UNIT - I

❖ Introduction to Android Application and History

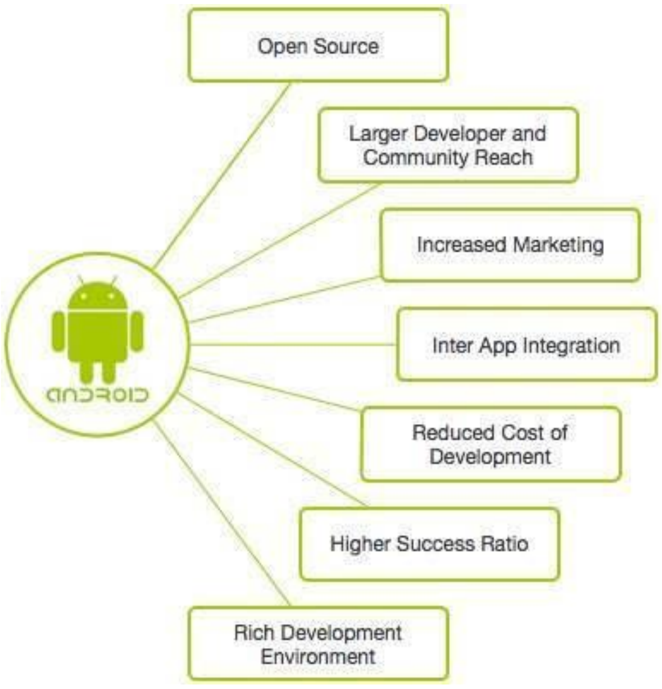
Android is an open source and Linux-based Operating System for mobile devices such as smartphones and tablet computers. Android was developed by the *Open Handset Alliance*, led by Google, and other companies.

Android offers a unified approach to application development for mobile devices which means developers need only develop for Android, and their applications should be able to run on different devices powered by Android.

The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007 where as the first commercial version, Android 1.0, was released in September 2008.

On June 27, 2012, at the Google I/O conference, Google announced the next Android version, 4.1 Jelly Bean. Jelly Bean is an incremental update, with the primary aim of improving the user interface, both in terms of functionality and performance.

The source code for Android is available under free and open source software licenses. Google publishes most of the code under the Apache License version 2.0 and the rest, Linux kernel changes, under the GNU General Public License version 2.


android
Open Source
Larger Developer and Community Reach
Increased Marketing
Inter App Integration
Reduced Cost of Development
Higher Success Ratio
Rich Development Environment

History of Android

The history of Android is a fascinating journey that spans several decades, characterized by innovation, collaboration, and evolution. From its humble beginnings, when it was competing with Nokia's mobile OS Symbian, the Windows Phone OS, and Blackberry OS, to its dominant position in the mobile operating system market, Android has significantly influenced how we interact with technology.

Here's an in-depth look at the history of Android:

1. Early origins and development (2003-2007)

The story of Android began in 2003 when Andy Rubin, Rich Miner, Nick Sears, and Chris White founded Android Inc. in Palo Alto, California. Their initial goal was to develop an advanced operating system for digital cameras. However, recognizing the potential of their project, they shifted their focus to creating an operating system for mobile devices.

2. Acquisition by Google (2005)

In 2005, Google, led by then-CEO Eric Schmidt, acquired Android Inc., laying the groundwork for what would become one of the most significant developments in the mobile industry. Google's acquisition of Android signaled its entry into the rapidly growing smartphone market and set the stage for developing a new mobile operating system.

3. Open Handset Alliance and the launch of Android (2007)

On November 5, 2007, the Open Handset Alliance (OHA) was unveiled. It comprised several prominent technology companies, including Google, HTC, Samsung, Motorola, and others. The OHA aimed to develop open standards for mobile devices and promote innovation in the mobile industry. Shortly after, on November 5, 2007, Google announced the first beta version of the Android operating system.

4. Android 1.0 and the first Android device (2008)

The first commercial version of Android, Android 1.0, was released on September 23, 2008. The HTC Dream, also known as T-Mobile G1, was the first smartphone to run on the Android operating system. The HTC Dream featured a touchscreen interface, a physical keyboard, and access to Google services such as Gmail, Maps, and YouTube.

Since its initial release, Android has undergone significant evolution with regular updates and new versions introduced to the market, as we will discuss later. The developer preview of Android 15 has been launched in 2024.

5. Growth and dominance in the mobile market

Over the years, Android has experienced tremendous growth, rapidly becoming the world's most popular mobile operating system. According to Statcounter, as of January 2022, Android holds over 72% of the global mobile operating system market share, far surpassing its competitors.

❖ Key Features of Android

Android is an open-source operating system, based on the Linux kernel and used in mobile devices like smartphones, tablets, etc. Further, it was developed for smartwatches and Android TV. Each of them has a specialized interface. Android has been one of the best-selling OS for smartphones. Android OS was developed by Android Inc. which Google bought in 2005.

Android offers a wide array of functionalities catering to both users and developers. Some of the key features of Android include:

- Open-source platform: Android is built on an open-source Linux kernel, allowing developers to access the source code, modify it, and contribute to its development. This openness fosters innovation and collaboration within the Android ecosystem.
- Customizable user interface: Android provides users with the ability to customize their device's user interface, including wallpapers, themes, widgets, and launchers. Users can personalize their devices to suit their preferences and style. This feature sets it apart from its closest competitor, iOS.
- Multitasking: Android supports multitasking, allowing users to run multiple apps simultaneously, switch between them seamlessly, and perform various tasks simultaneously. Users can also use split-screen mode to view two apps side by side.
- Google Play Store: Android users can access the Google Play Store, which offers a vast catalog of apps, games, movies, music, books, and more. The Play Store provides users a centralized platform to discover, download, and install content for their devices.

- Google Assistant: Android devices come with Google Assistant, a virtual assistant powered by artificial intelligence. Google Assistant can perform various tasks, answer questions, provide recommendations, and control smart home devices using voice commands.
- Security features: Android incorporates various security features to protect users' data and privacy. These features include app sandboxing, secure boot, encrypted file systems, Google Play Protect, and regular security updates from device manufacturers.
- Accessibility: Android includes a wide range of accessibility features to accommodate users with disabilities or special needs. These features include screen readers, magnification gestures, color inversion, text-to-speech, and more.
- Google Services integration: Android devices seamlessly integrate with Google services such as Gmail, Google Maps, Google Drive, Google Photos, and others. This integration provides users access to a suite of productivity tools, communication services, and cloud storage options.
- Development tools and support: Android provides developers with comprehensive development tools, including Android Studio, the official integrated development environment (IDE) for Android app development. Developers can also access extensive documentation, APIs, libraries, and resources to build high-quality apps for Android devices.

❖ Different Apps in android

Some Categories of the Android Applications:

1. E-Commerce Apps: E-commerce apps are an example of a B2B model. It helps to people to sell and borrow different items and it saves time and money. In e-commerce applications, we can do trading of commercial goods on online marketplaces. To buy specific items and goods, you simply need to make electronic transactions like UPI, Phonepe, etc. through your smartphone or computer. Flipkart, Amazon, OLX, and, Quiker are examples of e-commerce applications.
2. Educational Apps: Educational apps are too much used to improve knowledge and peoples get productivity. Apps for education can make people more interactive, more engaged, and perform better. Keeping teaching methods good is integral to getting students engaged in their studies and learning apps are a

fantastic way of achieving this. For example, Google Classroom, SoloLearn, edX, Duolingo, etc.

3. Social Media Apps: Social media apps give the opportunity to the peoples connect and communicate together. These apps are mainly used for sharing purposes and making fun. Many peoples use social media applications for influence, marketing/ business, entrepreneurship, etc. Instagram, Facebook, WhatsApp, YouTube, LinkedIn, etc. are examples of social media applications.
4. Productivity Apps: Productivity apps typically organize and complete complex tasks for you, anything from sending an email to figuring out a tip. The easy-to-use Google Drive app gives users access to all of the files saved to the cloud-based storage service across multiple devices. Productivity applications arise in many different forms and they often take a different approach to improving your workflow. For example, Hive, Todoist, Google Docs, etc.
5. Entertainment Apps: Entertainment apps are widely used apps worldwide. It contains OTT platforms and novels and other content. These platforms entertain people and give them much more knowledge about different things. Everyone is watching OTT platforms and those are trending these days, and their development is also in demand all over the world. Hotstar, Netflix, and Amazon prime video are the best examples of this entertainments applications.

