



---

# **Android Programming**

**Module Number: 01**

**Module Name: Introduction to Android and Android Studio**

## Aim

**To equip the students with the basics of Android and Architecture**

- Understand Android - Introduction, History, Features, Android Architecture, Versions/Evolution, Dalvik VM, Android Studio Environment, Android Application development, Application Folder Structure.

## Objectives

The Objectives of this module are:

- Explain Android - Introduction, History, Features, Android Architecture.
- Explain the Versions/Evolution, Dalvik VM.
- Installing Android Studio.
- Explain Android Studio Environment, First Android Application, Application Folder Structure.
- Explain about Manifest file, R.java file, Activity, Activity life cycle, Application Components, Resource Files.

## Outcome

At the end of this module, you are expected to:

- Classify the Android Architecture.
- Classify the Versions/Evolution, Dalvik VM.
- Basic Android Application, Application Folder Structure.
- Classify the Manifest file, R.java file, Activity, Activity life cycle.
- Application Components, Resource Files.

## Content

1. Introduction, History, Features
2. Android Architecture
3. Versions/Evolution, Dalvik VM
4. Installing Android Studio
5. Android Studio Environment
6. First Android Application
7. Application Folder Structure
8. Manifest file, R.java file
9. Activity, Activity life cycle
10. Application Components, Resource Files.

- 5

# Introduction to Android and Android Studio

## History

- Android, Inc. was founded in Palo Alto, California in October 2003 by Andy Rubin and his team. He is known as the “Father of Android”.
- The early intentions of the company were to develop an advanced operating system for digital cameras.
- Later, the idea got diverted to develop an Operating System for mobiles which could override Symbian and Windows phones.
- On July 2005, Google acquired Android Inc and helped the team in their development. The team proved themselves by launching its first product named **Android**, a mobile device platform built on the Linux kernel on **November 5, 2007**.



## (Continued) History

- The first Android running smartphone HTC Dream was released on October 22, 2008.
- Since then, Android has been releasing numerous updates which have incrementally improved the operating system, adding new features and fixing bugs in previous releases.
- First two versions of the Android are non-commercial versions, no devices have been launched using those two versions, and they are named Alpha and Beta.
- These Alpha and Beta are just internal release for developers.



## Android Features

An android device comes with numerous features. For example; making a phone call, sending SMS, and a lot more options when we use android along with the internet.

The following are the main features of Android OS:

- Application Framework
- Dalvik Virtual
- SQLite
- Media Support
- Optimised Graphics

## (Continued) Android Features

- **Application Framework** - Android Application Framework is defined as a collection of classes, libraries etc., which are used in developing or solving an issue in a program. Android Framework is a bind of Android SDK and various API libraries which are used to develop new applications using JAVA language on Android Platforms. Android Framework is efficient and faster than other Frameworks.
- **Dalvik Virtual Machine (DVM)** - Fastest interpreter to run dex files. DVM which has the same functionality as Java Virtual Machine (JVM), but works only on Mobile Platform. It consumes less memory and provides faster performance.

*For example,* in Android, the codes written is saved as a file with an extension of .dex called as DEX file. Jack (a compilation tool) is used to compile it into byte code (Machine readable code) and interpreted using DVM (Dalvik Virtual Machine).

## (Continued) Android Features

- **SQLite** - Android comes with built-in SQLite database implementation. SQLite, a open source SQL database which stores data to a text file on a device.
- **Media Support** - Android has inbuilt classes that can manage various audio and video formats like Mp3, MIDI etc. Unique Media Api's are used to access these media files. These API's are used to play Music, ringtones, videos etc.
- **Optimised Graphics** - Android has inbuilt classes that support optimised graphics VGA, 2D and 3D. Android equips custom 2D graphics library and 3D graphics based on the OpenGL ES 1.0 specification.

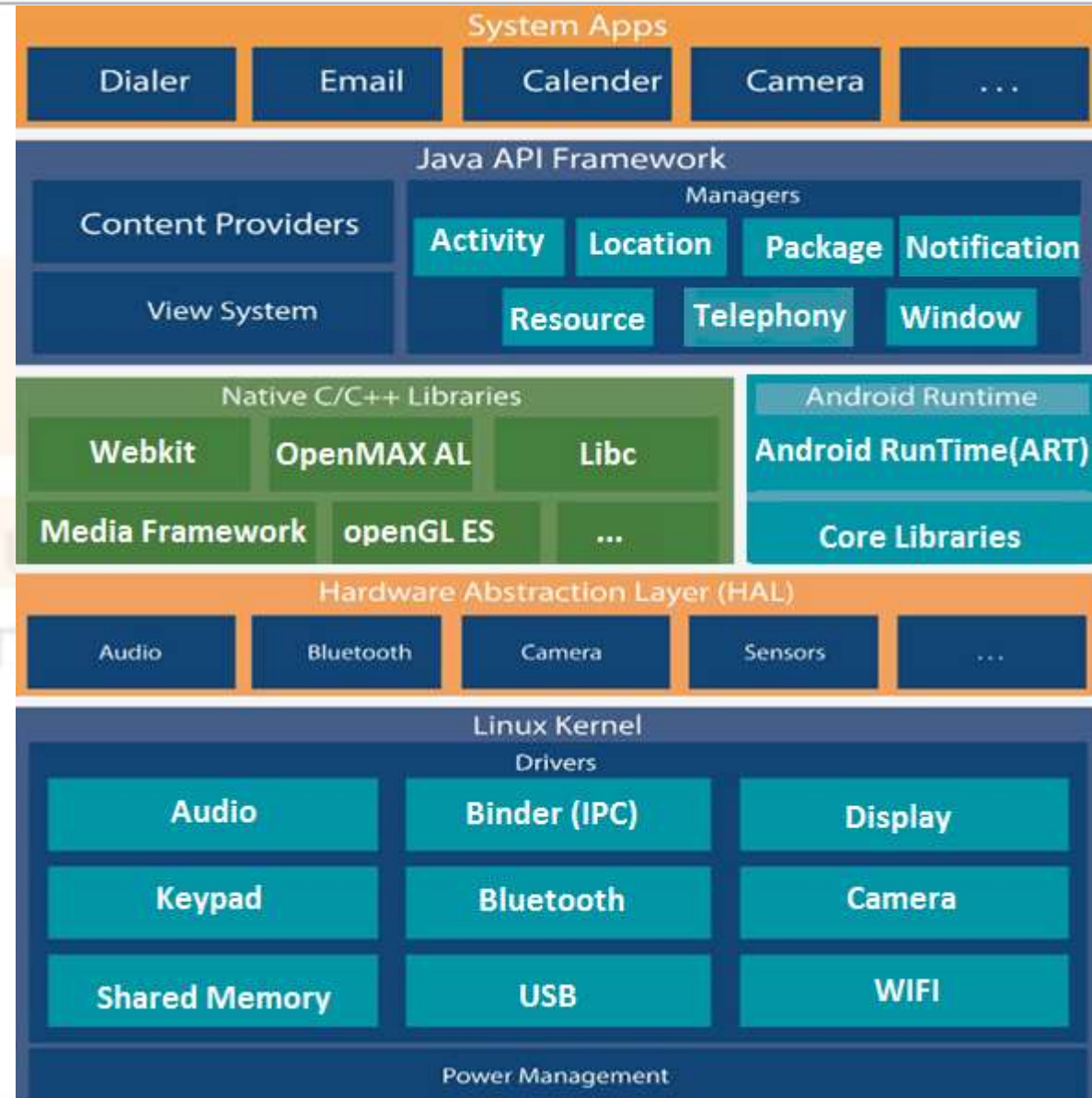
## Android Architecture

The architecture of Android consists of five layers:

- System Apps
- Java API Framework
- Native C/C++ Libraries and Android Runtime
- Hardware Abstraction Layer (HAL)
- Linux Kernel

# Introduction to Android and Android Studio

## Android Architecture



## Android Architecture Explanation

**System Apps:** This is the layer that consists of all the apps (both user installed and the predefined apps.)

- **Predefined Apps:** Android comes with a set of core apps for email, SMS messaging, calendars, internet browsing, contacts, and more. These apps are found naturally and also called as pre-defined apps.
- **Installed Apps:** Third party apps or the apps which are developed by you or any developer and is installed in your device.

## (Continued) Android Architecture Explanation

### Java API Framework:

This layer consists of three major components:

- **Built in Managers** - Android has some in-built managers. It carries the functionality of the assigned UI. It is the mediator for UI and Hardware components.  
**Example:** In app screen, if you click the button "SMS SEND," the SMS manager gets the input from UI and initiates the message sending process. Similarly, if you want to find a location, you can use Location Manager to provide the details.
- **Content Provider** – It is used to share the data between two applications.
- **View System** – It is the set of views to create UI (user interface) for the application.  
This layer consists of all the built-in managers such as, location manager, SMS manager, telephone manager, etc. In Android, the functionalities are handled by different managers. For example, if you want to send SMS programmatically, you should use SMS manager. There are such managers in the JAVA API framework, which is written in JAVA.



## (Continued) Android Architecture Explanation

### **Native C/C++ Libraries**

Android architecture consists of numerous libraries, which handles different kinds of functionalities. It supports java environment as well as writing a programming in C, C++ native applications.

### **Android Runtime (ART)**

Android Runtime was introduced from API level 21 onwards i.e., Android Version 5.0, which helps to run multiple virtual machines on low-memory devices. This layer consists of:

1. Dalvik Virtual Machine (DVM)
2. Core Libraries



## (Continued) Android Architecture Explanation

### Android Runtime (ART)

- **Dalvik Virtual Machine (DVM)**

DVM is developed by Google. It has the same functionality as JVM but works only for Mobiles. It consumes less memory and provides faster performance.

- **Core Libraries**

It is also called as **Android Library**, which consists of an in-built library function. It is used by the developer to create particular functionality in their application.

## (Continued) Android Architecture Explanation

### Hardware Abstraction Layer (HAL)

It is usually called as HAL interface. HAL is an **interface** between the Android Framework (API's) and hardware specific software and drivers.

**Example:** When a camera app is selected, the camera API's are triggered. HAL transfers this triggered request to the camera driver and camera hardware components and process it. HAL provides standard interface for hardware devices that helps to expose them to higher-level JAVA API framework. The Android system loads the library module to the hardware component, when a call is made to device hardware through framework API.

## (Continued) Android Architecture Explanation

### Linux Kernel

- It is the last layer of Android Architecture and the most important one.
- It takes care of power management, memory management, resource access, driver assignment, etc.
- It acts as the concept of Hardware drivers. All the hardware drivers like Audio, Bluetooth, Camera, Display, etc. are placed in this layer.
- This layer is the OS kernel layer.
- There is no user interaction with this layer.
- This layer consists of all the important and basic components that runs the system (the base of the whole system).

# Introduction to Android and Android Studio

## Android versions

Sl. No	Version Name	Version Number	API level	Release Date
1	Alpha	-	1	23 September 2008
2	Beta	1.1	2	9 February 2009
3	Cupcake	1.5	3	27 April 2009
4	Donut	1.6	4	15 September 2009
5	Éclair	2.0-2.1	5-7	26 October 2009
6	Froyo	2.2-2.2.3	8	20 May 2010
7	Ginger Bread	2.3-2.3.7	9-10	6 December 2010
8	Honey Comb	3.0-3.2.6	11-13	22 February 2011
9	Ice-cream sandwich	4.0-4.0.4	14-15	18 October 2011



# Introduction to Android and Android Studio

## (Continued) Android versions

Sl.No	Version Name	Version Number	API level	Release Date
10	Jellybean	4.1-4.3.1	16-18	9 July 2012
11	Kitkat	4.4-4.4.4	19-20	31 October 2013
12	Lollipop	5.0-5.1.1	21-22	12 November 2014
13	Marshmallow	6.0-6.0.1	23	5 October 2015
14	Nougat	7.0-7.1	24 - 25	22 August 2016
15	Oreo	8.0-8.1	26 - 27	21 August 2017
16	Pie	9.0-..	28-..	06 August 2018



## (Continued) Android versions

- All the versions are named in an alphabetical order after a food item name, especially a dessert or a sugary item.
- Every version name will have an API (Application Programming Interface) level number, which represents each version with a numerical value (for the identification purpose).
- In each version, there will be a performance enhancement and bug fixing of previous versions.



