

# Chapter 1

## 1. Mobile Programming

### 1.1. Introduction

Mobile Devices need some type of Operating System to run its services. Earlier OS were fairly simple, since they supported limited capabilities. Modern devices however, added many of the features of a full-fledged desktop computer like CPU, GPU, large storage, multitasking, multipurpose communication device etc. Modern OS combines a PC with other mobile specific features like Bluetooth, GPS, speech recognition, video camera, infrared etc, so mobile OS had to grow to support all these features. This results in the growth of market with a number of different technology platforms. Different competing platforms are driven by different actors. This requires app developers to study and choose the most winning platform available.

#### What is Mobile OS?

- ✓ A software platform on top of which other apps can run on.
- ✓ Experienced a 3 phase evolution
  - PC oriented, Embedded System Oriented and Smart-phone oriented evolution
- ✓ The architecture goes from complex to simple or something in between
- ✓ Evolution has driven by tech advancements in HW, SW and the Internet
- ✓ Resulted in a variety of OS platforms available on the market

Some of the mobile operating systems are:

- ✓ iOS:- Developed by Apple inc. and distributed exclusively for Apple Hardware
- ✓ Windows Phone:- A proprietary software Smartphone OS developed by Microsoft
- ✓ Android:- Developed by Open Handset Alliance (OHA), Led by Google

#### What is Android?

Android is a mobile operating system currently developed by Google, based on the Linux kernel and designed primarily for touch screen mobile devices such as smart phones and tablets. Android offers a unified approach to application development for mobile devices which means developers need to develop only for Android, and their applications should be able to run on different devices powered by Android. Android is an open-source operating system named Android. Google has made the code for all the low-level "stuff" as well as the needed middleware to power and use an electronic device, and gave Android free of cost to anyone who wants to write code and build the operating system from it. There is even a full application framework included, so third-party apps can be built and installed, then made available for the user to run as they like.

The "proper" name for this is the Android Open Source Project, and this is what people mean when they say things like Android is open and free.

Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The Android SDK provides the tools and APIs necessary to develop applications on the Android platform using the Java Programming language.

### 1.1.1. History and Version of Android

The history and versions of android are interesting to know. The code names of android ranges from A to J currently, such as **Aestro, Blender, Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean, KitKat, Lollipop, Marshmallow, Nougat, Oreo, Pie**. Let's understand the android history in a sequence.

1. Initially, **Andy Rubin** founded Android Incorporation in Palo Alto, California, United States in October, 2003.
2. In 17th August 2005, Google acquired android Incorporation. Since then, it is in the subsidiary of Google Incorporation.
3. The key employees of Android Incorporation are **Andy Rubin, Rich Miner, Chris White** and **Nick Sears**.
4. Originally intended for camera but shifted to smart phones later because of low market for camera only.
5. Android is the nick name of Andy Rubin given by coworkers because of his love to robots.
6. Google formed Open Handset Alliance (OHA) on 5<sup>th</sup> November 2007.

#### What is Open Handset Alliance?

The Open Handset Alliance is a group of 84 technology and mobile companies who have come together to accelerate innovation in mobile and offer consumers a richer, less expensive, and better mobile experience.

#### Which was the first commercially available Smartphone running Android?

The first commercially available Smartphone running Android was the **HTC Dream** it came to the market as the T-Mobile G1 in the USA, released on October 22 2008.

#### Android Versions Names

Android has its version named after sweets. The most recent is being Android 9.0 with Pie API level 28.

Date	Version	Code name	API Level
September 23, 2008	1.0	No codename	1
February 9, 2009	1.1	Petit Four	2
April 27, 2009	1.5	Cupcake	3
September 15, 2009	1.6	Donut	4
October 26, 2009	2.0 - 2.1	Éclair	5 - 7
May 20, 2010	2.2 – 2.2.3	Froyo	8
December 6, 2010	2.3 – 2.3.7	Gingerbread	9 - 10
February 22, 2011	3.0 – 3.2.6	Honeycomb	11 - 13
October 18, 2011	4.0 – 4.0.4	Ice Cream Sandwich	14 - 15
July 9, 2012	4.1 - 4.3	Jelly Bean	16 - 18
October 31, 2013	4.4 - 4.4.4	KitKat	19-20
November 12, 2014	5.0 - 5.1.1	Lollipop	21-22
October 5, 2015	6.0 - 6.0.1	Marshmallow	23
August 22, 2016	7.0 -7.1.2	Nougat	24-25
August 21, 2017	8.0	Oreo	26-27
August 6, 2018	9.0	Pie	28

