|  |
| --- |
| Android Tutorial |

|  |
| --- |
| 5-20-2018 |



# Introduction

Android is a Linux based operating system it is designed primarily for touch screen mobile devices such as smart phones and tablet computers. Android is developed in the Android Open Source Project (AOSP), led by Google. One of the most widely used mobile OS these days is android.

The Android operating system can be divided into the four areas as shown in the image below. Top two layers are used in android application development.

## Android OS Layers

Applications

**Home, Contacts, Phone, Browser, etc..**

Application Framework

**Activity Manager, Package Manager, Window Manager, Resource Manager, Content Providers, etc..**

Libraries

**SQLite, SSL, Media Framework, OpenGL, etc..**

Runtime

**Dalvik VM, Core Libraries**

Linux Kernel

**Display Driver, Camera Driver, Bluetooth Driver, Power Management, Wi-Fi Driver, Binder (IPC) Driver, etc..**

* Applications - Contains the applications, like the Browser, Camera, Gallery, Music and Phone
* Application framework - An API which allows high-level interactions with the Android system
* Libraries and runtime - The libraries for many common framework functions, like graphic rendering, data storage, web browsing. Also contains the Android runtime, as well as the core Java libraries for running Android applications.
* Linux kernel - Communication layer for the underlying hardware.

## Android Compilers

The compilers convert compiled **.class** files to executable **.dex** files in the **Dalvik** format for further execution in the Android environment. Following are the two important tools used:

### DEX

* **Dalvik** Virtual Machine is an Android Virtual Machine optimized for mobile devices.
* It optimizes the virtual machine for memory, battery life and performance.
* The **Dex** **compiler** converts the **.class files** into the **.dex file** that run on the Dalvik Virtual Machine.
* The Multiple .class files are converted into one dex file.
* Following flow shows compiling and packaging process from the source file-

**.java files**

**Source Code**

**.dex file**

**Packaging**

**(aapt)**

**Java Compiler**

**(Javac)**

**Dex Compiler**

**(dx)**

**.apk file**

**Resource + .dex**

**.class files**

**bytecode**

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

### Android ProGuard

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

## Android Versions

|  |  |  |  |
| --- | --- | --- | --- |
| Code name | Version | API level | Description |
| (no code name) | 1.1 | 2 | Android made its official public debut in 2008 with Android 1.0 , it didn't have a codename. The software did include a suite of early Google apps like Gmail, Maps, Calendar and YouTube, all of which were integrated into the OS |
| (no code name) | 1 | 1 |
| Cupcake | 1.5 | 3 | The early 2009's Cupcake release introduced numerous refinements to the Android interface, including the first on-screen keyboard, the framework for third-party app widget, and provided the platform's first-ever option for video recording. |
| Donut | 1.6 | 4 | Android 1.6, Donut, released in 2009, including the ability for the OS to operate on a variety of different screen sizes and resolutions, support for CDMA networks like Verizon |
| Eclair | 2.1 | 7 | Android 2.0 was released in October 2009. Then in January 2010, out came Android 2.1 with added animation features.  Google map navigation is its highlighted feature. Éclair include flash and digital zoom options for camera, live wallpapers, multi-touch support mechanism and, Bluetooth 2.1 support. |
| Eclair | 2.0 - 2.0.1 | 6-May |
| Froyo | 2.2.x | 8 | Short for Frozen Yoghurt, came 4 months after Android 2.1, Google got this version’s speed technically enhanced and a uniquely redesigned home screen. It ensured better functionality for the device, with the entire process streamlined. |
| Gingerbread | 2.3 - 2.3.7 | 10-Sep | Released in 2010, its unique features included several cameras, SIP internet calling, download manager, a few sensors like a barometer, gravimeter etc. |
| Honeycomb | 3.2.x | 13 | Google introduced Android 3.0 in February 2011. Made for tablets, versions 3.1 and 3.2 followed in rapid succession. Gingerbread was Android’s very first tablet-only update. |
| Honeycomb | 3.0 - 3.1 | 12-Nov |
| Ice Cream Sandwich | 4.0.1 - 4.0.4 | 14 - 15 | Released in 2011, Version 4.0 was the outcome of Google’s plan to get the tablet-only platform of Honeycomb synthesized with a mobile platform. Introduction of default font, and from this version onwards, Google effectively brought all its services under the umbrella, ‘Google Play’. |
| Jelly Bean | 4.1.x - 4.3.x | 16 - 18 | 2012 and 2013's Jelly Bean (Version 4.1) releases, “Google Now” was the most innovative aspect. Apart from the predictive feature, it had highly interactive notifications, unique in- built speech-to-text engine, popularly referred to as ‘voice typing’. |
| KitKat | 4.4 - 4.4.4 | 19 | Google officially unveiled 4.4, in 2013, debuted on Nexus 5, and can effectively run on quite many devices (with lower RAM) compared to earlier Android versions. 512 MB RAM was the recommended minimum. The introduction of Emoji on Google’s keyboards was yet another unique aspect of Kitkat. |
| Lollipop | 5.1 | 22 | Android 5.0 Lollipop was released in the fall of 2014, launched the Material Design standard, which became a core UI pattern. Came out with a brand new runtime. Battery saving feature ensuring excellent battery life, support for multiple users on phones. |
| Lollipop | 5 | 21 |
| Marshmallow | 6 | 23 | Android 6.0, released in 2015. Doze mode that cuts down the power consumption drastically when the device is idle, opt-in app permission, fully supported USB C, inbuilt fingerprint sensor support system, allowing the user to get a MicroSD card formatted and use it as internal storage, enjoying the same security level. |
| Nougat | 7.0 – 7.1.1 | 24 -25 | Released in 2016, provided Android with a native split-screen mode, a new bundled-by-app system for organizing notifications and a Data Saver feature. The launch of the Google Assistant — which came alongside the announcement of Google's first fully self-made phone, the Pixel contained Nougat |
| Oreo | 8 | 26 | 2017's Android 8.x Oreo released, including a native picture-in-picture mode, a notification snoozing option and notification channels that offer fine control over how apps can alert the user. |
| Android ‘P’ |  |  | Google released Android P to the general public as a beta in early May 2018 during Google I/O |

# Android Application Fundamentals

An Android application (app) is a single installable unit which can be started and used independently. Android apps can be written using Kotlin, Java, and C++ languages. An Android application consists of configuration files, Java source and resource files.

## Android SDK

Android development starts with the Android SDK (Software Development Kit). Many of the tools included in the SDK involve testing, debugging and packaging apps for Android. They provide a kind of bridge between Android Studio and a physical device or emulator so that the app can be appropriately packaged and then tested as its being developed. Android Studio will recommend necessary updates and it will call upon the required components when running or Building APK.

The Android SDK can be broken down into several components. These include:

* Platform-tools
* Build-tools
* SDK-tools
* The Android Debug Bridge (ADB)
* Android Emulator

The Android SDK tools compile the code along with any data and resource files into an **APK**, an **Android package**, which is an **archive file with an .apk suffix**. One APK file contains all the contents of an Android app and is the file that Android-powered devices use to install the app.

## Android Application

Each Android app lives in its own security sandbox, protected by the following Android security features:

* The Android operating system is a multi-user Linux system in which each app is a different user.
* By default, the system assigns each app a unique Linux user ID (the ID is used only by the system and is unknown to the app). The system sets permissions for all the files in an app so that only the user ID assigned to that app can access them.
* Each process has its own virtual machine (VM), so an app's code runs in isolation from other apps.
* By default, every app runs in its own Linux process. The Android system starts the process when any of the app's components need to be executed, and then shuts down the process when it's no longer needed or when the system must recover memory for other apps.

The Android system implements the *principle of least privilege*. That is, each app, by default, has access only to the components that it requires to do its work and no more. This creates a very secure environment in which an app cannot access parts of the system for which it is not given permission. However, there are ways for an app to share data with other apps and for an app to access system services:

* It's possible to arrange for two apps to share the same Linux user ID, in which case they are able to access each other's files. To conserve system resources, apps with the same user ID can also arrange to run in the same Linux process and share the same VM. The apps must also be signed with the same certificate.
* An app can request permission to access device data such as the user's contacts, SMS messages, the mountable storage (SD card), camera, and Bluetooth. The user has to explicitly grant these permissions. For more information, see Working with System Permissions.

## Android application components

| **Component** | **Description** |
| --- | --- |
| Application | An Android application can have one Application class which is instantiated before any other Android component. It is the last component which is stopped during application shutdown.  If not explicitly defined, Android creates a default application object for the application. |
| Activity | An activity is the entry point for interacting with the user. It represents a single screen with a user interface. An Android application can have several activities. Activities use views and fragments to create their user interface and to interact with the user.  An activity facilitates the following key interactions between system and app:   * Keeping track of what the user currently cares about (what is on screen) to ensure that the system keeps running the process that is hosting the activity. * Knowing that previously used processes contain things the user may return to (stopped activities), and thus more highly prioritize keeping those processes around. * Helping the app handle having its process killed so the user can return to activities with their previous state restored. * Providing a way for apps to implement user flows between each other, and for the system to coordinate these flows. (The most classic example here being share.) |
| Service | A *service* performs tasks without providing a user interface. They can communicate with other Android components and send notifications to the user. It is a component that runs in the background to perform long-running operations or to perform work for remote processes.  For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity. |
| Broadcast receiver (short: receiver) | A broadcast receiver is a component that enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements.  Because broadcast receivers are another well-defined entry into the app, the system can deliver broadcasts even to apps that aren't currently running. A receiver can be registered to listen to system messages and intents. A receiver gets notified by the Android system if the specified event occurs.  For example, you can register a receiver for the event that the Android system finished the boot process. Or you can register for the event that the state of the phone changes, e.g., someone is calling. |
| Content provider (short: provider) | A provider defines a structured interface to application data. A provider can be used for accessing data within one application, but can also be used to share data with other applications.  A content provider manages a shared set of app data that can be stored in the file system, in a SQLite database, on the web, or on any other persistent storage location that the app can access.  Android contains a SQLite database which is frequently used in conjunction with a content provider. The SQLite database would store the data, which would be accessed via the provider. |

## The manifest file

The components, settings and metadata of an Android application are described in the **AndroidManifest.xml** file. This file is known as the manifest file or the manifest. The manifest is read by the Android system during installation of the application. The Android system evaluates it and determines the capabilities of the application.

Activities, services and content provider components of the application must be statically declared in this file. Broadcast receiver can either defined here or registered dynamically at runtime.

The manifest does a number of things in addition to declaring the app's components, such as the following:

* Identifies any user permissions the app requires, such as Internet access or read-access to the user's contacts.
* Declares the minimum API Level required by the app, based on which APIs the app uses.
* Declares hardware and software features used or required by the app, such as a camera, bluetooth services, or a multitouch screen.
* Declares API libraries the app needs to be linked against (other than the Android framework APIs), such as the Google Maps library.

The primary task of the manifest is to inform the system about the app's components. For example, a manifest file can declare an activity as follows:

<?xml version="1.0" encoding="utf-8"?>  
<manifest ... >  
    <application android:icon="@drawable/app\_icon.png" ... >  
        <activity android:name="com.example.project.ExampleActivity"  
                  android:label="@string/example\_label" ... >  
        </activity>  
        ...  
    </application>  
</manifest>

# Android Application development

For this tutorial, a computer having 64-bit Windows 10 operating system will be used. The full configuration is shown below.



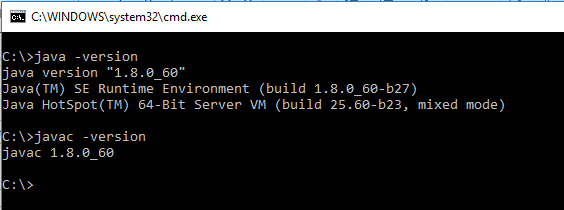
It is recommended to have at least 8GB RAM for Android Development, especially when Android Virtual Devices (AVD) are used to run the app.

In general, creating an Android app requires the SDK (Software Development Kit), an IDE (Integrated Development Environment) like Android Studio or Eclipse, the Java Software Development Kit (JDK) and a virtual device to test on. First, the development environment needs to be set up.

## Install Java

Since Java is used for Android development here, it should be downloaded and installed first, if not already available.

To check if Java is already available, open a command prompt and type java –version and javac –version commands, if it’s already available output should look like below,



If its not already available, Download Java 8 JDK (Java Development Kit) from:- <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

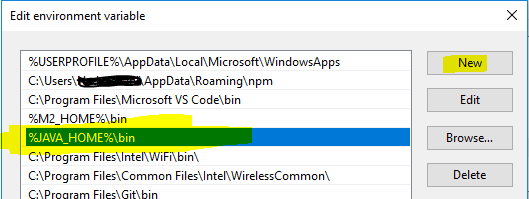
After the installation, set **JAVA\_HOME** and add entry to **path** variables in the environment as shown below,

Control Panel🡪System and Security🡪System🡪Advanced System Settings🡪Environment Variables

Add a new variable as “JAVA\_HOME” as shown below,



Edit Path Variable and add a new entry as %JAVA\_HOME%\bin, like shown below:-



## Android Studio

There are multiple ways to approach Android Development but by far the most official and powerful is to use Android Studio. This is the official IDE (Integrated Development Environment) for the Android platform, developed by Google and used to make the majority of the android apps that are used on a daily basis.