❖ Android Versions

Android has evolved through various versions since its inception. Each version brings new features, enhancements, and optimizations to the platform. Here is a list of the major Android versions released to date:

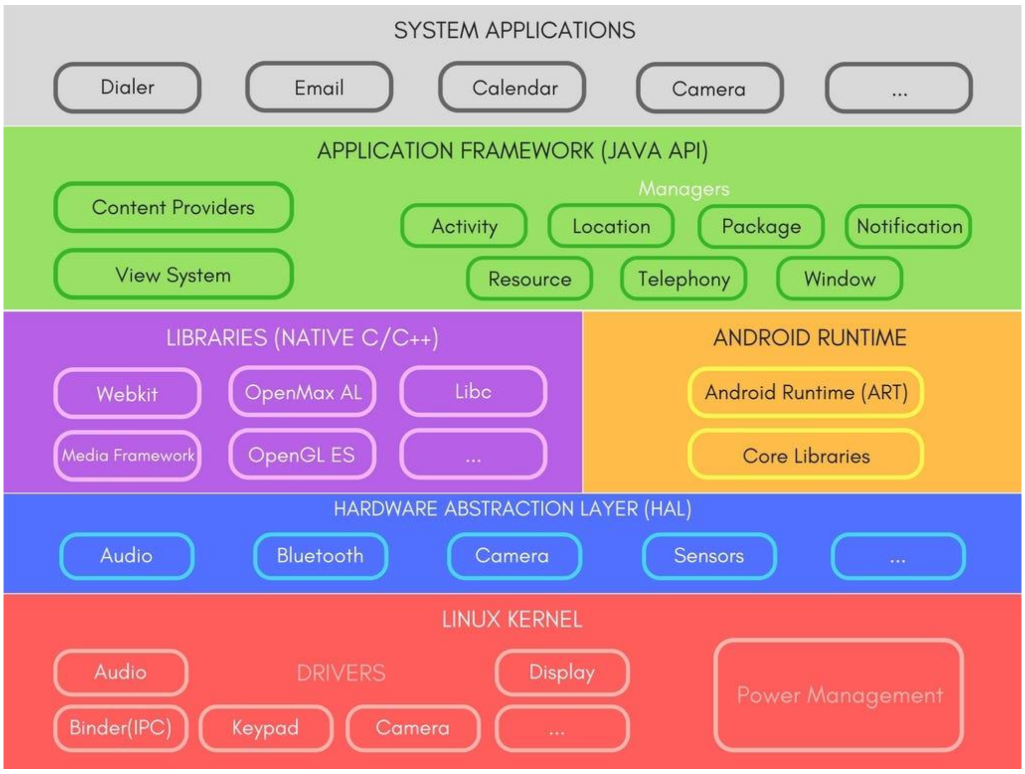
- Android 1.0 (Astro): The initial version of Android was released on September 23, 2008. It introduced basic functionalities such as web browsing, camera support, and access to Google services like Gmail and Google Maps.
- Android 1.1 (Bender): Released on February 9, 2009, Android 1.1 included minor updates and bug fixes to improve system stability and performance.
- Android 1.5 (Cupcake): Introduced on April 27, 2009, Android 1.5 brought significant improvements, such as an on-screen keyboard, support for third-party widgets, and video recording capabilities.
- Android 1.6 (Donut): Released on September 15, 2009, Android 1.6 featured updates to the user interface, improved search functionality, and support for CDMA networks.

- Android 2.0/2.1 (Eclair): Android 2.0 and 2.1, known collectively as Eclair, were released on October 26, 2009. Eclair introduced features such as multiple account support, Bluetooth 2.1, and an updated web browser.
- Android 2.2 (Froyo): Released on May 20, 2010, Android 2.2 (Froyo) introduced significant performance improvements, support for Adobe Flash Player, and the ability to install apps on external storage.
- Android 2.3 (Gingerbread): Introduced on December 6, 2010, Android 2.3 (Gingerbread) focused on refining the user interface, improving gaming performance, and adding support for near field communication (NFC).
- Android 3.0/3.1/3.2 (Honeycomb): Android 3.0 (Honeycomb) was released on February 22, 2011, and was specifically designed for tablets. It featured a redesigned user interface, support for multicore processors, and improved multitasking capabilities.
- Android 4.0 (Ice Cream Sandwich): Released on October 18, 2011, Android 4.0 (Ice Cream Sandwich) merged the tablet and smartphone versions of Android. It introduced features such as a new user interface, enhanced multitasking, and support for facial recognition.
- Android 4.1/4.2/4.3 (Jelly Bean): Android 4.1 (Jelly Bean) was released on July 9, 2012, followed by subsequent updates 4.2 and 4.3. Jelly Bean introduced features such as improved performance, enhanced notifications, and support for multiple user accounts on tablets.
- Android 4.4 (KitKat): Released on October 31, 2013, Android 4.4 (KitKat) focused on optimizing the operating system for low-end devices. It introduced features such as improved memory management, a new dialer app, and support for cloud printing.
- Android 5.0/5.1 (Lollipop): Android 5.0 (Lollipop) was released on November 12, 2014, followed by updates to 5.1. Lollipop introduced the Material Design language, improved performance, enhanced security features, and support for 64-bit processors.
- Android 6.0 (Marshmallow): Released on October 5, 2015, Android 6.0 (Marshmallow) introduced features such as app permissions, Google Now on Tap, and a new battery-saving feature called Doze.
- Android 7.0/7.1 (Nougat): Android 7.0 (Nougat) was released on August 22, 2016, followed by updates to 7.1. Nougat introduced features such as split-screen multitasking, enhanced notifications, and support for Daydream VR.

- Android 8.0/8.1 (Oreo): Android 8.0 (Oreo) was released on August 21, 2017, followed by updates to 8.1. Oreo introduced features, such as picture-in-picture mode, notification dots, and improved battery life, through background app limitations.
- Android 9 (Pie): Released on August 6, 2018, Android 9 (Pie) introduced features such as gesture-based navigation, adaptive battery, and digital wellbeing tools to help users monitor their smartphone usage.
- Android 10: Released on September 3, 2019, Android 10 introduced features such as a system-wide dark mode, improved privacy controls, and support for foldable smartphones.
- Android 11: Released on September 8, 2020, Android 11 focused on enhancing communication, privacy, and control with features like chat bubbles, one-time permissions, and improved media controls.
- Android 12: Released on October 4, 2021, Android 12 introduced a major visual overhaul with Material You design language, enhanced privacy features, and performance improvements.
- Android 13: Android 13 focused on user privacy with features like a photo picker and notification permission settings. Building on Android 12's tablet optimizations, Android 13 enhances system UI, multitasking, and compatibility modes.
- Android 14: Released on October 4, 2023, Android 14 enhances accessibility with features like 200% font scaling and customizable lock screens. Additionally, it introduces support for lossless audio formats and an improved magnifier for low-vision users.
- Android 15: It is the upcoming iteration of the Android operating system, slated for release in early 2025. It introduces advanced encryption features for secure data storage and transmission, among other features.

❖ Android Architecture

The Android architecture is a layered structure that defines the components and interactions within the Android operating system.



1. Linux kernel layer

At the core of the Android architecture lies the Linux kernel, which provides essential hardware abstraction, memory management, process management, security, and device drivers. The Linux kernel serves as the foundation upon which the Android operating system is built, offering low-level functionalities that interact directly with the underlying hardware components of the device.

Key features of the Linux kernel layer in the Android architecture include:

- Hardware abstraction: The Linux kernel abstracts hardware functionalities, allowing the upper layers of the Android stack to interact with hardware components through standardized interfaces.
- Process management: The Linux kernel manages processes and threads, allocating system resources such as CPU time, memory, and input/output operations.
- Memory management: The Linux kernel handles memory allocation, virtual memory management, and memory protection to ensure efficient utilization of system resources.
- Security mechanisms: The Linux kernel enforces security policies through access control mechanisms, permissions, and secure execution environments.

- Device drivers: The Linux kernel provides device drivers to facilitate communication between the operating system and hardware peripherals, such as display drivers, camera drivers, input/output drivers, and network drivers.

2. Hardware abstraction layer (HAL)

Above the Linux kernel layer resides the HAL, which abstracts hardware-specific functionalities and provides standardized interfaces for device drivers and hardware components. The HAL enables device manufacturers to develop drivers for specific hardware configurations while ensuring compatibility with the Android framework.

Key components of the hardware abstraction layer include:

- HAL modules: HAL modules encapsulate hardware-specific functionalities, such as camera, audio, display, sensors, and input devices, into standardized interfaces accessible to higher-level software layers.
- Interface definitions: The HAL defines standardized interfaces, known as Hardware Abstraction Interfaces (HAIs), which specify the methods and parameters for interacting with hardware components.
- Vendor-specific implementations: Device manufacturers like Samsung or Huawei provide vendor-specific implementations of HAL modules tailored to their hardware configurations, ensuring seamless integration with the Android framework.

3. Native libraries layer

The native libraries layer consists of libraries written in C and C++ programming languages that provide core system functionalities and support for native code execution within Android applications. These libraries augment the capabilities of the Java-based Android framework and enable developers to access low-level system resources and hardware features.

Key native libraries in the Android architecture include:

- libc: The C standard library provides fundamental programming utilities and functions for memory management, string manipulation, input/output operations, and system calls.
- libm: The Math library contains mathematical functions and operations, including arithmetic, trigonometric, exponential, and logarithmic functions.
- libz: The Zlib library implements data compression and decompression algorithms, facilitating file compression and decompression operations within Android applications.

- libjpeg/libpng: These libraries provide support for image processing and manipulation, including JPEG and PNG image format decoding and encoding.
- OpenGL/OpenGL ES: The OpenGL (for desktop) and OpenGL ES (for embedded systems) libraries enable hardware-accelerated graphics rendering and 3D rendering within Android applications.

4. Android runtime layer

The Android runtime (ART) layer is responsible for executing and managing Android applications bytecode compiled from Java or Kotlin source code. ART employs ahead-of-time (AOT) compilation to convert bytecode into native machine code, enhancing runtime performance and reducing memory overheads.

Key components of the Android Runtime layer include:

- ART compiler: The ART compiler translates bytecode into native machine code during the installation or upgrade of Android applications, improving runtime performance and efficiency.
- Dalvik virtual machine (legacy): In earlier versions of Android, the Dalvik virtual machine executed bytecode in the form of Dalvik Executable (DEX) files. Dalvik employed just-in-time (JIT) compilation to convert bytecode into native code at runtime.

5. Java API framework layer

The Java application programming interface (API) framework layer comprises a comprehensive set of libraries, APIs, and runtime environments that facilitate the development of Android applications using Java or Kotlin programming languages.

