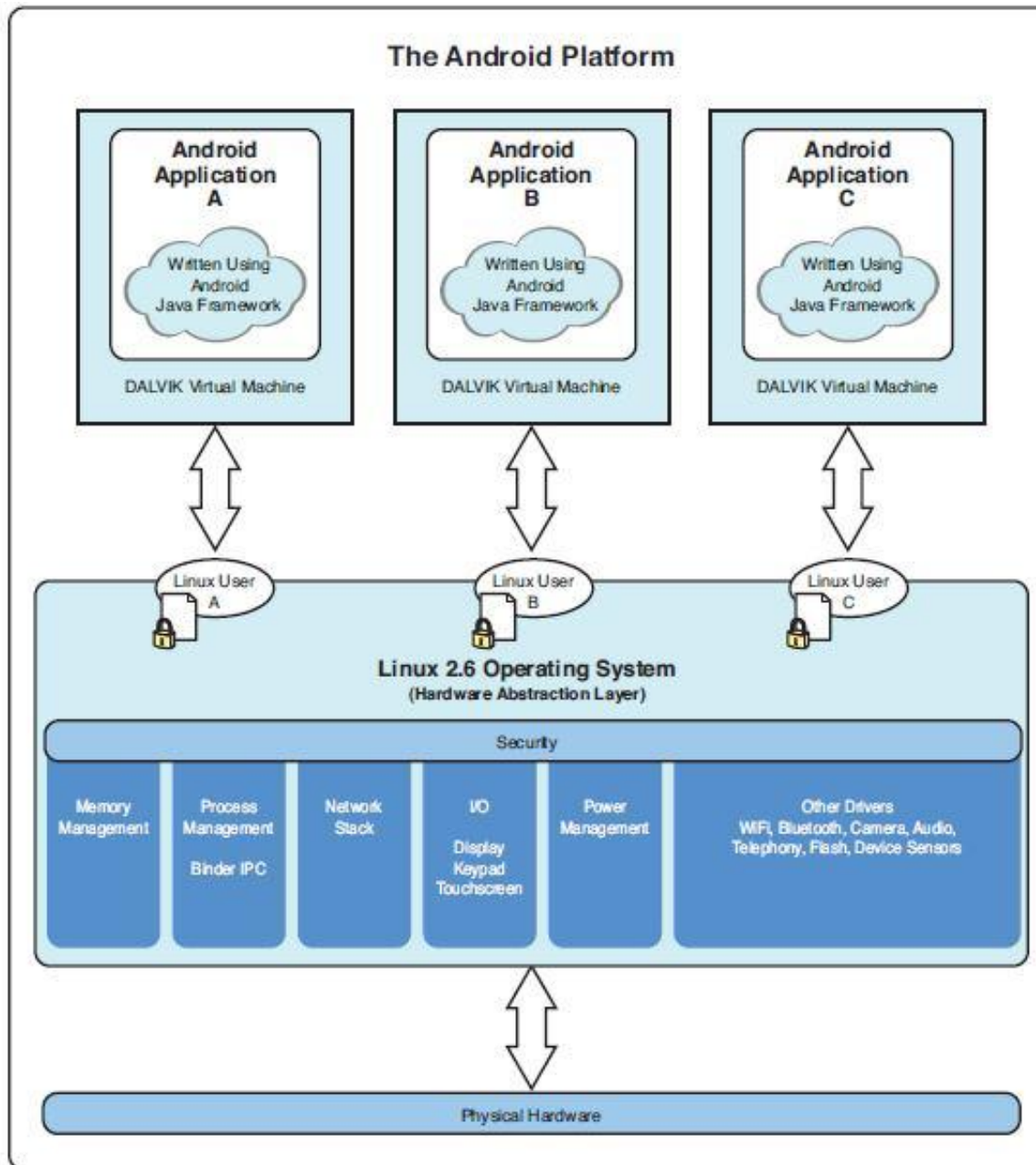
# The Android Platform :

- 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.

# The Android Platform :

- 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.

# The Android Platform :



# Android's Underlying Architecture :

- The Android platform is designed to be more fault-tolerant (सहनशील) than many of its predecessors (पुरोगामी).
- 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's Underlying Architecture :

- The Linux Operating System
  - The Linux 2.6 kernel handles core system services and acts as **hardware abstraction layer (HAL)** between the physical hardware of the handset and the Android software stack.

# Android's Underlying Architecture :

- Some of the core functions the kernel handles include Enforcement of application permission and security, Low-level memory management, Process management and threading, The network stack n Display, Keypad input, camera, Wi-Fi, Flash memory, audio and binder driver access.

# Android Application runtime Environment

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

# Security and Permissions :

- The integrity of the Android platform is maintained through a variety of security measures.
- These measures help to ensure that the user's data is secure and that the device is not subjected to malware.



# 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.

# Applications as Operating System Users

- 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 Permission

- To access shared resources on the system, Android application 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.

# Explicitly Defined Application Permission

- Applications also require permission to access shared data containing private and personal information, such as user preferences, user's location and contact information.
- The application can declare any number of different permission types, such as read-only or read write permissions, for finer control over the application.

# Marketplace Developer Registration

- To publish applications on the popular Android Market, developers must create a developer account.
- The Android Market is managed closely and no malware is tolerated (सह्यन्).

# Android Programming Languages Choices

- Android applications are written in Java.
- For now the Java language is the developer's only choice on the Android platform.

# Commonly Used Packages :

- 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 (including SGL and OpenGL ES) sensors

# Commonly Used Packages :

- Audio and visual media formats (MPEG4, MP3, Still images)
- Access to optional hardware such as location-based services (LBS), Wi-Fi, Bluetooth, and hardware sensors



# Android application framework :

- 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 platform gives developers a wide range of ways to make use of internet access.
- Some offer high-level access, such as the integrated WebKit browser component.
- Android Platform release 2.0 support the SDK version 5.

# Android SDK :

- The Android SDK is more than a library of Java classes and API calls. It also includes a number of tools to assist in application development.

# Android SDK Features :

- No licensing, distribution or development fees or release approval processes.
- Wi-Fi hardware access.
- GSM, EDGE and 3G networks for telephony or data transfer, enabling you to make or receive calls or SMS messages, or to send and retrieve data across mobile networks.

# Android SDK Features :

- No licensing, distribution or development fees or release approval processes.
- Wi-Fi hardware access.
- GSM, EDGE and 3G networks for telephony or data transfer, enabling you to make or receive calls or SMS messages, or to send and retrieve data across mobile networks.

# Android SDK Features :

- Comprehensive APIs for location-based services such as GPS.
- Full multimedia hardware control, including playback and recording with camera and microphone.
- APIs for using sensor hardware, including accelerometers and the compass.

# Android SDK Features :

- Libraries for using Bluetooth for peer-to-peer data transfer.
- IPC (Inter Process Communications) message passing.
- Shared data stores.
- Background applications and processes.
- Home-screen Widgets, Live Folders and Live Wallpaper.

# Android SDK Features :

- The ability to integrate application search results into the system search.
- An integrated open source HTML5 WebKit-based browser.
- Full support for application that integrate map controls as part of their user interface.
- Media libraries for playing and recording a variety of audio/video or still image formats.



# Versions Of Android SDK :

- Android 1.0 SDK 1
- Android 1.1 SDK 2
- Android 1.5 SDK 3
- Android 1.6 SDK 4
- Android 2.0 SDK 5
- Android 2.0.1 SDK 6
- Android 2.1.x SDK 7
- Android 2.2.x SDK 8

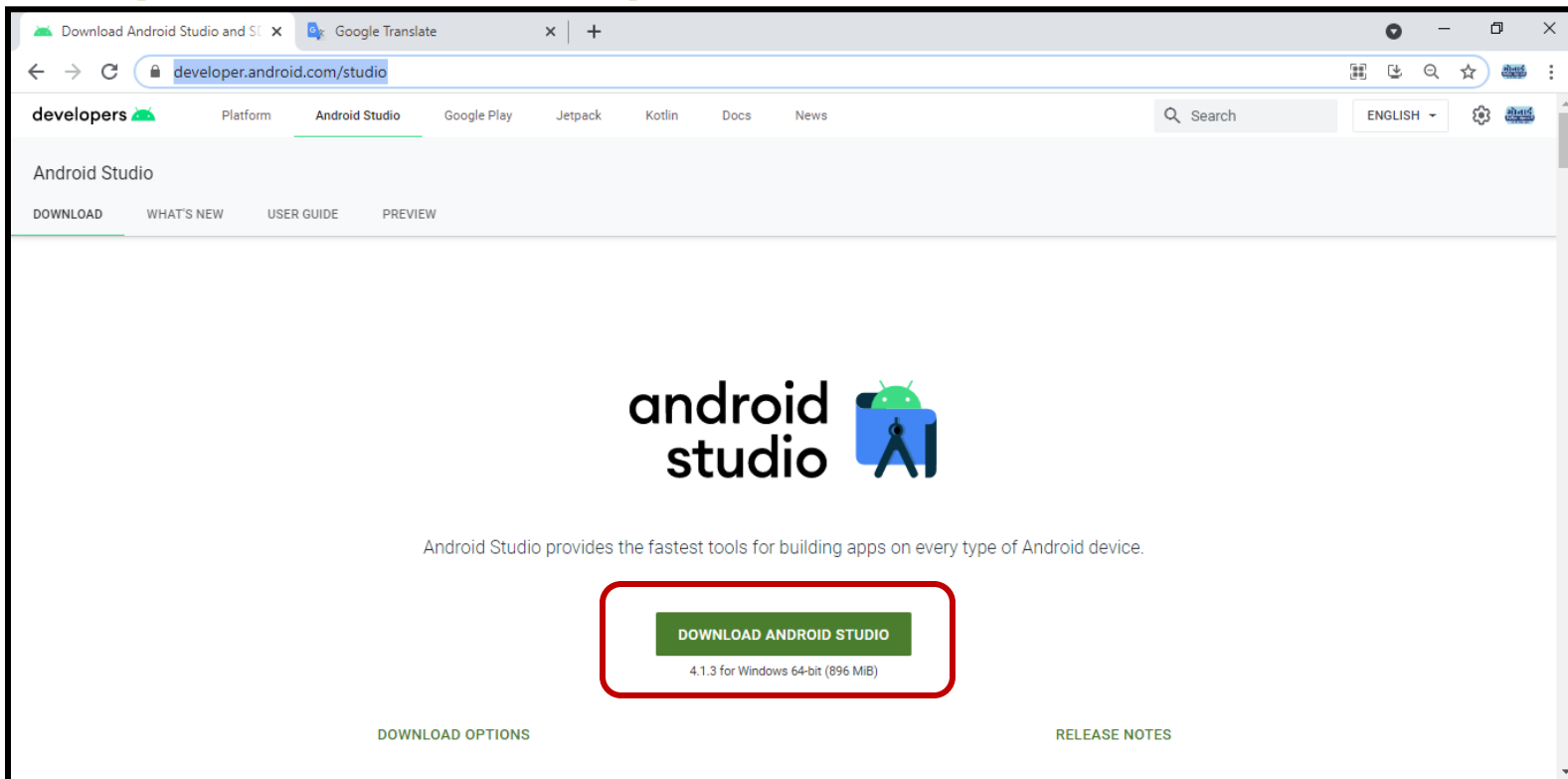
# Versions Of Android SDK :

- Android 2.3, 2.3.1, 2.3.2 SDK 9
- Android 2.3.3, 2.3.4 SDK 10
- Android 3.0.x SDK 11
- Android 3.1.x SDK 12
- Android 3.2 SDK 13
- Android 4.0, 4.0.1, 4.0.2 SDK 14
- Android 4.0.3 SDK 15

# Download And Install ANDROID :

- To install Android studio first of all go to website

<https://developer.android.com/studio>



# Building a sample Android application

- Now double-click on downloaded file...
- Install Android studio...

 android-studio-ide-201.7042882-windows

- After installation of Android Studio,  
**Download** or **Copy** support file for  
Android studio.