Android Studio was first announced at a Google I/O conference in 2013 and was released to the general public in 2014 after various beta versions. Prior to its release, Android development was handled predominantly through Eclipse IDE, which is a more generic Java IDE that also supports numerous other programming languages.

An IDE is what gives the main UI to enter the code, highlights mistakes, offers suggestions and lets user run and test apps conveniently. It creates the files required, provides basic layouts and generally it saves a lot of time and effort.

## Download and installation of Android Studio

Android Studio is designed specifically for Android development (unlike the second most popular option, Eclipse). This means, along with download the software,

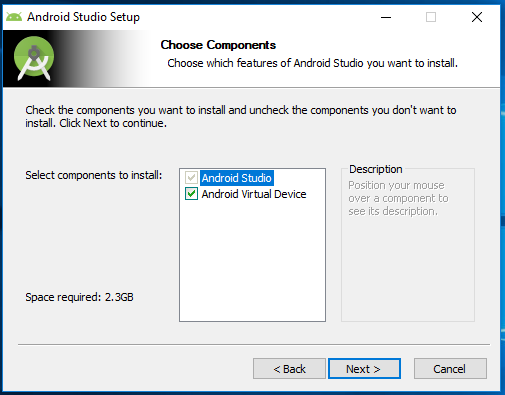
* The Android SDK (a selection of tools including the Android platform itself)
* The Android Virtual Device (AVD), which is an emulator to test apps

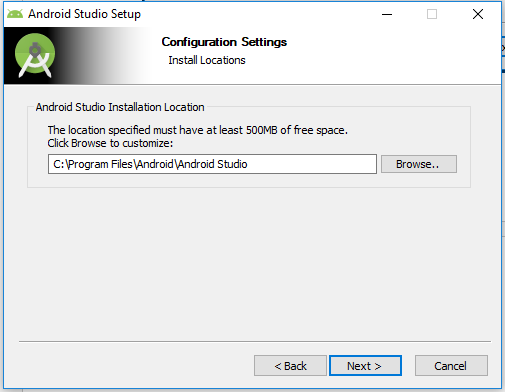
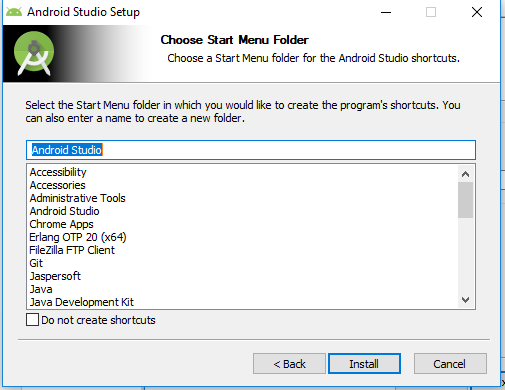
is received. When going through the installation, make sure leave those checkboxes boxes ticked to confirm that these additional components are installed.

Download Android Studio from: - <https://developer.android.com/studio/> , make sure to match the download with your OS and get 32bit or 64bit version accordingly.

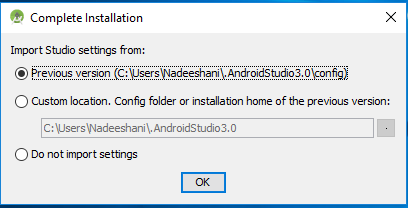
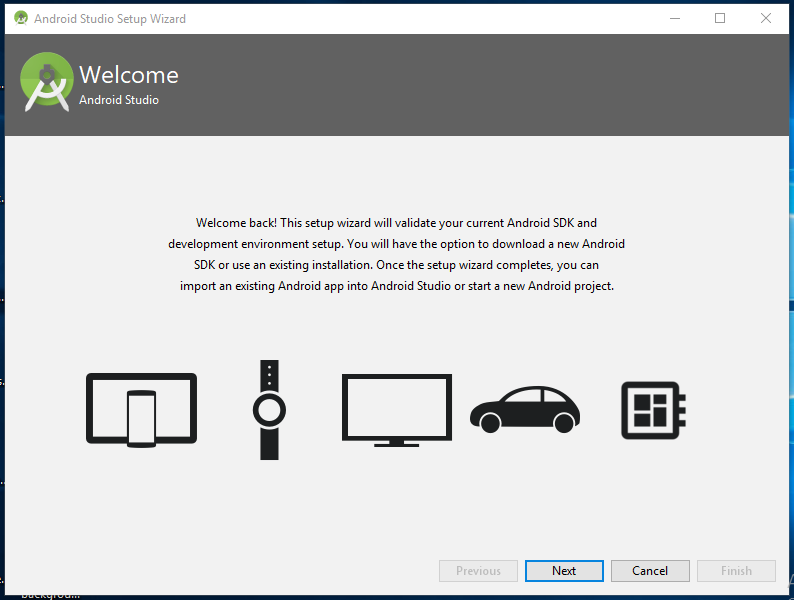
Installation for Windows is simple, just launch the .exe which was downloaded.

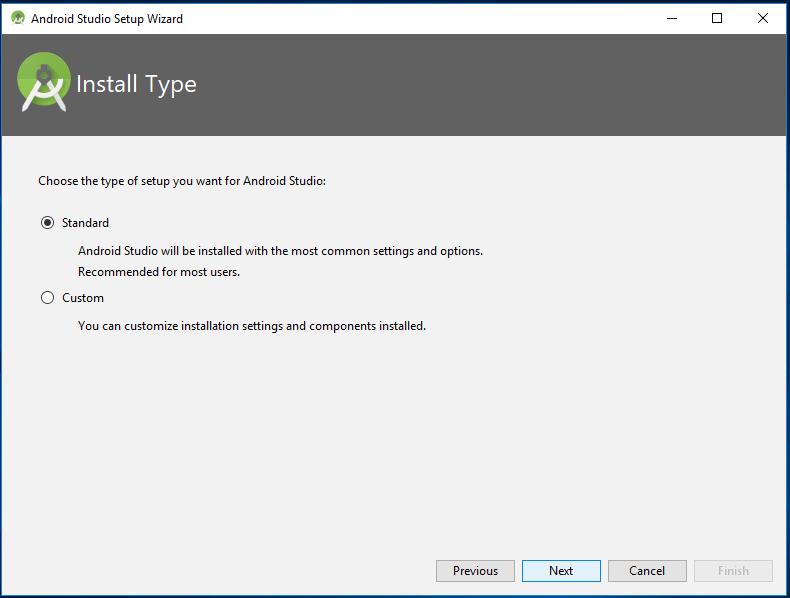
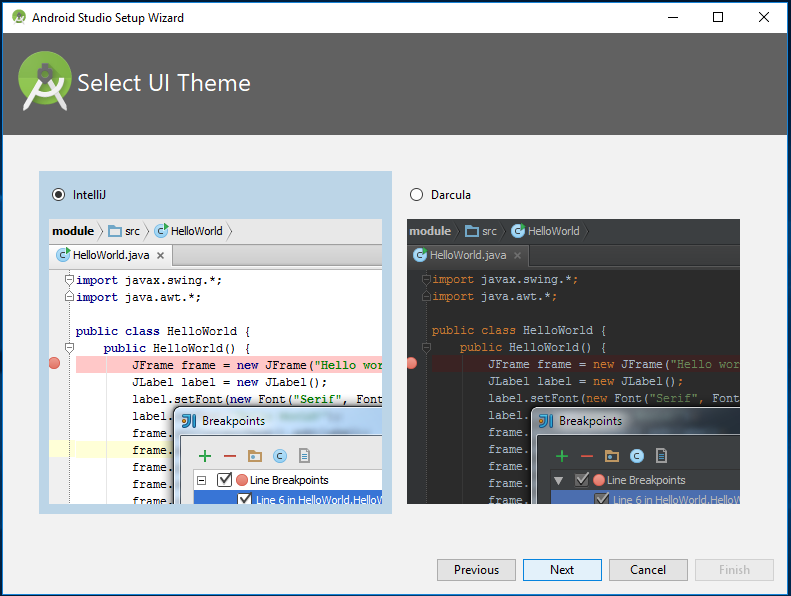
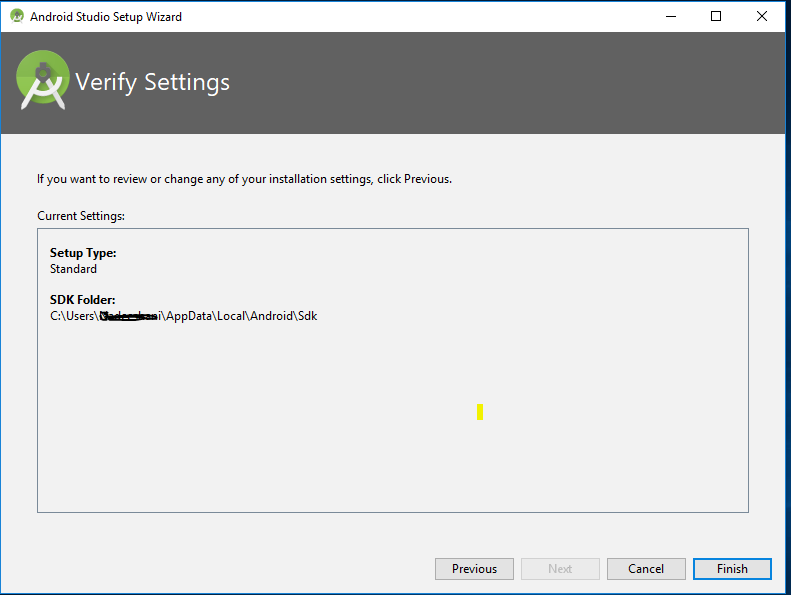
* Initial Screen  
  
* Screen2 - Make sure “Android Virtual Device” options is checked



* Screen 3 – Choose a location to install Android Studio  
  
* Screen 4 – Allows user to decide if a shortcut is created or not, and then finish setup by clicking on “Install” button  
  

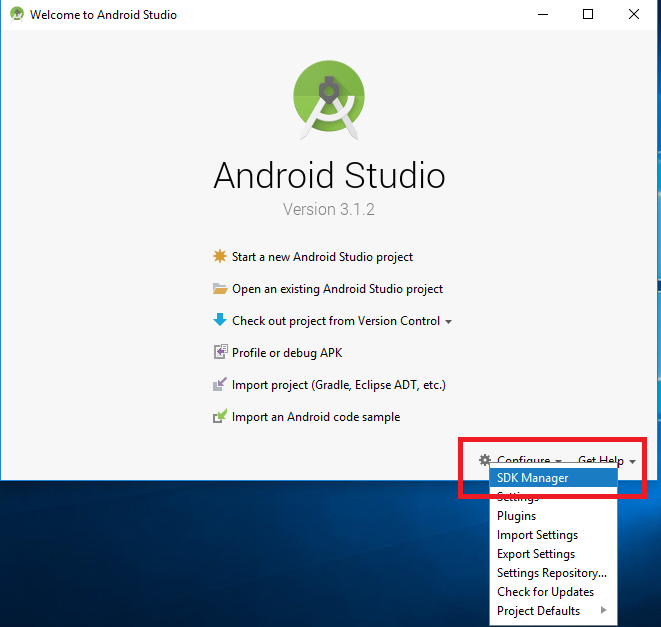
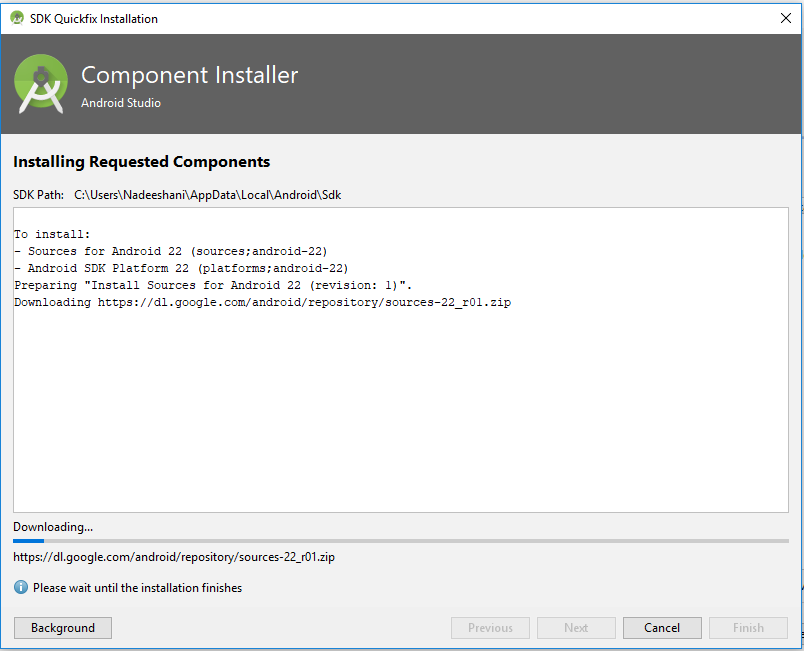
### Staring Android Studio for first time

* Screen 1 – Allows to import settings from a previous version if previously installed, for a fresh installation, just select “Do not import settings” option  
  
* Screen 2 :- In next step, the SDK set up will initiate, either to validate and select already existing SDK installed or to download and install a new SDK.  
  