September 3, 2019	10.0	Android 10	29
September 8, 2020	11.0	Android 11	30

## 1.2. Android Architecture

Android is a software stack. **Android software stack** is categorized into five parts:



Figure 1.2.1 Android Architecture

### Linux Kernel

In android the operating system is Linux Kernel. It is the heart of android architecture that exists at the root of android architecture. Linux kernel is responsible for device drivers, power management, memory management, device management and resource access. You can find the Linux Kernel version running on your device under “About phone” or “About tablet” in Android’s settings.

### Functions of Linux Kernel/ Operating System

- ✓ **Memory Management:** Allocate a memory to a new file, Free the memory when a specific file is deleted etc
- ✓ **Power Management:** providing power to various devices like Bluetooth, camera etc
- ✓ **Resource Management:** It provides resources to each process, thus providing the ability to do multiple operations at the same time. i.e. Surfing Internet, Listening Songs etc
- ✓ **Driver Management:** It handles installation of various drivers.

**Middle Ware:****1. Native Applications/Libraries:**

Just by using Java we cannot interact with native applications. (Here native applications means that programs written in some other languages like C, C++, Assembly language which are specific to hardware and operating system). Thus we need the support of native libraries for interacting with such low level media components. On the top of linux kernel, there are **Native libraries** such as WebKit, OpenGL, FreeType, SQLite, Media, C runtime library (libc) etc. The WebKit library is responsible for browser support, SQLite is for database, FreeType for font support, Media for playing and recording audio and video formats.

**2. Application Framework:**

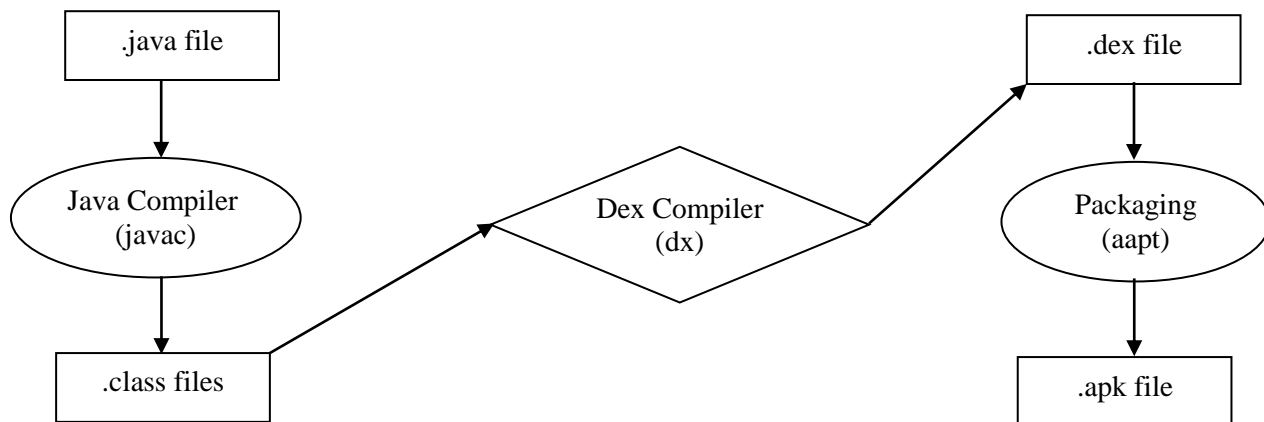
The android framework is the set of API's that allow developers to quickly and easily write apps for android phones. Android framework includes **Android API's** such as UI (User Interface), telephony, resources, locations, Content Providers (data) and package managers. It provides a lot of classes and interfaces for android application development. Application developers are allowed to make use of these services in their applications.