- **Copy Folder :**

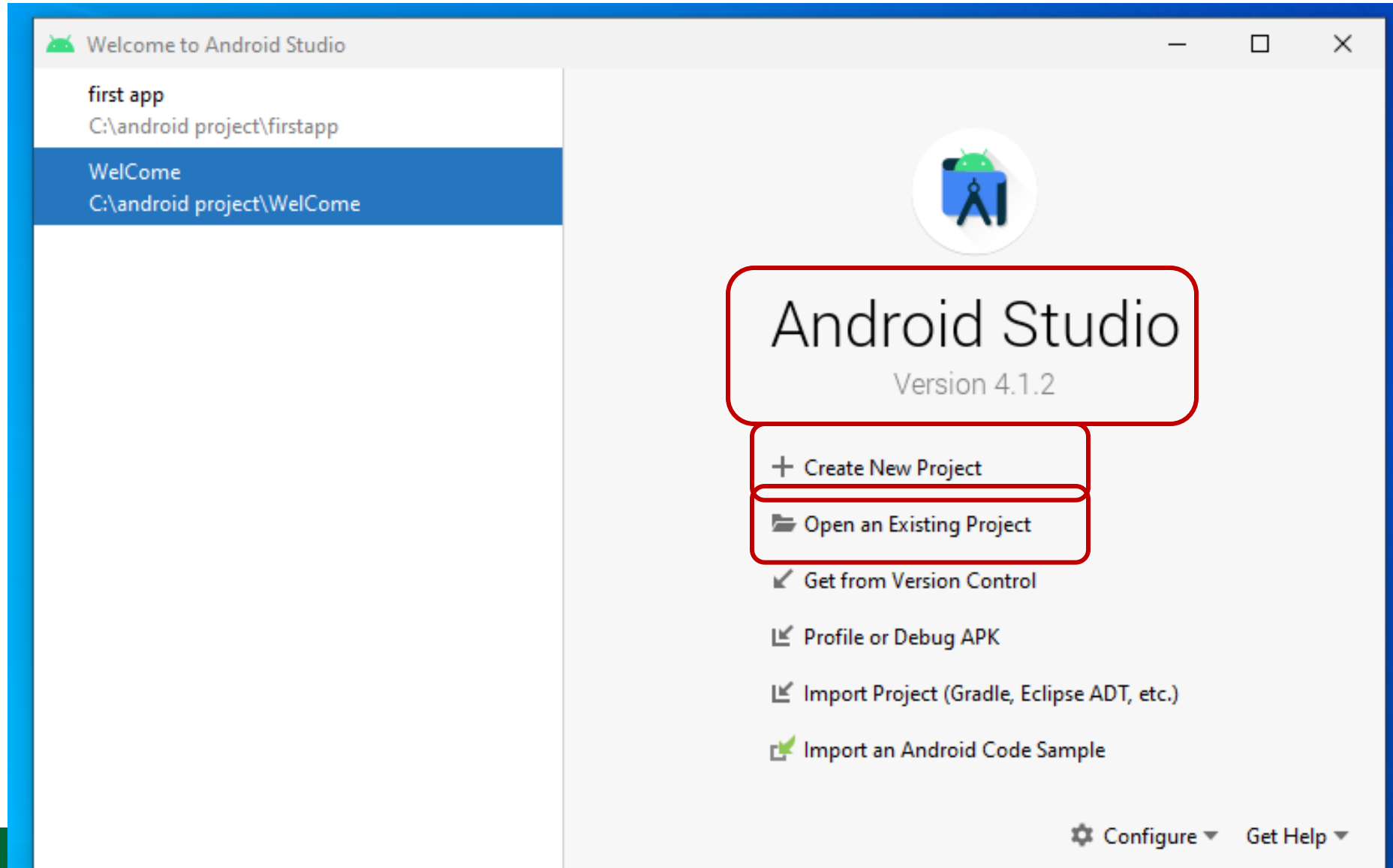
**C:\Users\admin\AppData\Local\Android\Sdk**

# Android IDE First Screen :



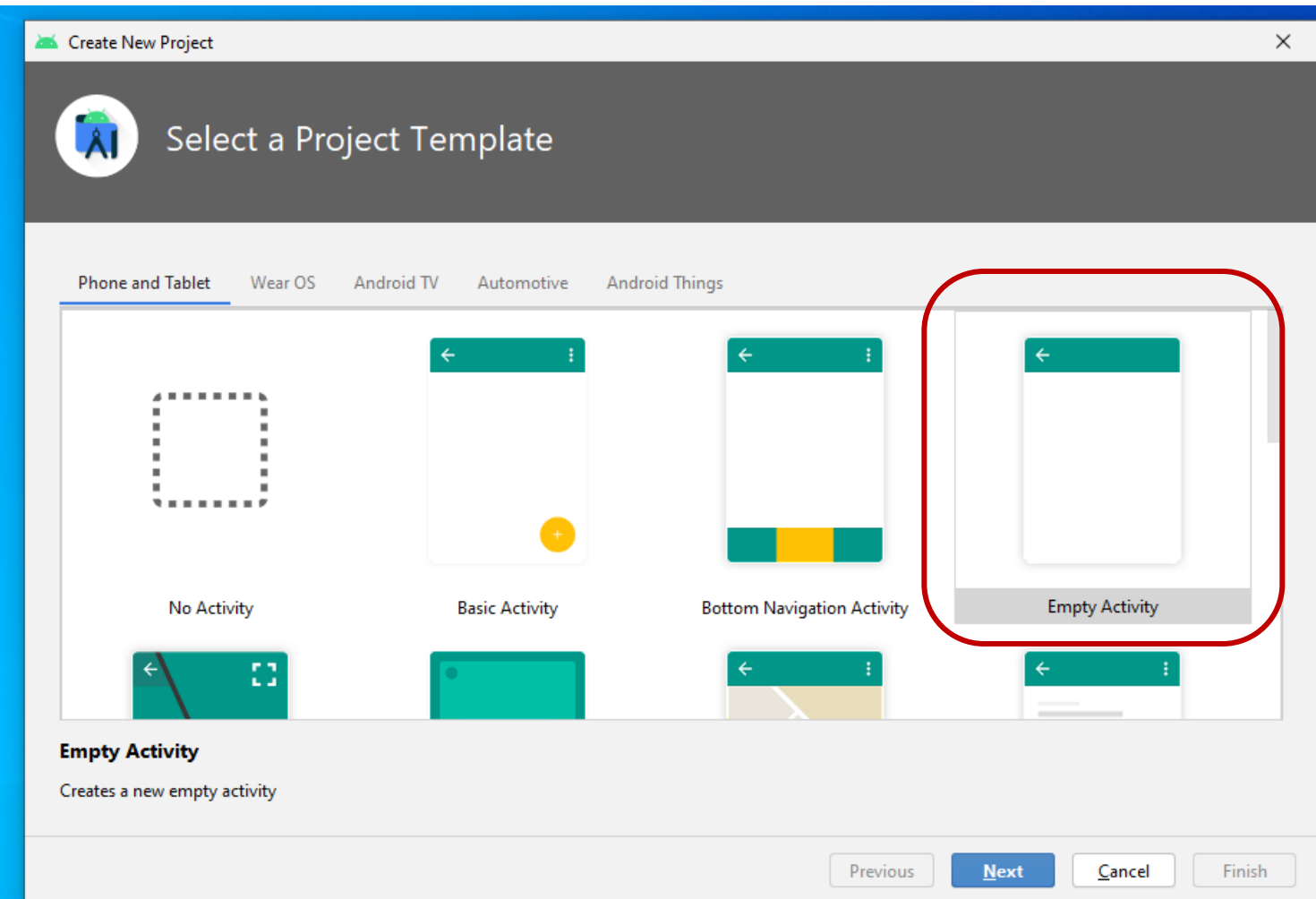
# Android IDE **Second** Screen :

- **To Create or Open** a project...



# Android : Create a New Project

- When we use Create a New Project command we can see following screen...



# Android : Create a New Project

- Click Next button, we can see...

Create New Project

Configure Your Project

←

Empty Activity

Name  
My Application

Package name  
com.example.myapplication

Save location  
C:\android project\MyApplication

Language  
Java

Minimum SDK  
API 16: Android 4.1 (Jelly Bean)

**i** Your app will run on approximately 99.8% of devices.  
[Help me choose](#)

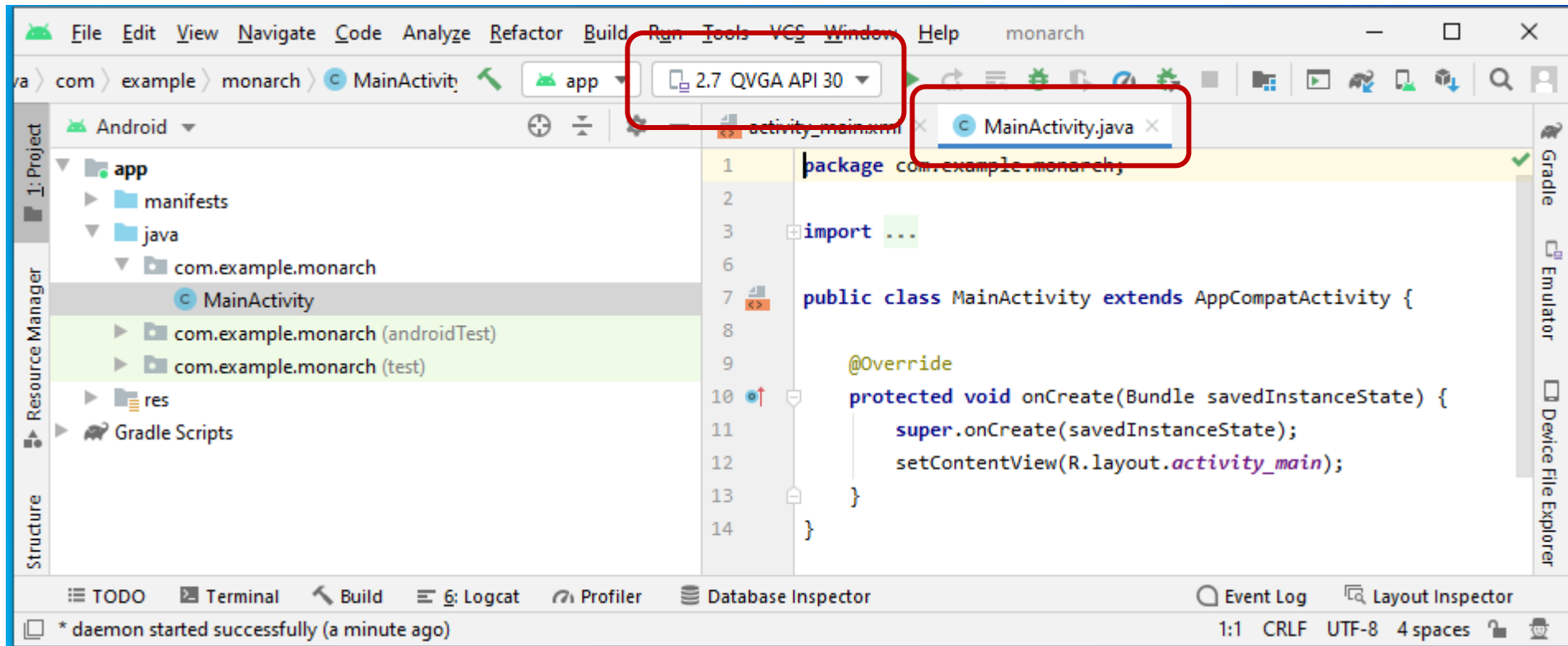
**!** project location should not contain whitespace, as this can cause problems with the NDK tools.

Previous Next Cancel **Finish**



# Android : Create a New Project

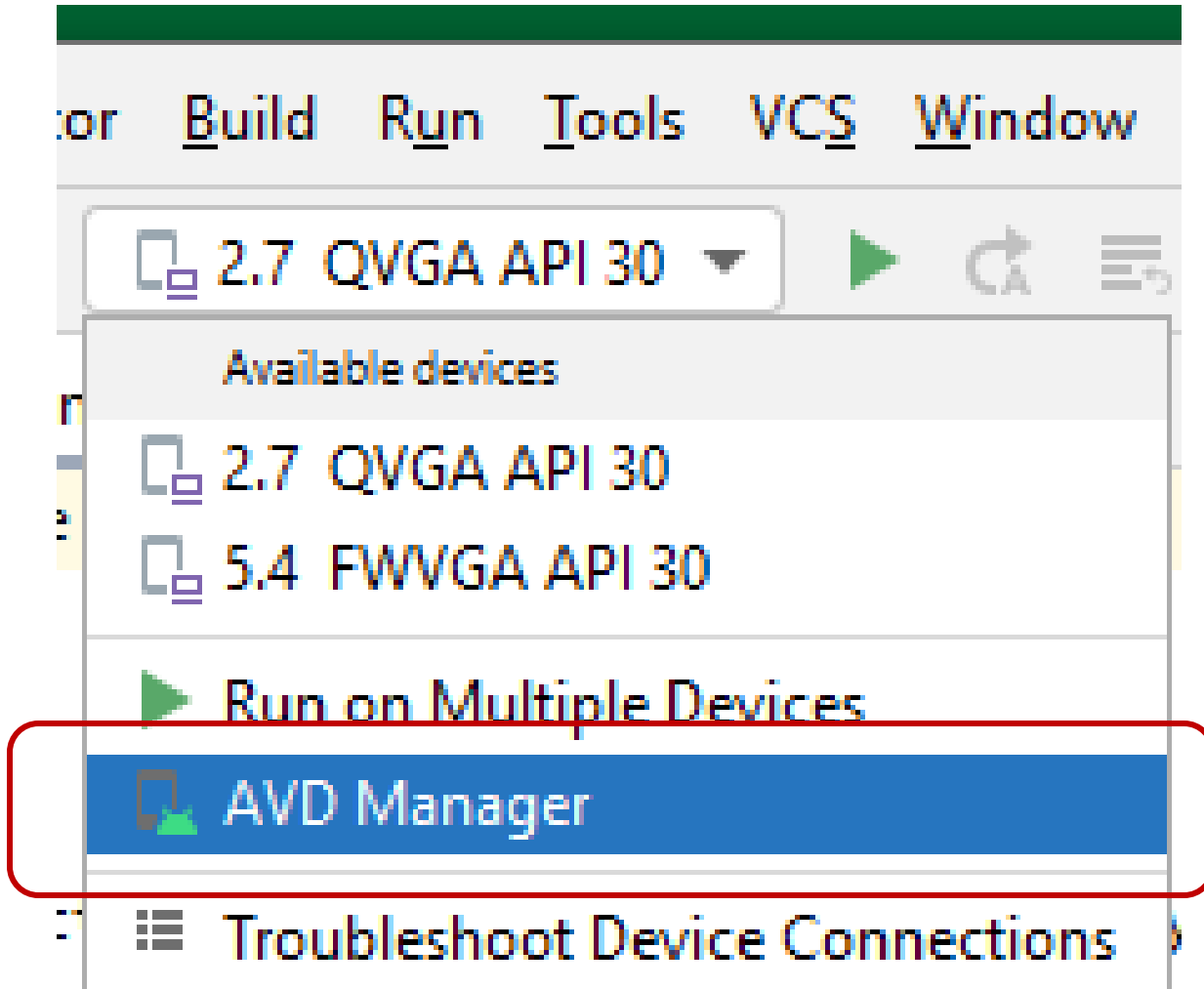
Click Finish button, we can see...