* Screen 3:- Allows to select install type, Standard or Custom. Select the option “Standard” here, it will install all the required settings to get started with.  
  
* Screen 4 – Allows to select a UI theme for the IDE, choose according to the preference, this can be changed later.  
  
* Screen 5:- Confirmation screen – If everything is selected properly, proceed with SDK installation by clicking on finish  
  

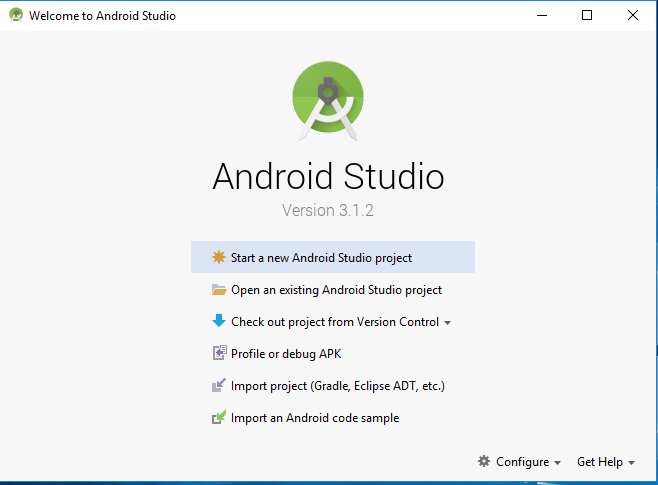
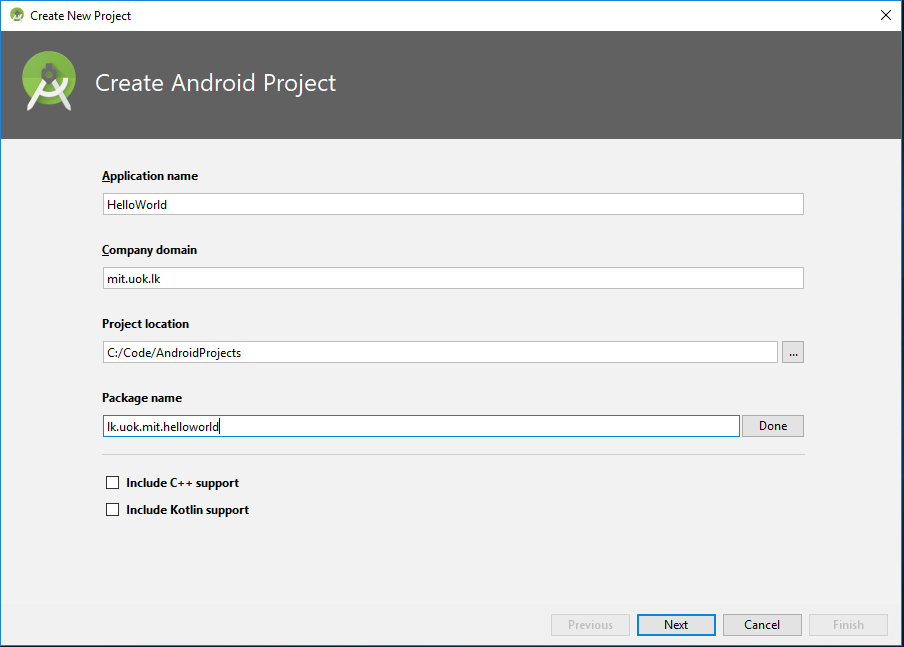
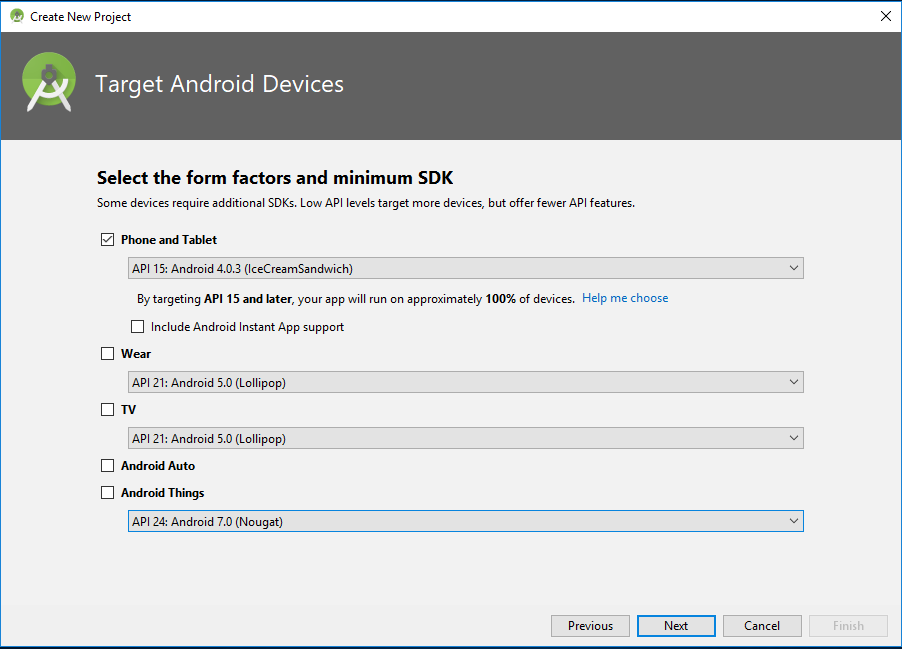
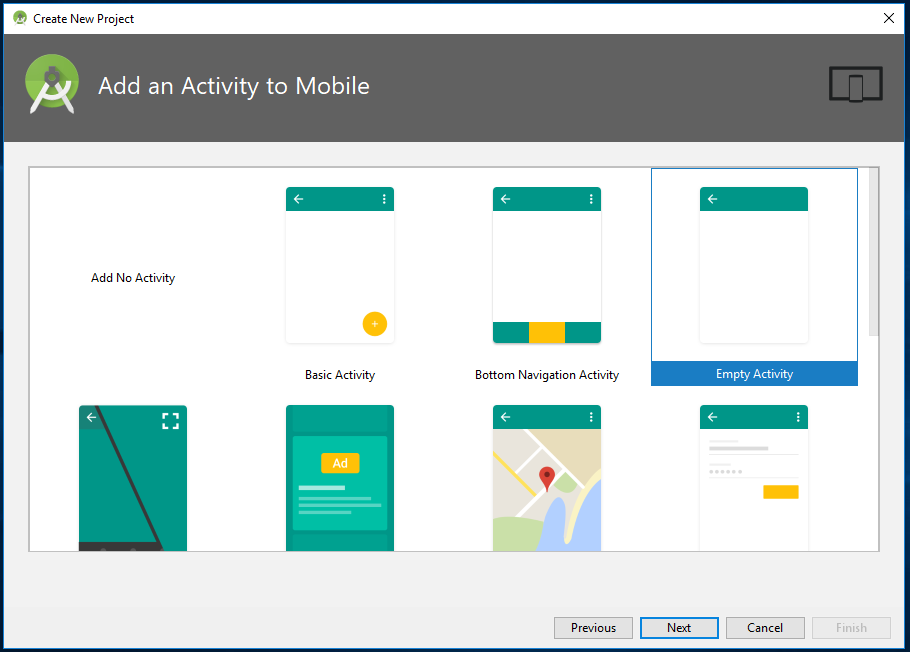
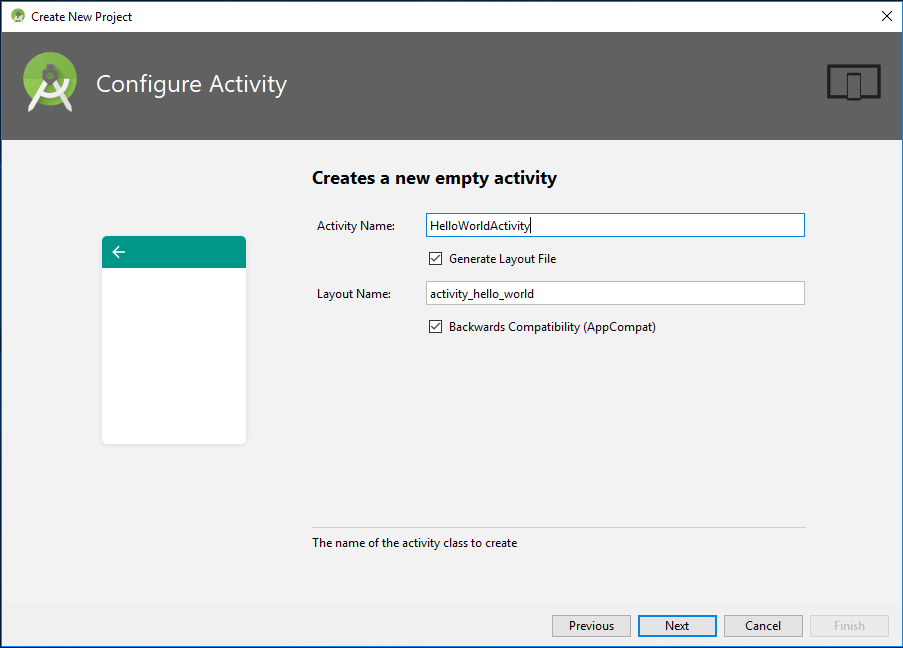
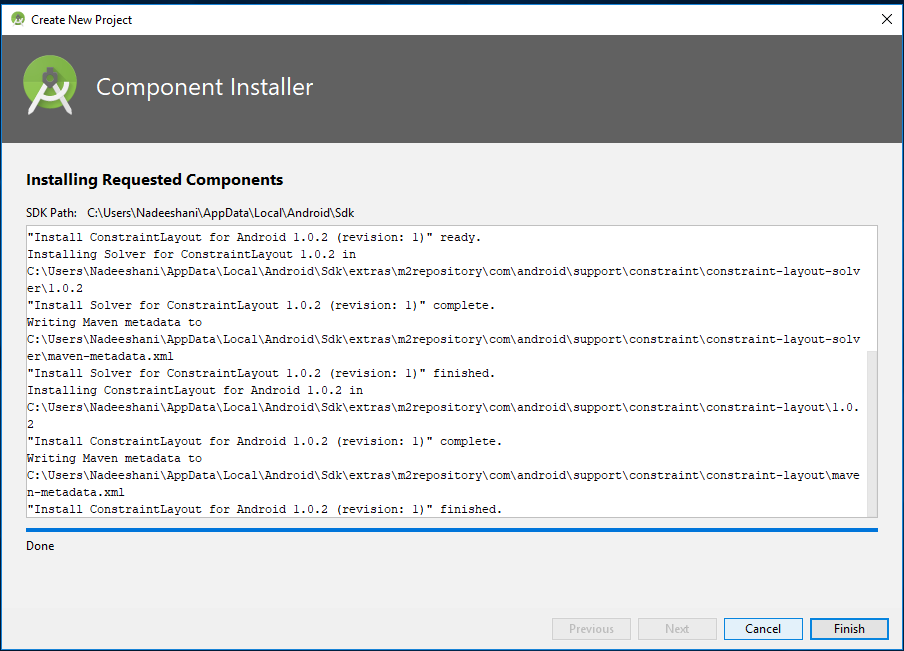
### Android Studio’s SDK Manager

To configure Android SDK through Android Studio, follow the steps given below,

* Screen 1 : - From the welcome screen of Android Studio, select “SDK Manager” option from “Configure” drop down as shown below,  
  
* Screen 2:- SDK Manager, Standard Development Kits are available for each release Android version, it’s sufficient to install the latest available stable version. At least one SDK platform should be installed. If nothing is installed, select the latest stable version “Oreo” as for current date and click on “Apply” button. Confirm the dialog when asked.  
  
* The installation progress could be viewed as below,  
  

## Starting a New Project

After Android studio is installed, and the SDK is properly set up as shown above, we can start to develop a new Android Application. The steps are shown below,

* Screen 1 : - From the welcome screen of Android Studio, select the option “Start a new Android Studio Project” from list as shown below  
  
* Screen 2 :- Fill the fields as shown in the image below, and click on “Next”  
  
  + Application name :- To enter the desired name of the application
  + Company domain :- Will be used to identify the project along with app name
  + Project Location :- The physical path in the machine where the source code is saved
  + Package Name :- name of the package where first activity class is created
* Screen 3:- Select Target Android Devices, from this screen user is allowed to select the minimum targeted SDK version for the app to run for different device groups.  
  **Selecting a lower SDK would allow the new application to run on older devices as well, covering vast percentage of devices, but latest API features introduces with newer SDK releases will not be available for the Application to use.** Therefore the minimum SDK should be selected carefully based on the purpose of the application.  
  Since we would be creating a simple HelloWorld application, its ok to use API 15 for Phone ans Tablets. Other devices are not required, hence they are not checked.  
  