**3. Android Runtime:**

It comprises of Dalvik Virtual Machine and Core Libraries which is responsible to run android application. DVM is like JVM but it is optimized for mobile devices. It consumes less memory and provides fast performance.

**Applications:**

Applications layer is the top most layer of Android Architecture. All applications using android framework uses android runtime and libraries, while android runtime and native libraries are using Linux Kernel.

**Dalvik Virtual Machine:**

Android applications can be developed using languages such as C++, C#, Java etc. But the official language is Java.

**JDK** (Java Development Kit) compile our Java code and it produce a file called **.class** file. It is also called as Java bytecode. Java Bytecode is a platform independent code because it can run on multiple platforms.

In android Dex compiler take all the .class file as input and it produce a single lightweight file called **.dex** file. Dex compile produces a single .dex file for multiple .class files. Dex is termed as Dalvik Executable Code. DVM takes this single .dex file as input and generates the final machine code.

With Android version 2.2(Froyo) Google introduced **JIT(Just in Time)** compilation in Dalvik. Till Android Version 5.0(Lollipop) Dalvik was the runtime used by Android which is being replaced by **ART(Android Run Time)** in later versions.

#### **Why do we need a .dex file when actually even .class file is platform independent?**

1. DVM is Register based which is designed to run on low memory unlike JVM which is stack based. Thus though DVM we are able to achieve small memory footprint with higher battery life.
2. Java tools are free but the JVM is not free, so that android developers from Google have made their own virtual machine and made it as free.

#### **ART vs Dalvik**

ART is a new Android runtime which was introduced experimentally in the 4.4 release KitKat. If you have KitKat then you can see the preview in “setting >Developer Option>Select Runtime”.

#### **Why did Google switched from Dalvik to ART?**

Android apps are deployed in Dalvik bytecode, which is portable, unlike native code. In order to be able to run the app on a device, the code has to be compiled to machine code.

**Dalvik** is based on JIT compilation. It means that each time you run an app, the part of the code required for its execution is going to be translated (compiled) to machine code at that moment. As you progress through the app, additional code is going to be compiled and cached, So that the system can reuse the code while the app is running. Since JIT compiles only a part of the code, it has a smaller memory footprint and uses less physical space on the device.

**ART**, on the other hand compiles the intermediate language, Dalvik bytecode, into a system independent binary. The whole code of the app will be pre-compiled during installation (once), thus removing the lag that we see when we open an app on our device. With no need for JIT compilation, the code should execute much faster. Except for the potential speed increase, the use of ART can provide an important secondary benefit. As ART runs app machine code directly (native execution), it does not hit the CPU as hard as **Just-in-Time** code compiling on Dalvik. Less CPU usage results in less battery drain, which is a big plus for portable devices in general.

So why was not ART implemented earlier?

- ✓ First of all, the generated machine code requires more space than the existing bytecode.
- ✓ Moreover, the code is pre-compiled at install time, so the installation process takes a bit longer.
- ✓ Furthermore, it also corresponds to a larger memory footprint at execution time.
- ✓ This means that fewer apps run concurrently

When first Android devices hit the market, memory and storage capacity were significantly smaller and presented a bottleneck for performance. This is the reason why a JIT approach was the preferred option at that time. Today, memory is much cheaper and thus more abundant, even on low-end devices, so ART is a logical step forward.

#### **Android Application Components**

- ✓ **Activities:** An activity in Android represents a single screen with which a user can interact with. An application can have more than one Activity and each Activity operates independently, but can be linked to one another and each Activity you create must be defined in your application's manifest file. Each Activity in android will be subclass of Activity class defined in Android SDK. Thus public class MainActivity extends Activity (if MainActivity is the name of the Activity).
- ✓ **Services:** Long running background process without any need of user interaction is known as Service. For example, as service might play music in the background while the user is in different application, or it might fetch data over the network without blocking user interaction with an activity. Each service in android will be subclass of service class defined in Android SDK. Thus public class MyService extends Service (if MyService is the name of the service).
- ✓ **Broadcast Receiver:** They handle communication between Android OS and applications. For example the notification that the device battery is low, the sign of earphone as soon as you plug the headset. Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs.
- ✓ **Content Provider:** content providers are used to share data between the applications. In android, the data cannot be shared directly between the two applications.