The Java API Framework exposes high-level functionalities and system services to developers, enabling them to create rich, interactive, and feature-rich applications. Key components of the Java API Framework layer include:

- Android software development kit (SDK): The Android SDK provides a collection of development tools, libraries, sample code, and documentation for building Android applications. It includes the Android Debug Bridge (ADB), Android Studio IDE, Android Emulator, and various command-line utilities.
- Core libraries: The core libraries contain essential classes and packages for application development, including data structures, utilities, I/O operations, networking, graphics, and user interface components.
- Android framework APIs: The Android framework APIs expose system-level functionalities and services, such as activity management, resource handling,

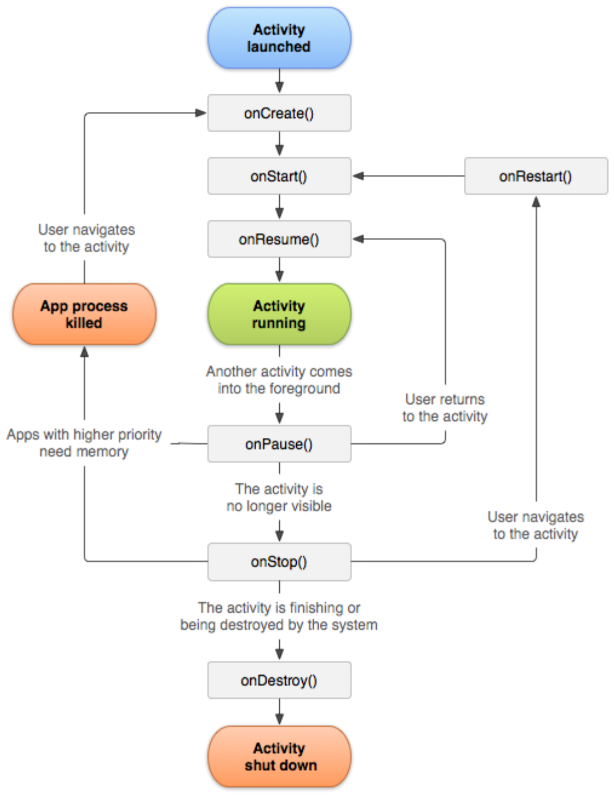
content providers, intents, services, and user interface components (views, layouts, widgets).

6. Application layer

At the topmost layer of the Android architecture is the application layer, which consists of user-installed applications, system applications, and system services running on the Android platform. This layer encompasses a diverse range of applications, including productivity tools, multimedia players, games, social networking apps, communication apps, and more.

❖ Android Activity Lifecycle

In Android, an activity is referred to as one screen in an application. It is very similar to a single window of any desktop application. An Android app consists of one or more screens or activities.
Each activity goes through various stages or a lifecycle and is managed by activity stacks. So when a new activity starts, the previous one always remains below it. There are four stages of an activity.


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Android activities have a specific lifecycle that they go through as the user interacts with them. Understanding the activity lifecycle is crucial for developing robust and responsive Android apps. In this post, we will go over the various states that an activity can be in and the corresponding methods that are called during those states.

The activity lifecycle is composed of the following states:

1. onCreate(): This method is called when the activity is first created. It is responsible for initializing the activity's UI, such as inflating the layout and finding the views.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //initialize views and other UI related tasks
}
```

2. onStart(): This method is called when the activity becomes visible to the user. It is a good place to start animations and other visual changes.

```
@Override
protected void onStart() {
    super.onStart();
    //start animations and other visual changes
}
```

3. onResume(): This method is called when the activity becomes the foreground activity. It isthe state in which the user can interact with the activity. This is where you should register any listeners or start any services that need to be running while the activity is in the foreground.

```
@Override
protected void onResume() {
    super.onResume();
    //register listeners and start services
}
```

4.onPause(): This method is called when the activity is no longer the foreground activity. It is a good place to unregister listeners, save any data that needs to be saved, and stop any services that don't need to be running in the background.

```
@Override
protected void onPause() {
    super.onPause();
    //unregister listeners, save data and stop services
}
```

5. onStop(): This method is called when the activity is no longer visible to the user. It is a good place to stop animations and other visual changes.

```
@Override
protected void onStop() {
    super.onStop();
    //stop animations and other visual changes
}
```

6.onDestroy(): This method is called when the activity is about to be destroyed. It is a good place to release any resources and clean up any remaining data.

```
@Override
protected void onDestroy() {
    super.onDestroy();
    //release resources and clean up data
}
```

It's important to note that an activity can also be in a "stopped" state if it is temporarily obscured by another activity, but it will still retain its state and not be destroyed. In this case, the onStop() method will be called, but not onDestroy(). The onRestart() method will be called when the activity becomes visible again.

```
@Override
protected void onRestart() {
    super.onRestart();
    //refresh data or perform any other necessary tasks
}
```

In conclusion, the activity lifecycle is an important aspect of Android development and understanding the various states and methods that are called during those states can help you create more robust and responsive apps. It’s important to remember that the activity can be in different states depending on the actions of the user and it’s crucial to handle the data and resources accordingly.

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toast toast = Toast.makeText(getApplicationContext(), "onCreate
Called", Toast.LENGTH_LONG).show();
    }

    protected void onStart() {
        super.onStart();
        Toast toast = Toast.makeText(getApplicationContext(), "onStart
Called", Toast.LENGTH_LONG).show();
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Toast toast = Toast.makeText(getApplicationContext(), "onRestart
Called", Toast.LENGTH_LONG).show();
    }
    protected void onPause() {
        super.onPause();
        Toast toast = Toast.makeText(getApplicationContext(), "onPause
Called", Toast.LENGTH_LONG).show();
    }

    protected void onResume() {
        super.onResume();
        Toast toast = Toast.makeText(getApplicationContext(), "onResume
Called", Toast.LENGTH_LONG).show();
    }

    protected void onStop() {
        super.onStop();
        Toast toast = Toast.makeText(getApplicationContext(), "onStop Called",
Toast.LENGTH_LONG).show();
    }

    protected void onDestroy() {
        super.onDestroy();
        Toast toast = Toast.makeText(getApplicationContext(), "onDestroy
Called", Toast.LENGTH_LONG).show();
    }
}
```

❖ Install Android Studio on Windows

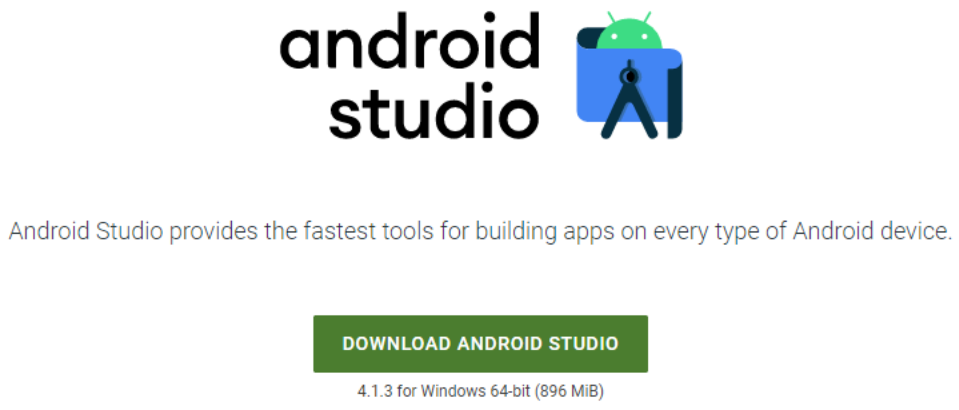
Android Studio System Requirements for Windows

- Microsoft Windows 7/8/10 (32-bit or 64-bit)
- 4 GB RAM minimum, 8 GB RAM recommended (plus 1 GB for the Android Emulator)
- 2 GB of available disk space minimum, 4 GB recommended (500 MB for IDE plus 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution

Steps to Install Android Studio on Windows

Step 1: Head over to this link (https://developer.android.com/studio/#downloads) to get the Android Studio executable or zip file.

Step 2: Click on the Download Android Studio Button.

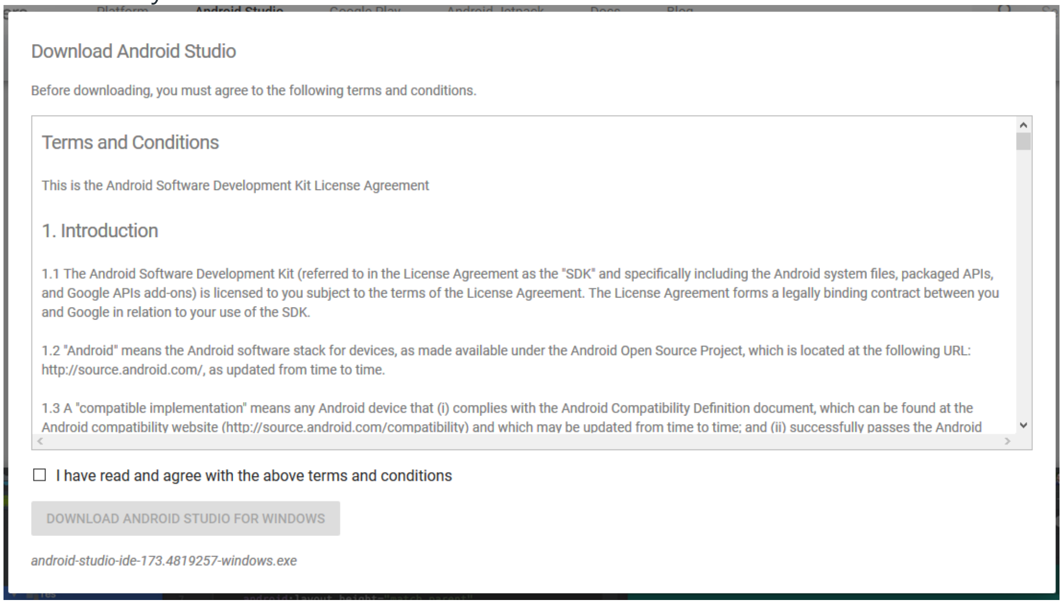

android studio

Android Studio provides the fastest tools for building apps on every type of Android device.

DOWNLOAD ANDROID STUDIO

4.1.3 for Windows 64-bit (896 MiB)

Click on the “I have read and agree with the above terms and conditions” checkbox followed by the download button.


Download Android Studio

Before downloading, you must agree to the following terms and conditions.