* Screen 4 :- Allows to add an Activity (Screen) to our project, select “Empty Activity” option and click “Next”  
  
* Screen 5:- Allows us to configure the “Empty Activity” we added from screen above. Change Activity Name field in to “HelloWorldActivity”, notice the change in “Layout Name” field and the format of the name.  
  
  + Since a new Activity is created, its worth to note important points about an activity here
  + As stated in above sections, an Activity can be considered as a user interface to interact with the user
  + It consist of two items, a “**sample.java**” file as **Activity class** and a **sample.xml** file as the **interface design (the actual layout of the app)**
  + **XML** is a markup language that defines the layout of a document – much like HTML which is used for creating websites. It’s not really ‘programming’ but it is a kind of code.
  + The java class can access UI elements defined in **sample.xml** file through an auto generated class called **R.java**
  + Android **R.java** is an **auto-generated** file by aapt (Android Asset Packaging Tool) that **contains resource IDs for all the resources of res/ directory**.
  + If you create any component in the activity .xml file, id for the corresponding component is automatically created in this file. This id can be used in the activity source file to perform any action on the component.
  + The naming convention should be noted for the xml file and java file of the Activity, for java class, java naming conventions apply, such as camel case for class names, etc. When naming the xml file, separate each camel case word by an underscore and add prefix as “activity”  
    eg:- Java class – **HelloWorld.java** , and Layout xml – **activity\_hello\_world.xml**
* Screen 6 :- The project initiation completes with following screen, click on “Finish”  
  

### Gradle Build Tool

When working with new android projects, the term “Gradle” will come up. It has been used as the build tool for android projects. This is a ‘build automation tool’ which essentially helps Android Studio to turn all those different files into one single APK.



Gradle is an advanced general purpose build management system based on Groovy and Kotlin. Gradle supports the automatic download and configuration of dependencies or other libraries. It supports Maven and Ivy repositories for retrieving these dependencies. This allows reusing the artifacts of existing build systems. Gradle supports multi-project and multi-artifact builds. Gradle has the notion of projects and tasks.

In simple terms, when developing an application, various types of libraries are used, its same for android development. To manage these dependencies and to execute certain tasks Gradle is used.

***You should be able to leave Gradle to do its thing most of the time, but you will occasionally need to jump into the build.gradle files if you want to add a new ‘dependency’ allowing advanced features for your app.***

Sometimes, if things stop working, you can choose Build > Clean Project and this will essentially reaffirm where all the files are and what their roles are.

#### Projecs and tasks in Gradle

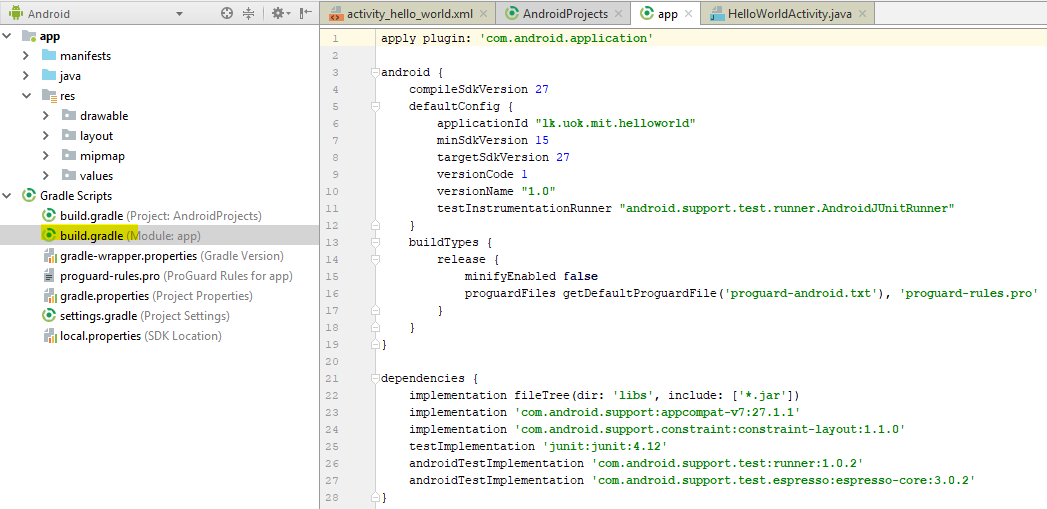
A Gradle build consists of one or more projects. Projects can be something which should be built or something that should be done. Each project consists of tasks. A task represents a piece of work which a build performs, e.g., compile the source code or generate the Javadoc.

#### The Gradle build file - build.gradle

A project using Gradle describes its build via a build.gradle file. This file is located in the root folder of the project. The build file for Gradle builds is based on a Domain Specific Language (DSL). In this file you can use a combination of declarative and imperative statements.

There are normally going to be **two of these Gradle build files** in an android project, one for the **whole project** and one for the **‘module’ (the app)**.

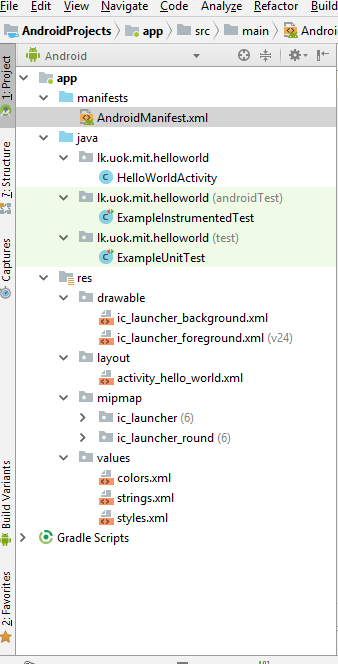
This build file defines a project and its tasks. Gradle is a general purpose build system hence this build file can perform any task. The following is a sample build file.



This content is auto generated by android studio and developers do not often have to be concerned about. But Its required to understand purpose of Gradle in android projects.

For the most part, Android Studio performs application builds in the background without any intervention from the developer, handled using the Gradle system.

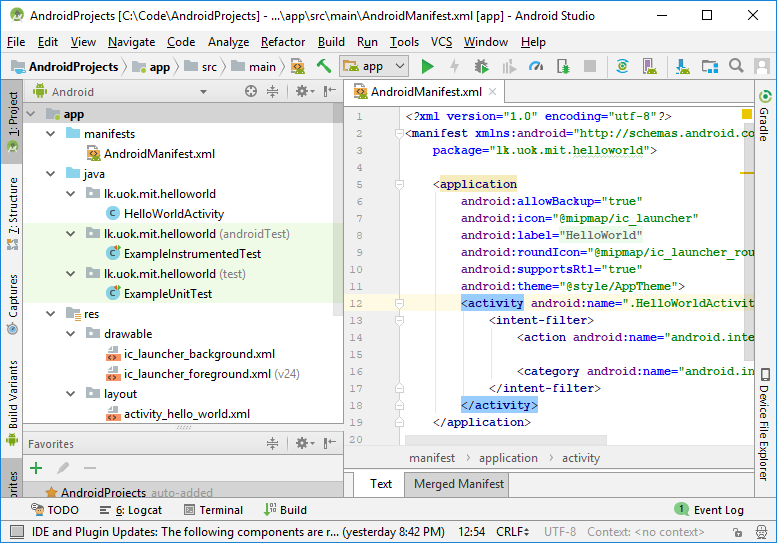
### Android Project Structure



* Android app actually consists of multiple files
* It’s Android Studio’s duty to keep these all in one place for the developer.
* The main window on the right of the screen allows to view individual scripts and files
* the tabs along the top here let you switch between what’s open at any given time.
* The file/folder structure of the newly created app could be seen in the “Project” view of the “Android Studio” as shown in the image here.
* Each project module appears as a folder at the top level of the project hierarchy and contains these three elements at the top level:
  + **manifests**/ : Manifest files for the module
  + **java**/ : Source files for the module.
  + **res**/ : Resource files for the module
* **res** folder is where all the external resources for the application such as **images, layout XML files**, **strings**, **animations**, audio files etc. are stored.
* Sub folders of **res**:
  + **drawable**:-contains the bitmap file to be used in the program.
  + **layout**: contains XML files that define the User Interface of the application
  + **menu**: XML files that define menus for the application goes into this folder
  + **mipmap**: The mipmap folders is used for placing the app icons only. Any other drawable assets should be placed in the relevant drawable folders as mentioned below.
  + **values** contains below files:
    - **colors.xml**. has default colors for the chosen theme, can add/remove custom own colors
    - **strings.xml**. has resources for all your strings, makes it easy to translate app
    - **styles.xml**. has styles for the app, giving it a consistent look for all UI elements.
* There are different folders to store drawables.
  + **drawable-ldpi**,
  + **drawable-mdpi**,
  + **drawable-hdpi**,
  + **drawable-xdpi** etc.
* The folders are to provide alternative image resources to specific screen configurations.
  + ldpi - low density screens
  + mdpi - medium density screens
  + hdpi - high density screens
* **android system will choose images according to the pixel density of the device**

### Android Studio User Interface

* The Android Studio main window is made up of several logical areas identified in figure below.



The **toolbar** lets you carry out a wide range of actions, including running your app and launching Android tools.

The **navigation bar** helps you navigate through your project and open files for editing. It provides a more compact view of the structure visible in the Project window.

The **editor window** is where you create and modify code. Depending on the current file type, the editor can change. For example, when viewing a layout file, the editor displays the Layout Editor.

The **tool window** **bar** runs around the outside of the IDE window and contains the buttons that allow you to expand or collapse individual tool windows.

The **tool windows** give you access to specific tasks like project management, search, version control, and more. You can expand them and collapse them.

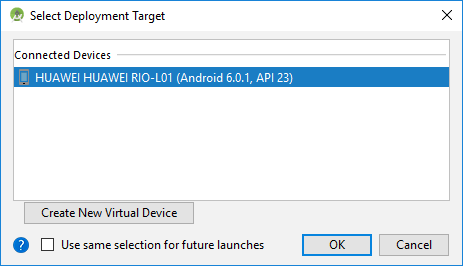
The **status bar** displays the status of your project and the IDE itself, as well as any warnings or messages.

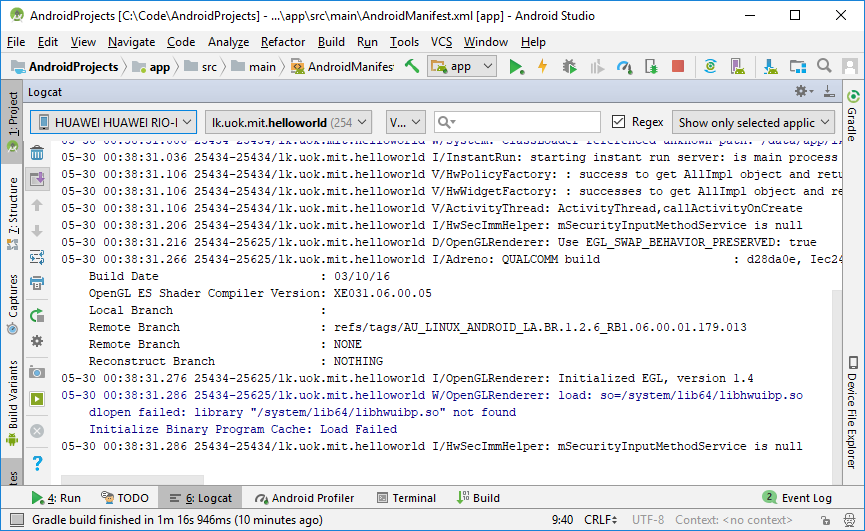
### Debugging (Running) Android Applications

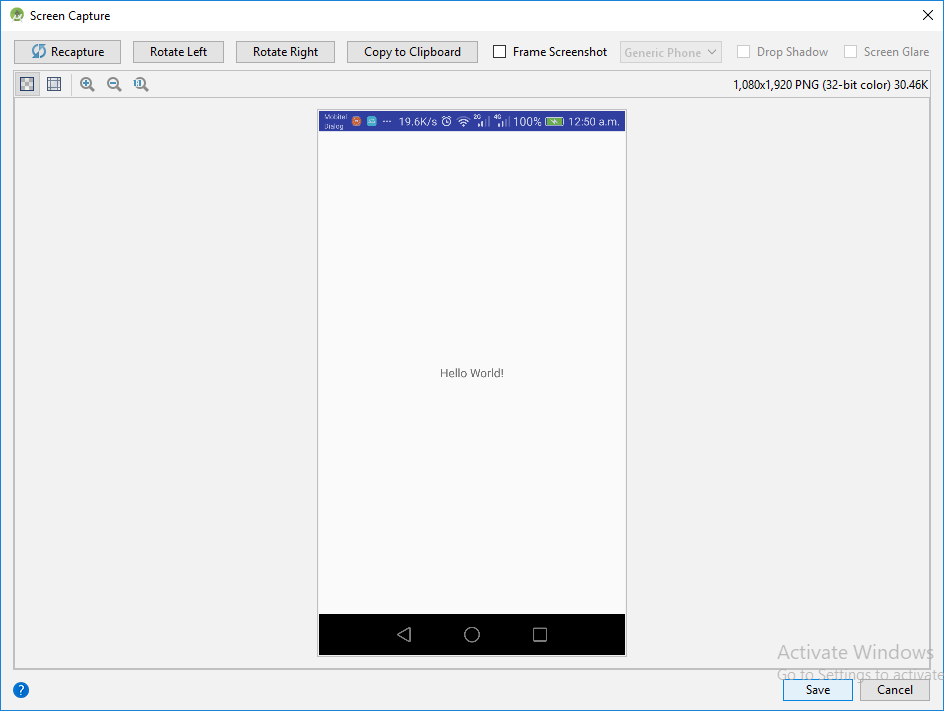
* Once the app is ready to be tested, there are two options available.
  + Run it on your physical device
  + Create a virtual device (emulator) to test it on.
* Running it on a physical device is simple and less resource consuming.
* The device should be containing an Android version greater than the minimum API level set in the app’s manifest file