### Additional Components

There are additional components which will be used in the construction of above mentioned entities, their logic, and wiring between them. These components are:

- ✓ **Fragments:** Represent a behavior or a portion of user interface in an Activity.
- ✓ **Views:** UI elements that are drawn onscreen including buttons, lists forms etc.
- ✓ **Layouts:** View hierarchies that control screen format and appearance of the views.
- ✓ **Intents:** Messages wiring components together.
- ✓ **Resources:** External elements, such as strings, constants and drawable pictures.
- ✓ **Manifest:** Configuration file for the application.

## 1.3. Getting Started

In order to write an Android application, we need a Java Development Kit (JDK), Java Runtime Environment (JRE) and development environment. The common Android application development environments are Android Studio, Eclipse and NetBeans. But officially Android Studio is the recommended IDE for developing Android Projects. Android Studio is a very useful tool made by Google for all Android Developers.

Android Studio includes everything you need to begin developing Android apps. Included in the download kit, are the Software Development Kit (SDK), with all the Android libraries we may need, and the infrastructure to download the many Android emulator instances, so that we can initially run our application, without needing a real device.

**Android Emulator** is used to run, debug and test the android application. If you don't have the real device, it can be the best way to run, debug and test the application. It uses an open source processor emulator technology called **QEMU**. The emulator tool enables you to start the emulator from the command line. You need to write: *emulator -avd <AVD NAME>*

In case of Eclipse IDE, you can create AVD by **Window menu > AVD Manager > New**.

If you want to test the android applications using a real device, unlock the developer's option in Android as follows: Go to **Settings>General>About Phone**. Then scroll and select Software **Information>Build Number**. Now quickly tap on "Build Number" seven times and you will see the message "You are now a developer".