Terms and Conditions

This is the Android Software Development Kit License Agreement

1. Introduction

1.1 The Android Software Development Kit (referred to in the License Agreement as the "SDK" and specifically including the Android system files, packaged APIs, and Google APIs add-ons) is licensed to you subject to the terms of the License Agreement. The License Agreement forms a legally binding contract between you and Google in relation to your use of the SDK.

1.2 "Android" means the Android software stack for devices, as made available under the Android Open Source Project, which is located at the following URL: http://source.android.com/, as updated from time to time.

1.3 A "compatible implementation" means any Android device that (i) complies with the Android Compatibility Definition document, which can be found at the Android compatibility website (http://source.android.com/compatibility) and which may be updated from time to time; and (ii) successfully passes the Android

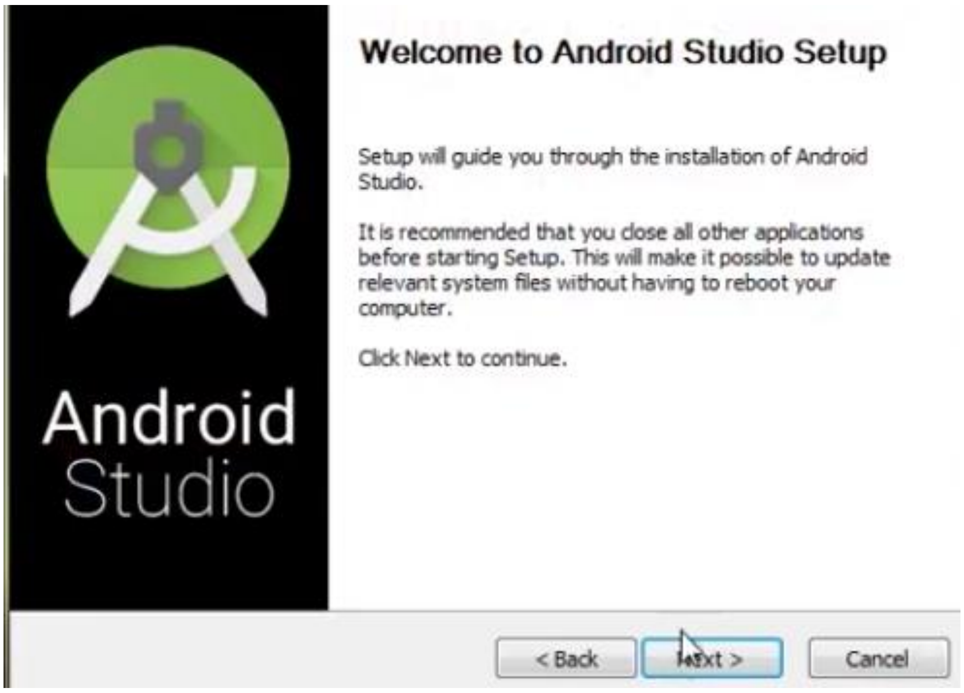
☐ I have read and agree with the above terms and conditions

DOWNLOAD ANDROID STUDIO FOR WINDOWS

android-studio-ide-173.4819257-windows.exe

Click on the Save file button in the appeared prompt box and the file will start downloading.

Step 3: After the downloading has finished, open the file from downloads and run it. It will prompt the following dialog box.

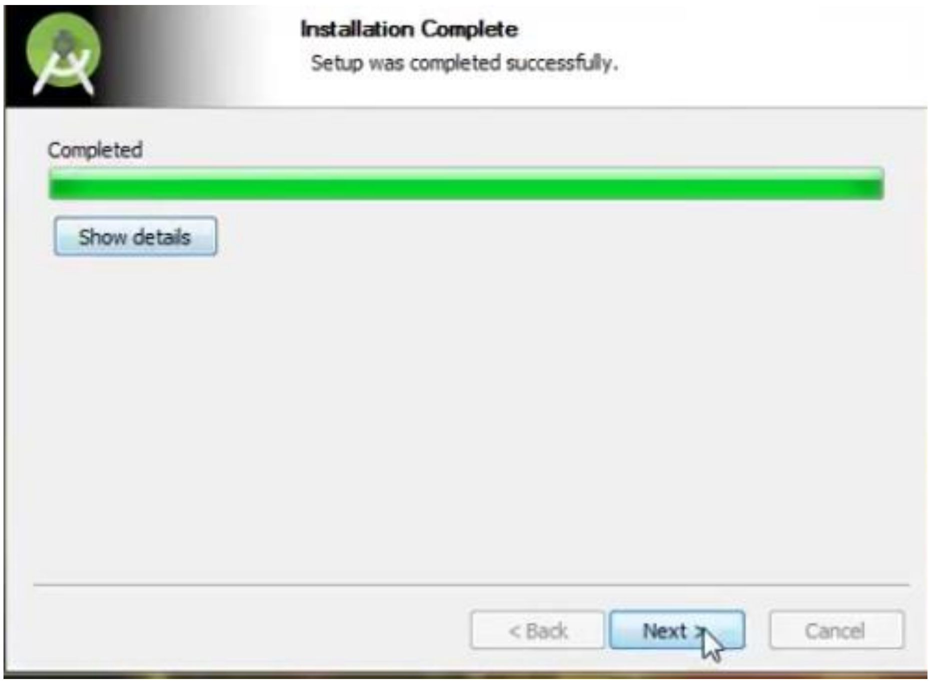

Welcome to Android Studio Setup

Setup will guide you through the installation of Android Studio.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

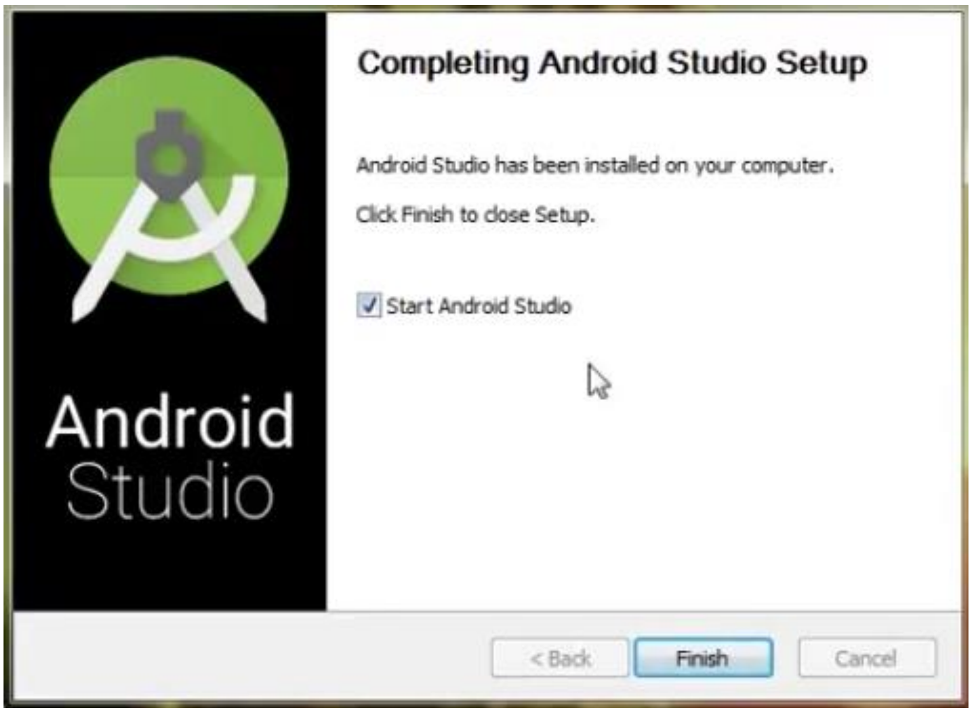
Click Next to continue.

Click on next. In the next prompt, it'll ask for a path for installation. Choose a path and hit next.
Step 4: It will start the installation, and once it is completed, it will be like the image shown below.


Installation Complete
Setup was completed successfully.

Completed

Click on next.

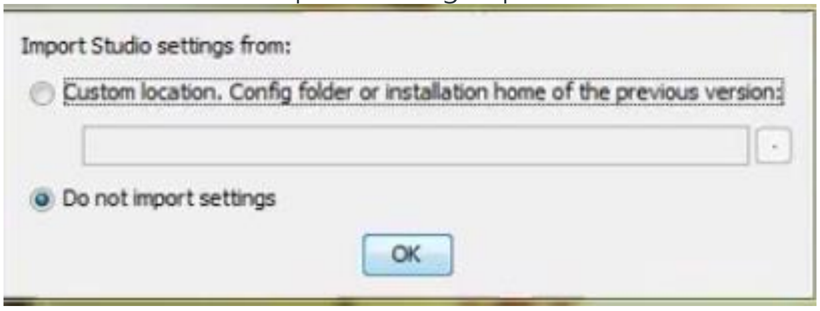

Completing Android Studio Setup

Android Studio has been installed on your computer.

Click Finish to close Setup.

[✓] Start Android Studio

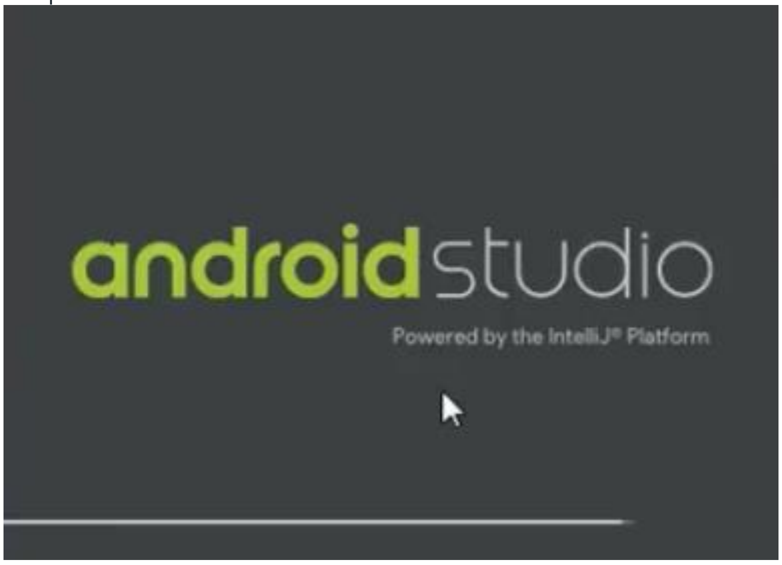
Step 5: Once " Finish " is clicked, it will ask whether the previous settings need to be imported [if the android studio had been installed earlier], or not. It is better to choose the 'Don't import Settings option'.