#### Run it on your physical device

1. Plug the device in via USB to the computer
   1. **If the OS of the computer does not detect the device**
      1. For Windows: Install a USB driver for Android Debug Bridge (adb).
      2. For an installation guide and links to OEM drivers, see the Install OEM USB Drivers document at <https://developer.android.com/studio/run/oem-usb.html>
2. **Allow USB debugging in the device**
   1. On the device,
      1. open the Settings app,
      2. select Developer options and enable it first,
      3. and then enable USB debugging.
   2. If **Developer Options** is **not available** in the settings;
      1. **Go to the settings menu, and scroll down to "About phone." Tap it**
      2. Scroll down to the bottom again, where you see "**Build number**."
      3. Tap on “**Build number**” **repeatedly**.
         1. A message should appear on the screen that says “You are now ‘n’ steps away from being a developer.”
         2. Keep tapping until it says “**You are now a developer**.“
      4. Now go back to settings and “Developer Options” should be available now
3. **Allow installations from unknown sources in the device**
   1. On the device,
      1. open the Settings app,
      2. select Advanced Settings,
      3. and then select Security,
      4. Unknown Sources (Allow installation of non-official apps), and allow it
      5. This option has be set as allowed, otherwise device will allow only to install apps from Google Play
4. Hit the green **play** button at the top of the android studio, or ‘Run > Run App’.  
     
   
5. If the device is successfully connected and detected by Android Studio, when clicking on run it should display the device as shown below,



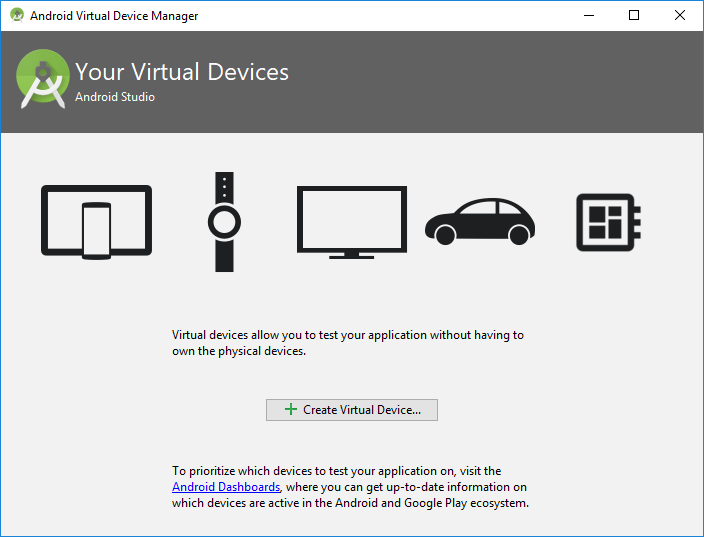
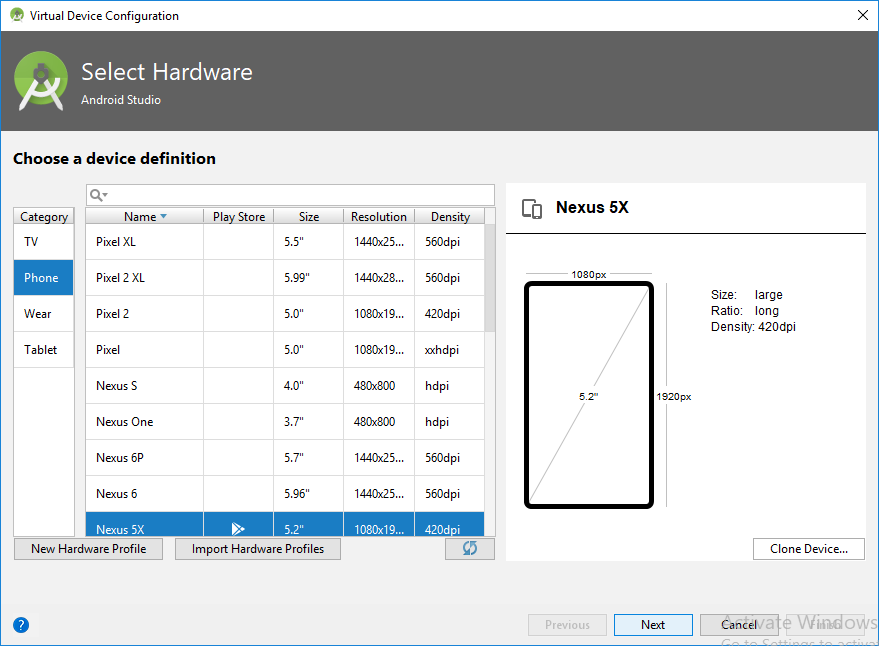
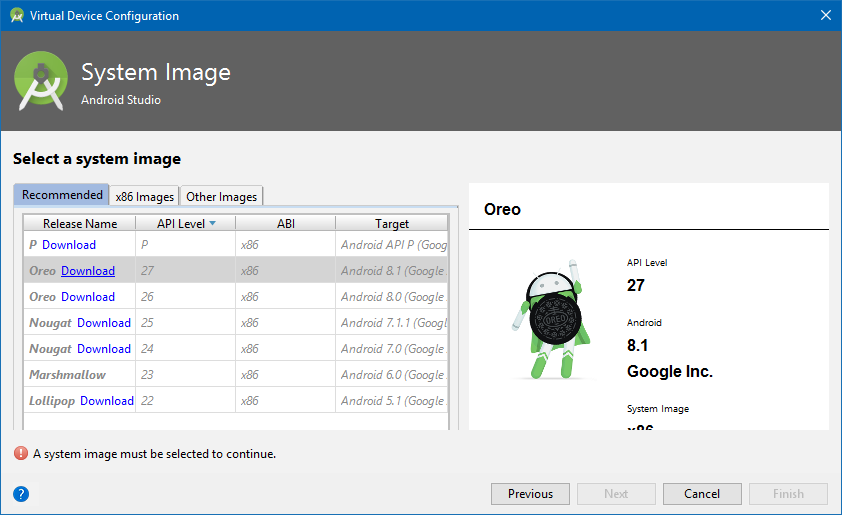
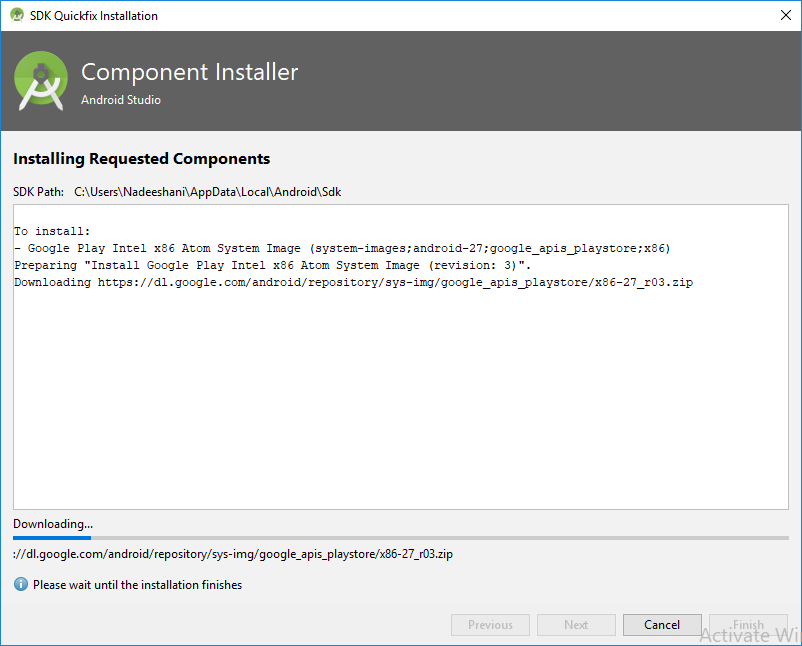
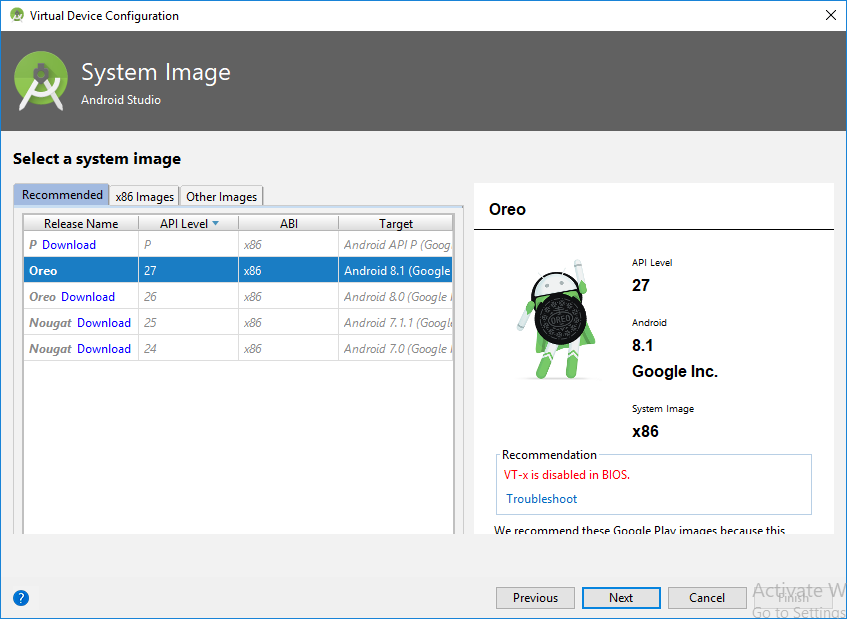
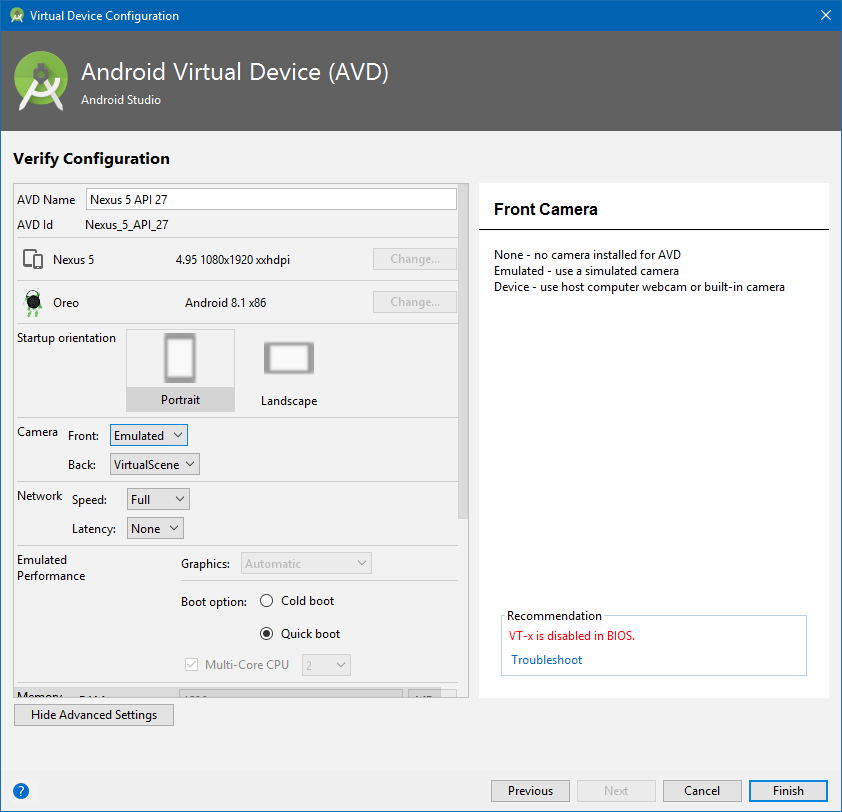
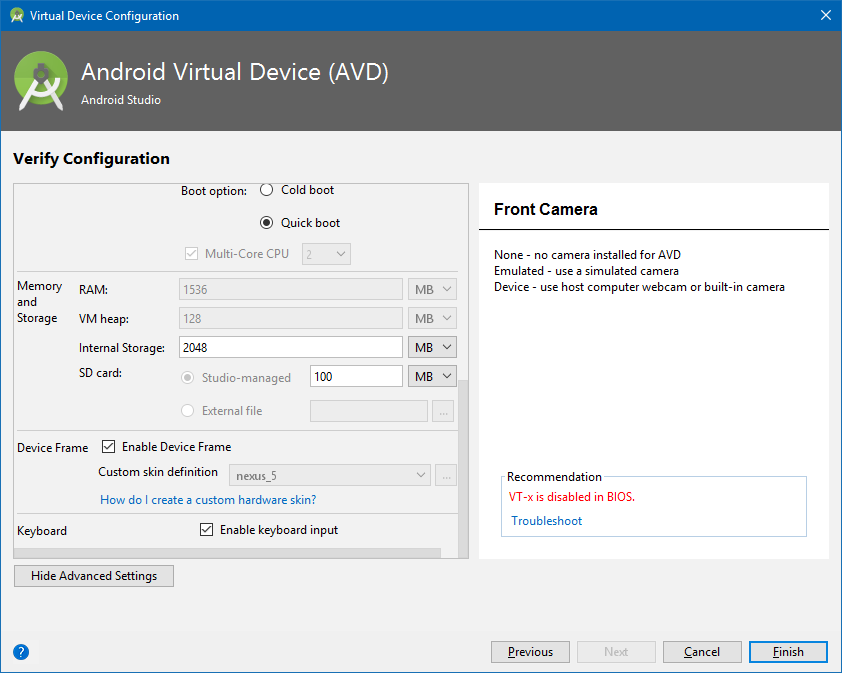
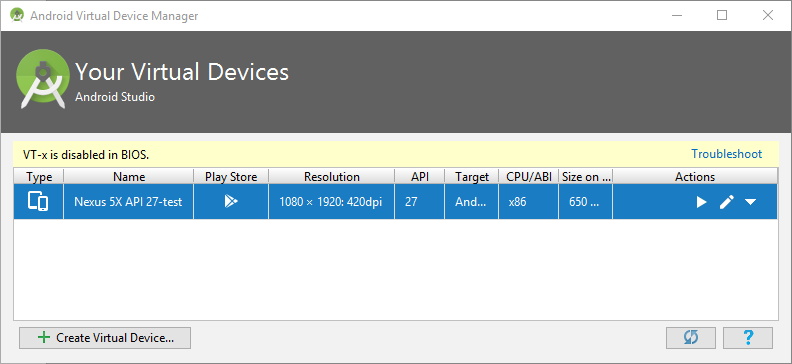
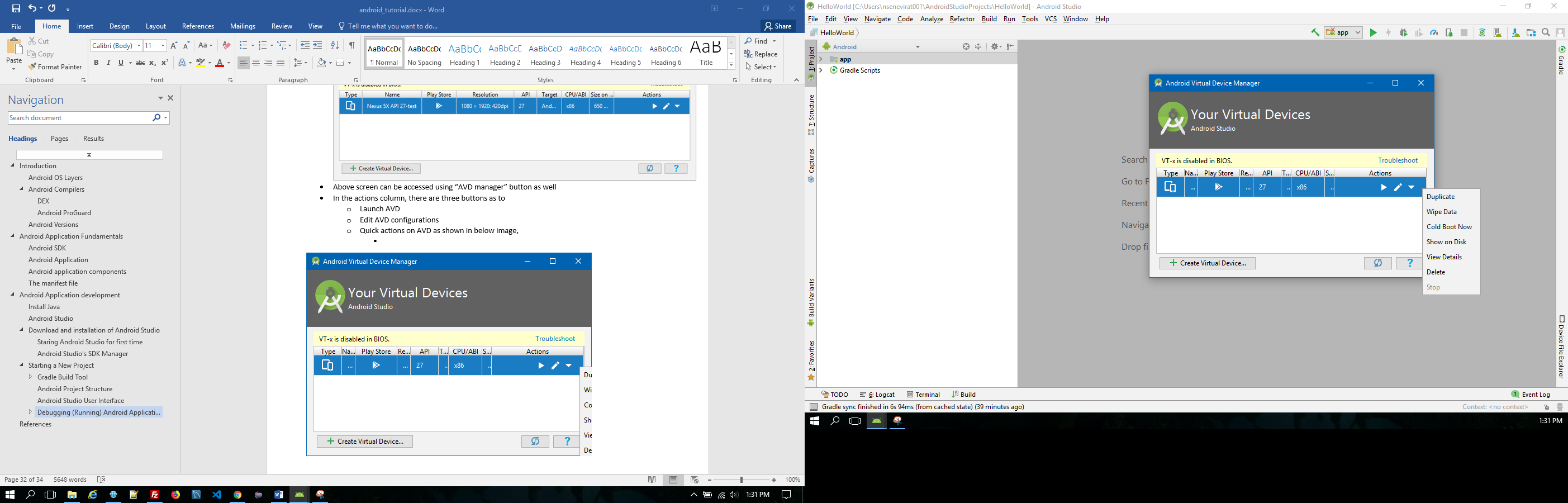
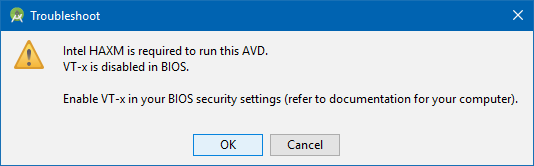
* While the app is running in the device, live reports of the app status could be seen through the ‘**logcat’** tab in the Android Monitor, found in the lower half of the screen as shown below.   
    