Click to create Mobile using AVD Manager

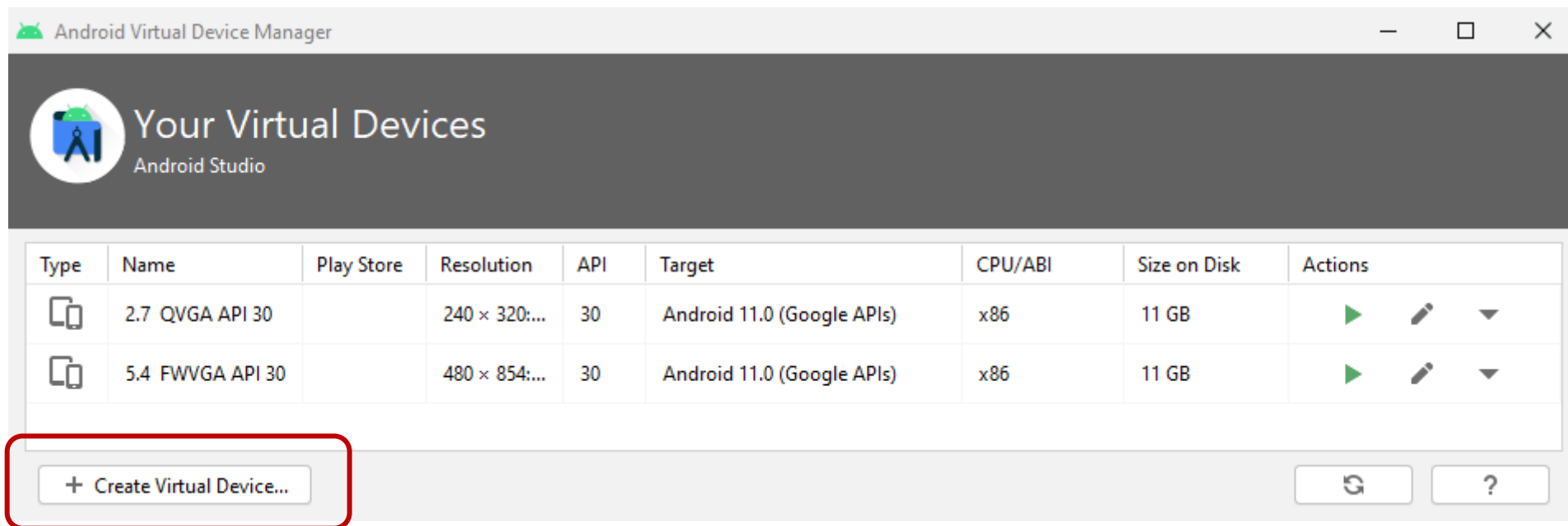
# Android : Create a New Project

- Select AVD Manager...



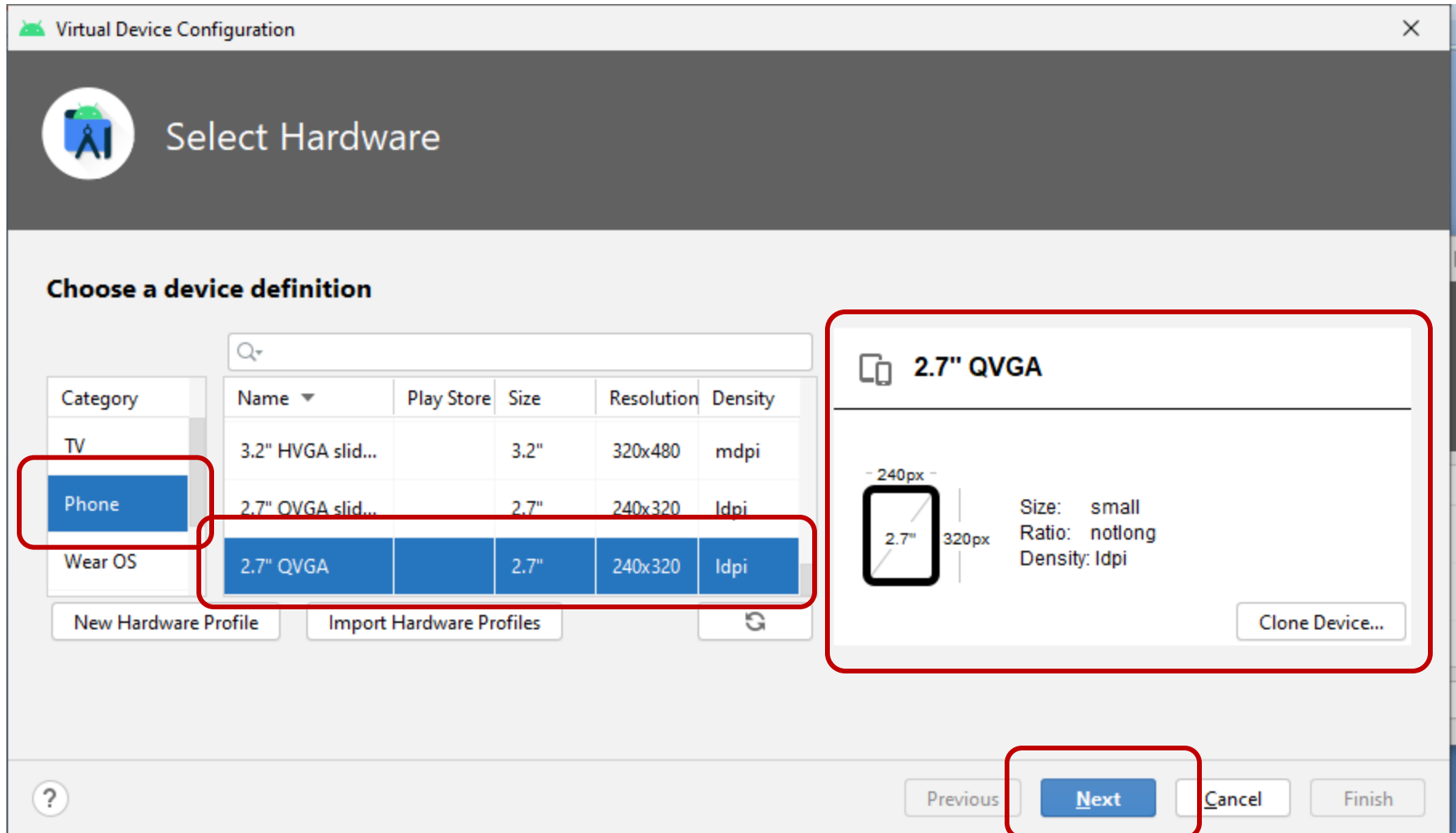
# Android : Create a New Project

- Select AVD Manager...
- Create Your Virtual Devices...



# Android : Create a New Project

## Create Your Virtual Devices...



# Android : Create a New Project

## Select System Image

Virtual Device Configuration


System Image

Select a system image

Recommended x86 Images Other Images

Release Name	API Level	ABI	Target
R	30	x86	Android 11.0 (Google A
<a href="#">Download</a>	29	x86	Android 10.0 (Google AP
<a href="#">Pie Download</a>	28	x86	Android 9.0 (Google API
<a href="#">Oreo Download</a>	27	x86	Android 8.1 (Google API
<a href="#">Oreo Download</a>	26	x86	Android 8.0 (Google API

R



API Level  
**30**

Android  
**11.0**

Google Inc.

Previous Next Cancel Finish

# Android : Create a New Project

## Give a proper name to AVD :

Virtual Device Configuration

Android Virtual Device (AVD)

### Verify Configuration

AVD Name: 2.7 QVGA API 30 2

2.7" QVGA      2.7 240x320 mdpi      Change...

R      Android 11.0 x86      Change...

Startup orientation

Portrait      Landscape

Show Advanced Settings

AVD Name

The name of this AVD.

Previous      Next      Cancel      Finish

# Android : Create a New Project

Click on **activity\_main.xml**, we can see

The screenshot shows the Android Studio IDE with the following details:

- Project Structure:** The left sidebar shows the project structure for 'monarch' > 'app' > 'src' > 'main' > 'res' > 'layout'. The 'MainActivity' class is selected under 'com.example.monarch'.
- Code Editor:** The main editor displays the content of 'activity\_main.xml'. The file name 'activity\_main.xml' is highlighted in the editor's title bar. The XML code is as follows:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     tools:context=".MainActivity">
8
9     <TextView
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:text="Hello World!"
13         app:layout_constraintBottom_toBottomOf="parent"
14         app:layout_constraintLeft_toLeftOf="parent"
15         app:layout_constraintRight_toRightOf="parent"
16         app:layout_constraintTop_toTopOf="parent" />
17
18 </androidx.constraintlayout.widget.ConstraintLayout>
```
- UI Elements:** The 'Code' tab is selected in the editor's top right corner. The 'Design' tab is also visible. The 'Layout Validation' and 'Emulator' panels are visible on the right side of the IDE.
- Bottom Bar:** The bottom bar contains various tool windows: TODO, Terminal, Build, Logcat, Profiler, Database Inspector, Event Log, and Layout Inspector. A status message at the bottom left reads: '\* daemon started successfully (6 minutes ago)'. The bottom right corner shows '1:1 CRLF UTF-8 4 spaces'.

# Android : Create a New Project

## Split View :

The screenshot displays the Android Studio interface with a split view. The left pane shows the XML code for `activity_main.xml`, and the right pane shows the corresponding design view. A red box highlights the `Split` button in the top right corner of the editor area, which is used to toggle between code and design views. The code in the left pane is as follows:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     tools:context=".MainActivity">
8
9
10    <TextView
11        android:layout_width="wrap_content"
12        android:layout_height="wrap_content"
13        android:text="Welcome to my First Application"
14        app:layout_constraintBottom_toBottomOf="parent"
15        app:layout_constraintLeft_toLeftOf="parent"
16        app:layout_constraintRight_toRightOf="parent"
17        app:layout_constraintTop_toTopOf="parent" />
18 </androidx.constraintlayout.widget.ConstraintLayout>
```

The design view on the right shows a white background with a blue text box containing the text "Welcome to my First Application". The `Split` button is located in the top right corner of the editor area, next to the `Code` and `Design` tabs.



# Building a sample Android application

- To building an application in Android, we must have knowledge of some Android commands...
  - Like,
    - Types of Widget...
      - Textview
      - EditText
      - ImageView
      - Button

# TextView Control :

- TextView control is used to display a text message on mobile screen.
- We can use multiple properties to display text on screen.
- Basically main property of TextView Control is **text property** by which we can display any text on mobile screen.
- Example :  
`android:text="Put Text Here"`

# Def.01 WAP to print a Welcome Message.

- **Note :** When you are writing your journals at that times of moment you have to design the layout with proper codes.