## Dalvik Virtual Machine

- The **Dalvik Virtual Machine (DVM)** is an android virtual machine optimised for mobile devices.
- It optimises the virtual machine for *memory, battery life* and *performance*.
- DVM is developed by Google.
- It has the same functionality as JVM but works only for mobiles.
- It consumes less memory and provides faster performance.
- In **JAVA**, the written code is saved with the extension of **.java** called as JAVA file.
- Later, it is compiled as bytecode (Machine readable code) and interprets using JVM (Java Virtual Machine).
- Similarly, in Android, the written code is saved with an extension **.dex** called DEX file. **Jack** (a compilation tool) is used to compile it into bytecode (Machine readable code) and interprets using DVM (Dalvik Virtual Machine).

## Installing Android Studio

**Download Android studio form the below link**

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

To install Android Studio on Windows, proceed as follows:

1. Launch the .exe file you downloaded.
2. Follow the setup wizard to install Android Studio and any necessary SDK tools.

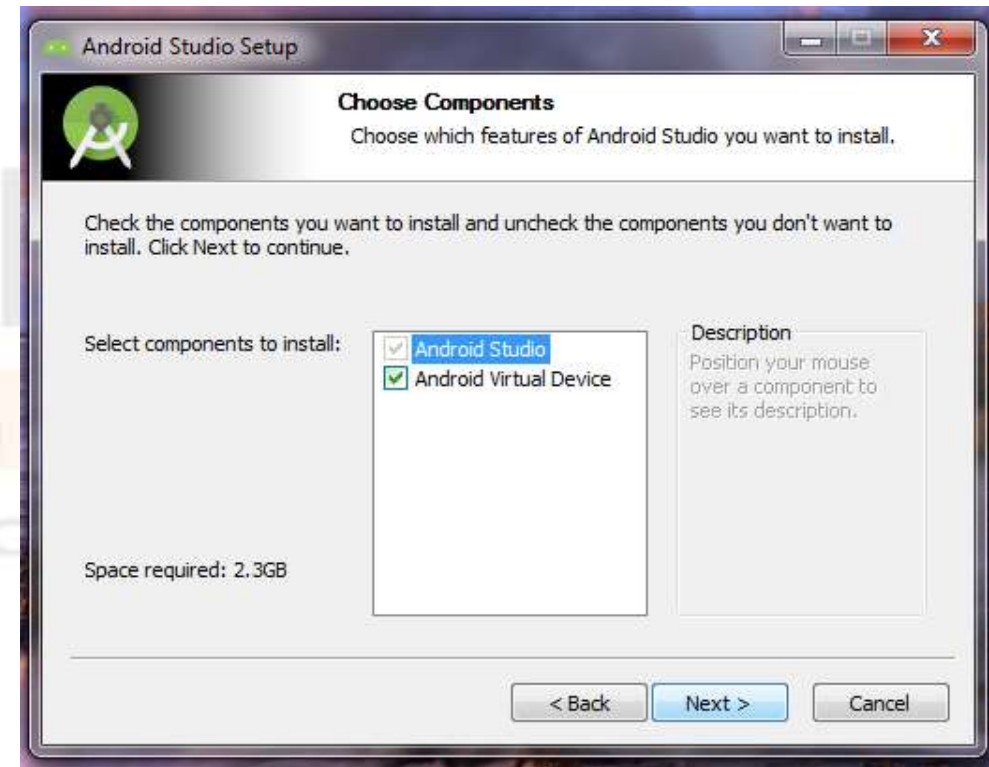


# Introduction to Android and Android Studio

## (Continued) Installing Android Studio



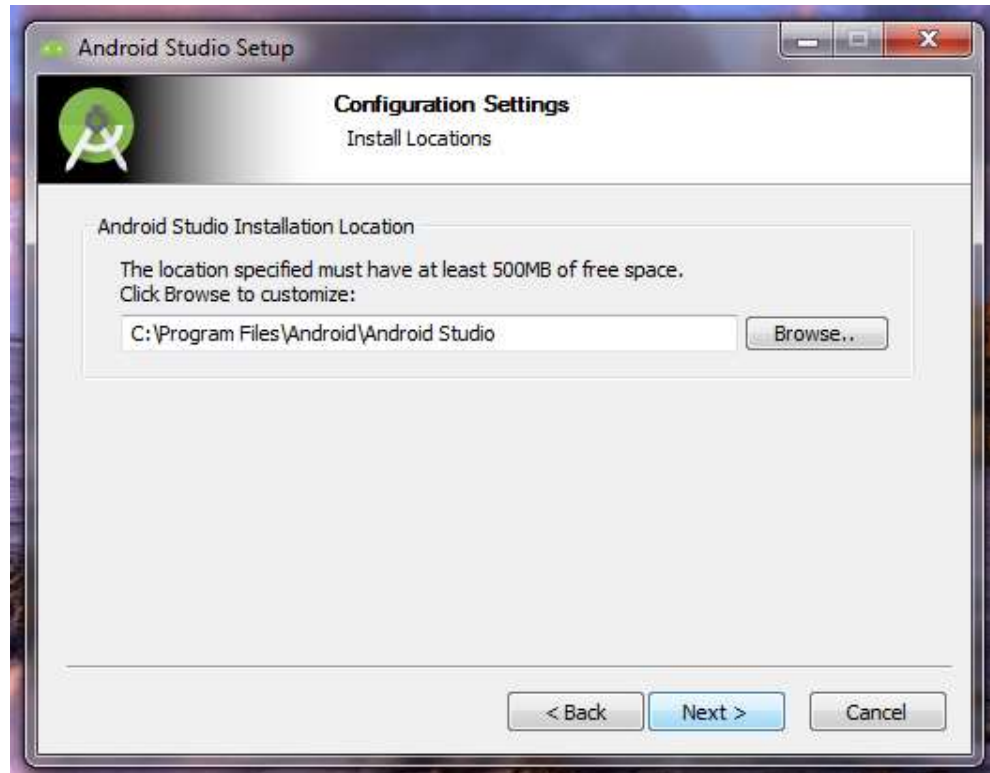
3. Click Next



4. Select all options and Click Next

# Introduction to Android and Android Studio

## (Continued) Installing Android Studio



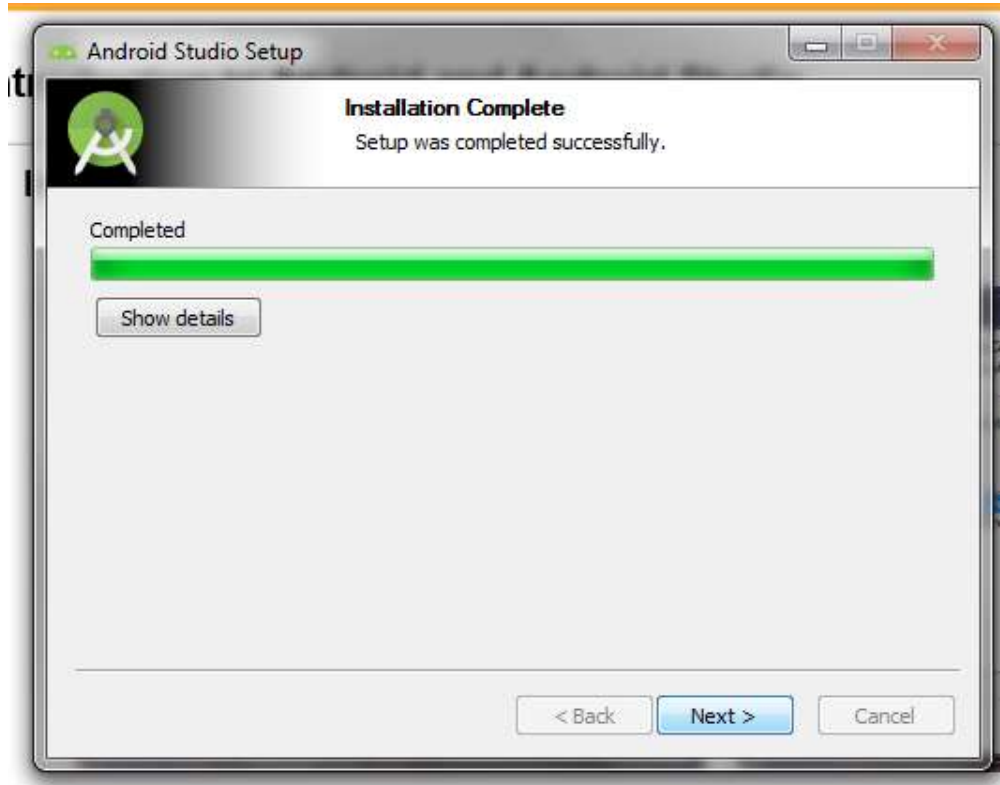
5. Select folder location and  
Click Next