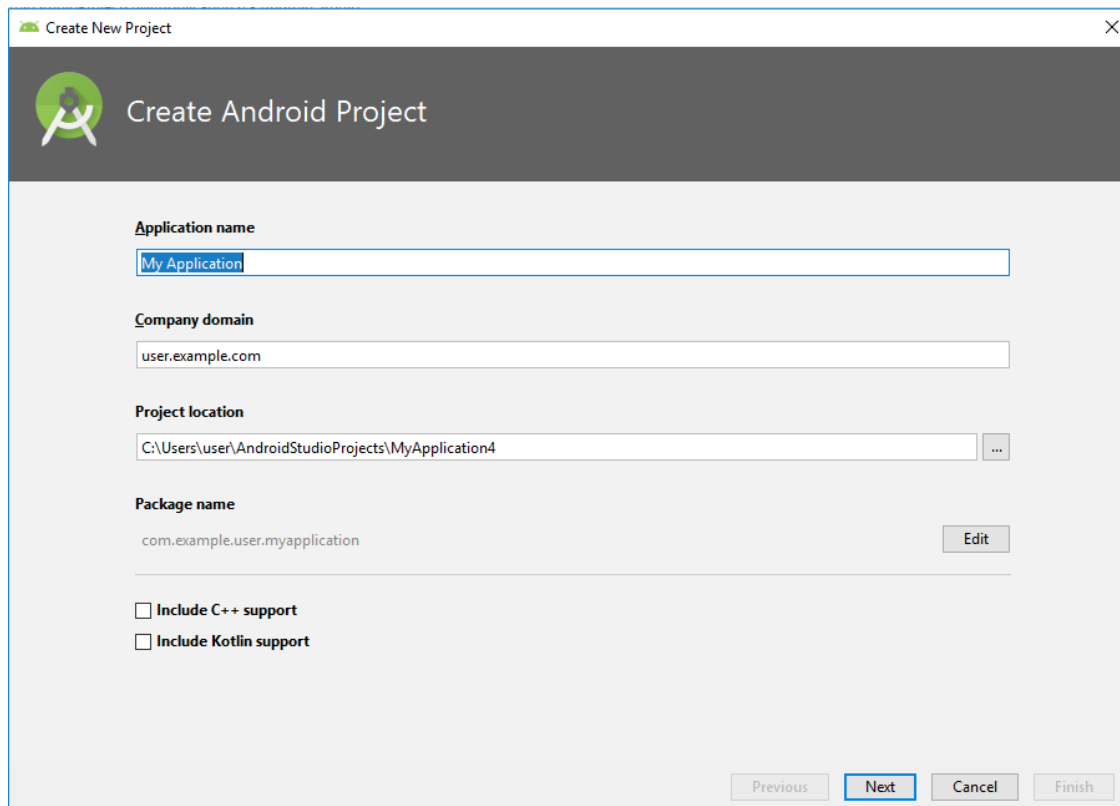
## Android Studio Installation

- ✓ Install JDK and JRE in your system
- ✓ Launch Android Studio .exe. If the JDK is not detected automatically you need to mention its path explicitly.
- ✓ Proceed further to include all the components
- ✓ Click Next. Until you complete the installation.

## Building your first app (Hello World)

Step1: Start a new Android Studio Project

Step2: Fill Application Name and Package Name and click next



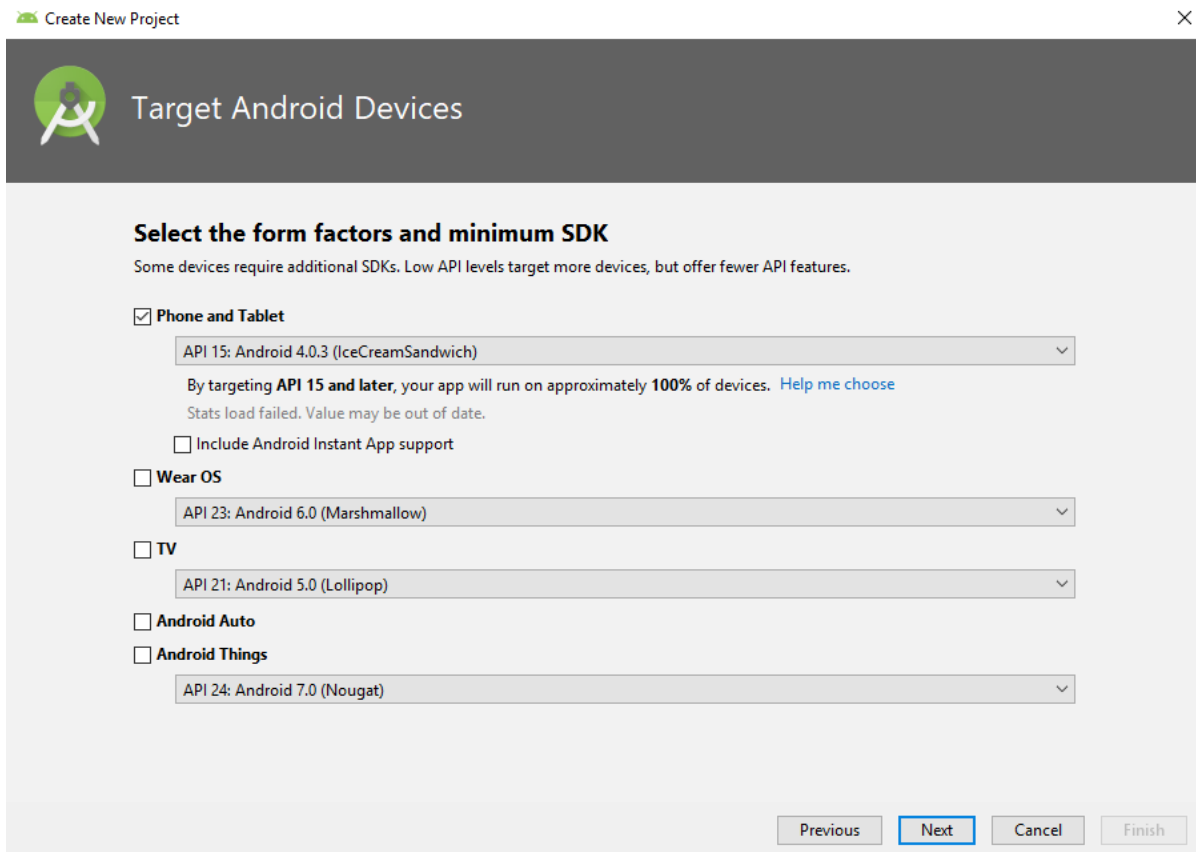
The screenshot shows the 'Create New Project' dialog in Android Studio. The dialog has a title bar 'Create New Project' with a close button. Below the title bar is a header area with the Android Studio logo and the text 'Create Android Project'. The main area contains several input fields and checkboxes:

- Application name:** A text field containing 'My Application'.
- Company domain:** A text field containing 'user.example.com'.
- Project location:** A text field containing 'C:\Users\user\AndroidStudioProjects\MyApplication4' with a browse button '...'.
- Package name:** A text field containing 'com.example.user.myapplication' with an 'Edit' button.
- Include C++ support:** A checkbox that is currently unchecked.
- Include Kotlin support:** A checkbox that is currently unchecked.

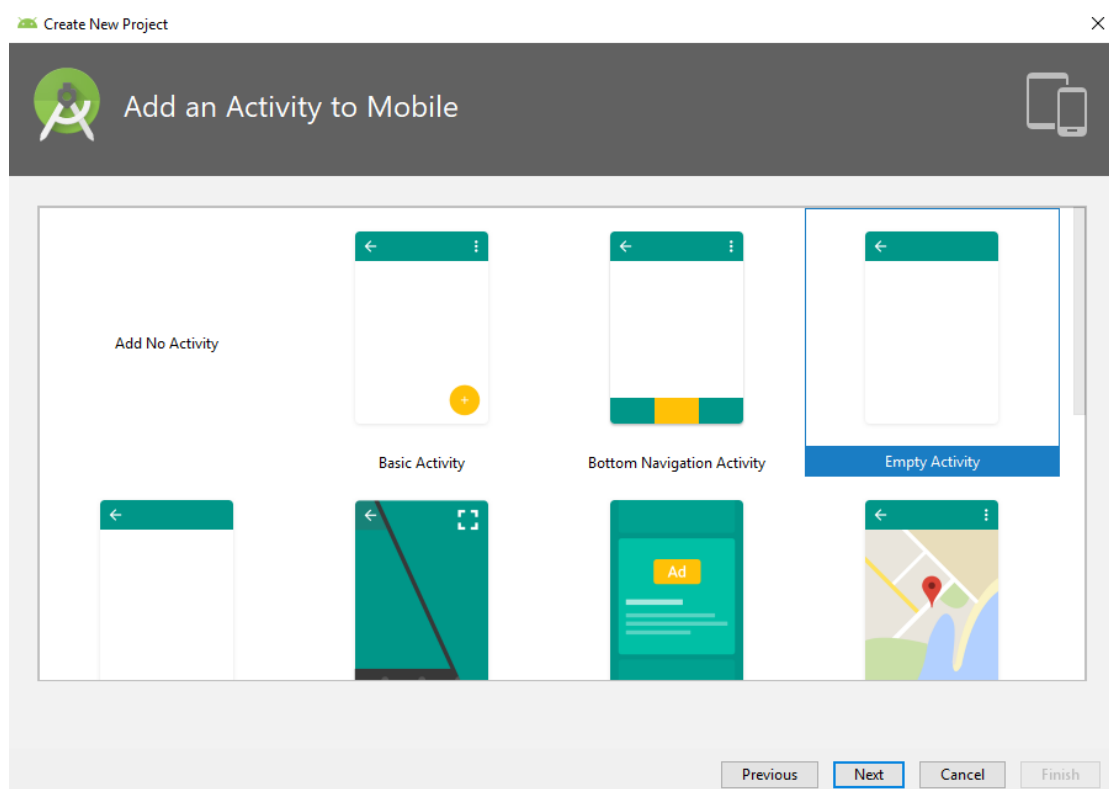
At the bottom of the dialog, there are four buttons: 'Previous', 'Next' (highlighted with a blue border), 'Cancel', and 'Finish'.

Step3: Select Phone and Tablet in Target Android Device and Proceed





Step4: Select Empty Activity and click Next.