Import Studio settings from:

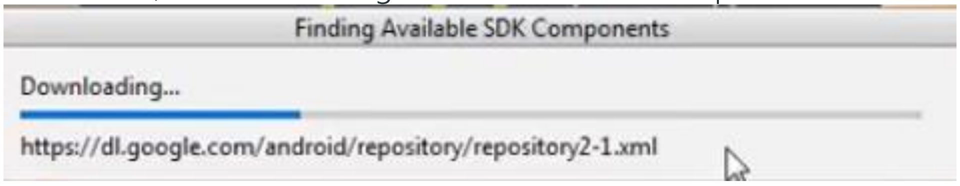
○ Custom location. Config folder or installation home of the previous version:

● Do not import settings

Click the OK button.
Step 6: This will start the Android Studio.



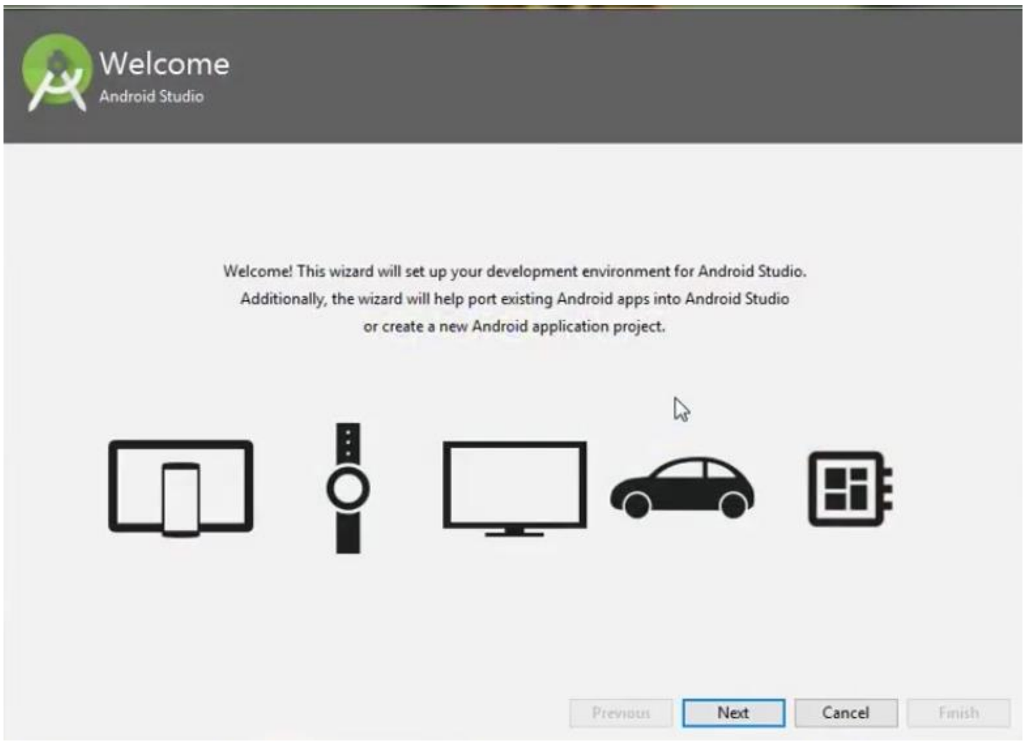
Meanwhile, it will be finding the available SDK components.


Finding Available SDK Components

Downloading...

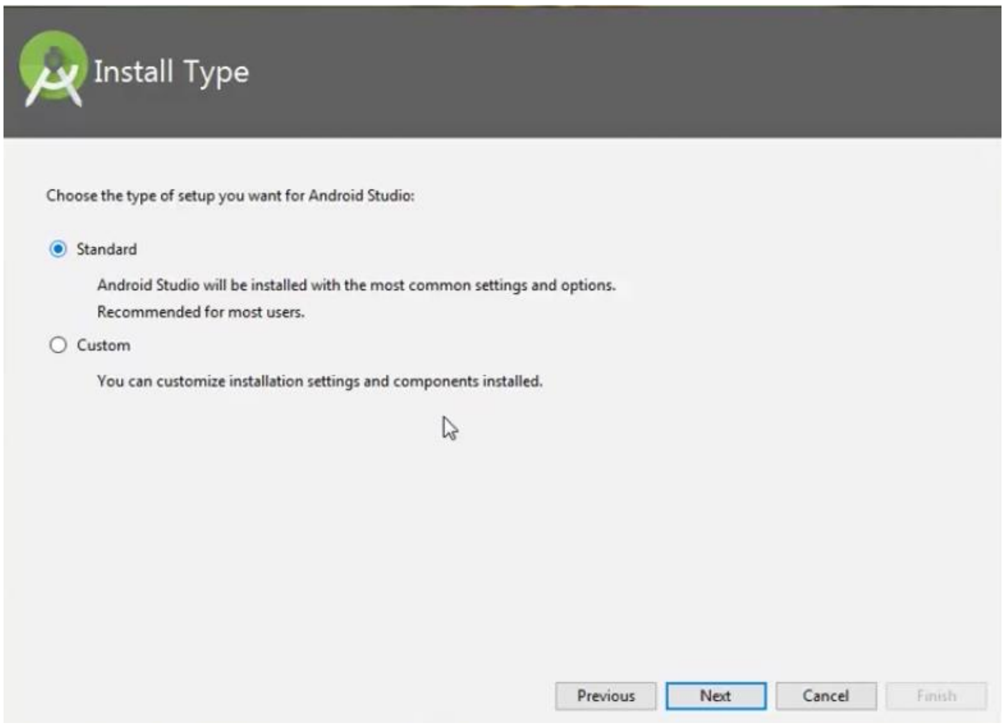
https://dl.google.com/android/repository/repository2-1.xml

Step 7: After it has found the SDK components, it will redirect to the Welcome dialog box.



Welcome! This wizard will set up your development environment for Android Studio. Additionally, the wizard will help port existing Android apps into Android Studio or create a new Android application project.

Click on Next .



Choose the type of setup you want for Android Studio:

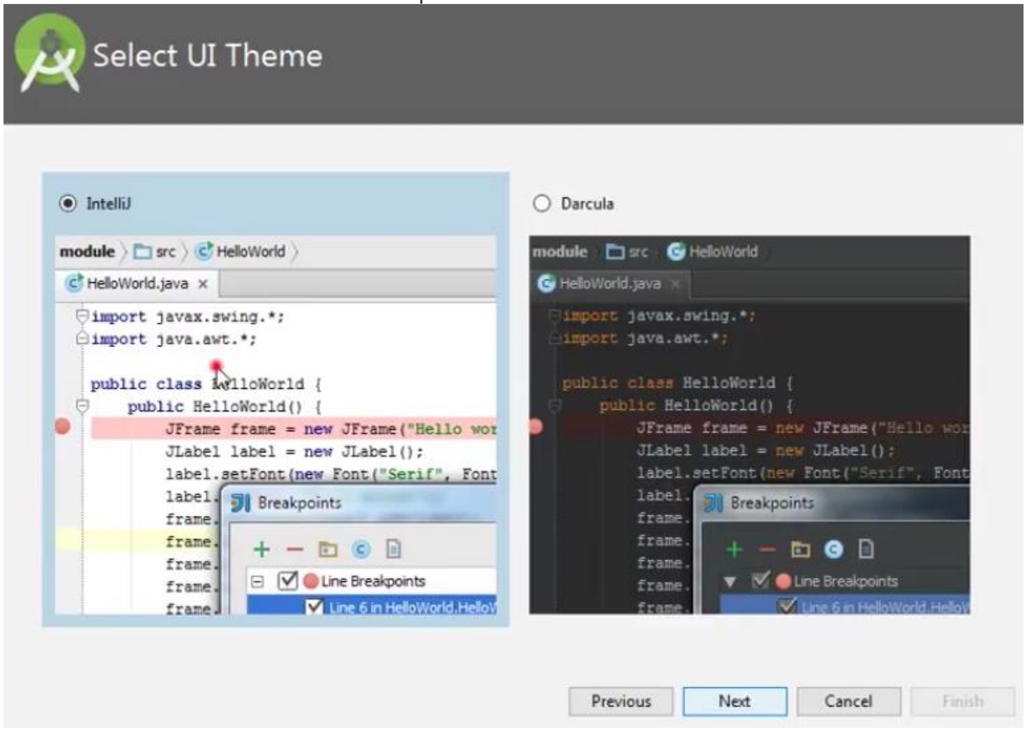
(●) Standard

Android Studio will be installed with the most common settings and options. Recommended for most users.