  
* Should something go wrong causing the app to crash or become unresponsive, then red text will appear and this will give a description of the problem.
* It essentially saves a lot of time versus blindly trying to guess what went wrong.
* User should make sure to filter the types of messages required to see here.
* Apart from logs, “**logcat”** window has options to allow developer to
  + **capture the screen shot from device**
  + **record device’s screen as a video**



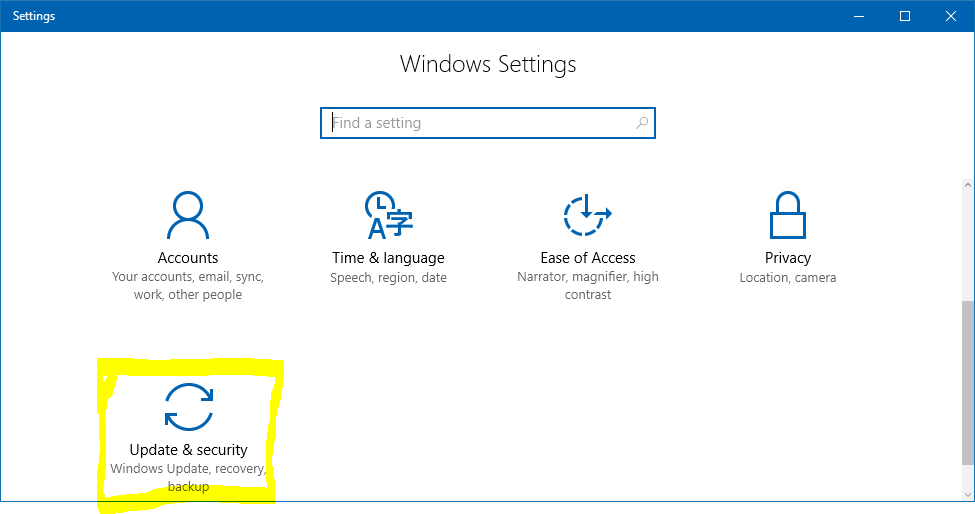
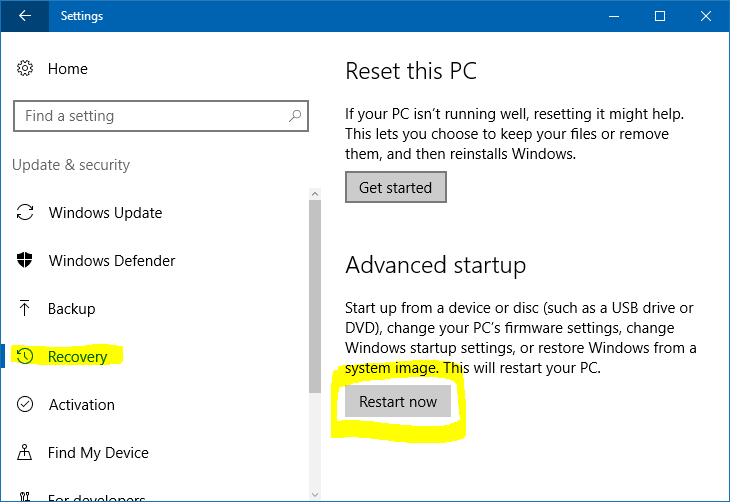
#### Create a virtual device (emulator) to test it on.