6. Click Install

# Introduction to Android and Android Studio

## (Continued) Installing Android Studio



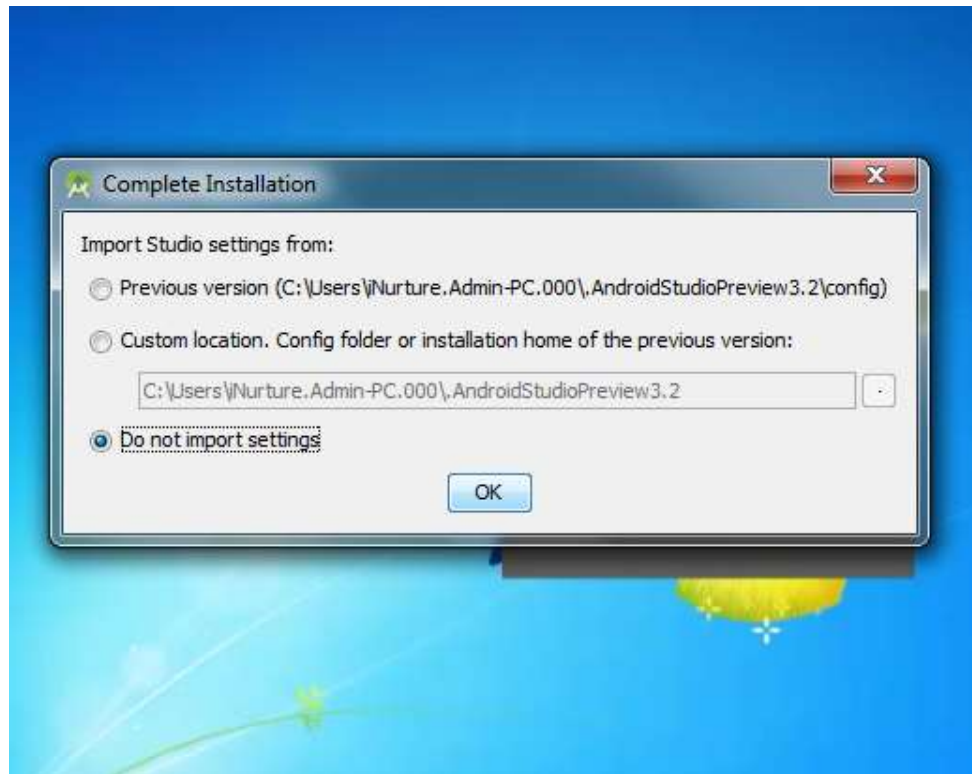
7. Click Next



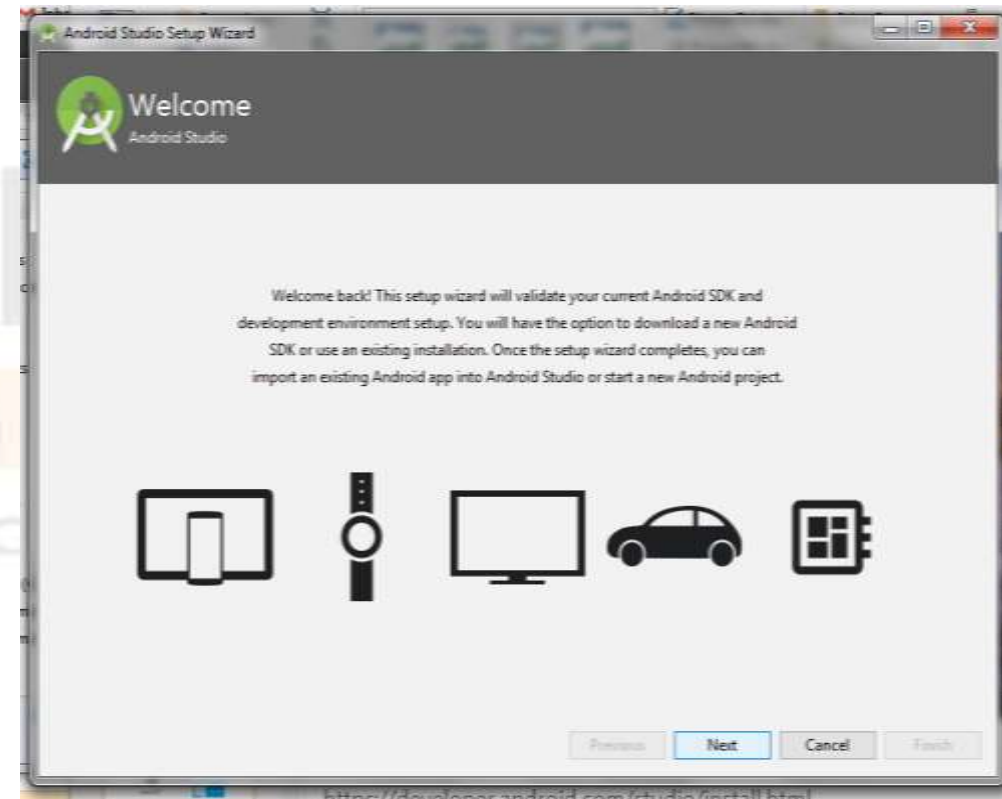
8. Click Finish

# Introduction to Android and Android Studio

## (Continued) Installing Android Studio



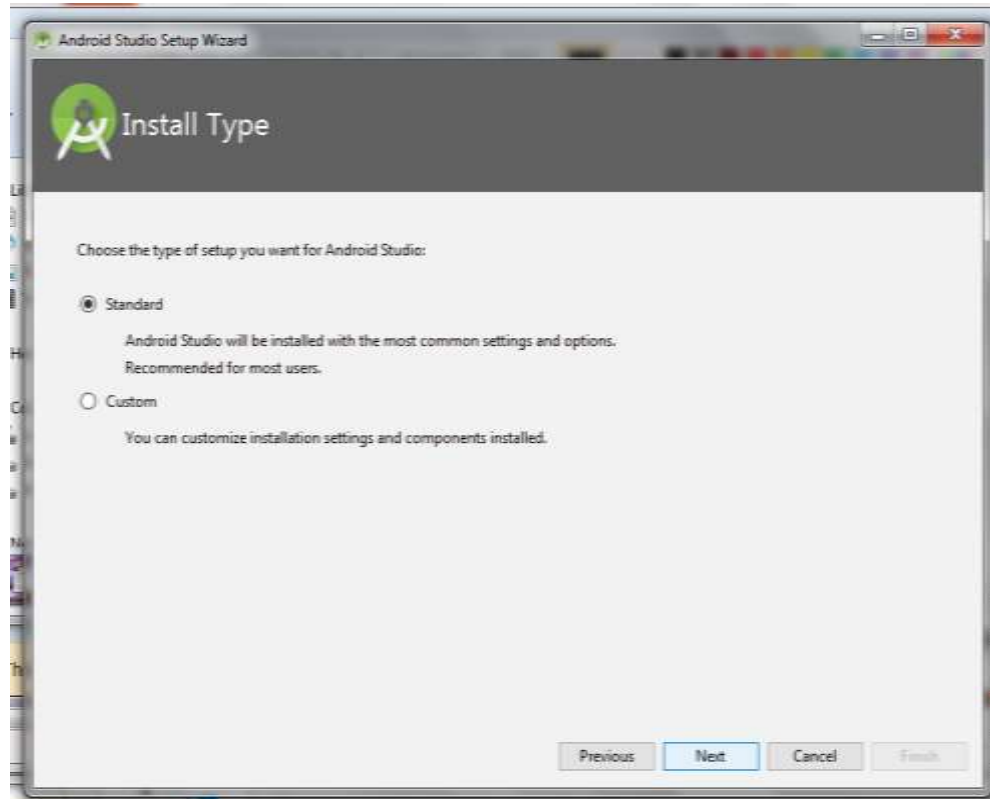
9. Select Do not import settings / Do not have existing setting Click OK



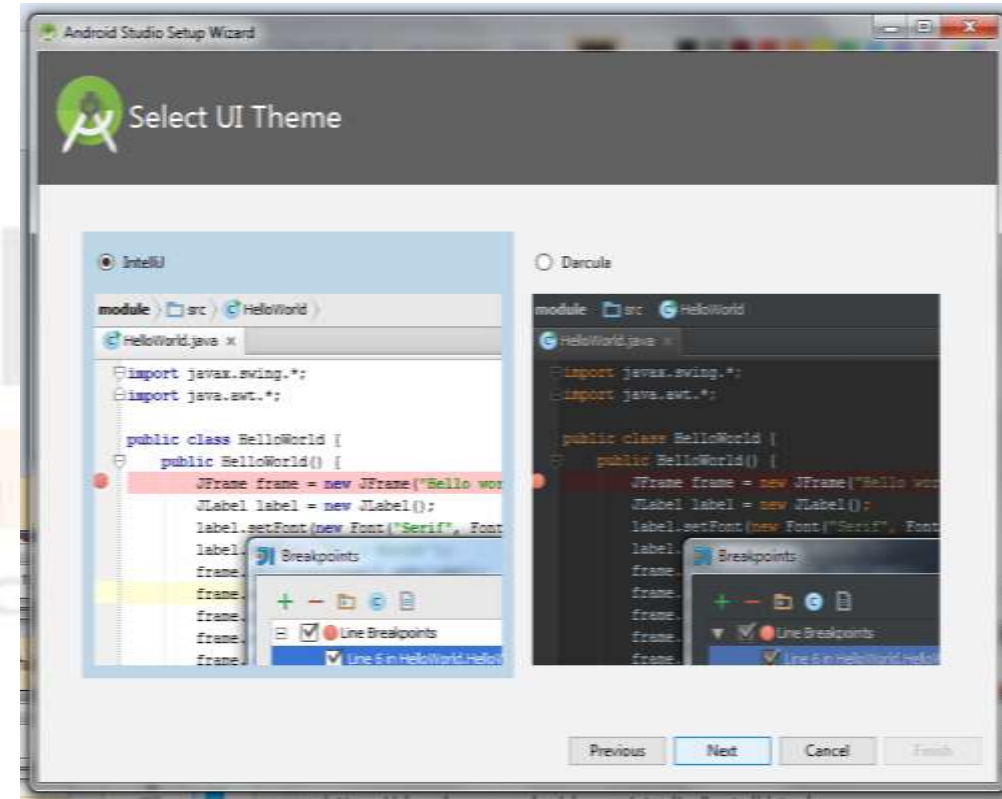
10. Click Next

# Introduction to Android and Android Studio

## (Continued) Installing Android Studio



11. Select Standard Click Next



12. Click Next

For more details please check this link : <https://developer.android.com/studio/install.html>



## Android Studio Environment / Android Development Environment

- To develop an Android application, we have to set a development environment.
- Using Android SDK., a developer can code a program, compile and interpret into working application.
- Under the topic Android Development Environment, we discuss about the architecture of Android platform and its different layers that are used to develop a complete application.

## (Continued) Android Studio Environment

### Android Virtual Device (AVD)

- Android virtual device is used for checking the output of the code we developed.
- The UI preview on the Android studio is just a graphical view and does not have any functionality in it.
- AVD is a virtual device same as a physical device, but it is a virtual dummy. We can perform almost every operation that we perform in our mobile device.
- With the help of device monitor we can make a dummy incoming call to the virtual device for the testing purpose of our app.
- Not only dummy incoming call, we can send a SMS to the virtual device and can assume a location to the virtual device.
- The memory allocation, threads analysis and lots more can be accomplished using Android Device Monitor, this you can find it in the following path:

**Tools → Android Device Monitor**

## Creating Android Virtual Device (AVD)

- **Android Virtual Device (AVD)**

Click on the AVD manager icon at the top (toolbar) of the Android studio.