# Def.01 coding

## <TextView

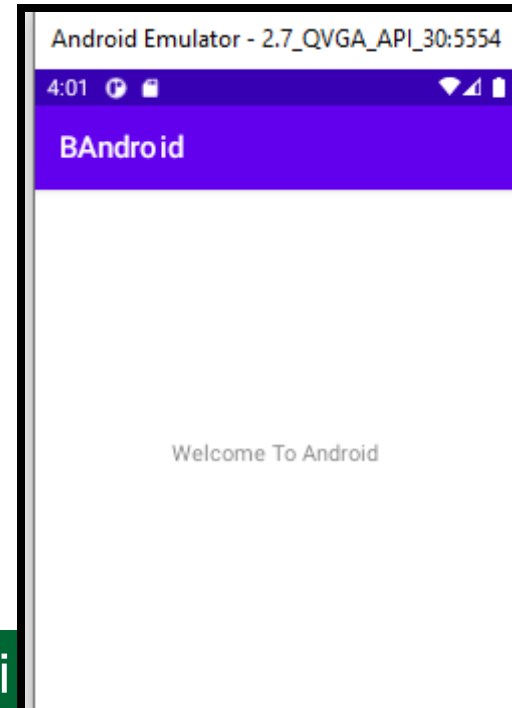
android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Welcome To Android"

app:layout\_constraintBottom\_  
toBottomOf="parent"

app:layout\_constraintLeft\_  
toLeftOf="parent"



# Def.01 coding

```
app:layout_constraintRight_  
        toRightOf="parent"  
app:layout_constraintTop_  
        toTopOf="parent" />
```

# TextView Control :

## ■ **layout\_height property**

□ To Set TextView's height we can use `android:layout_height` property.

□ We can set height as,

`layout_height="match_parent"`

`layout_height="100dp"`

# TextView Control :

## ■ **layout\_width property**

- To Set TextView's width we can use

android:layout\_width property.

- We can set width as,

layout\_width="match\_parent"

layout\_width="200dp"

# TextView Control :

- **android:textColor** property
- To Set TextView's text-Color we can use **textColor** Property.
  - We can set color as,  
**android:textColor="#673AB7"**
- To set color :
  - Use color code as "**#000000**" to set **black** as default color.



# TextView Control :

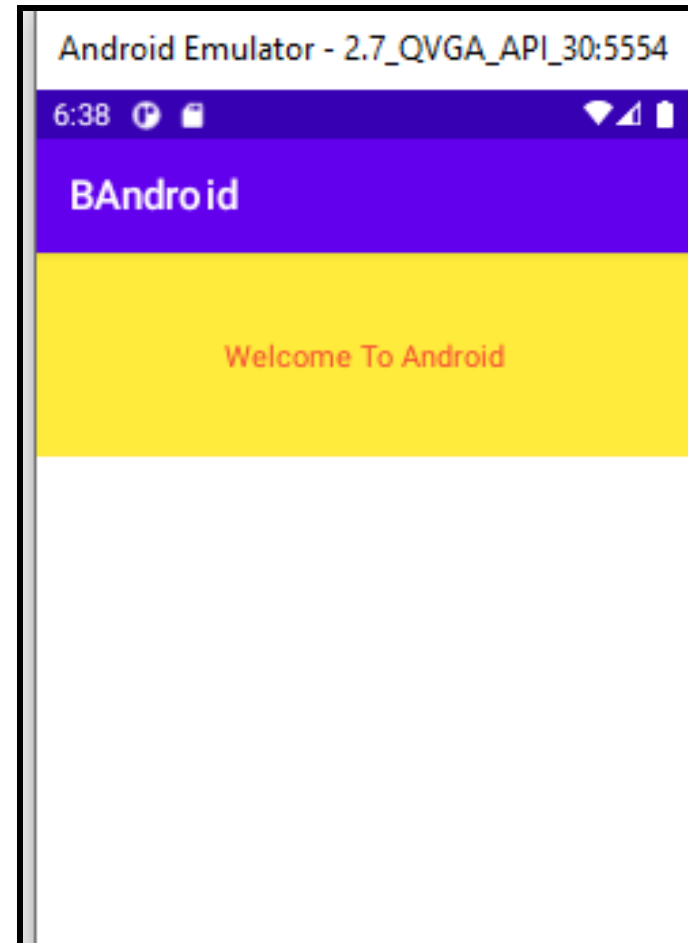
- **android:background property**
- To Set TextView's background-Color we can use background Property.
  - We can set color as,  
`android:background="#C5B5B5"`

# TextView Control :

- **android:gravity="center|bottom|top"**
- To Set TextView's alignment using gravity property to Center | Bottom | Top we can use this Property.

# Def\_\_ WAP to display following output...

- Use following properties :
  - ❑ layout\_height
  - ❑ layout\_width
  - ❑ textColor
  - ❑ background
  - ❑ gravity



# TextView Control :

- **android:textAllCaps="true"**
  - To Set TextView's All fonts as CAPITAL LETTERS we can use **textAllCaps** property.

# TextView Control :

- **android:textSize="20dp"**
  - To Set TextView's fonts size we can use textSize property.
- **android:textStyle="bold|italic"**
  - To Set TextView's textStyle, we can use textStyle property.

# TextView Control :

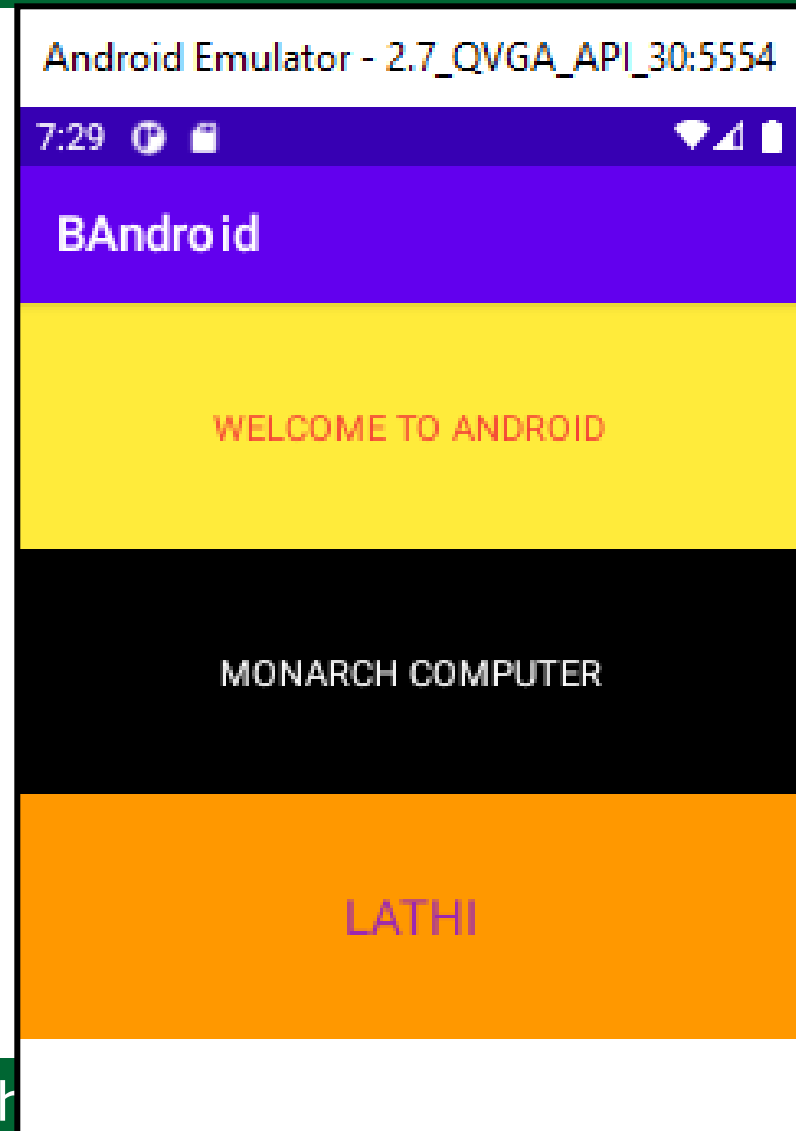
- **android:id="@+id/name"**
- To Set TextView's id, we can use **id** property.
- **"@+id"**
  - Its compulsory property to set controls ID.
- **"/name"**
  - Its used to set a name for ID.
  - Like, **id="@+id/txtName"**

# TextView Control :

- **android:layout\_below="@id/name"**
  - To Set more than one TextView's with its location, we can use **layout\_below** property.
  - We can display as many textViews As required.
  - **Example :**  
**android:layout\_below="@id/txtWelcome"**

# Def \_\_ WAP to display following output.

- Using,
  - id
  - layout\_below
  - textSize
  - textStyle
  - textAllCaps





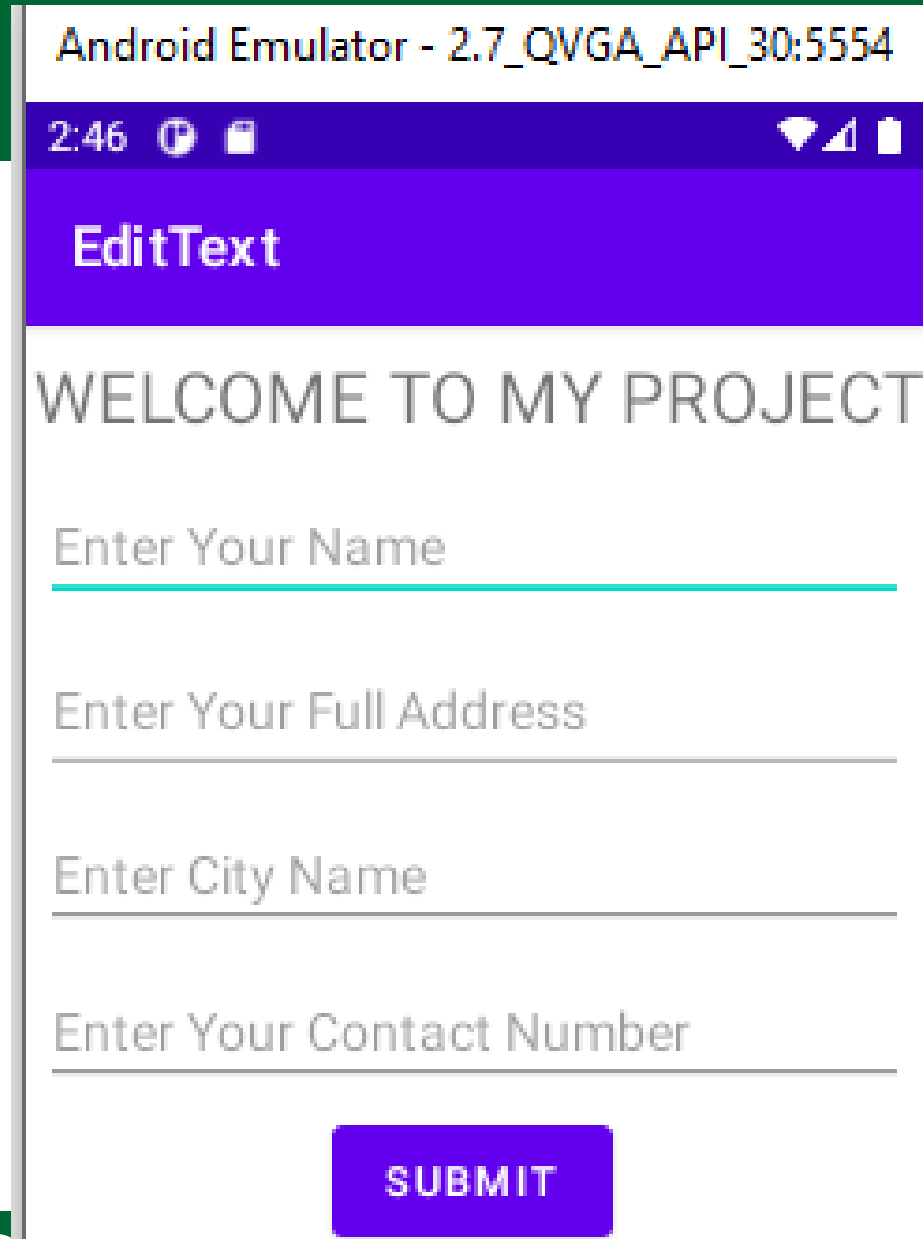
# EditText Control :

- EditText control is used to take user input on mobile screen.
- We can use multiple properties to enter text on screen.
- Useful properties :
  - ▣ **id, hint, textColorHint, layout\_margin, background**
- Example :

```
android:hint="Enter Your Name"
```

# Def\_\_ WAP to display following output...

- Using control,
  - EditText
  - Button
- Using properties,
  - id
  - hint
  - layout\_gravity



# Def\_\_ WAP to display following output...

- Using control,
  - EditText
  - Button
- Using properties,
  - id
  - hint
  - layout\_gravity

# Code Example...

**<EditText**

**android:layout\_width="match\_parent"**

**android:layout\_height="wrap\_content"**

**android:id="@+id/edtName"**

**android:hint="Enter Your Name"**

**android:layout\_margin="5dp"/>**

# Code Example...

## <Button

android:**layout\_width**="100dp"

android:**layout\_height**="50dp"

android:id="@+**id/btnSubmit**"

android:**text**="Submit"

android:**layout\_gravity**="center"/>

# How To Use Control with JAVA programming?

- We had already learnt how to design an application. Now we also have to know how to use coding with application.