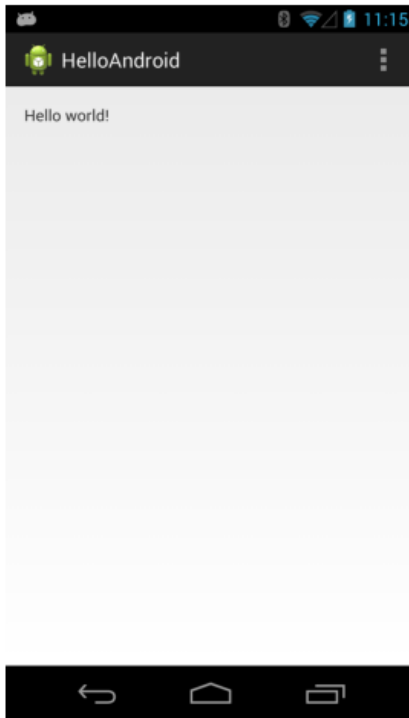
Step5 Click next and Finish.



## Running Application

In Android Studio, click the app module in the Project window and then select Run>Run 'app'(or Run Icon in the toolbar). In the Select Deployment Target window, select your device and click ok.

### Output



## Android Project Structure

When you start a new project, Android Studio creates the necessary structure for all your files and makes them visible in the project window on the left side of the IDE *click View>Tool Windows>Project*. By default, Android Studio displays our project files in the **Android view**.

### **Android App Module:**

When you create a new project, the default module name is “**app**”. A module is a collection of source files and build settings that allow you to divide your project into discrete units of functionality. Your Project can have one or more modules and one module may use another module as a dependency. Each module can be independently built, tested and debugged.

Module provides a container for your app’s source code, resource files, and app level settings such as the module-level build file and Android Manifest file. To add a new module to your project *click File>New>New Module*. In the create new module window, Android Studio offers different app modules such as Phone & Tablet, Android Wear, Android TV and Glass Modules. Each module provides essential files and some code templates that are appropriate for the corresponding app or device type.

### **Android Manifest.xml:**

Android Manifest.xml describes the fundamental characteristics of an app and each of its components. It works as an interface between your android OS and your application, so if you forget to declare your components in this file then it will not be considered by the OS.

The default AndroidManifest.xml file looks like this:

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

### Android Java

It contains the Java source code files separated by package names and JUnit test code. All the Activity classes are stored inside java folder. For e.g. The Activity class called MainActivity.java is stored in this directory under the package name you mentioned in the wizard.

### Android Resource (res)

It contains all non-code resources, such as xml layouts, UI strings and bitmap images, divided into corresponding sub-directories. Some of the examples of resource types are:

- **Animation Resources**- defines predetermined animations. Animations are saved in *res/anim/* and accessed from the *R.anim* class. Frame animations are saved in *res/drawable/* and accessed from the *R.drawable* class.
- **Color State List Resource**-it defines color resources that changes based on the view state. It is saved in *res/color/* and accessed from the *R.color* class
- **Drawable Resource**- defines various graphics with bitmaps or XML. It is saved in *res/drawable/* and accessed from the *R.drawable* class.
- **Layout Resource**- defines the layout for your application UI. It is saved in *res/layout/* and accessed from the *R.layout* class.
- **Menu Resource**- defines the contents of your application menus. It is saved in *res/menu/* and accessed from the *R.menu* class.
- **String Resource**- defines string, string arrays and plurals (and include string formatting and styling). It is saved in *res/values/* and accessed from the *R.string*, *R.array* and *R.plurals* classes.
- **Style Resource**- defines the look and format for UI elements. It is saved in *res/values/* and accessed from the *R.style* class.

### Android Strings

A string resource provides text strings for your application with optional text styling and formatting.

Types of resources that can provide your application with strings:

- **String:** XML resource that provides a single string.
- **String Array:** XML resource that provides an array of strings.

### Android Styles

A style resource defines the format and look for a UI. It can specify properties such as height, padding, font color, font size, background color etc. Styles in Android share a similar philosophy to cascading style sheets in web design i.e they allow you to separate the design from the content.