Or

**Tools → Android → Android virtual device**

It will open a window

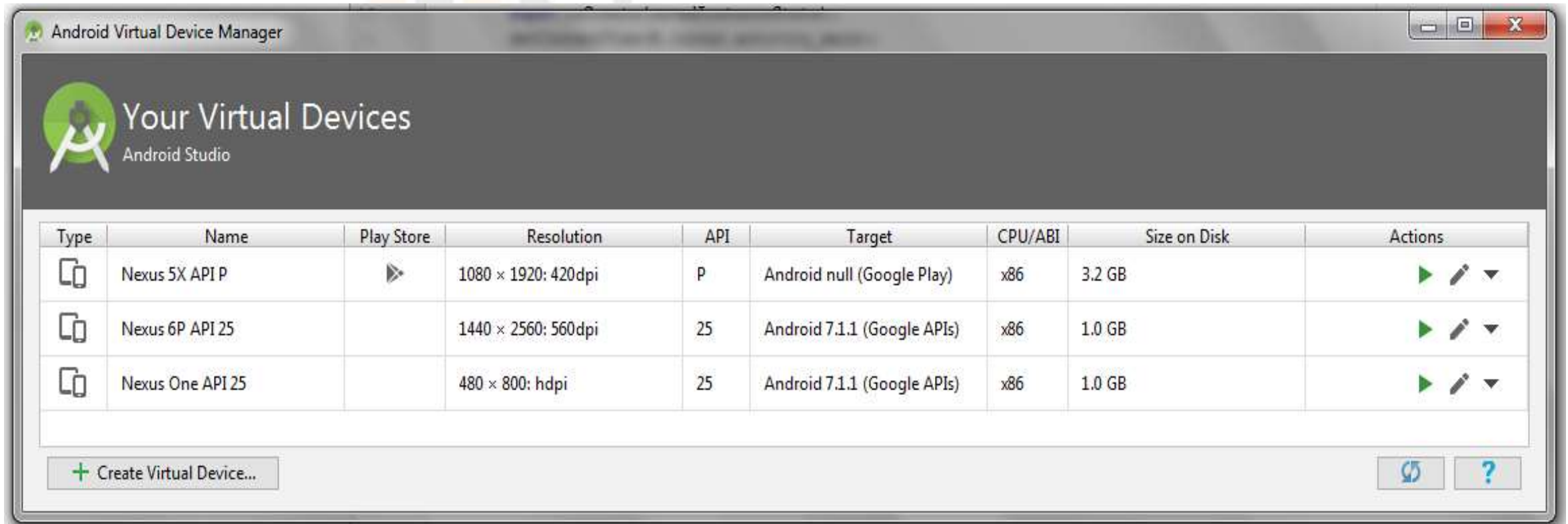




# Introduction to Android and Android Studio

## (Continued) Creating Android Virtual Device (AVD)

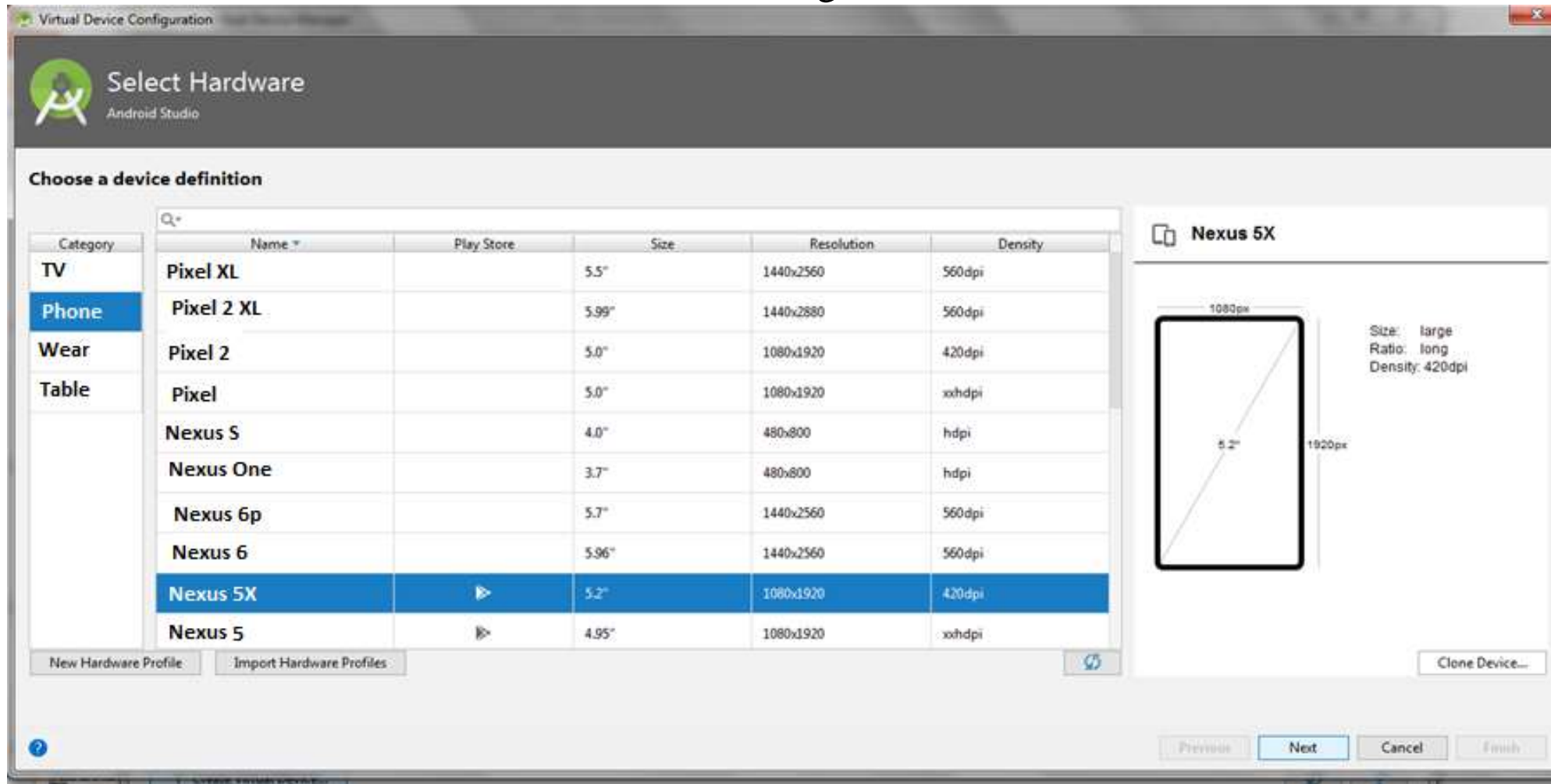
This shows the previously created virtual devices. You can run on any of those AVD also, yet if you want to create a new emulator just click on the **'create virtual device'** button that is located below.



# Introduction to Android and Android Studio

## (Continued) Creating Android Virtual Device (AVD)

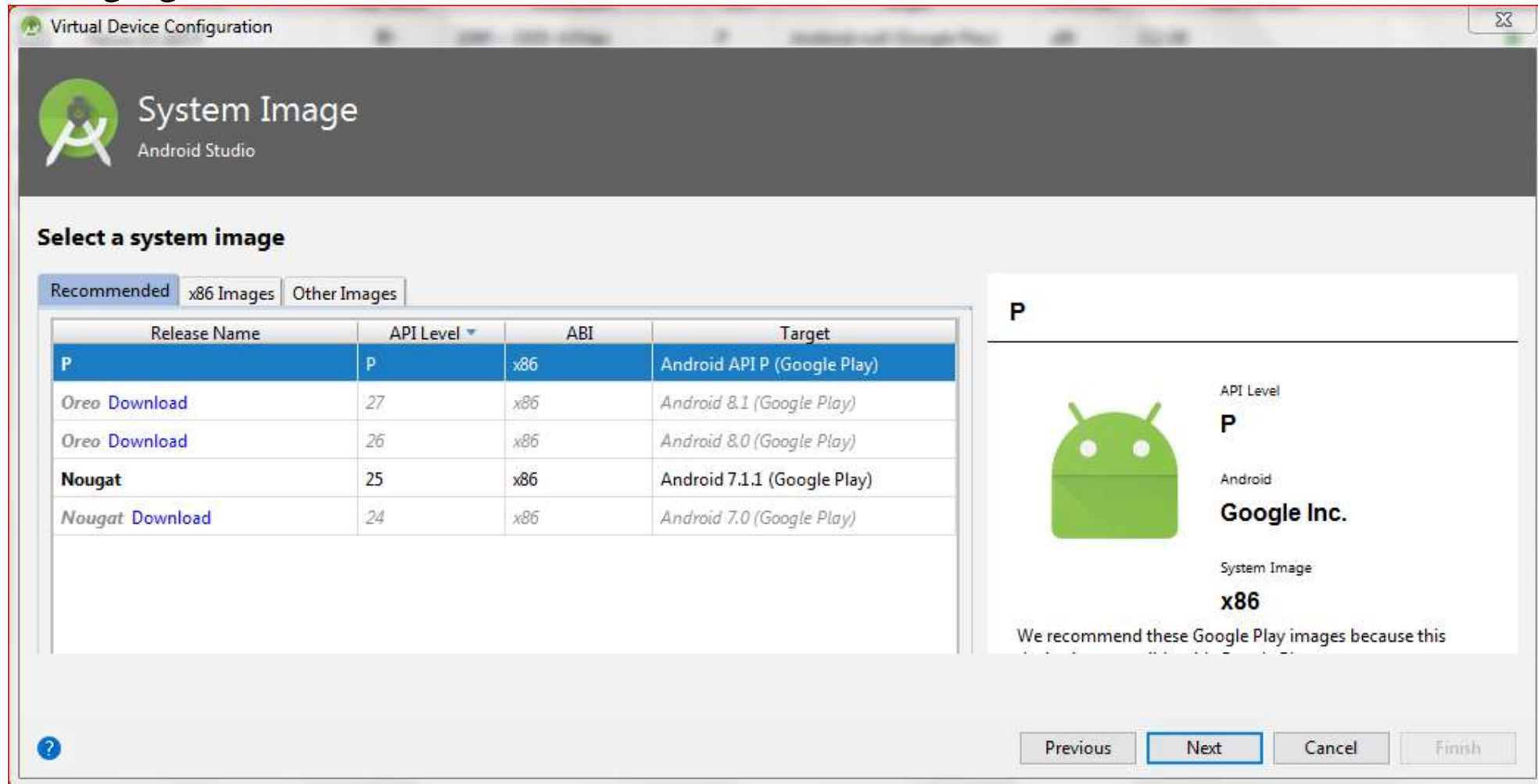
Select the device size and its features as shown in the figure and click on Next.



# Introduction to Android and Android Studio

## (Continued) Creating Android Virtual Device (AVD)

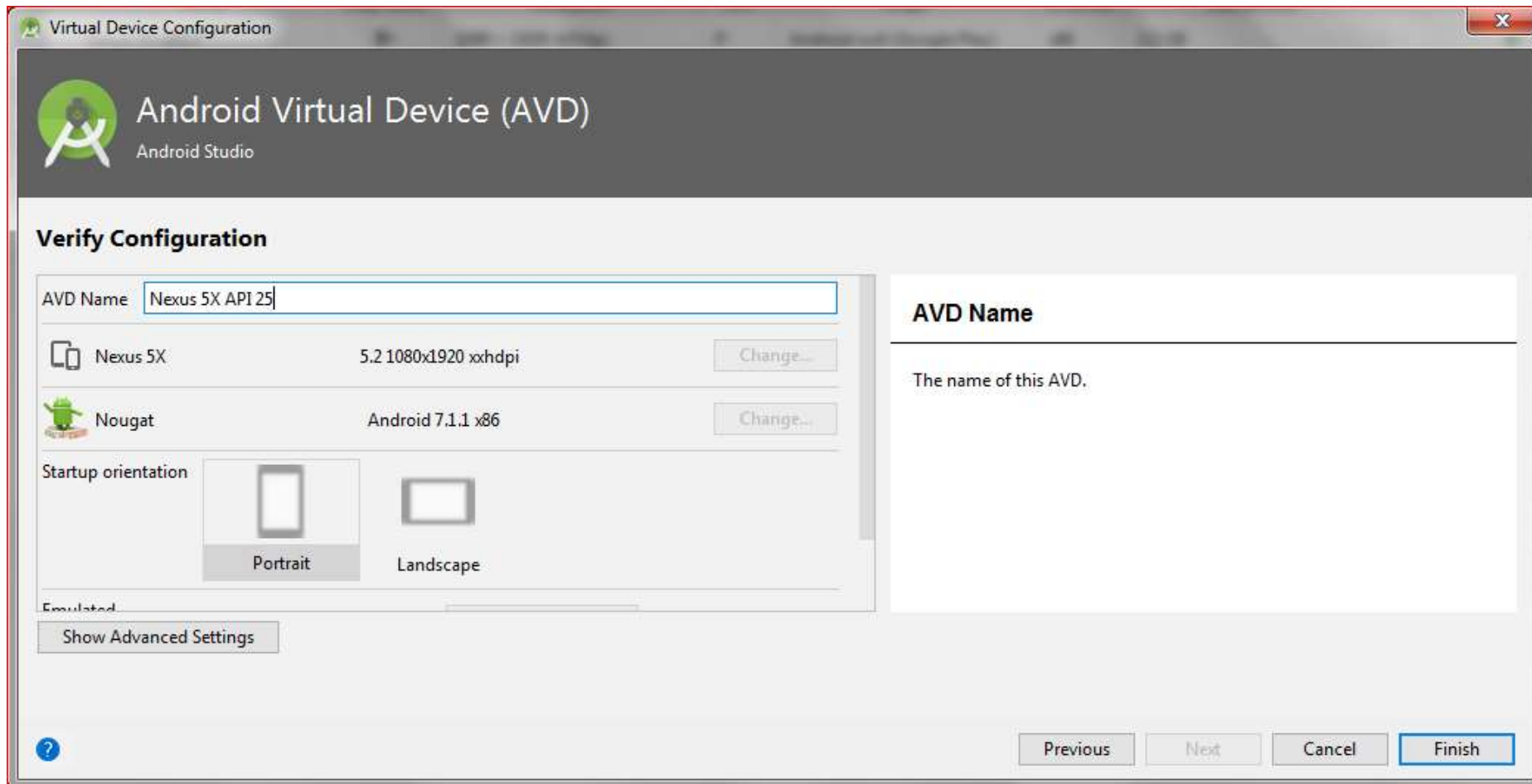
The following figure shows the OS version selection. Click Next.



# Introduction to Android and Android Studio

## (Continued) Creating Android Virtual Device (AVD)

Here, select the OS version of the AVD device and select the configurations based on your requirements as shown in the figure and finally click on Finish.



# Introduction to Android and Android Studio

## Emulator Device

The Emulator does not include virtual hardware for the following:

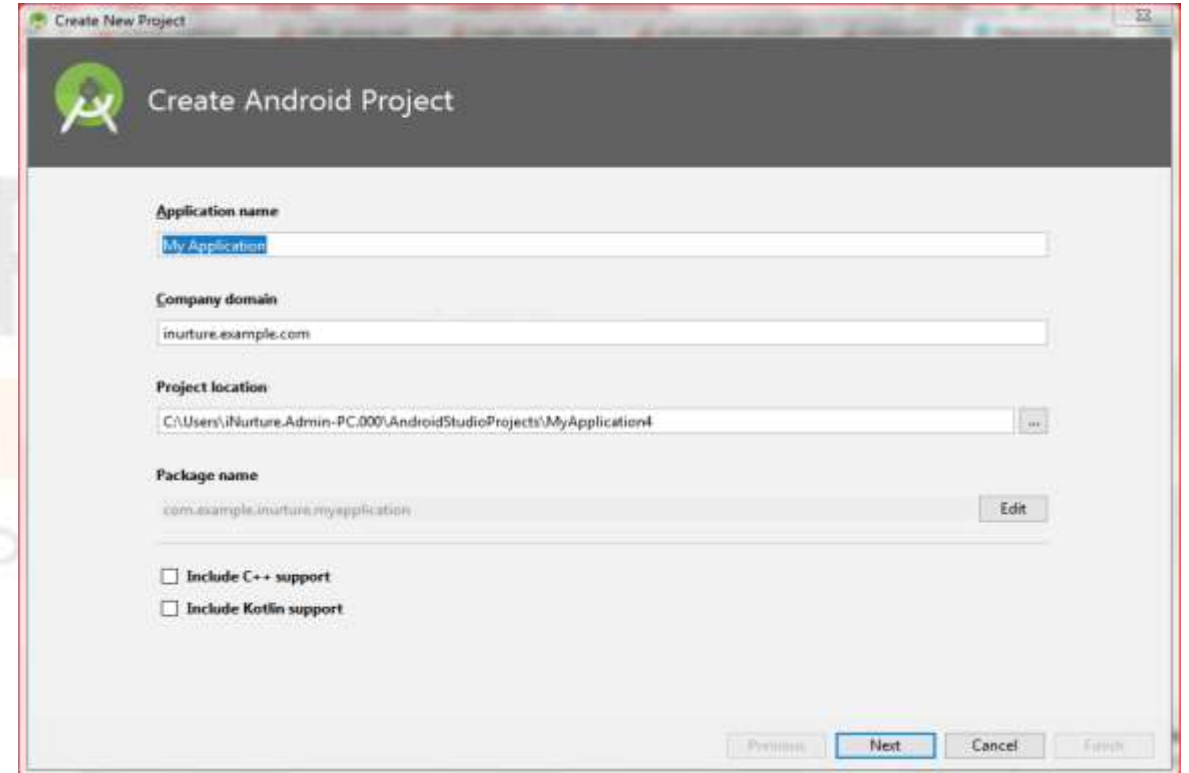
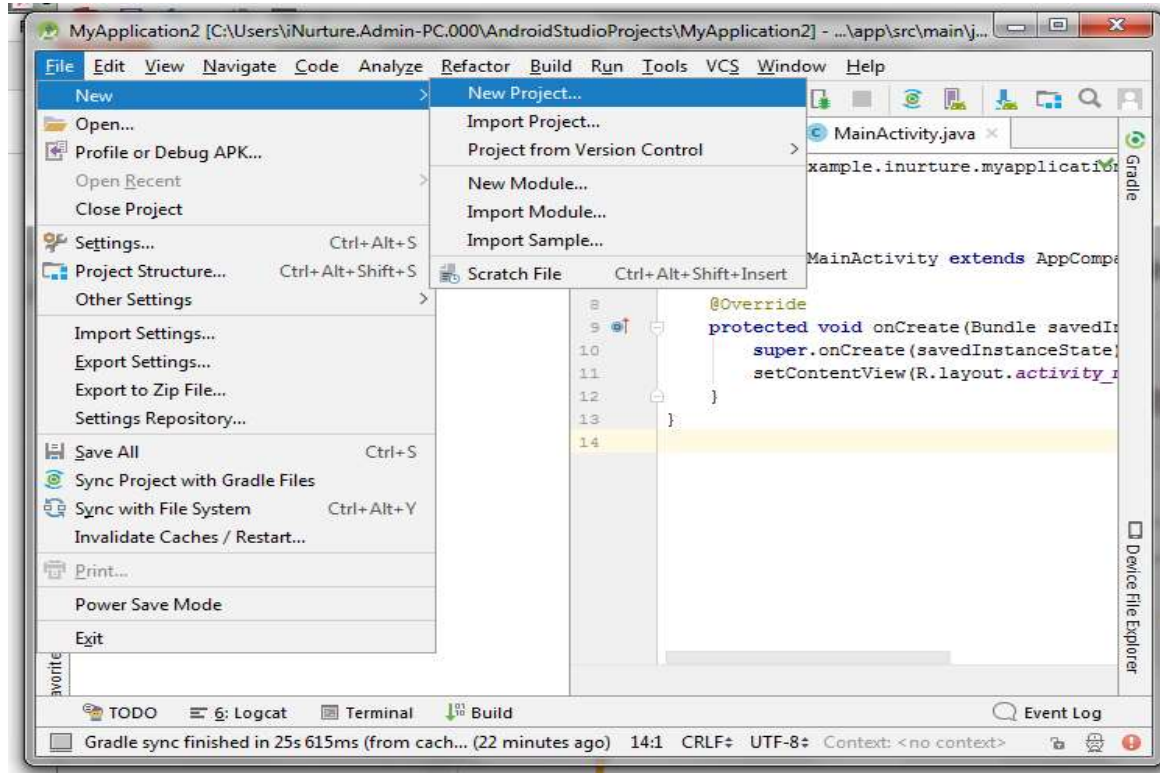
- Wi-Fi
- Bluetooth
- NFC
- SD card insert/eject
- Device-attached headphones and
- USB





# Introduction to Android and Android Studio

## First Android Application

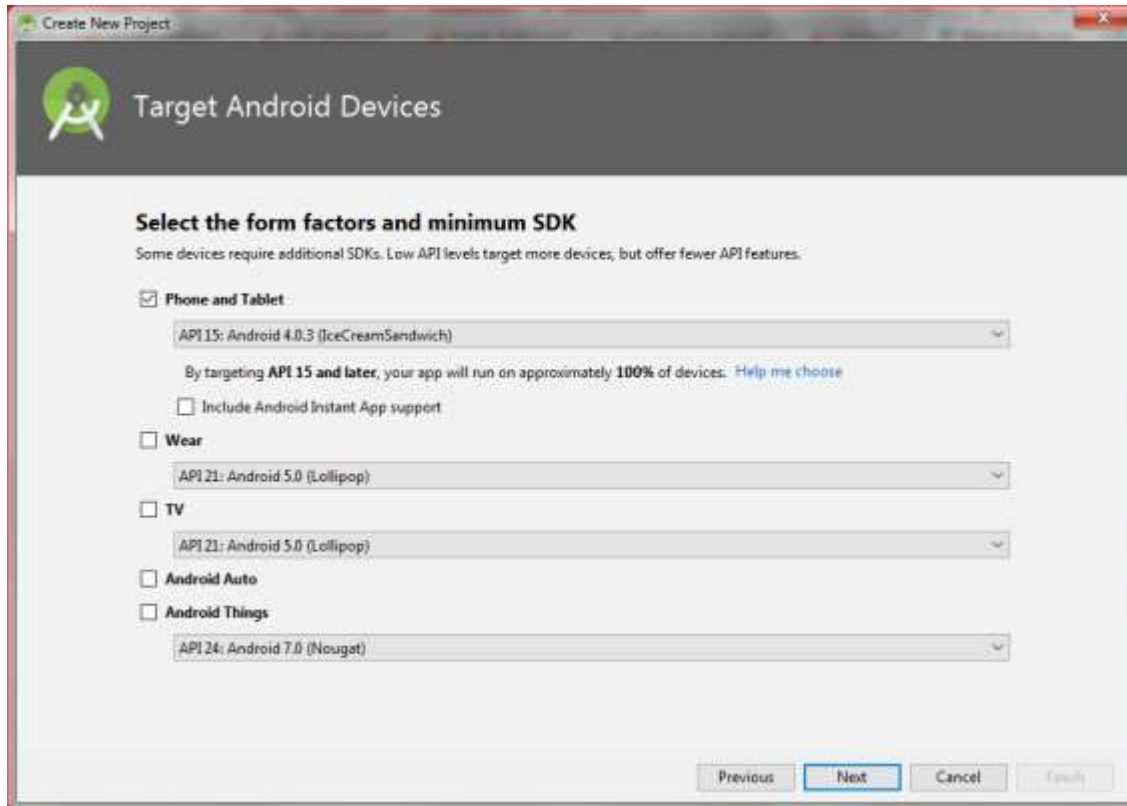


1. Open Android Studio: Go to  
File → New → New Project

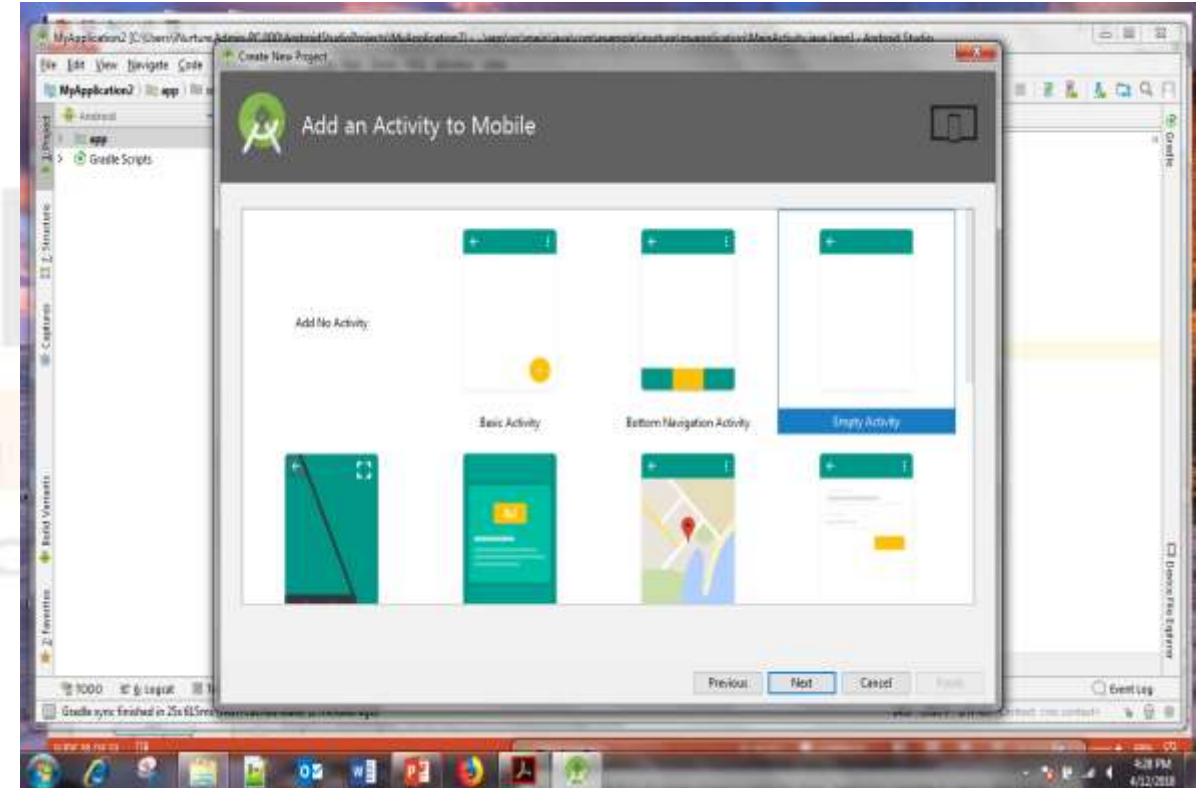
2. Pass application name and location Click Next