□ Activity\_main.xml



- **.xml** file is used to front designing.

□ MainActivity.java



- **.java** file is used for programming.

# MainActivity.java

```
package com.example.welcome;

import androidx.appcompat.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);
    }
}
```

# MainActivity.java

```
package com.example.welcome;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;

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



# MainActivity.java : declare object

- Before using any object we have to declare that object with **object type** and **id(name)** properties.

MainActivity extends AppCompatActivity {

    EditText **edtName, edtWelcome;**

    Button **btnSubmit;**

    @Override        }

# MainActivity.java : findViewById

- We have to set object with Resource.

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    edtName=findViewById(R.id.edtName);  
    btnSubmit=findViewById(R.id.btnSubmit)  
}
```

# MainActivity.java : setOnClickListener

- To set Click event we have to define that using following syntax.

```
btnSubmit.setOnClickListener
```

```
    (new View.OnClickListener() {
```

```
        @Override
```

```
        public void onClick(View v) {
```

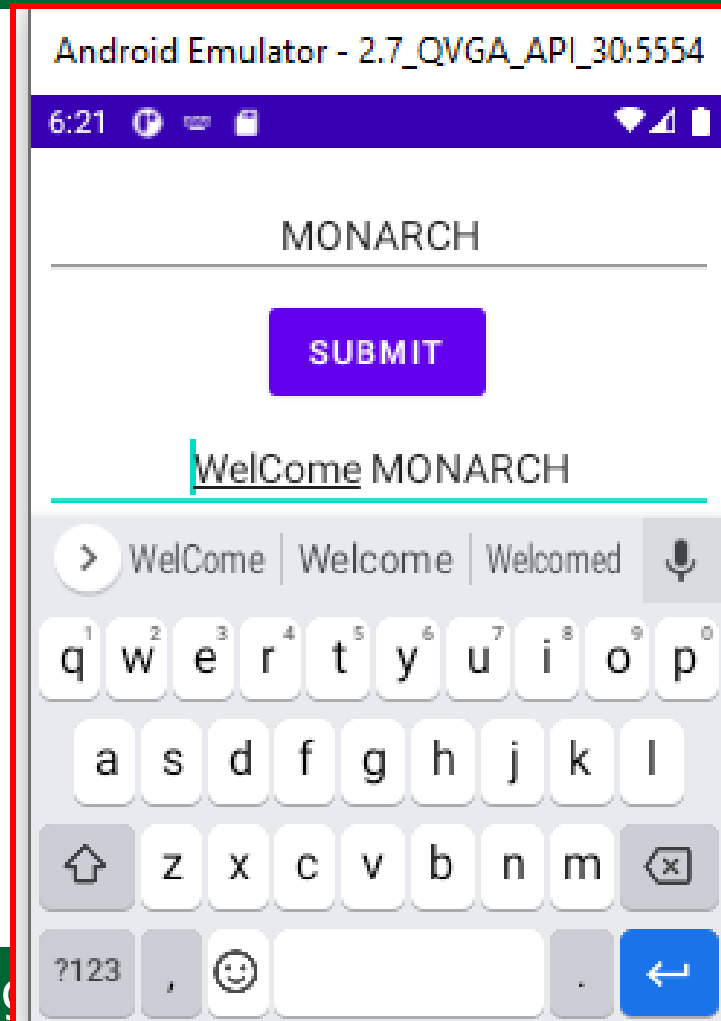
```
            String str="WelCome "+edtName.getText();
```

```
            edtWelcome.setText(str);    }
```

```
    });
```

Def. \_\_\_ **Design an app** to Enter Your Name and when press submit button it will **display a Welcome message.**

- We have to maintain two files...
  - Activity\_main.xml
    - For designing
  - MainActivity.java
    - For coding.



# Def. Cont. : Activity\_main.xml

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

# Def. Cont. : Activity\_main.xml

**<TextView**

android:layout\_width="match\_parent"

android:layout\_height="50dp"

android:gravity="center"

android:text="Welcome Message"

android:textAllCaps="true"

android:textSize="25dp"

/>

# Def. Cont. : Activity\_main.xml

**<EditText**

**android:id="@+id/edtName"**

**android:textAlignment="center"**

**android:layout\_width="match\_parent"**

**android:layout\_height="wrap\_content"**

**android:layout\_margin="5dp"**

**android:hint="Enter Your Name" />**

# Def. Cont. : Activity\_main.xml

## <Button

android:**id**="@+id/btnSubmit"

android:**layout\_width**="100dp"

android:**layout\_height**="50dp"

android:**layout\_gravity**="center"

android:**text**="Submit" />



# Def. Cont. : Activity\_main.xml

## <EditText

android:id="@+id/edtWelcome"

android:textAlignment="center"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_margin="5dp" />

## </LinearLayout>

## Def. Cont. : MainActivity.java

```
package com.example.edittext;  
  
import androidx.appcompat.app.AppCompatActivity;  
import androidx.appcompat.view.menu.ShowableListMenu;  
  
import android.os.Bundle;  
  
import android.view.View;  
  
import android.widget.Button;  
import android.widget.EditText;  
import android.widget.TextView;  
import android.widget.Toast;
```

# Def. Cont. : MainActivity.java

```
public class MainActivity extends AppCompatActivity {
```

```
    EditText edtName, edtWelcome;
```

```
    Button btnSubmit;
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

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

```
        edtName = findViewById(R.id.edtName);
```

```
        btnSubmit=findViewById(findViewById(R.id.btnSubmit));
```

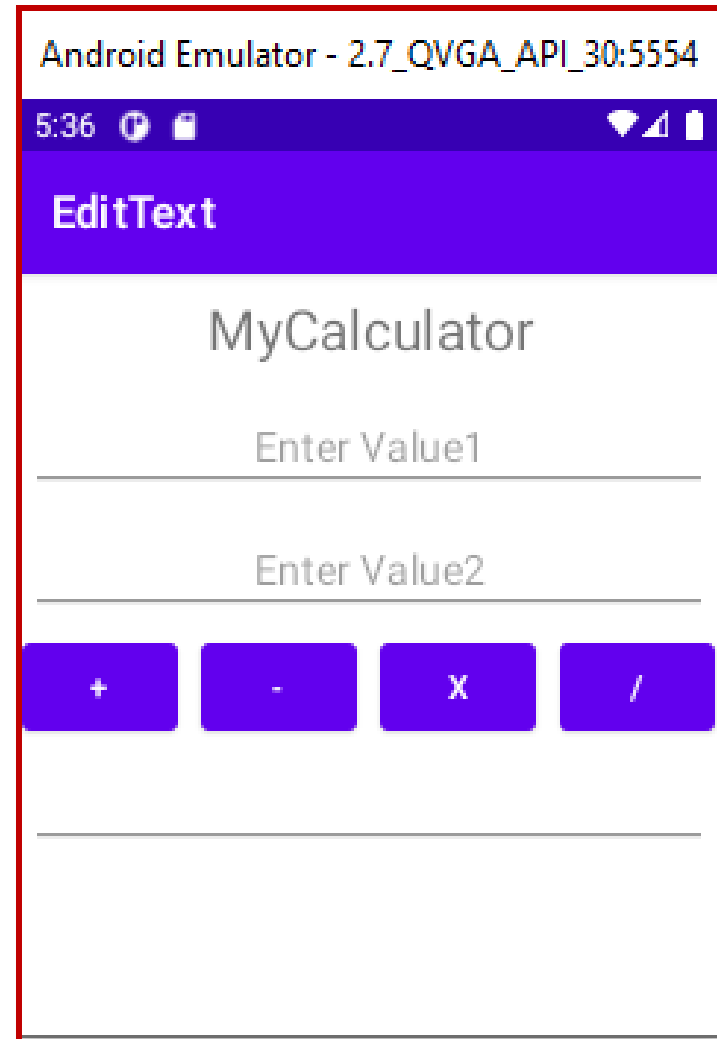
```
        edtWelcome=findViewById(findViewById(R.id.edtWelcome));
```

# Def. Cont. : MainActivity.java

```
btnSubmit.setOnClickListener  
    (new View.OnClickListener(){  
@Override  
    public void onClick(View v) {  
        String str="WelCome "+edtName.getText();  
        edtWelcome.setText(str); }  
    });  
    }  
}
```

# Def. \_\_\_ **Design an app** to Enter two values and process buttons **+, -, \*, /**.

- We have to maintain two files...
  - `Activity_main.xml`
    - For designing
  - `MainActivity.java`
    - For coding.



# Def. Cont. : Activity\_main.xml

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

# Def. Cont. : Activity\_main.xml

## <TextView

android:layout\_width="match\_parent"

android:layout\_height="50dp"

android:gravity="center"

android:text="MyCalculator"

android:textSize="25dp"/>

# Def. Cont. : Activity\_main.xml

## <EditText

android:id="@+id/**edtV1**"

android:textAlignment="center"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_margin="5dp"

android:hint="**Enter Value1**" />



# Def. Cont. : Activity\_main.xml

## <EditText

android:id="@+id/**edtV2**"

android:textAlignment="center"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_margin="5dp"

android:hint="**Enter Value2**" />

# Def. Cont. : Activity\_main.xml

<LinearLayout

```
    android:layout_width="match_parent"  
    android:layout_height="50dp"  
    android:orientation="horizontal"  
    android:gravity="center">
```

<**Button**

```
    android:id="@+id/btnPlus"  
    android:layout_width="70dp"  
    android:layout_height="50dp"  
    android:layout_gravity="center"  
    android:layout_margin="5dp"  
    android:text="+" />
```

# Def. Cont. : Activity\_main.xml

## <Button

```
android:id="@+id/btnMinus"  
android:layout_width="70dp"  
android:layout_height="50dp"  
android:layout_gravity="center"  
android:layout_margin="5dp"  
android:text="-" />
```

# Def. Cont. : Activity\_main.xml

## <Button

```
android:id="@+id/btnMulti"  
android:layout_width="70dp"  
android:layout_height="50dp"  
android:layout_gravity="center"  
android:layout_margin="5dp"  
android:text="x" />
```

# Def. Cont. : Activity\_main.xml

## <Button

android:id="@+id/**btnDivision**"

android:layout\_width="70dp"

android:layout\_height="50dp"

android:layout\_gravity="center"

android:layout\_margin="5dp"

android:text="/" />

# Def. Cont. : Activity\_main.xml

```
</LinearLayout>
```

```
<EditText
```

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

```
    android:textAlignment="center"
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
```

```
    android:layout_margin="5dp"/>
```

```
</LinearLayout>
```

## Def. Cont. : MainActivity.java

```
package com.example.edittext;  
  
import androidx.appcompat.app.AppCompatActivity;  
import androidx.appcompat.view.menu.ShowableListMenu;  
  
import android.os.Bundle;  
  
import android.view.View;  
  
import android.widget.Button;  
import android.widget.EditText;  
import android.widget.TextView;  
import android.widget.Toast;
```

# Def. Cont. : MainActivity.java

```
public class MainActivity extends
    AppCompatActivity {
    EditText edtV1, edtV2, edtOutput;
    Button btnPlus, btnMinus, btnMulti, btnDivision;
    @Override
    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```



# Def. Cont. : MainActivity.java

```
edtV1 = findViewById(R.id.edtV1);
```

```
edtV2 = findViewById(R.id.edtV2);
```

```
btnPlus=findViewById(R.id.btnPlus);
```

```
btnMinus=findViewById(R.id.btnMinus);
```

```
btnMulti=findViewById(R.id.btnMulti);
```

```
btnDivision=findViewById(R.id.btnDivision);
```

```
edtOutput=findViewById(R.id.edtOutput)
```

# Def. Cont. : MainActivity.java

```
btnMinus.setOnClickListener(new View.OnClickListener(){
```

```
    @Override
```

```
public void onClick(View v) {
```

```
    String strv1 = "" + edtV1.getText();
```

```
    String strv2 = "" + edtV2.getText();
```

```
    Integer inv1 = Integer.parseInt(strv1);
```

```
    Integer inv2 = Integer.parseInt(strv2);
```

```
    Integer tot = inv1 - inv2;
```

```
    edtOutput.setText(tot.toString()); } };
```

# Def. Cont. : MainActivity.java

```
btnPlus.setOnClickListener(new View.OnClickListener(){
```

```
    @Override
```

```
    public void onClick(View v) {
```

```
        String strv1 = "" + edtV1.getText();
```

```
        String strv2 = "" + edtV2.getText();
```

```
        Integer inv1 = Integer.parseInt(strv1);
```

```
        Integer inv2 = Integer.parseInt(strv2);
```

```
        Integer tot = inv1 + inv2;
```

```
        edtOutput.setText(tot.toString()); } };
```

# Def. Cont. : MainActivity.java

```
btnMulti.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        String strv1 = "" + edtV1.getText();  
        String strv2 = "" + edtV2.getText();  
        Integer inv1 = Integer.parseInt(strv1);  
        Integer inv2 = Integer.parseInt(strv2);  
        Integer tot = inv1 * inv2;  
        edtOutput.setText(tot.toString()); } });
```

# Def. Cont. : MainActivity.java

```
btnDivision.setOnClickListener(new View.OnClickListener(){
```

```
    @Override
```

```
public void onClick(View v) {
```

```
    String strv1 = "" + edtV1.getText();
```

```
    String strv2 = "" + edtV2.getText();
```

```
    Integer inv1 = Integer.parseInt(strv1);
```

```
    Integer inv2 = Integer.parseInt(strv2);
```

```
    Integer tot = inv1 / inv2;
```

```
    edtOutput.setText(tot.toString()); } };
```

```
} }
```

# Complete Following Definitions...

- **Def. \_\_\_** WAP to print Hello MONARCH in center in MainActivity. (Using LinearLayout)
- **Def. \_\_\_** WAP to print Hello MONARCH in center in MainActivity. (Using RelativeLayout)
- **Def. \_\_\_** WAP to Enter Marks of three subject from and print its total and percent.
- **Def. \_\_\_** WAP to Enter Your Age in **Year** and print your age in **Month, Days, Hours, Minutes** and **Second**.