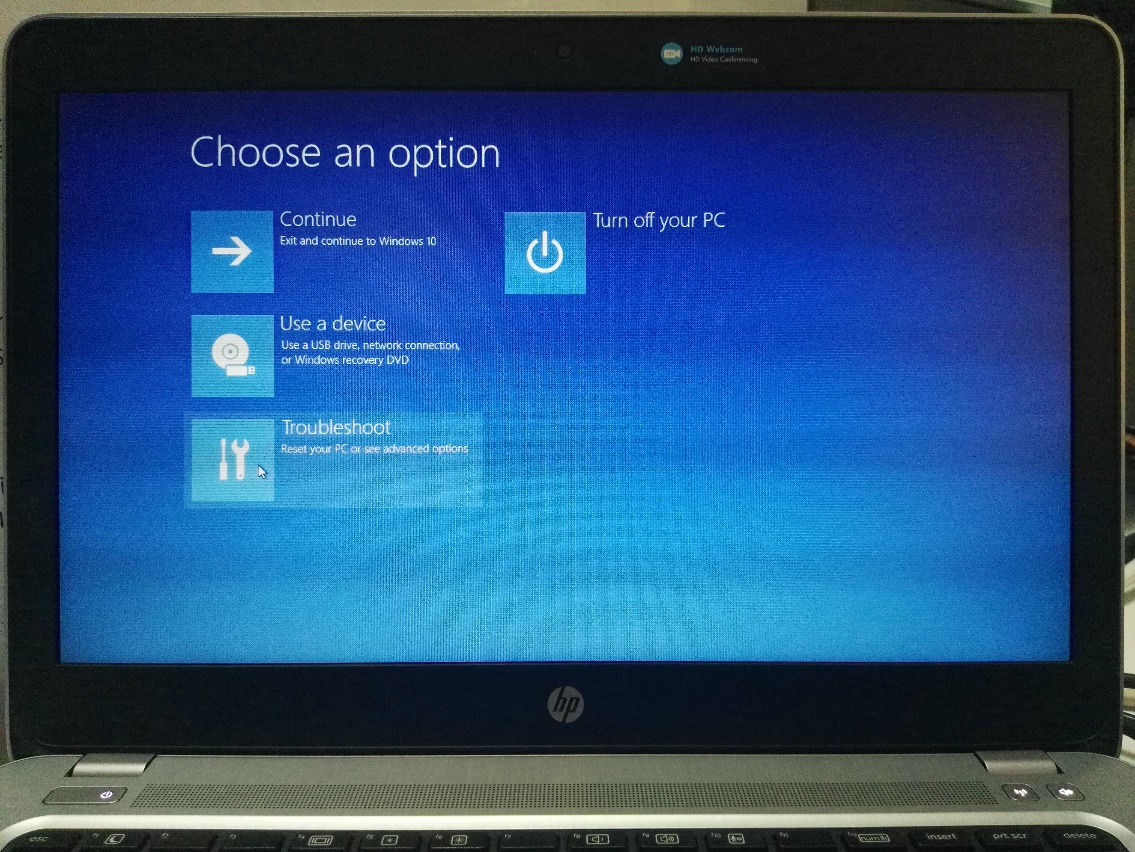
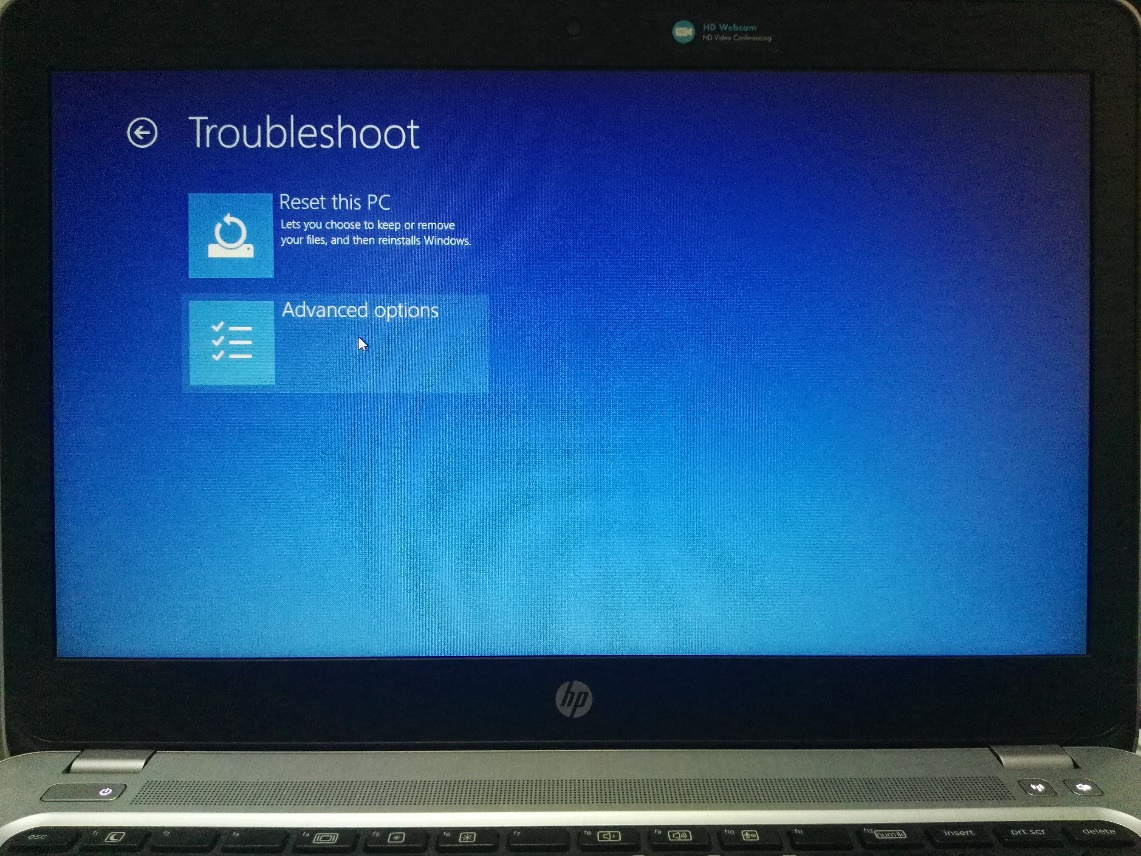
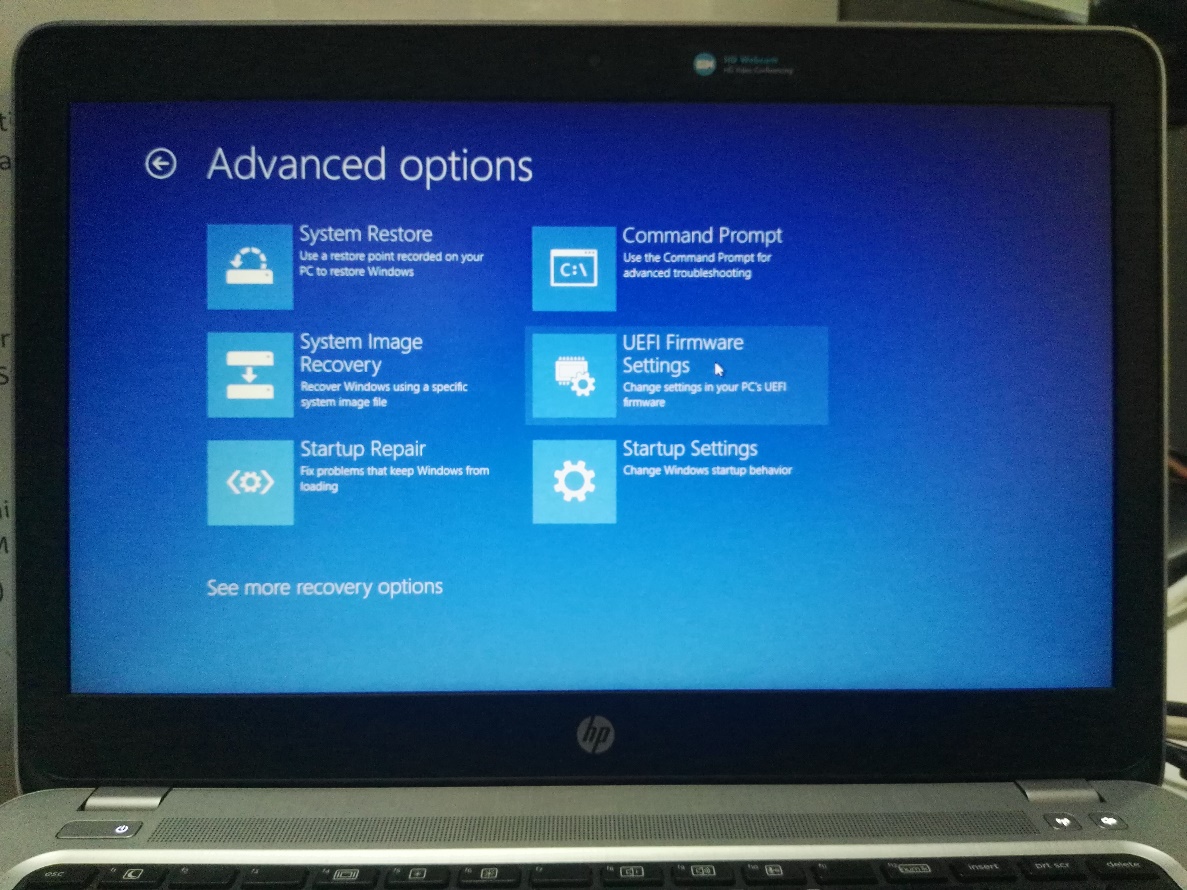
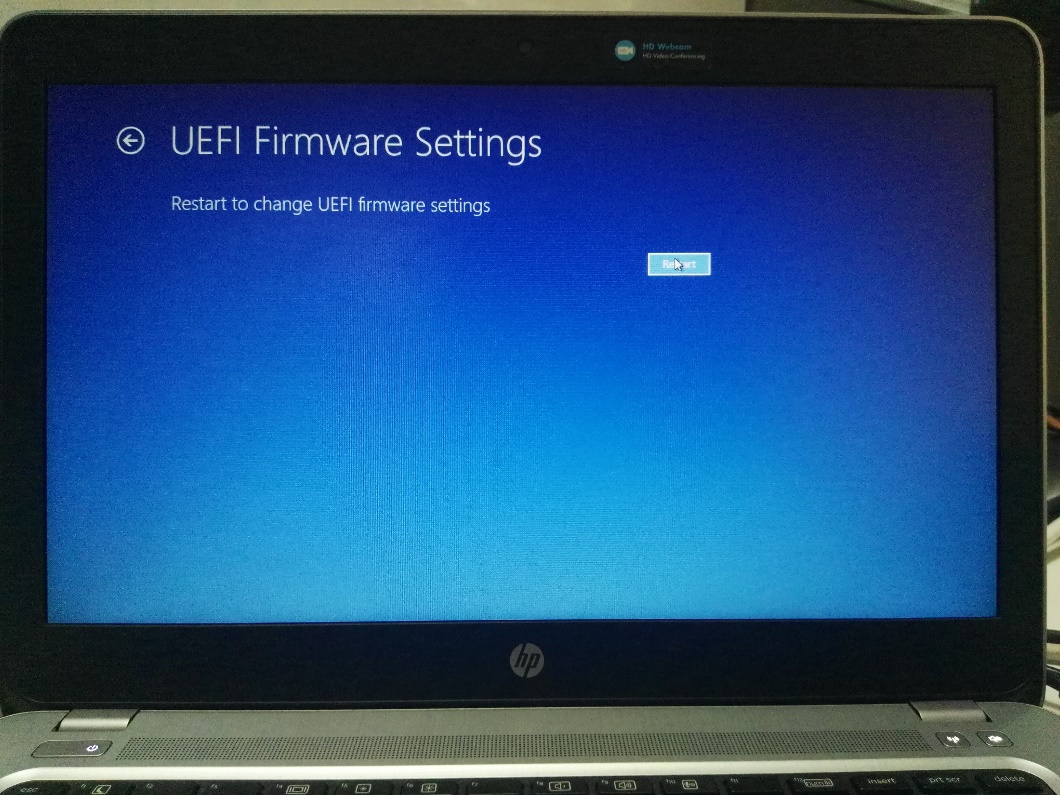
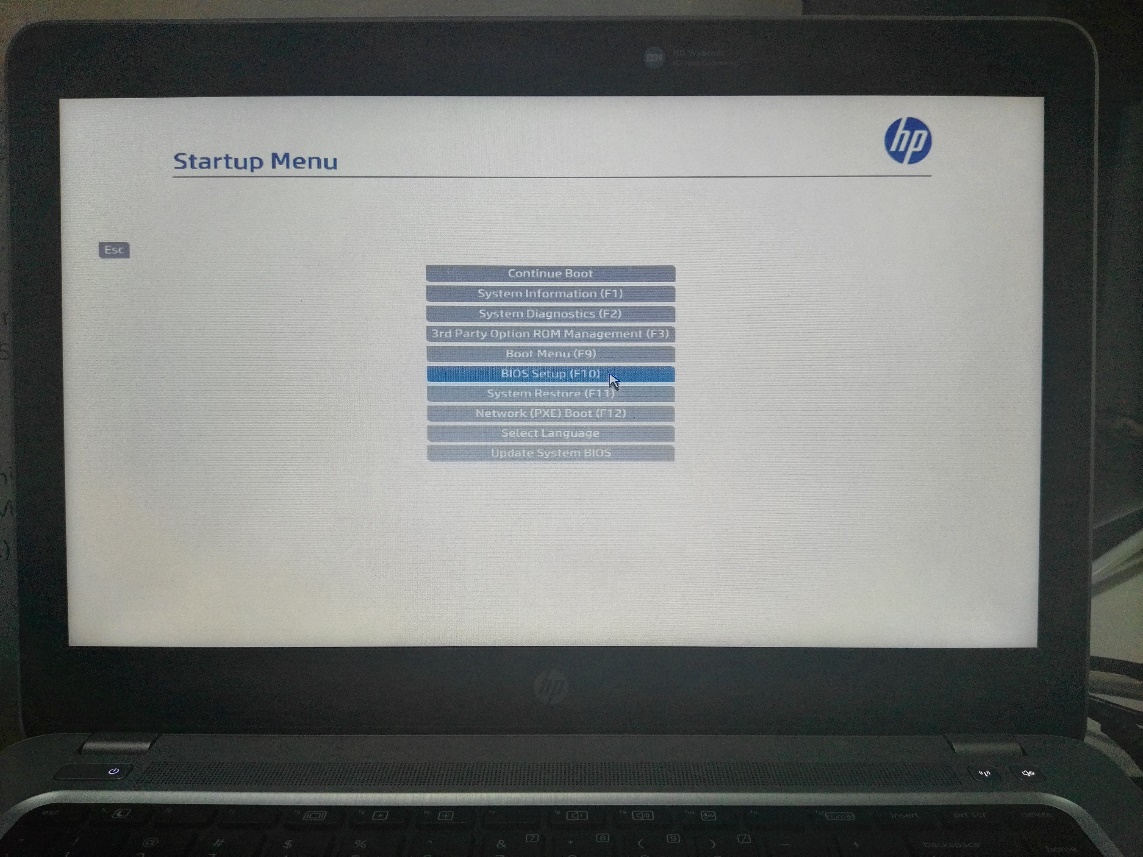
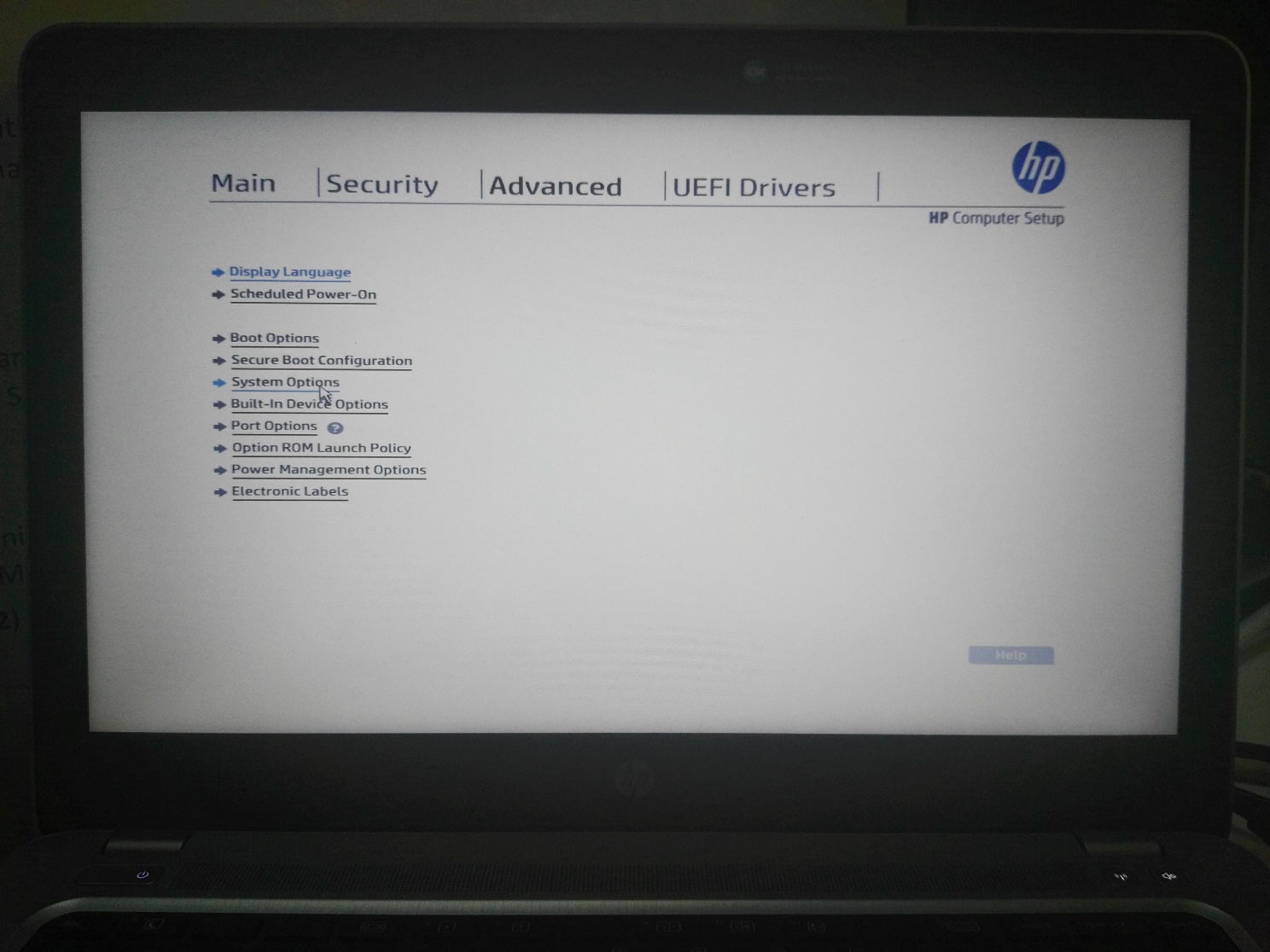
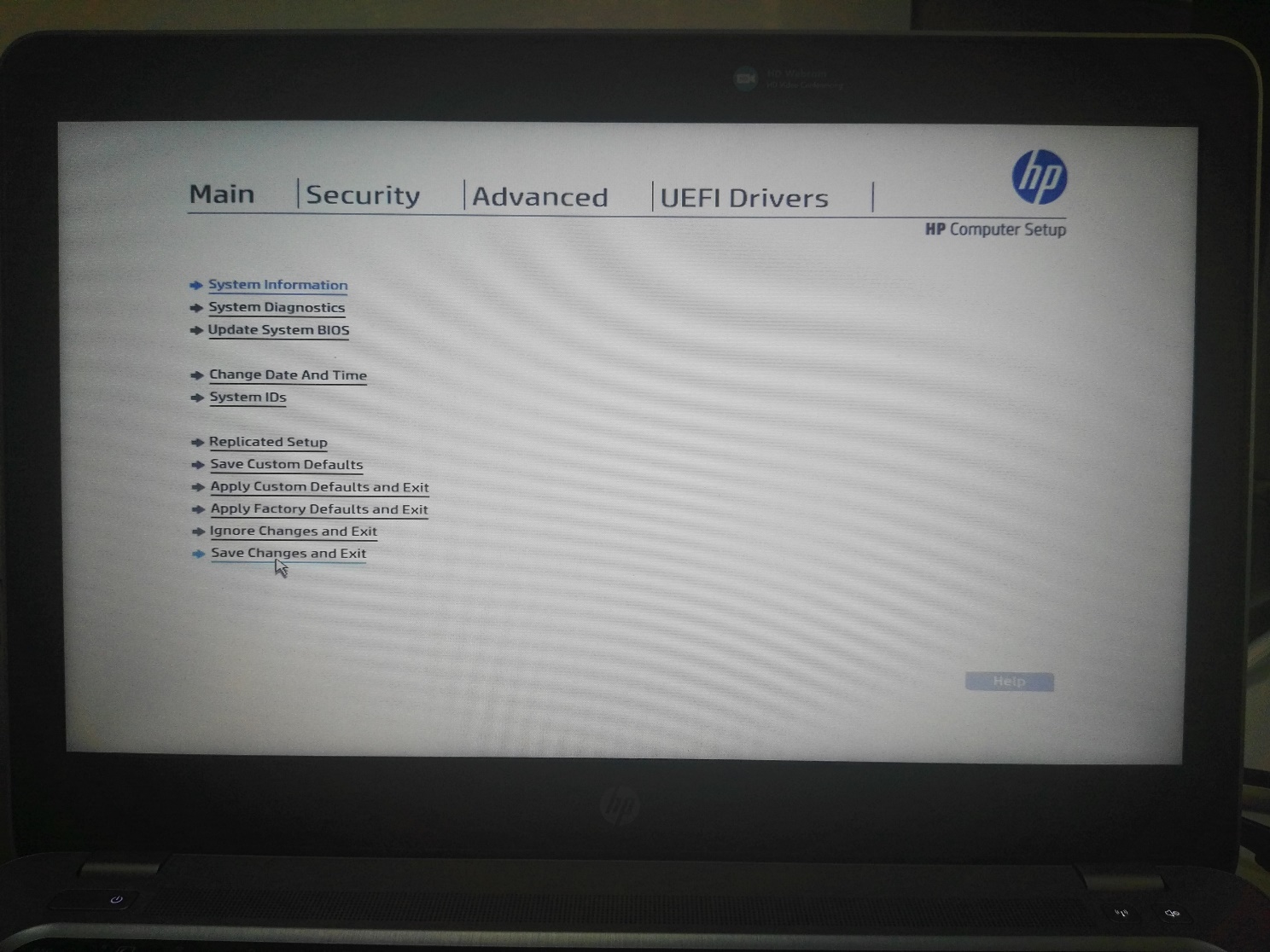
* Usually an android application developer will have at least one android device in possession to run the apps and test while developing
* However, one of the biggest challenges for Android developers is fragmentation,
* it’s not good enough that the app works on a single device, it also needs to work on
  + Devices with different screen sizes
  + Devices with different versions of android
  + Devices with different resource powers e.g.:- very underpowered.
  + And even different device types eg:- Phones, Tablets, Wearable, TVs, etc.
* This is where the ‘Android Virtual Device’ comes in.
* This is essentially an emulator that you can use to mimic the look and performance of any other Android device, setting such things as screen size, power and Android version.