# Introduction to Android and Android Studio

## (Continued) First Android Application



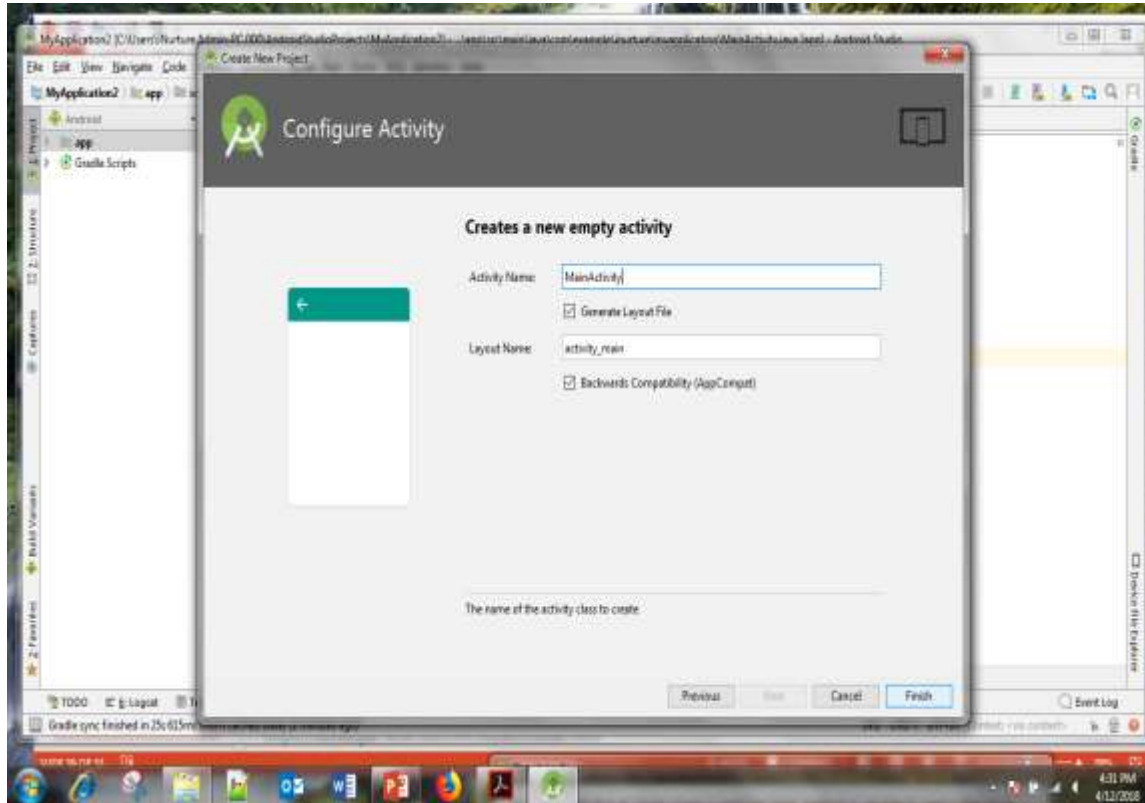
3. Select SDK Version and Click Next



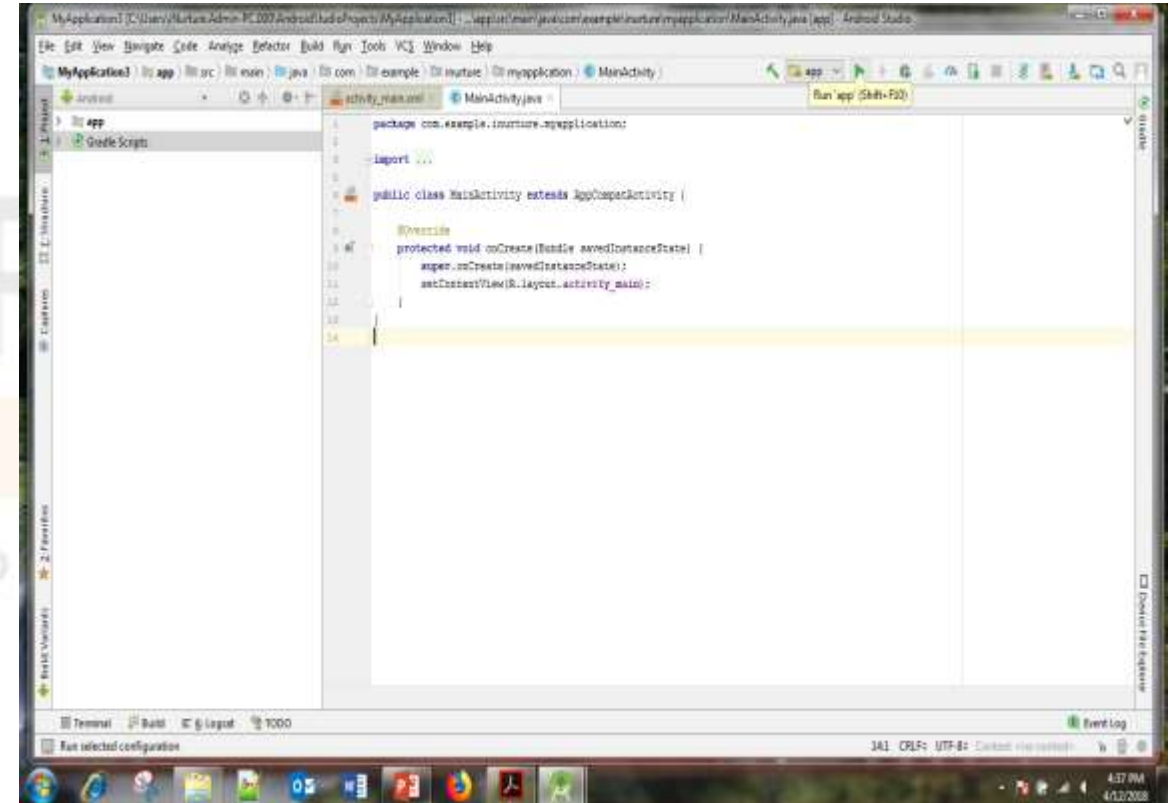
4. Select Activity type and Click Next

# Introduction to Android and Android Studio

## (Continued) First Android Application



5. Pass activity name, layout name Click Finish

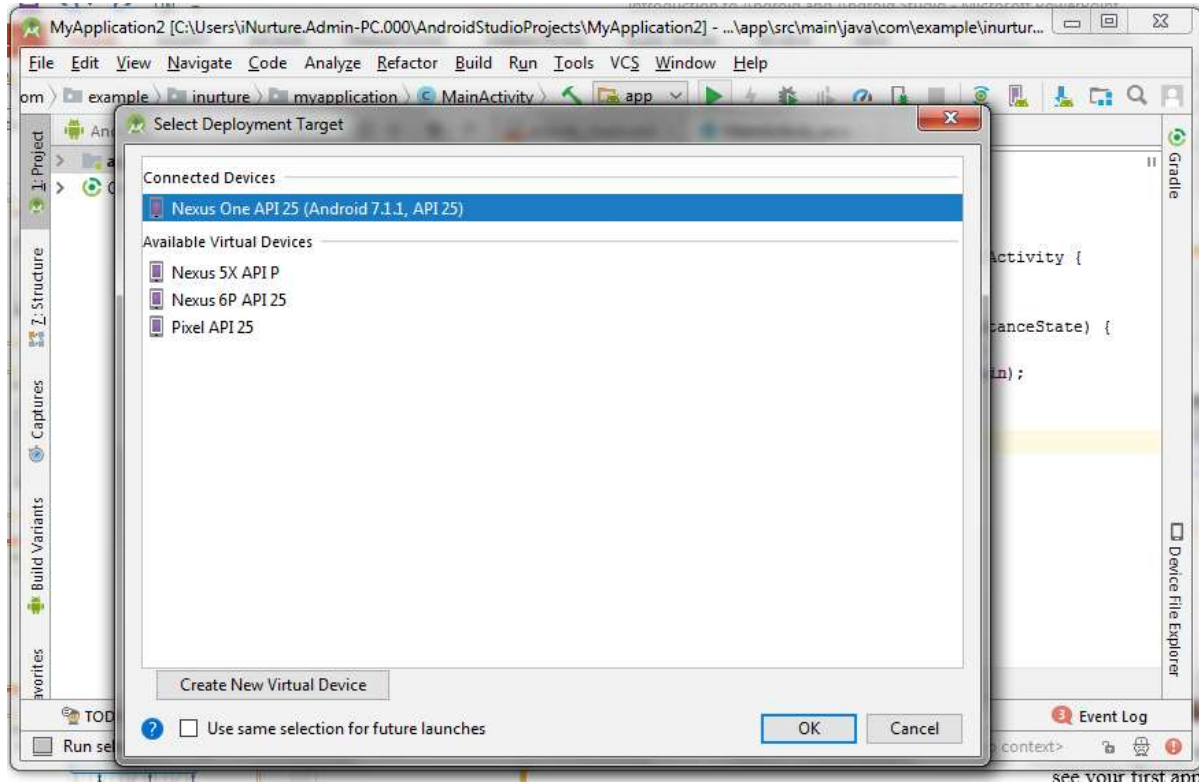


6. Your activity is ready to run now. Click Run to see your first app with welcome message.

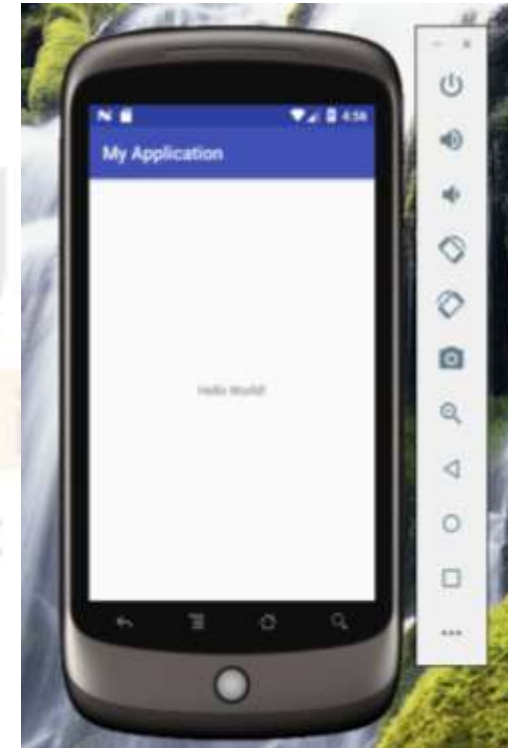


# Introduction to Android and Android Studio

## (Continued) First Android Application



7. Once you click Run button from menu you can see all your configured and connected devices. You can choose your device and click OK

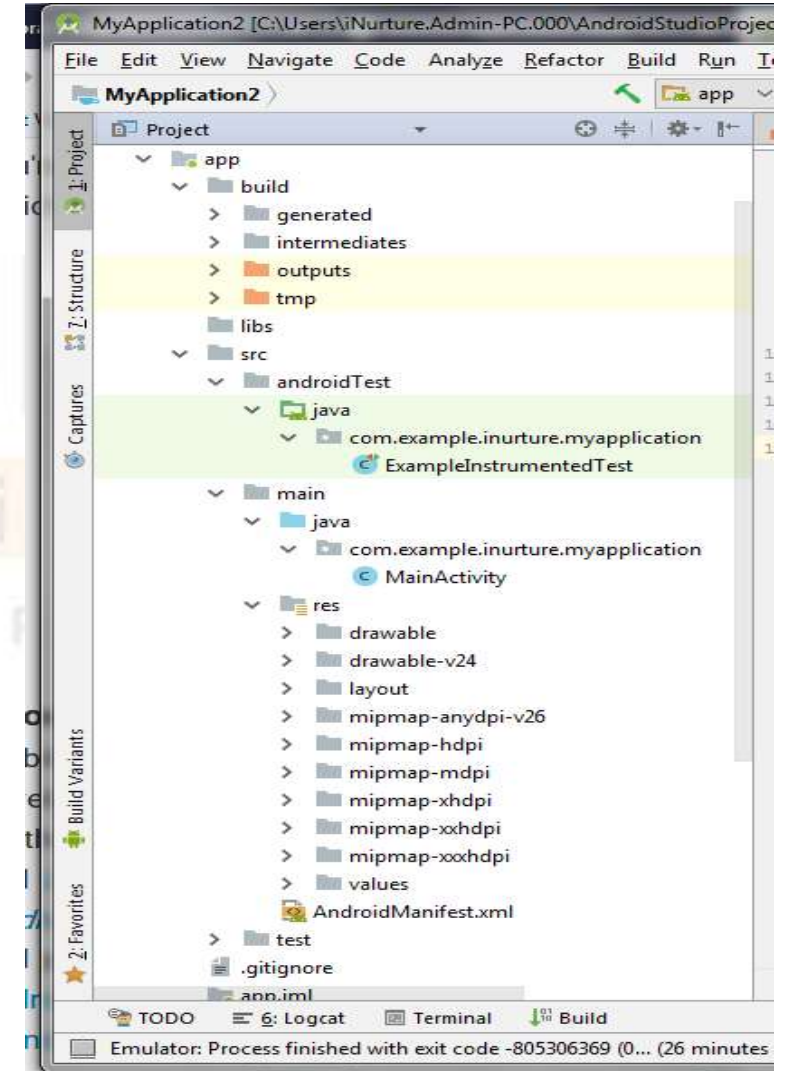
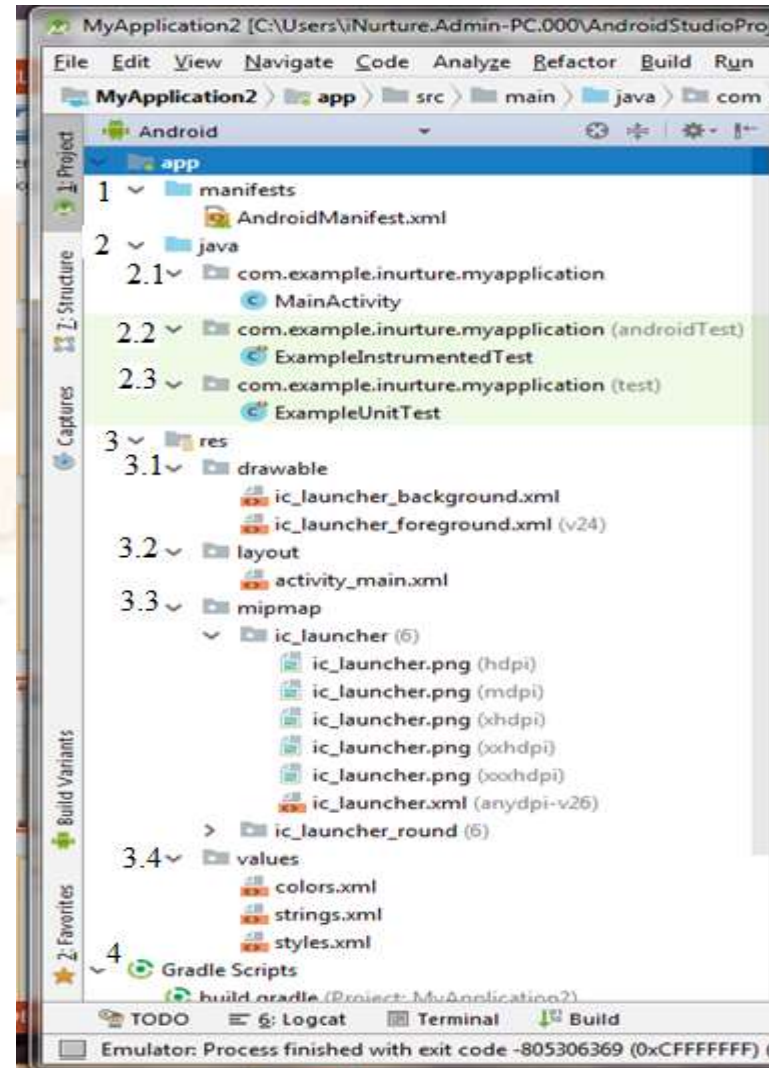