# Anatomy (Structure) of an Android Application :

- In any java project we can see anatomy (structure) of an Android Application as shown here...
- **Src**
  - Java source code files will be available here.

# Anatomy (Structure) of an Android Application:

## ■ Gen

- ❑ The gen directory in an Android project contains auto generated files. We can see **R.Java** inside this folder which is a generated class which contains references to certain resources of the project.
- ❑ **R.java** is automatically created by the Eclipse IDE and any manual changes are not necessary.



# Anatomy (Structure) of an Android Application:

## ■ Res

- **Android supports** resources like **images** and certain **XML configuration** files these can be keeping separate from the source code. All these resources should placed **inside the res folder**.
- This res folder will be having sub-folders to keep the resources based on its type.

# Anatomy (Structure) of an Android Application:

## ■ /res/values

- Used to define strings, colors, dimensions, styles and static arrays strings or integers.
- By convention each type is stored in a separate file, e.g. strings are defined in the **res/values/strings.xml** file.

# Anatomy (Structure) of an Android Application:

- `/res/values-v11`
  - **Res/values** is the values of the API version 11, and `/res/values-values-v14` is the values of the API version 14.
  - **/res/values** :
    - Layout for normal screen size or default.

# Anatomy (Structure) of an Android Application:

- ❑ **/res/values-small :**
  - Layout for small screen size
- ❑ **/res/values-large:**
  - Layout for large screen size
- ❑ **/res/values-xlarge :**
  - Layout for extra-large screen size
- ❑ **/res/values-xlarge-land :**
  - Layout for extra-large in landscape orientation.

# Anatomy (Structure) of an Android Application:

- ❑ **/res/values-sw600dp**
  - Layout for tablets or Layout for 7" tablets (600dp wide and bigger)
- ❑ **/res/values-sw720dp**
  - Layout for 10" tablets (720dp wide and bigger)
- ❑ **/res/values-w600dp**
  - Layout for Multi-pane (any screen with 600dp available width or more)

# Anatomy (Structure) of an Android Application:

## ■ **/res/layout**

- This folder contains the layouts to be used in the application.
- A layout resource defines the architecture for the UI in an Activity or a component of a UI.
- These are resource directories in an application that provides different layout designs for different screen sizes.

# Anatomy (Structure) of an Android Application:

## ❑ **/res/menu**

- This folder contains menu resources to be used in the application (Options Menu, Context Menu, or submenu)

## ❑ **/res/drawable**

- Drawable folder are resource directories in an application that provides different bitmap drawables for medium, high and extra high density screens.

# Anatomy (Structure) of an Android Application:

- /res/drawable-ldpi
  - Bitmap for lower density.
- /res/drawable-mdpi
  - Bitmap for medium density.
- /res/drawable-hdpi
  - Bitmap for high density.
- /res/drawable-xhdpi
  - Bitmap for extra high density.
- /res/drawable-xxhdpi
  - Bitmap for X extra high density.



# Anatomy (Structure) of an Android Application:

## □ **libs :**

- External library files will be placed in this folder. If you want to any external library in you project place the library jar inside this folder and it will be added to the classpath automatically.

# Anatomy (Structure) of an Android Application:

## □ **assets :**

- This folder contains raw hierarchy of files and directories, with no other capabilities. It is just an unstructured hierarchy of files, allowing you to put anything you want there and later retrieve as raw byte streams.

# Anatomy (Structure) of an Android Application:

## ❑ **bin :**

- Bin folder is the are used by the compiler to prepare the files to be finally packaged to the application's APK file.

This includes

- ❑ Compiling your Java code into class files.
- ❑ Putting your resources (including images) into a structure to be zipped into the APK.

# Anatomy (Structure) of an Android Application:

## □ **AndroidManifest.xml**

- All the android applications will have an AndroidManifest.xml file in the root directory.
- This file will contain essential information about the application to the Android system, information the system must have before it can run any of the application's code. This control file describes the nature of the application and each of its components.

# Anatomy (Structure) of an Android Application:

## ❑ **ic\_launcher-web.png** :

- This is an icon to be used in Google play. Application on Google Play require a high fidelity version of the application icon. It is not used in your actual app or the launcher, so it not packaged in the APK. The specifications for the high-resolution icon are :

- ❑ 32-bit PNG with an alpha channel
- ❑ 512 x 512 pixels
- ❑ Maximum size of 1024KB

- ❑ **Proguard-project.txt :**
  - Everything in proguard.project.txt file will be in commented out state, because in general most people don't have any project specific needs, just to run ProGuard tool with standard settings.

# Anatomy (Structure) of an Android Application:

## ▣ Proguard-project.txt :

- The ProGuard tool shrinks, optimizes, and obfuscates your code by removing unused code and renaming classes, fields and methods with semantically obscure names. The result is a smaller sized **.apk** file that is more difficult to reverse engineer.

# Anatomy (Structure) of an Android Application:

## ▣ **Project.properties :**

- project.properties is the main project's properties file containing information such as the build platform target and the library dependencies has been renamed from default.properties in older SDK versions. This file is integral to the project.



# Android Terminologies :

- The list below defines some of the basic terminology of the Android platform.
- **.apk file :**
  - Android application package file is known as .apk file.
  - Each Android application is compiled and packaged in a single file that includes all of the application's code (.dex files), resources, assest and manifest file.

# Android Terminologies :

- **.apk file :**
  - The application package file can have any name but must use the .apk extension.
  - For example,
    - **Myfirstproj.apk**

# Android Terminologies :

## ■ **.dex file :**

- ❑ Compiled Android application code file.
- ❑ Android program are compiled into .dex (Dalvik Executable) files, which are in turn zipped into a single .apk file on the device. **.dex** files can be created by automatically translating compiled applications written in the Java programming language.

# Android Terminologies :

## ■ Action :

- An action is a string value assigned to Intent. Action strings can be defined by Android or by a third-party developer.
- For example,
  - `android.intent.action.`

# Android Terminologies :

## ■ adb :

- ❑ Android Debug Bridge, a command-line debugging application included with the SDK. It provides tools to browse the device, copy tools on the device, and forward ports for debugging.
- ❑ If you are developing in Eclipse using the ADT Plugin, adb is integrated into your development environment.

# Android Terminologies :

## ■ **Application :**

- ❑ Form a component viewpoint, an Android application consists of one or more activities, services, listeners and intent receivers.
- ❑ From a source file viewpoint, an Android application consists of code, resources, assets and a single manifest.
- ❑ During compilation, these files are packaged in a single file called an application package file. (.apk)

# Android Terminologies :

## ■ **Canvas :**

- Canvas is the simplest, easiest way to draw 2D objects on the screen.

However, it does not support hardware acceleration, as OpenGL ES does. The base class is Canvas.

# Android Terminologies :

## ■ **Content Provide :**

- A content provider is built on the Content Provider class, which handles content query strings of a specific format to return data in a specific format.



# Android Terminologies :

## ■ Dalvik :

- The Android platform's virtual machine. The Dalvik VM is an interpreter-only virtual machine that executes files in the Dalvik Executable (*.dex*).
- The **(.dex)** format, a format that is optimized for efficient storage and memory-mappable execution.

# Android Terminologies :

## ■ Dalvik :

- The virtual machine is register-based and it can run classes compiled by a Java language compiler that have been transformed into its native format, using the included "dx" tool.

# Android Terminologies :

## ■ Dalvik :

- The VM runs on top of Posix-compliant operating system, which it relies on for underlying functionality (such as threading and low level memory management).

# Android Terminologies :

## ■ Dalvik :

- The Dalvik core class library is intended to provide a familiar development base for those used for those used to programming with Java Standard Edition, but it is geared specifically to the needs of a small mobile device.

# Android Terminologies :

## ■ **DDMS :**

- ❑ Dalvik Debug Monitor Service, a GUI debugging application included with the SDK.
- ❑ It provides screen capture, log dump and process examination capabilities. If we are developing in Eclipse using the ADT Plugin, DDMS is integrated into our development environment.

# Android Terminologies :

## ■ **Dialog :**

- A floating window that acts as a lightweight form. A dialog can have button controls only and is intended to perform a simple action (such as button choice) and perhaps return a value.

# Android Terminologies :

## ■ Dialog :

- A dialog is not intended to persist in the history stack, contain complex layout, or perform complex actions. Android provides a default simple dialog for you with optional buttons, though you can define your own dialog layout. The base class for dialogs is `Dialog`.

# Android Terminologies :

## ■ **Drawable :**

- A drawable is typically loaded into another UI element, for example as a background image. A drawable is not able to receive events, but does assign various other properties such as “state” and scheduling, to enable subclasses such as animation objects or image libraries.



# Android Terminologies :

## ■ Drawable :

- Many drawable objects are loaded from drawable resource files – XML or bitmap files that describe the image. Drawable resources are compiled into subclasses or `android.graphics.drawable`.

# Android Terminologies :

## ■ Intent :

- An Intent object is an instance of Intent. It included several criteria fields that you can supply, to determine what application/activity receives the Intent and what the receiver does when handling the Intent.

# Android Terminologies :

## ■ Intent :

- Available criteria include the desired action, a category, a data string, the MIME type of the data, a handling class, and others.
- An application sends Intent to the Android system, rather than sending it directly to another application/activity.

# Android Terminologies :

## ■ Intent :

- The application can send the Intent to a single target application or it can send it as a broadcast, which can in turn be handled by multiple application sequentially.

# Android Terminologies :

## ■ Intent Filter :

- A filter object that an application declares in its manifest file, to tell the system what types of Intents each of its components is willing to accept and with what criteria.
- Through an intent filter, an application can express interest specific data types, Intent actions, URIs formats, and so on.

# Android Terminologies :

## ■ **Broadcast Receiver :**

- An application class that listens for Intents that are broadcast, rather than being sent to a single target application/activity.
- The system delivers a broadcast Intent to all interested broadcast receivers, which handle the Intent sequentially.

# Android Terminologies :

## ■ **Layout Resource :**

- ❑ An XML file that describes the layout of an Activity screen.

## ■ **Manifest File :**

- ❑ An XML file that each application must define, to describe the application's package name, version, components (activities, intent filters, services), imported libraries and describes the various activities, and so on.

# Android Terminologies :

- **Nine-patch / 9-patch / Ninepatch image :**
  - A resizable bitmap resource that can be used for backgrounds or other images on the device.



# Android Terminologies :

## ■ OpenGL ES :

- ❑ Android provides OpenGL ES libraries that you can use for fast, complex 3D images. It is harder to use than a Canvas object, but better for 3D objects.
- ❑ The `android.opengl` and `javax.microedition.khronos.opengles` packages expose OpenGL ES functionality.

# Android Terminologies :

## ■ Resources :

### □ Non programmatic application

components that are external to the compiled application code, but which can be loaded from application code using a well-known reference format.

# Android Terminologies :

- Resources :
  - Android supports a variety of resource types, but a typical application's resources would consist of UI strings, UI layout components, graphics or other media files, and so on.

# Android Terminologies :

## ■ Resources :

- An application uses resources to efficiently application would include a separate set of resources for each supported local or device type and it could include layout resources that are specific to the current screen orientation (landscape or portrait).

# Android Terminologies :

## ■ **Service :**

- An object of class Service that runs in the background (without any UI presence) to perform various persistent actions, such as playing music or monitoring network activity.