( ) Custom

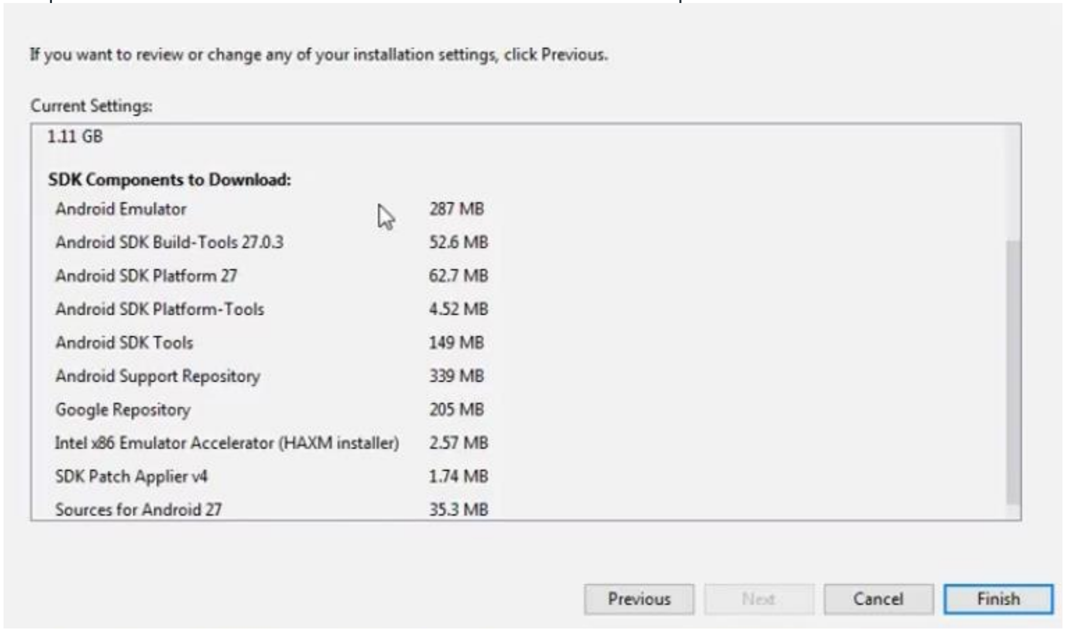
You can customize installation settings and components installed.

Choose Standard and click on Next. Now choose the theme, whether the Light theme or the Dark one. The light one is called the IntelliJ theme whereas the dark theme is called Dracula . Choose as required.



```
import javax.swing.*;
import java.awt.*;

public class HelloWorld {
    public HelloWorld() {
        JFrame frame = new JFrame("Hello wor
        JLabel label = new JLabel();
        label.setFont(new Font("Serif", Font
```

Click on the Next button.
Step 8: Now it is time to download the SDK components.



If you want to review or change any of your installation settings, click Previous.

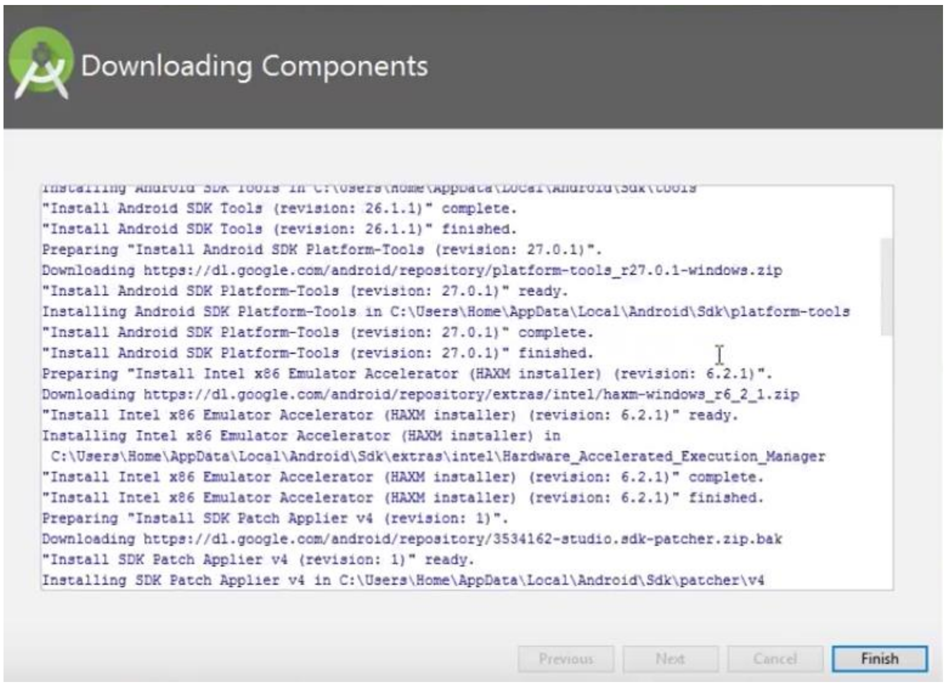
Current Settings:

1.11 GB

SDK Components to Download:

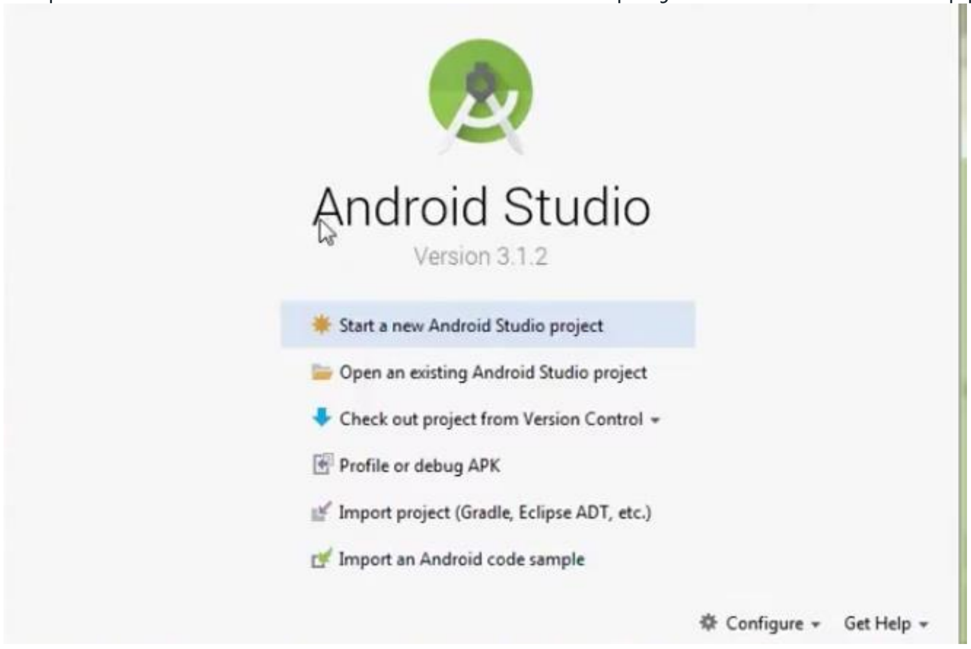
Android Emulator	287 MB
Android SDK Build-Tools 27.0.3	52.6 MB
Android SDK Platform 27	62.7 MB
Android SDK Platform-Tools	4.52 MB
Android SDK Tools	149 MB
Android Support Repository	339 MB
Google Repository	205 MB
Intel x86 Emulator Accelerator (HAXM installer)	2.57 MB
SDK Patch Applier v4	1.74 MB
Sources for Android 27	35.3 MB

Click on Finish. Components begin to download let it complete.



```
Installing Android SDK Tools in C:\Users\Home\AppData\Local\Android\Sdk\tools
"Install Android SDK Tools (revision: 26.1.1)" complete.
"Install Android SDK Tools (revision: 26.1.1)" finished.
Preparing "Install Android SDK Platform-Tools (revision: 27.0.1)".
Downloading https://dl.google.com/android/repository/platform-tools_r27.0.1-windows.zip
"Install Android SDK Platform-Tools (revision: 27.0.1)" ready.
Installing Android SDK Platform-Tools in C:\Users\Home\AppData\Local\Android\Sdk\platform-tools
"Install Android SDK Platform-Tools (revision: 27.0.1)" complete.
"Install Android SDK Platform-Tools (revision: 27.0.1)" finished.
Preparing "Install Intel x86 Emulator Accelerator (HAXM installer) (revision: 6.2.1)".
Downloading https://dl.google.com/android/repository/extras/intel/haxm-windows_r6_2_1.zip
"Install Intel x86 Emulator Accelerator (HAXM installer) (revision: 6.2.1)" ready.
Installing Intel x86 Emulator Accelerator (HAXM installer) in
 C:\Users\Home\AppData\Local\Android\Sdk\extras\intel\Hardware_Accelerated_Execution_Manager
"Install Intel x86 Emulator Accelerator (HAXM installer) (revision: 6.2.1)" complete.
"Install Intel x86 Emulator Accelerator (HAXM installer) (revision: 6.2.1)" finished.
Preparing "Install SDK Patch Applier v4 (revision: 1)".
Downloading https://dl.google.com/android/repository/3534162-studio.sdk-patcher.zip.bak
"Install SDK Patch Applier v4 (revision: 1)" ready.
Installing SDK Patch Applier v4 in C:\Users\Home\AppData\Local\Android\Sdk\patcher\v4
```

The Android Studio has been successfully configured. Now it's time to launch and build apps. Click on the Finish button to launch it.
Step 9: Click on Start a new Android Studio project to build a new app.


Android Studio
Version 3.1.2
Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Profile or debug APK
Import project (Gradle, Eclipse ADT, etc.)
Import an Android code sample
Configure
Get Help

To run your first android app in Android Studio you may refer to Running your first Android app.