8. If you choose virtual device in step 7 you can see your output in Emulator devices like this.

# Introduction to Android and Android Studio

## Application Folder Structure

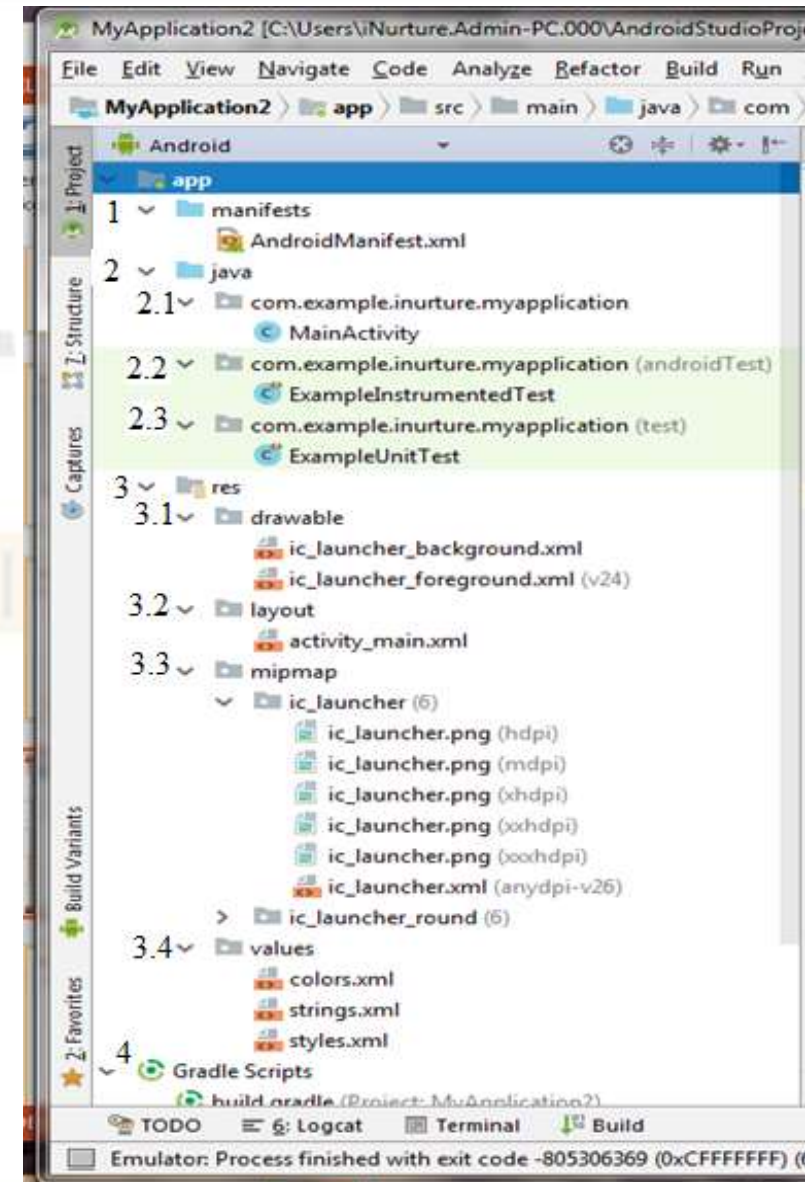
We can see the folder structure in different formats as shown in the figure.



# Introduction to Android and Android Studio

## Application Folder Structure

1. app → manifests
  1. AndroidManifest.xml
2. java
  1. /path/project name
    1. MainActivity
  2. /path/project name(Android Test)
    1. ExampleInstrumentedTest
  3. /path/project name(test)
    1. ExampleUnitTest
3. res
  1. drawable
  2. layouts
  3. mipmaps
    1. ic\_launcher
    2. ic\_launcher\_round
4. Gradle Script



## (Continued) Application Folder Structure

### 1. **AndroidManifest:**

Every app project must have an **AndroidManifest.xml** file (with precisely that name) at the root of the project source set. The manifest file describes essential information about your app to the Android build tools, the Android operating system, and Google Play.

### 2. **Java :**

Contains the Java source code files, separated by package names, including JUnit test code.

1. /path/project name  
    Contains all activities
2. /path/project name (Android Test)  
    Contains all android test
3. /path/project name (test)  
    Contains all UnitTest



## (Continued) Application Folder Structure

### 3. res :

Contains all non-code resources, such as XML layouts, UI strings, and bitmap images, divided into corresponding sub-directories like (drawable, layouts, mipmaps).

#### 1. drawable:

Bitmap files (.png, .9.png, .jpg, .gif) or XML files that are compiled into the following drawable resource subtypes.

#### 2. layouts:

XML files that define a user interface layout. A layout resource defines the architecture for the UI in an Activity or a component of a UI.

#### 3. mipmaps:

Drawable files for different launcher icon densities.

1. ic\_launcher

2. ic\_launcher\_round

## (Continued) Application Folder Structure

### 4. Gradle Script:

1. build.gradle (module)

This defines the module-specific build configurations.

2. build.gradle (project)

This defines your build configuration that apply to all modules. This file is integral to the project, so you should maintain them in revision control with all other source code.



## Manifest file

- The **AndroidManifest.xml** file contains information of your package, including components of the application such as activities, services, broadcast receivers, content providers etc.
- Every application must have an **AndroidManifest.xml** file (with precisely that name) in its root directory.
- Manifest file provides essential information about your app to the Android system before it can run any of the app's code.

## (Continued) Manifest file

### Importance of Android Manifest File:

- Package name of the application is declared in the manifest file. Package name is a unique name of the Java package of the application.
- All the components of the application like activities, services, broadcast receivers and content providers are described in this file.
- Permissions for an application to access protected parts of the API and interact with other applications are declared in this file.
- Minimal level of Android API required by the application is declared here.

## Manifest file Example

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.inurture.myapplication"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
/>
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
```

## (Continued) Manifest file Example

```
<activity android:name=".SecondActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
    </intent-filter>
</activity>
<service>
    <intent-filter> . . . </intent-filter>
    <meta-data/>
</service>
<receiver>
    <intent-filter> . . . </intent-filter>
    <meta-data />
</receiver>
</application>
</manifest>
```

## Manifest file Elements

### Elements of Android Manifest File:

#### **<action>:**

Every Intent Filter should be declared along with its action to perform.

<action> element is used to declare that action.

If action is not declared, no intent objects will get through the filter.

**Syntax:** <action>....</action>

## Manifest file Attributes

- <action>
- <activity>
- <application>
- <category>
- <data>
- <intent-filter>
- <manifest>
- <meta-data>
- <path-permission>
- <permission>
- <uses-permission>
- <uses-sdk>





## (Continued) Manifest file Attributes

### Attributes used:

android:name:

### <activity>:

This element is used to declare all the activity class of an application. All activities must be represented by <activity> elements in the manifest file. Those that are not declared, will not be seen by the system and will never be run.

**Syntax:** <activity> ..... </activity>

## (Continued) Manifest file Attributes

### Activity Attributes used:

android:allowEmbedded=["true" | "false"] android:autoRemoveFromRecents=["true" | "false"]  
android:banner="drawable resource"  
android:clearTaskOnLaunch=["true" | "false"]  
android:configChanges=["mcc", "mnc", "locale", "touchscreen", "keyboard", "keyboardHidden",  
"navigation", "screenLayout", "fontScale", "uiMode", "orientation", "screenSize", "smallestScreenSize"]  
android:name="string"  
android:noHistory=["true" | "false"]  
android:parentActivityName="string"  
android:permission="string" android:screenOrientation=["unspecified" | "behind" | "landscape" |  
"portrait" | "reverseLandscape" | "reversePortrait" | "sensorLandscape" | "sensorPortrait" |  
"userLandscape" | "userPortrait" | "sensor" | "fullSensor" | "nosensor" |  
"user" | "fullUser" | "locked"]

## (Continued) Manifest file Attributes

### Attributes used:

android:name:

### <activity>:

This element is used to declare all the activity class of an application. All activities must be represented by <activity> elements in the manifest file. Those that are not declared, will not be seen by the system and will never be run.

**Syntax:** <activity> ..... </activity>

## (Continued) Manifest file Attributes

### Attributes used:

android:name:

### <activity>:

This element is used to declare all the activity class of an application. All activities must be represented by <activity> elements in the manifest file. Those that are not declared, will not be seen by the system and will never be run.

**Syntax:** <activity> ..... </activity>

## (Continued) Manifest file Attributes

### **<application>:**

This element is used to declare the application related components. It occurs only once in the manifest file. Application components like service, broadcast receivers, activities, etc. are declared under this element.

**syntax:** <application> ..... </application>

### **Attributes used:**

- android:allowTaskReparenting=["true" | "false"]
- android:allowBackup=["true" | "false"]
- android:backupAgent="string"
- android:backupInForeground=["true" | "false"]
- android:banner="drawable resource"
- android:extractNativeLibs=["true" | "false"]
- android:fullBackupContent="string"

## (Continued) Manifest file Attributes

### **<category>:**

This element is used to add a category name to an intent filter.

**syntax:** <category android:name="string" />

### **Attribute used:**

android:name

### **<data>:**

This element is used to add a data specification to an intent filter.

**syntax:** <data> ..... </data>

### **Attributes used :**

android:scheme="string"

android:host="string"

android:port="string"

android:path="string"

android:pathPattern="string"

android:pathPrefix="string"

android:mimeType="string" />



## (Continued) Manifest file Attributes

### **<intent-filter>:**

<intent-filter> is a sub-element of <activity>, <service>, <receiver>, elements.

It is used to specify the type of intent to which the activity, service or broadcast receiver is going to respond.

<intent-filter> can also be used inside <service>, <receiver>, <activity-alias> elements.

**syntax:** <intent-filter> ..... </intent-filter>

### **Attribute used:**

android:icon="drawable resource"

android:label="string resource"

android:priority="integer"

## (Continued) Manifest file Attributes

### **<manifest>:**

- It is the root element of AndroidManifest.xml file.
- This element can occur only once. xmlns and package attributes are mandatory in this element.
- It must contain an <application> element and specify xmlns:android and package attributes.

**syntax:** <manifest>.....</manifest>

### **Attribute used:**