# Android Terminologies :

## ■ **Surface :**

- An object of type Surface representing a block of memory that gets composited to the screen.
- A surface holds a Canvas object for drawing and provides various helper methods to draw layers and resize the surface. We should not use this class directly; use SurfaceView instead.

# Android Terminologies :

## ■ **SurfaceView :**

- ❑ A View object that wraps a Surface for drawing and exposes methods to specify its size and format dynamically.
- ❑ A SurfaceView provides a way to draw independently of the UI thread for resource-intensive operations (such as games or camera previews), but it uses extra memory as a result.
- ❑ SurfaceView supports both Canvas and OpenGL ES graphics. The base class is SurfaceView.

# Android Terminologies :

## ■ Theme :

- A set of properties (text size, background color, and so on) bundled together to define various default display settings.
- Android provides a few standards themes, listed in R.style (starting with "Theme\_").



# Android Terminologies :

## ■ URIs and Android :

- ❑ Android uses URI string as the basis for requesting data in a content provider (such as to retrieve a list of contacts) and for requesting actions in an Intent (such as opening a Web page in a browser).
- ❑ The URI scheme and format is specialized according to the type of use, and an application can handle specific URI schemes and strings in any way it wants. Some URI schemes are reserved by system components.

# Android Terminologies :

## ■ **View :**

- An object that draws to a rectangular area on the screen and handles click, keystroke, and other interaction events.
- A View is a base class for most layout components of an Activity or Dialog screen (text boxes, windows, and so on).

# Android Terminologies :

## ■ View :

- It receives calls from its parent object to draw itself, and informs its parent object about where and how big it would like to be (which may or may not be respected by the parent).

# Android Terminologies :

## ■ **Viewgroup :**

- ❑ A container object that groups a set of child Views. The viewgroup is responsible for deciding where child views are positioned and how large they can be, as well as for calling each to draw itself when appropriate.
- ❑ Some viewgroups are invisible and are for layout only, while others have an intrinsic UI (for instance, a scrolling list box).

# Android Terminologies :

## ■ **Widget :**

- ❑ One of a set of fully implemented View subclasses that render from elements and other UI components, such as a text box or popup menu.
- ❑ Because a widget is fully implemented, it handles measuring and drawing itself and responding to screen events.
- ❑ Widgets are all in the `android.widget` package.

# Android Terminologies :

## ■ **Window :**

- ❑ In an Android application, an object derived from the abstract class Window that specifies the elements of a generic window, such as the look and feel (title bar text, location and content of menus, and so on).
- ❑ Dialog and Activity use an implementation of this class to render a window. You do not need to implement this class or use windows in your application.

# Application Context, Activities, Services, Intents

- Context :
  - Context is probably the most used element is created in android applications.

# Application Context, Activities, Services, Intents

## ■ Application :

- The application object is created whenever one of your Android components is started. It is started in a new process with a unique ID under a single user. Even if you do not specify one in your. This object provides the following lifecycle methods:



# Application Context, Activities, Services, Intents

## ❑ **onCreate () :**

- called before the first components of the application starts

## ❑ **onLowMemory() :**

- called when the Android system requests that the application cleans up memory

# Application Context, Activities, Services, Intents

- ❑ **onTerminate()** :
  - only for testing, not called in production.

# Application Context, Activities, Services, Intents

- ❑ **onconfigurationChanged()** :
  - called whenever the configuration changes The application object starts before any component and runs at least as long as another component for the application runs.

# Priority System :

- ❑ If the Android system needs to terminate processes it follows the following priority system.
- ❑ Priorities :
  - Foreground
  - Visible
  - Service
  - Background
  - Empty

# Priority System :

Process status	Description	Priority
Fore ground	An application in which the user is interacting with an activity, or which has service which is bound to such an activity. Also if a service is executing one of its lifecycle methods or a broadcast receiver which runs its onReceive() method.	1

# Priority System :

Process status	Description	Priority
Visible	User is not interacting with the activity, but the activity is still (partially) visible or the application has a service which is used by a inactive but visible activity.	2
Service	Application with a running service which does not qualify for 1 or 2.	3

# Priority System :

Process status	Description	Priority
Back ground	Application with only stopped activities and without a service or executing receiver. Android keeps them in a least recent used (LRU) list and if requires terminates the one which was least used.	4
Empty	Application without any active components.	5

# Activity LifeCycle :

- Activity lifecycle :

- An activity can be in different states which are described by the following table.



# Activity state :

State	Description
Running	Activity is visible and interacts with the user.
Paused	Activity is still visible but partially obscured, instance is running but might be killed by the system.

# Activity state :

State	Description
Stopped	Activity is not visible, instance is running but might be killed by the system.
Killed	Activity has been terminated by the system or by a call to its finish() method.

# Activity state

- The user should not notice if an activity which is still part of an activity stack has been terminate or not. For this the developer needs to store the state of the activity at the right point in time and restore it.

# Activity state :

- ❑ He also should stop any unnecessary actions if the activity is not visible anymore to save system resources.
- ❑ The android system defines a lifecycle for activities via predefined (lifecycle) methods.

# Activity lifecycle methods :

Method	Purpose
onCreate()	Called when the activity is created. Used to initialize the activity, for example create the user interface.
onRause()	Called once another activity gets visible again and the user starts interacting with the activity again. Used to initialize fields, register listeners, bind to services, etc.

# Activity lifecycle methods :

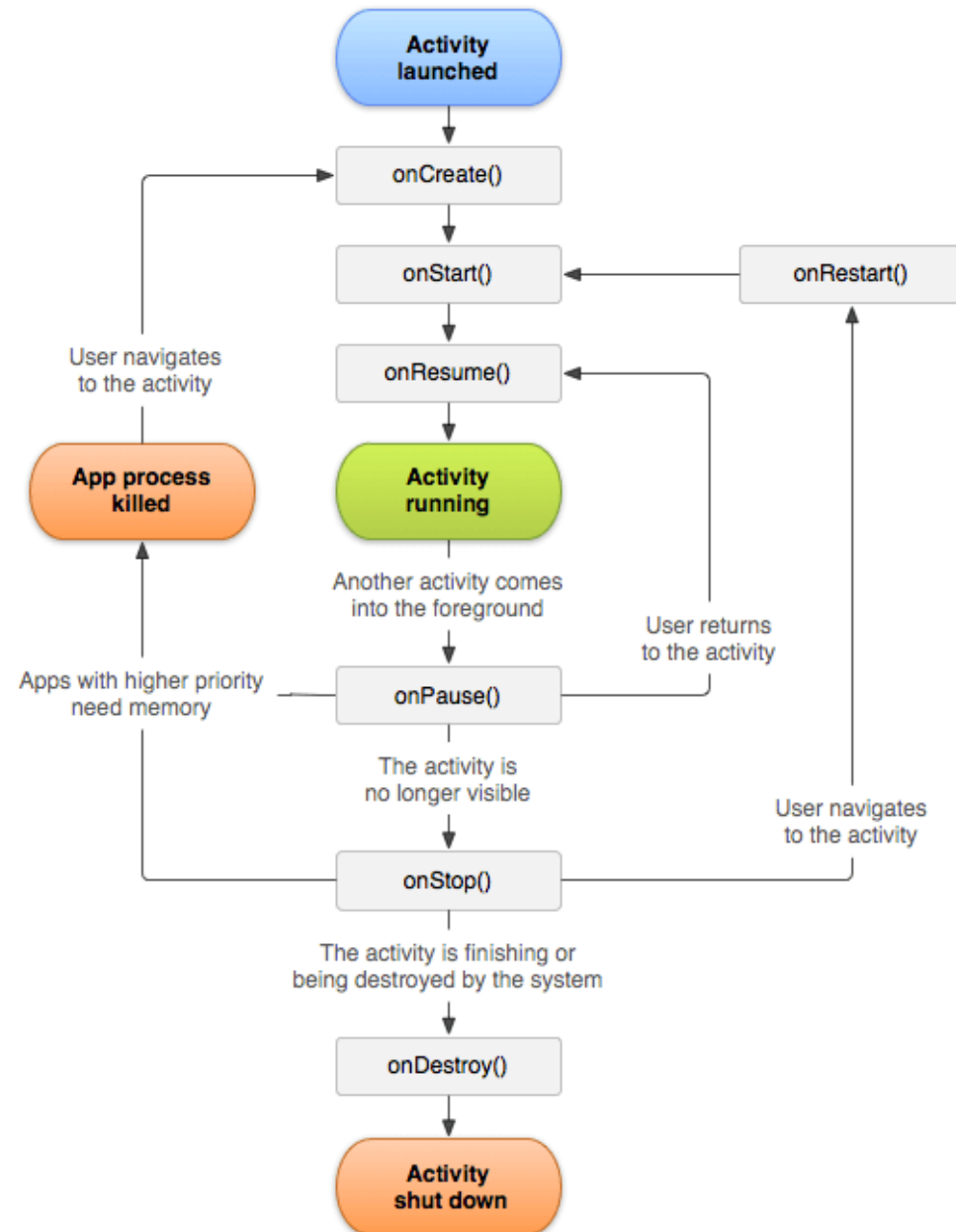
Method	Purpose
onPause()	<p>Called once another activity gets into the foreground. Always called before the activity is not visible anymore. Used to release resources or save application data.</p> <p><b>For example</b>, you unregister listeners, intent receivers, unbind from services or remove system service listeners.</p>

# Activity lifecycle methods :

Method	Purpose
onStop()	<p>Called once the activity is no longer visible. Time or CPU intensive shut-down operations, such as writing information to a database should be done in the onStop() method.</p> <p>This method is guaranteed to be called as of API 11.</p>

# Activity lifecycle methods :

- The life cycle of an activity with its most important methods in displayed here,





# Services :

- A service is a component that runs in the background to perform long-running operations without needing to interact with the user.
- For Example,
  - A service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.

# Services :

- A service can essentially take two states :

State	Description
Started	<p>A service is <b>started</b> when an application component, such as an activity, starts in by calling <code>startService()</code>.</p> <p>Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.</p>

# Services :

- A service can essentially take two states :

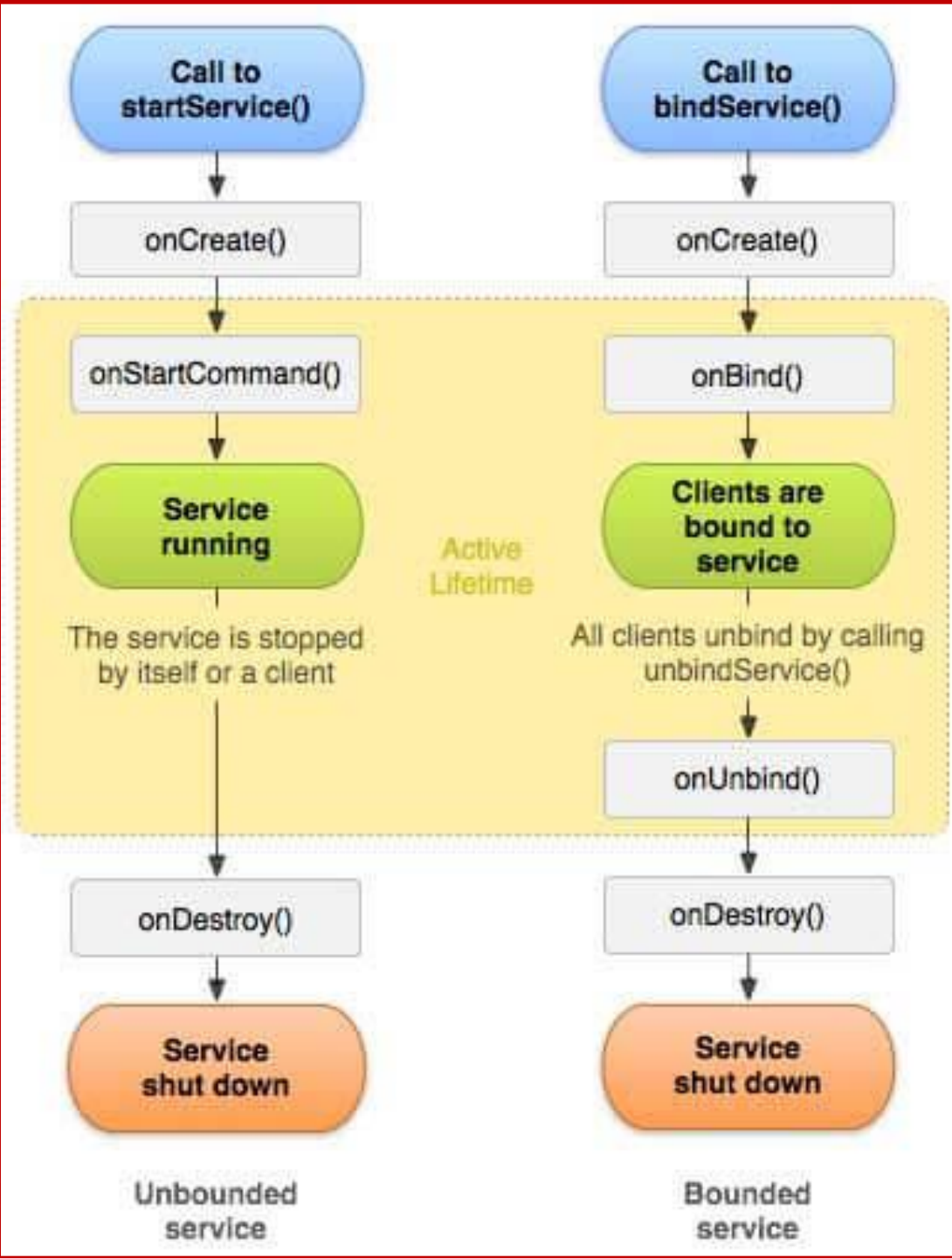
State	Description
Bound	<p>A service is <b>bound</b> when an application component binds to it by calling <code>bindService()</code>.</p> <p>A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC).</p>

# Services :

- A service has lifecycle callback methods that you can implement to monitor changes in the service's state and you can perform work at the appropriate stage.
- The diagram on the left shows the lifecycle when the service is created with `startService()` and the diagram on the right shows the lifecycle when the service is created with `bindService()`.