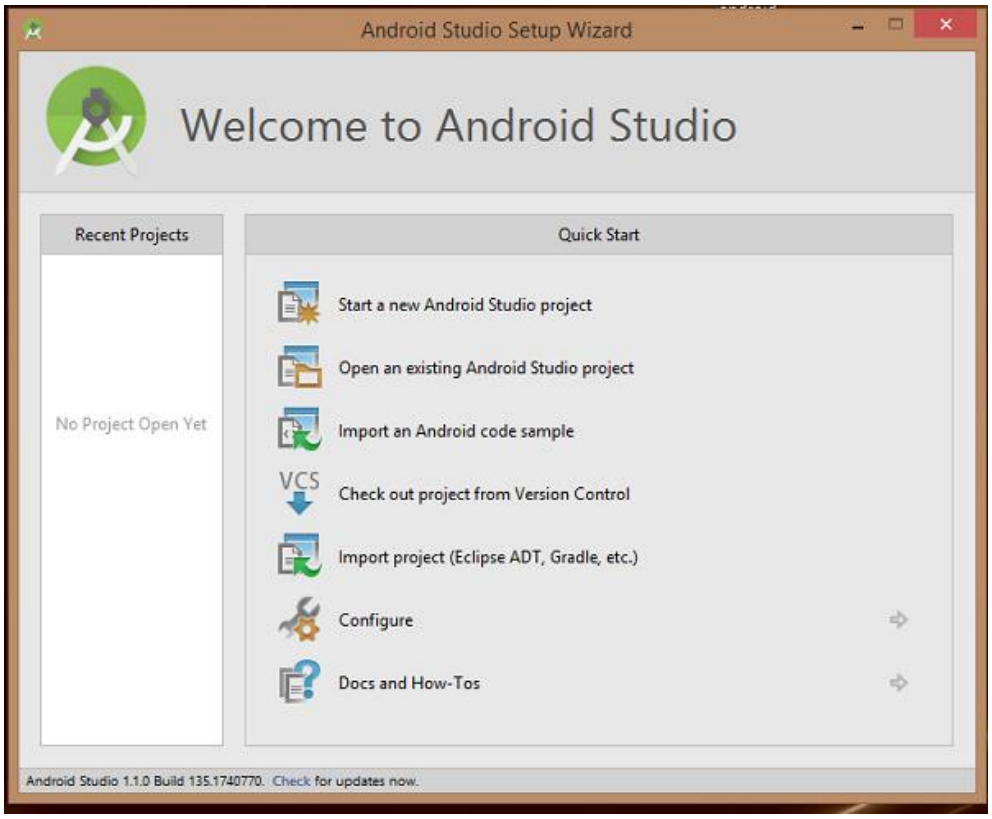
❖ First Android Application

Let us start actual programming with Android Framework. Before you start writing your first example using Android SDK, you have to make sure that you have set-up your Android development environment properly as explained in Android - Environment Set-up tutorial. I also assume that you have a little bit working knowledge with Android studio.

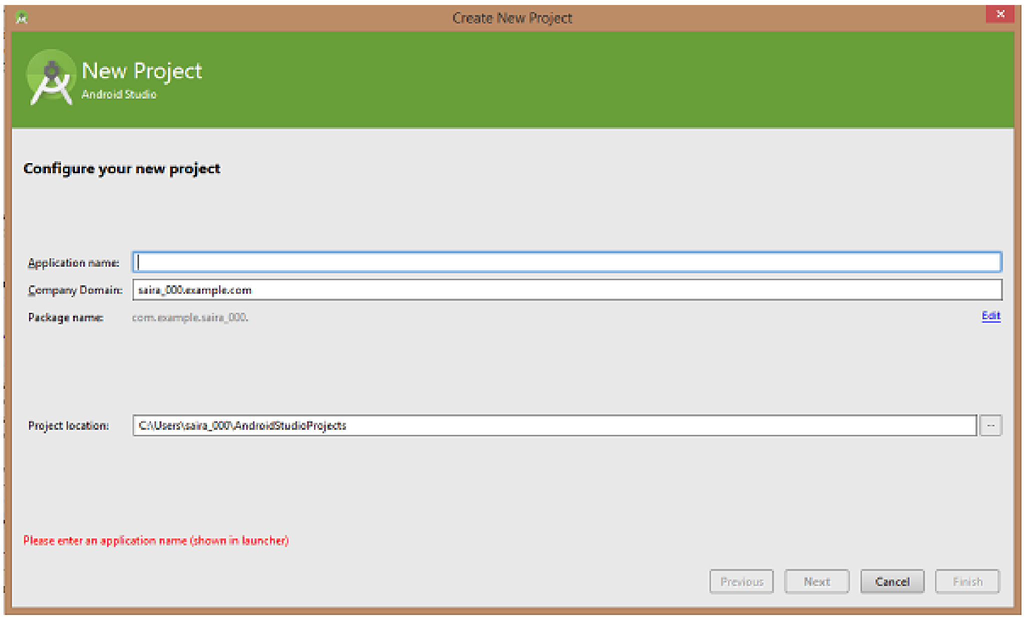
So let us proceed to write a simple Android Application which will print "Hello World!".

Create Android Application

The first step is to create a simple Android Application using Android studio. When you click on Android studio icon, it will show screen as shown below


Android Studio Setup Wizard
Welcome to Android Studio
Recent Projects
No Project Open Yet
Quick Start
Start a new Android Studio project
Open an existing Android Studio project
Import an Android code sample
VCS
Check out project from Version Control
Import project (Eclipse ADT, Gradle, etc.)
Configure
Docs and How-Tos
Android Studio 1.1.0 Build 135.1740770. Check for updates now.

You can start your application development by calling start a new android studio project. in a new installation frame should ask Application name, package information and location of the project.–


Create New Project

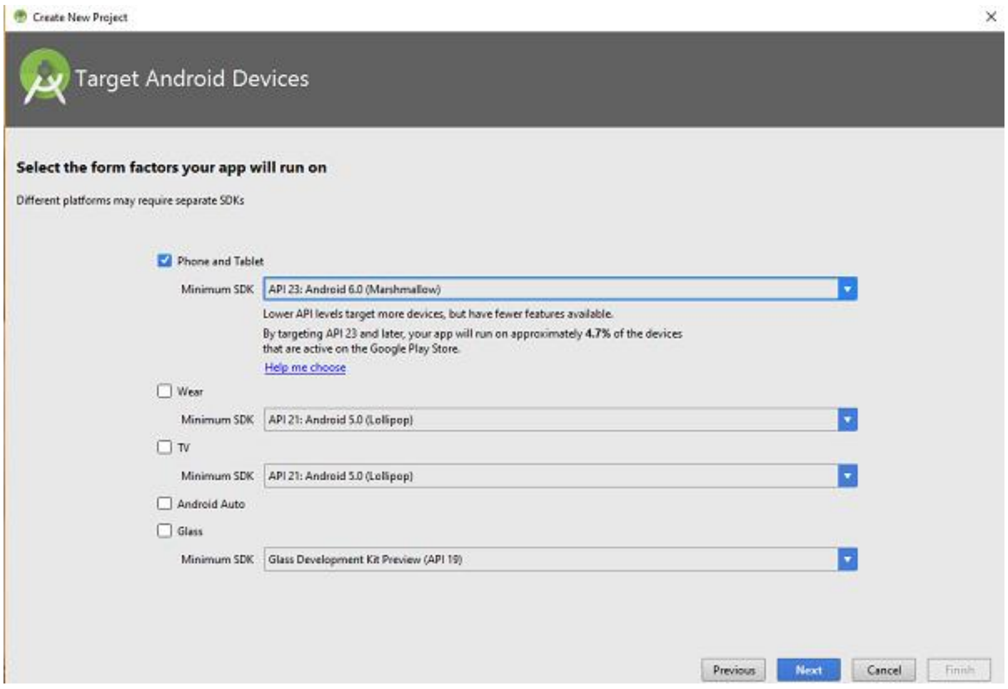
New Project
Android Studio

Configure your new project

Application name:
Company Domain: saira_000.example.com
Package name: com.example.saira_000.
Project location: C:\Users\saira_000\AndroidStudioProjects

Please enter an application name (shown in launcher)

After entered application name, it going to be called select the form factors your application runs on, here need to specify Minimum SDK, in our tutorial, I have declared as API23: Android 6.0(Mashmallow) –


Create New Project

Target Android Devices

Select the form factors your app will run on

Different platforms may require separate SDKs

☑ Phone and Tablet
Minimum SDK: API 23: Android 6.0 (Marshmallow)
Lower API levels target more devices, but have fewer features available.
By targeting API 23 and later, your app will run on approximately **4.7%** of the devices that are active on the Google Play Store.
Help me choose

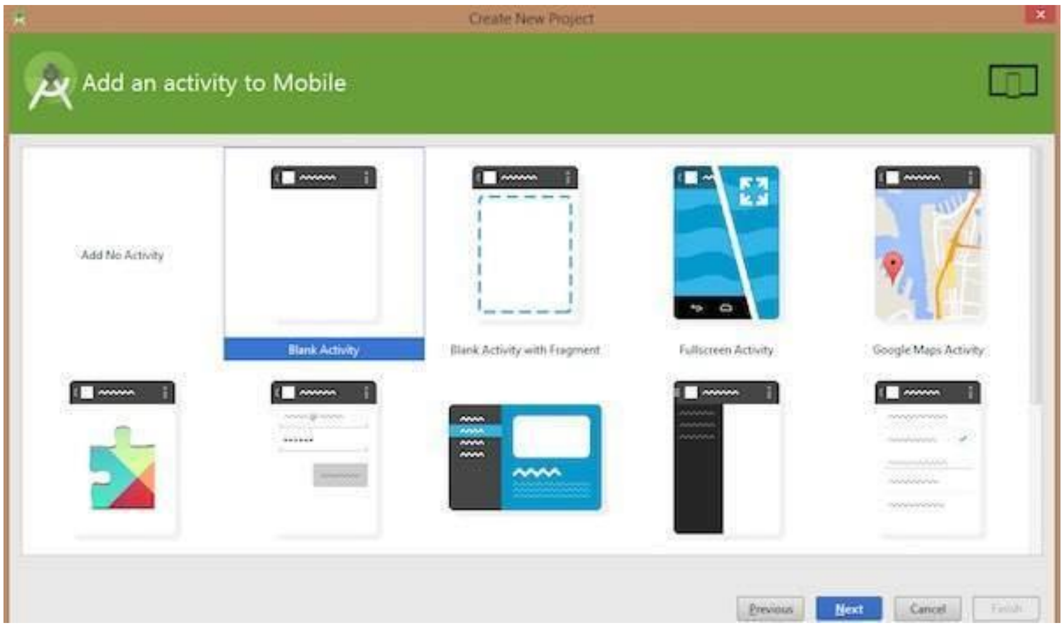
☐ Wear
Minimum SDK: API 21: Android 5.0 (Lollipop)