##### AVD Manager (Android Virtual Device Manager)

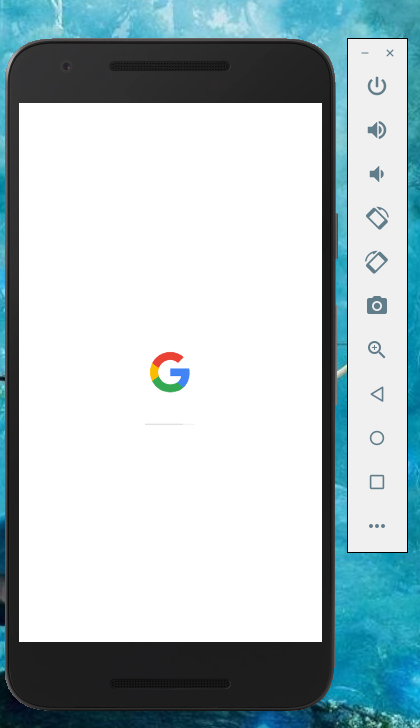
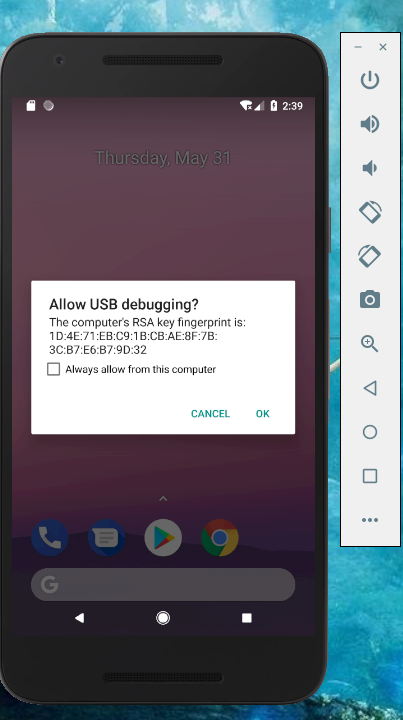
* To use a virtual device, first need to build one by downloading the required components and setting the specifications as required.
* To do this, navigate to Tools > Android > AVD Manager, or select “**AVD Manager**” from toolbar  
  
* The below window will appear, from there click on “Create Virtual Device”  
  
* Then choose the required hardware under required category from below window. It shows the selected device’s screen sizes and resolutions, here a Nexus 5X is selected;  
  
* As shown below, in the next step, choose the Android platform required to run from, this will be an android system image of the selected device based on API version selected.
* If the selected Android version to run hasn’t been downloaded yet, then the option will be presented next to it as to download.  
  
* After clicking on “Download” the system image will start to download and a progress window will appear like below.  
  
* Once an android system image of selected device is downloaded, the “Download” link will disappear next to OS name, and when selecting it, “Next” button will be enabled, Note the error message “**VT-x is disabled in BIOS**” under recommendation section.  
  
* When clicking on “Next”, the configuration screen will appear like below  
  
* In this screen a name for AVD could be given in “AVD Name” field, note that the same will be used as the ID, to uniquely identify the device.
* When naming the device, make sure to indicate the device name, and API level since multiple AVDs can be defined and it will be easier to identify the device later.
* Click on “Show Advanced Settings” to view all the settings for AVD
* The startup orientation could be selected as either “Portrait” or “Landscape”, this depends on the nature of the application which is developed.  
  Then there are options to emulate cameras, and there is an option to user the webcam as devices camera
  + None:- no camera installed for AVD
  + Emulated – use a simulated camera
  + Device – use host computer webcam or built-in camera
* Network Speed and latency – Initial state of the network speed and delay
* User can select the boot option either as
  + Cold Boot:- Starts emulator just as a device is powering on
  + Quick Boot:- Starts from the status which was saved when device last exited
* configuration screen – next part  
  
* There is an option to select memory and storage amounts
  + Internal Storage - allocate space as just as targeted device of the app, non – removable space on actual device
  + SD card - allocate space as just as targeted device of the app removable space on actual device
* Option to enable device frame :- if this is enabled, the actual device’s outer frame will appear in emulator, to mimic the actual device
* Option to enable keyboard input:- enable to interact with emulator using computer’s keyboard and on screen keyboard
* Once the configurations are done as required click on finish button, and it will bring the screen show below,  
  
* Above screen can be accessed using “AVD manager” button as well
* In the actions column, there are three buttons as to
  + Launch AVD - to start the AVD
  + Edit AVD configurations – to change the AVD configuration added upon creation
    - Quick actions on AVD as shown in below image,  
      
* Click on Launch button to launch AVD  
  
* Above error message occurred when trying to launch AVD, it’s the same error mentioned above.
* **Follow the steps given below to enable VT-x in computer’s BIOS, ONLY if the same error occurs for you!, otherwise IGNORE the next section under “Enable VT-x in computer’s BIOS”**

##### Enable VT-x in computer’s BIOS

* The steps given here will be different for each computer based on the brand of the computer (HP/Dell/Toshiba, etc.) and the OS version
* **Note:- below steps involves Restarting the OS, please save all the work before going forward**
* **Please note** that below steps are for a **HP ProBook** having **windows 10**
  1. In windows OS, go to “Start Menu” -- > “Settings” and get window shown below and then go to “Update & Security” option  
     
  2. Now in below screen go to “Recovery” option and in that click on “Restart now”  
     

* 1. Once system restarts, below screen appears, select the option “**Troubleshoot**” from it  
     
  2. From troubleshoot screen, select the option – “**Advanced Option**”  
     
  3. From Advanced Option Screen, select the option, “**UEFI Firmware Settings**”  
     
  4. In UEFI Firmware Settings, click on “**Restart**” button  
     
  5. After restarting, system goes to BIOS screen, from that either press “F10” key or click on “BIOS Setup” option  
     
  6. In BIOS Setup, go to “**Advanced**” section and then go to “**System Options**”  
     
  7. Under “Systems Options” enable “Virtualization Technology (VTx)” as shown in below image; and then click on “Back Arrow” to go back to “Main” section  
     
  8. From the “Main tab ” select the option “Save Changes and Exit”  
     
  9. From the confirm message shown, click on “Save”, and then system will restart.

##### Android Virtual device;

* After launching the AVD it will look like below,  
  While booting up:-  
  
* After Booting process completes,  
  
* As shown in above screen shot, device prompts to allow USB debugging, click OK on that.

##### Running the app on Virtual Device;

# References

*Application Fundamentals*. (2018, May 2018). Retrieved from Android Developers: https://developer.android.com

MULLIS, A. (2017, November 11). *Android Studio tutorial for beginners*. Retrieved from Android Authority: https://www.androidauthority.com/android-studio-tutorial-beginners-637572/

SINICKI, A. (2017, December 16). *Android SDK tutorial for beginners*. Retrieved from Android Authority: https://www.androidauthority.com/android-sdk-tutorial-beginners-634376/

Vogel, L. (2016, June 20). *Getting started with Android development - Tutorial*. Retrieved from vogella: http://www.vogella.com/tutorials/Android/article.html#high-level-overview-of-android-development