# Services :

- Unbounded Service
- Bounded Service



# Services :

- To create a service, you create a Java class that extends the Service base class or one of its existing subclasses. The **Service** base class defines various callback methods and the most important are give below.
- You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

# Services :

- `onStartCommand()`
  - The system calls this method when another component, such as an activity, requests that the service be started, by calling `startService()`.
  - If you implement this method, it is your responsibility to stop the service when its work is done, by calling `stopSelf()` or `stopService()` methods.

# Services :

- `onBind()`
  - ❑ The system calls this method when another component wants to bind with the service by calling `bindService()`.
  - ❑ If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an `Ibinder` object.
  - ❑ You must always implement this method, but if you don't want to allow binding, then you should return `null`.



# Services :

- **onUnbind()**
  - The system calls this method when all clients have disconnected from a particular interface published by the service.
- **onRebind()**
  - The system calls this method when new clients have connected to the service, after it had previously been notified that all had disconnected in its onUnbind.

# Services :

- onCreate()
  - The system calls this method when the service is first created using onStartCommand() or onBind().
  - This call is required to perform one-time setup

# Services :

## ■ onDestroy()

- The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc.

# Intents and Intent Filters :

- Intent is a messaging object we can use to request an action from another app component.
- Although intents facilitate communication between components in several ways, there are three fundamental use-cases:
  - To start an activity
  - To start a service
  - To deliver a broadcast

# Intents and Intent Filters :

- To start an activity
  - An Activity represents a single screen in an app. We can start a new instance of an Activity by passing an Intent to `startActivity()`.
  - The Intent describes the activity to start and carries any necessary data.

# Intents and Intent Filters :

- To start an activity
  - If you want to receive a result from the activity when it finishes, call `startActivityForResult()`.
  - Your activity receives the result as a separate Intent object in your activity's `onActivityResult()` callback.

# Intents and Intent Filters :

- To start a Service :
  - A Service is a component that performs operations in the background without a user interface.
  - We can start a service to perform a one-time operation (such as download a file) by passing an Intent to `startService()`.

# Intents and Intent Filters :

- To start a Service :
  - The Intent describes the service to start and carries any necessary data.
  - If the service is designed with a client-server interface, we can bind to the service from another component by passing an Intent to `bindService()`.



# Intents and Intent Filters :

- To deliver a broadcast :
  - A broadcast is a message that any app can receive. The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging. You can deliver a broadcast to other apps by passing an Intent to `sendBroadcast()`, `sendOrderedBroadcast()`, or `sendStickyBroadcast()`.

# Intent Types :

- There are two types of intents :
  - Explicit Intents
  - Implicit Intents

# Intent Types :

## ■ Explicit Intents

- Explicit intents specify the component to start by name (the fully-qualified class name). We can typically use an explicit intent to start a component in our own app, because we know the class name of the activity or service we want to start.
- For example,
  - Start a new activity in response to a user action or start a service to download a file in the background.

# Intent Types :

## ■ Implicit Intents

- Implicit intents do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it.
- For Example,
  - If we want to show the user a location on a map, we can use an implicit intent to request that another capable app show a specified location on a map.

# Create an Intent :

- An Intent object carries information that the Android system uses to determine which component to start (such as the exact component name or component category that should receive the intent), plus information that the recipient component uses in order to properly perform the action (such as the action to take and the data to act upon).

# Forcing an app chooser :

- When there is more than one app that responds to your implicit intent, the user can select which app to use and make that app the default choice for the action.
- This is nice when performing an action for which the user probably wants to use the same app from now on, such as when opening a web page (users often prefer just one web browser).

# Forcing an app chooser :

- However, if multiple apps can respond to the intent and the user might want to use a different app each time, he should explicitly show a chooser dialog.
- The chooser dialog asks the user to select which app to use for the action every time.

# Receiving an Implicit Intent :

- To advertise which implicit intents your app can receive, declare one or more intent filters for each of your app components with an `<intent-filter>` element in your manifest file.
- Each intent filter specifies the type of intents it accepts based on the intent's **action**, **data**, and **category**.



# Restricting access to components :

- Using an intent filter is not a secure way to prevent other apps from starting your components.
- Although intent filters restrict a component to respond to only certain kinds of implicit intents, another app can potentially start your app component by using an explicit intent if the developer determines your component names.

# Restricting access to components :

- If it's important that only your own app is able to start one of your components, set the exported attribute to "false" for that components.

# Using a Pending Intent :

- A PendingIntent object is a wrapper around an Intent object. The primary purpose of a PendingIntent is to grant permission to other application to use the contained Intent as if it were executed from your app's own process.

# Intent Resolution :

- When the system receives an implicit intent to start an activity, it searches for the best activity for the intent by comparing the intent to intent filters based on three aspects :
  - The intent action
  - The intent data (both URI and data types)
  - The intent category

# Intent matching :

- Intents are matched against intent filters not only to discover a target component to activate, but also to discover something about the set of components on the device.
- For example,
  - The home app populates the app launcher by finding all the activities with intent filters that specify the ACTION\_MAIN action and CATEGORY\_LAUNCHER category.

# Android Manifest File and Its Common Settings :

- Manifest file for an android application is a resource file which contains all the details needed by the android system about the application.
- It is a key file that works as a bridge between the android developer and the android platform.

# Android Manifest File and Its Common Settings :

- It helps the developer to pass on functionality and requirements of our application to Android.
- This is an xml file which must be named as `AndroidManifest.xml` and placed at application root. Every Android app must have `AndroidManifest.xml` file.

# Android Manifest File and Its Common Settings :

- **AndroidManifest.xml** allows us to define.
  - Basic building blocks of application like activities, services and etc.
  - Details about permissions.
  - Set of classes needed before launch.



# Elements of AndroidManifest.xml

- Following are the elements (listed alphabetically) that can appear in AndroidManifest.xml, this list are restricted and we cannot add our own elements to it.
  - `<action>`, `<activity>`, `<activity-alias>`,  
`<application>`, `<category>`, `<data>`,  
`<grant-uri-permission>`,

# Elements of AndroidManifest.xml

- <instrumentation>, <intent-filter>, <manifest>, <mata-data>, <permission>, <permission-group>, <permission-tree>, <provider>, <receiver>, <service>, <supports-screens>, <uses-configuration>, <uses-feature>, <uses-library>, <uses-permission>, <uses-sdk>

# Elements for Application Properties :

- Uses-permission :
  - Used to specify permissions that are requested for the purpose of security.
- Permission :
  - Used to set permissions to provide access control for some specific component of the application.

# Elements for Application Properties :

- **Permission-group :**
  - Does the same as above for a set of components.
- **Permission-tree :**
  - Refer one specific name of the component which is the owner or parent of the set of component.

# Elements for Application Properties :

## ■ Instrumentation :

- Enables to know interaction between Android system and application.

## ■ Uses-sdk :

- Specifies the platform compatibility of the application.

# Elements for Application Properties :

- Use-configuration :
  - Specifies set of hardware and software requirement of the application.
- Uses-feature :
  - Specifies single hardware and software requirement and their related entity.

# Elements for Application Properties :

- **Supports-screens, compatible-screens :**
  - Both these tags deals with screen configuration mode and size of the screen and etc.
- **Supports-gl-texture :**
  - Specifies texture based on which the application is filtered. Elements for Application Components. These should be enclosed in <application> container.

# Elements for Application Properties :

- **Activity :**
  - Has the set of attributes based on user interface.
- **Activity-alias :**
  - Specifies target activities.
- **Service :**
  - Has the operation provided by any library or API, running in background that is not visible.



# Elements for Application Properties :

## ■ Receiver :

- That makes to receive message broadcasted by the same application or by outside entity.

## ■ Provider :

- Provides some structure to access application data.

# Elements for Application Properties :

- Uses-library :
  - It specifies set of library files need to run the application.
- This entire information has to be known by the system to run any file of the application. So that this file has to be created at the time of installing and not at the time of running the application.

# Structure Of AndroidManifest.xml

**<manifest>**

<Elements for Application properties  
should come here – refer above for list>

**<application>**

<Elements for application components  
should come here – refer above  
for list>

**</application>**

**<manifest>**

# Resources Overview :

- You should always externalize resources such as images and strings from your application code, so that you can maintain them independently.
- Externalizing your resources also allows you to provide alternative resources that support specific device configurations such as different languages or screen sizes, which becomes increasingly important as more Android-powered devices become available with different configurations.

# Resources Overview :

- In order to provide compatibility with different configurations, you must organize resources in your project's res/ directory, using various sub-directories that group resources by type and configuration.

# Resource Types :

- We should place each type of resource in a specific subdirectory of your project's res/ directory.
- For Example,
  - Here's the file hierarchy for a simple project.

# Resource Types :

MyProject/

**src/**

MyActivity.java

**res/**

drawable/

icon.png

**layout/**

main.xml

info.xml

**values/**

strings.xml

# Resource Types :

Directory	Resource Type
animator/	XML files that define property animations.
anim/	XML files that define animations. (Property animations can also be saved in this directory, but the animator/ directory is preferred for property animations to distinguish between the two types.)



# Resource Types :

Directory	Resource Type
color/	XML files that define a state list of colors.
layout/	XML files that define a user interface layout.
menu/	XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu

# Resource Types :

Directory	Resource Type
drawable/	<p>Bitmap files (.png, jpg, gif) or XML files that are compiled into the following drawable resource subtypes:</p> <ul style="list-style-type: none"><li>• Bitmap files</li><li>• Nine-Patches (re-sizable bitmaps)</li><li>• State lists</li><li>• Shapes</li><li>• Animation drawables</li><li>• Other drawables</li></ul>

# Resource Types :

Directory	Resource Type
raw/	<p>Arbitrary files to save in their raw form. To open these resources with a raw <code>InputStream</code>, call <code>Resources.openRawResource()</code> with the resource ID, which is <code>R.raw.filename</code>.</p>

# Resource Types :

Directory	Resource Type
raw/ Cont...	<p>However, if you need access to original file names and file hierarchy, you might consider saving some resources in the assets/ directory (instead of res/raw/). Files in assets/ are not given a resource ID, so you can read them only using AssetManager.</p>

# Resource Types :

Directory	Resource Type
values/	<p>XML files that contain simple values, such as strings, integers, and colors. Whereas XML resource files in other res/ subdirectories define a single resource based on the XML filename, files in the values/ directory describe multiple resources.</p>

# Resource Types :

Directory	Resource Type
values/	<p>For a file in this directory, each child of the <code>&lt;resources&gt;</code> element defines a single resource.</p> <p>For example, a <code>&lt;string&gt;</code> element creates an <code>R.string</code> resource and <code>&lt;color&gt;</code> element creates an <code>R.color</code> resource.</p>

# Resource Types :

Directory	Resource Type
values/	<p>Because each resource is defined with its own XML element, you can name the file whatever you want and place different resource types in one file. However, for clarity, you might want to place unique resource types in different files.</p>

# Resource Types :

Directory	Resource Type
values/	<p>For example, here are some filename conventions for resources you can create in this directory:</p> <ul style="list-style-type: none"><li>• arrays.xml for resource arrays</li><li>• colors.xml for color values</li><li>• dimens.xml for dimension values</li><li>• strings.xml for string values</li><li>• styles.xml for styles</li></ul>



# Resource Types :

Directory	Resource Type
xml/	<p>Arbitrary XML files that can be read at runtime by calling <code>Resources.getXML()</code>.</p> <p>Various XML configuration files must be saved here, such as a searchable configuration.</p>

# ::: Android :::

**Chapter 01 OVER :**

Thanks...