☐ TV
Minimum SDK: API 21: Android 5.0 (Lollipop)

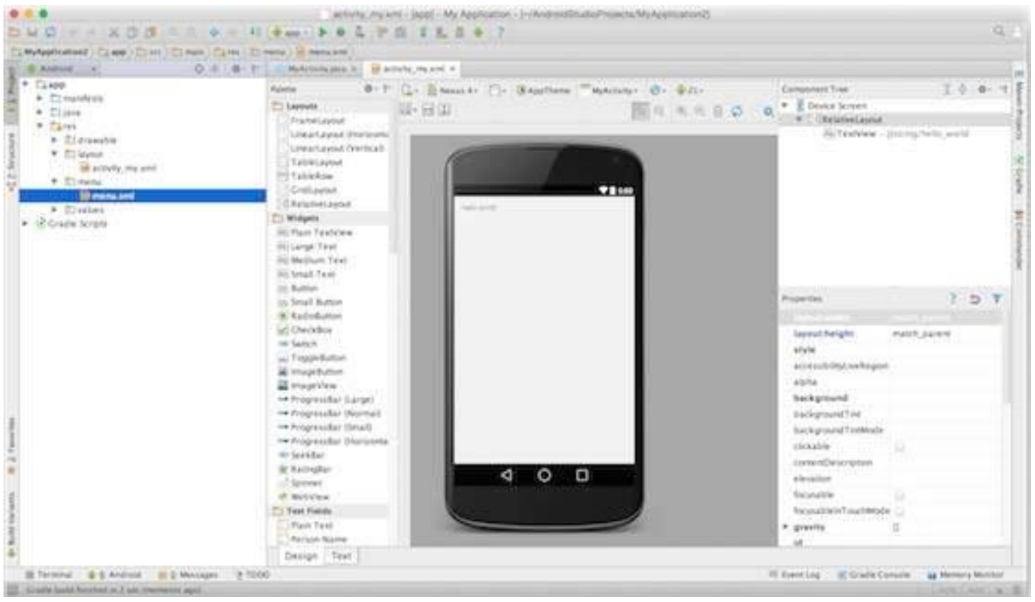
☐ Android Auto

☐ Glass
Minimum SDK: Glass Development Kit Preview (API 19)

The next level of installation should contain selecting the activity to mobile, it specifies the default layout for Applications.

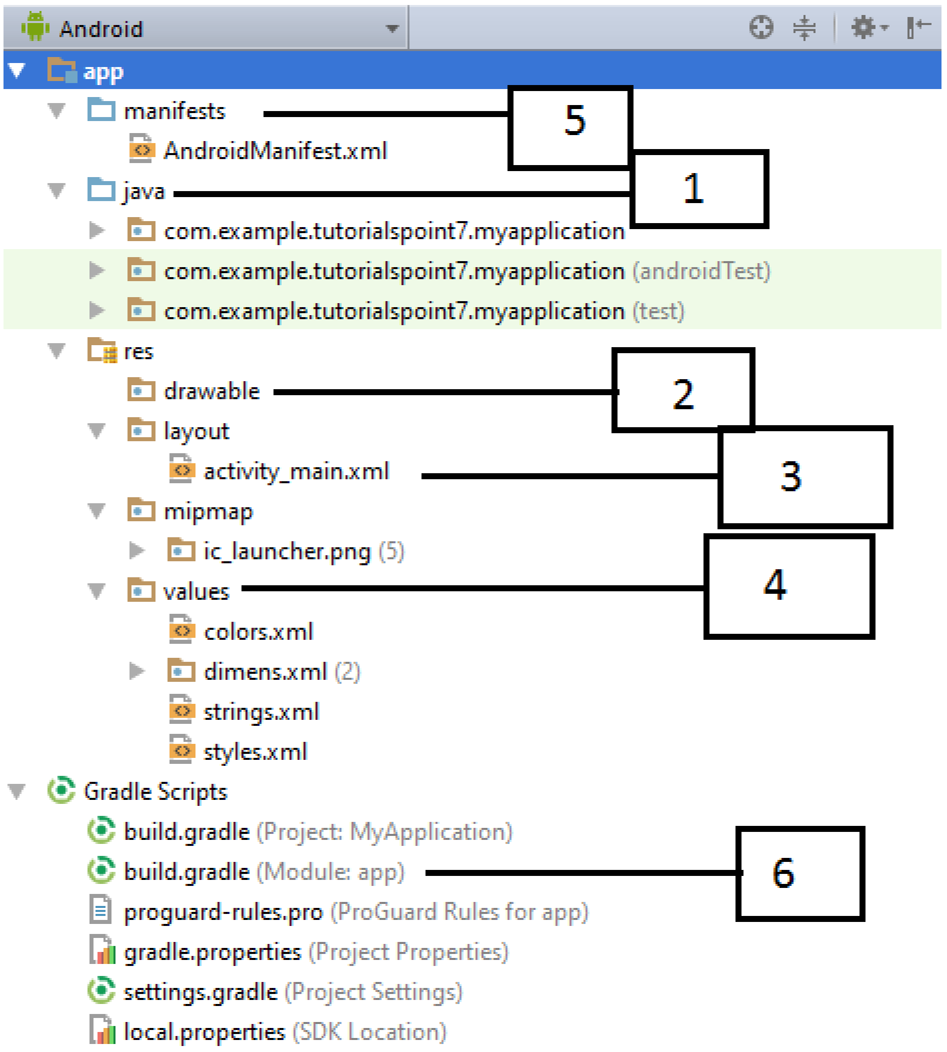


At the final stage it going to be open development tool to write the application code.



Anatomy of Android Application

Before you run your app, you should be aware of a few directories and files in the Android project –


Android
app
manifests
AndroidManifest.xml
java
com.example.tutorialspoint7.myapplication
com.example.tutorialspoint7.myapplication (androidTest)
com.example.tutorialspoint7.myapplication (test)
res
drawable
layout
activity_main.xml
mipmap
ic_launcher.png (5)
values
colors.xml
dimens.xml (2)
strings.xml
styles.xml
Gradle Scripts
build.gradle (Project: MyApplication)
build.gradle (Module: app)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
5
1
2
3
4
6

Sr.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

Following section will give a brief overview of the important application files.

The Main Activity File

The main activity code is a Java file MainActivity.java. This is the actual application file which ultimately gets converted to a Dalvik executable and runs your application. Following is the default code generated by the application wizard for *Hello World!* 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.

The Manifest File

Whatever component you develop as a part of your application, you must 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, so if you do not declare your component in this file, then it will not be considered by the OS. 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. Attribute *android:icon* will point to the application icon available under *res/drawable-hdpi*. The application uses the image named ic_launcher.png located in the drawable folders

The <activity> tag is used to specify an activity and *android:name* attribute specifies the fully qualified class name of the *Activity* subclass and the *android:label* attributes specifies a string to use as the label for the activity. You can specify multiple activities using <activity> tags.

The action for the intent filter is named *android.intent.action.MAIN* to indicate that this activity serves as the entry point for the application. The category for the intent-filter is named *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 explained below.
Hence, *@string/app_name* refers to the *app_name* string defined in the strings.xml file, which is "HelloWorld". Similar way, other strings get populated in the application.

Following is the list of tags which you will use in your 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

The Strings File

The strings.xml file is located in the *res/values* folder and it contains all the text that your application uses. For example, the names of buttons, labels, default text, and similar types of strings go into this file. This file is responsible for their textual content. For example, a default strings file will look like as following file –

```
<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>
```

The Layout File

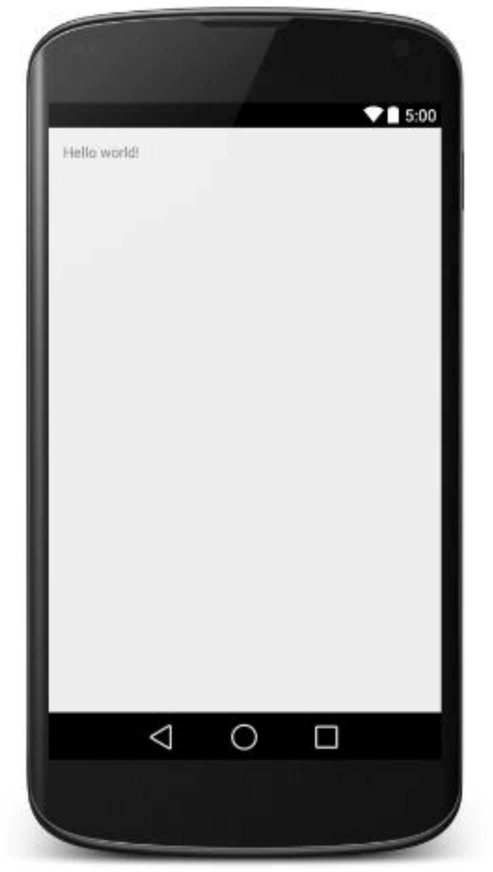
The activity_main.xml is a layout file available in *res/layout* directory, that is referenced by your application when building its interface. You will modify this file very frequently to change the layout of your application. For your "Hello World!" application, this file will have following content related to default layout –

```
<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>
```

This is an example of simple *RelativeLayout* which we will study in a separate chapter. The *TextView* is an Android control used to build the GUI and it have various attributes like *android:layout_width*, *android:layout_height* etc which are being used to set its width and height etc.. The *@string* refers to the strings.xml file located in the res/values folder. Hence, @string/hello_world refers to the hello string defined in the strings.xml file, which is "Hello World!".

Running the Application

Let's try to run our Hello World! application we just created. I assume you had created your AVD while doing environment set-up. 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 –



Congratulations!!! you have developed your first Android Application.