```
xmlns:android="http://schemas.android.com/apk/res/android"  
package="string"  
android:sharedUserId="string"  
android:sharedUserLabel="string resource"  
android:versionCode="integer"  
android:versionName="string"  
android:installLocation=["auto" | "internalOnly" | "preferExternal"] >
```

## (Continued) Manifest file Attributes

### **<meta-data>:**

This element is used to declare a name-value pair for an item of additional, arbitrary data that can be supplied to the parent component.

Ex: `<meta-data android:name="color" android:value="@string/red" />`

**syntax:** `<meta-data>....</meta-data>`

### **Attribute used:**

`android:name="string"`

`android:resource="resource specification"`

`android:value="string"`

## (Continued) Manifest file Attributes

### **<path-permission>:**

This element defines the path and required permissions for a specific subset of data within a content provider. This element can be specified multiple times to supply multiple paths.

**syntax:** <path-permission>....<path-permission/>

### **Attribute used:**

android:path="string"  
android:pathPrefix="string"  
android:pathPattern="string"  
android:permission="string"  
android:readPermission="string"  
android:writePermission="string"

## (Continued) Manifest file Attributes

### **<permission>:**

This element is used to declare a security permission that can be used to limit access to specific components or features of this or other applications.

**syntax:** <permission>.....</permission>

### **Attribute used:**

android:description="string resource"  
android:icon="drawable resource"  
android:label="string resource"  
android:name="string"  
android:permissionGroup="string"  
android:protectionLevel=["normal" | "dangerous" |  
"signature" | "signatureOrSystem"]

## (Continued) Manifest file Attributes

### **<uses-permission>:**

- This element is used to specify any special permissions required to run this application.
- These permission are requested to the user for proper functioning of the application.
- Permissions are granted by the user when the application is installed (on devices running Android 5.1 and lower) or while the app is running (on devices running Android 6.0 and higher).

**syntax:** <uses-permission>.....</uses-permission>

### **Attribute used:**

android:name="string"

android:maxSdkVersion="integer"



## (Continued) Manifest file Attributes

### <uses-sdk>:

- This element is used to specify the api-level value, which specifies the application compatibility android versions.
- The API Level expressed by an application will be compared to the API Level of a given Android system, which may vary among different Android devices.

**syntax:** <uses-sdk>....</uses-sdk>

### Attribute used:

android:minSdkVersion="integer"

android:targetSdkVersion="integer"

android:maxSdkVersion="integer"

## R.java file

### Android R.java

- 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\_main.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.
- If you delete R.jar file, android creates it automatically.

## R.java file

*/\* AUTO-GENERATED FILE. DO NOT MODIFY.*

*\* This class was automatically generated by the aapt tool from the resource data it found. It should not be modified by hand.\*/*

**package** com.example.android.viewflipper;

**public final class** R {

**public static final class** anim {

**public static final int** left\_in=0x7f040000;

**public static final int** left\_out=0x7f040001;

**public static final int** right\_in=0x7f040002;

**public static final int** right\_out=0x7f040003;

  }

**public static final class** attr {

  }

**public static final class** dimen {

*/\*\* Default screen margins, per the Android Design guidelines. Example customization of dimensions originally defined in res/values/dimens.xml (such as screen margins) for screens with more than 820dp of available width. This would include 7" and 10" devices in landscape (~960dp and ~1280dp respectively). \*/*

**public static final int** activity\_horizontal\_margin=0x7f050000;

**public static final int** activity\_vertical\_margin=0x7f050001;

  }

**public static final class** drawable {

**public static final int** android\_image=0x7f020000;

**public static final int** ic\_launcher=0x7f020001;

**public static final int** image\_five=0x7f020002;

**public static final int** image\_four=0x7f020003;

**public static final int** image\_one=0x7f020004;

## (Continued) R.java file

```
    public static final int image_six=0x7f020005;
    public static final int image_three=0x7f020006;
    public static final int image_two=0x7f020007;
    public static final int logo=0x7f020008;
}
public static final class id {
    public static final int action_settings=0x7f090002;
    public static final int relativeLayout=0x7f090000;
    public static final int viewFlipper=0x7f090001;
}
public static final class layout {
    public static final int activity_main=0x7f030000;
}
public static final class menu {
    public static final int main=0x7f080000;
}
public static final class string {
    public static final int action_settings=0x7f060002;
    public static final int app_name=0x7f060000;
    public static final int hello_world=0x7f060001;
}
public static final class style {
    public static final int AppBaseTheme=0x7f070000;
    public static final int AppTheme=0x7f070001;
}
}
```

## Activity

- An **Activity** is like a window or single screen which holds view items like button, edit text, text view etc. through which user interacts with the application.
- An application usually consists of multiple activities that are bound to each other.
- An application always starts with the activity which is marked with keyword "**main()**".
- Once an Activity is started, it first calls the **onCreate() callback** method, so the code that is written under this method will be executed first.
- When the Activity gets completed **onDestroy()** method is called.

## Creating an Activity Class

- An Activity class is created as a subclass of Activity.
- In your Activity, Call back methods are implemented which are called by the system when the Activity transit between various states of its lifecycle.

### Two Main Callback Methods are:

#### **onCreate():**

This method is called, whenever an Activity is started. This method is used to initialize all the components.

#### **onPause():**

This method is called, when the user is leaving from one activity to another. The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.

## Class Declaration:

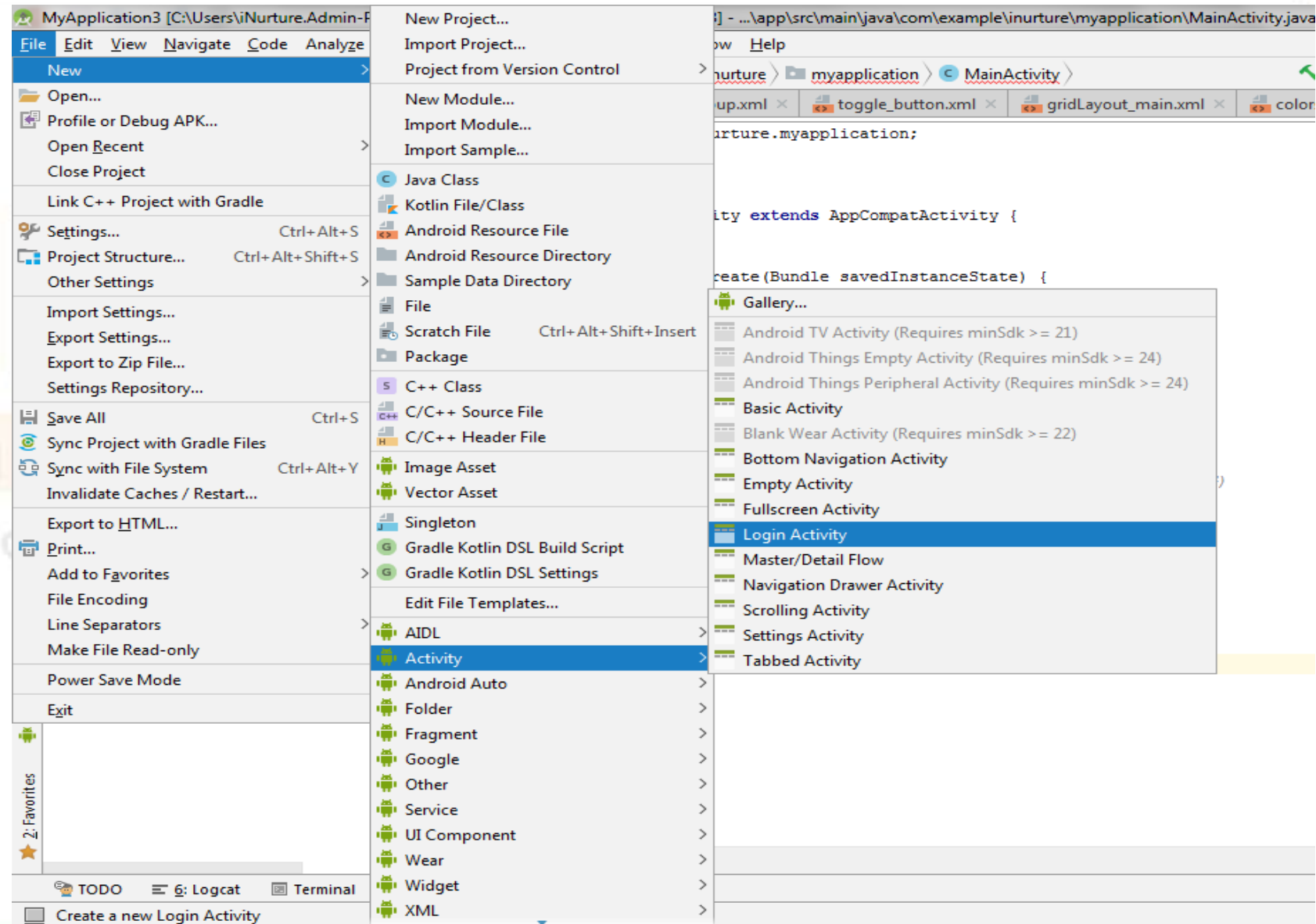
```
public class MainActivity extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    /** Called when another activity is taking focus. */
    @Override
    protected void onPause()
    {
        super.onPause();
        Log.d(msg, "The onPause() event");
    }
}
```



# Introduction to Android and Android Studio

## Create Activity

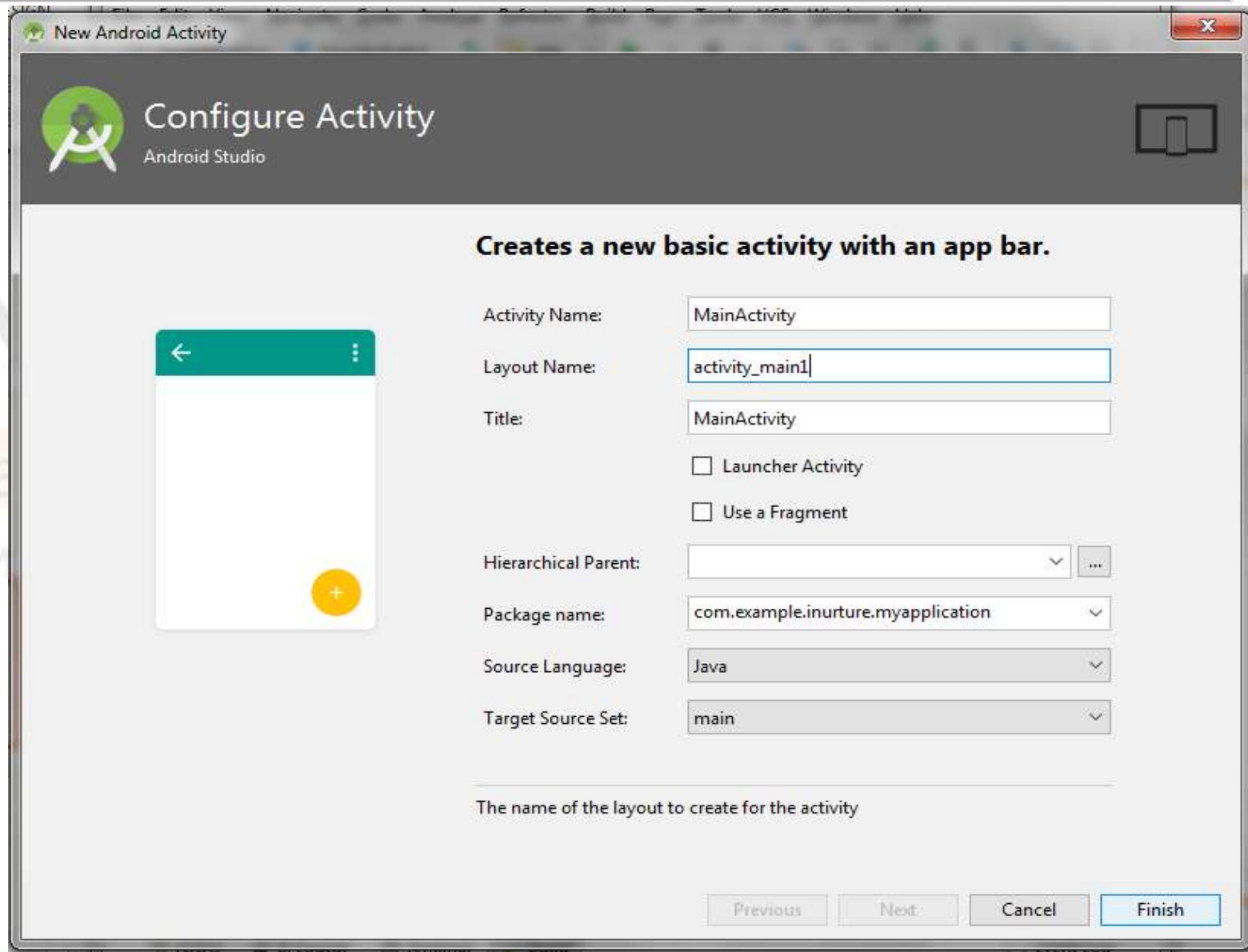
- Right click app →
- New →
- Activity →
- Chose activity type →



# Introduction to Android and Android Studio

## (Continued) Create Activity

- Enter your Activity name
- Layout Name
- Title (Activity and Title mostly same)



New Android Activity

**Configure Activity**  
Android Studio

Creates a new **basic activity with an app bar.**

Activity Name: MainActivity

Layout Name: activity\_main1

Title: MainActivity

☐ Launcher Activity

☐ Use a Fragment

Hierarchical Parent: [Dropdown]

Package name: com.example.inurture.myapplication

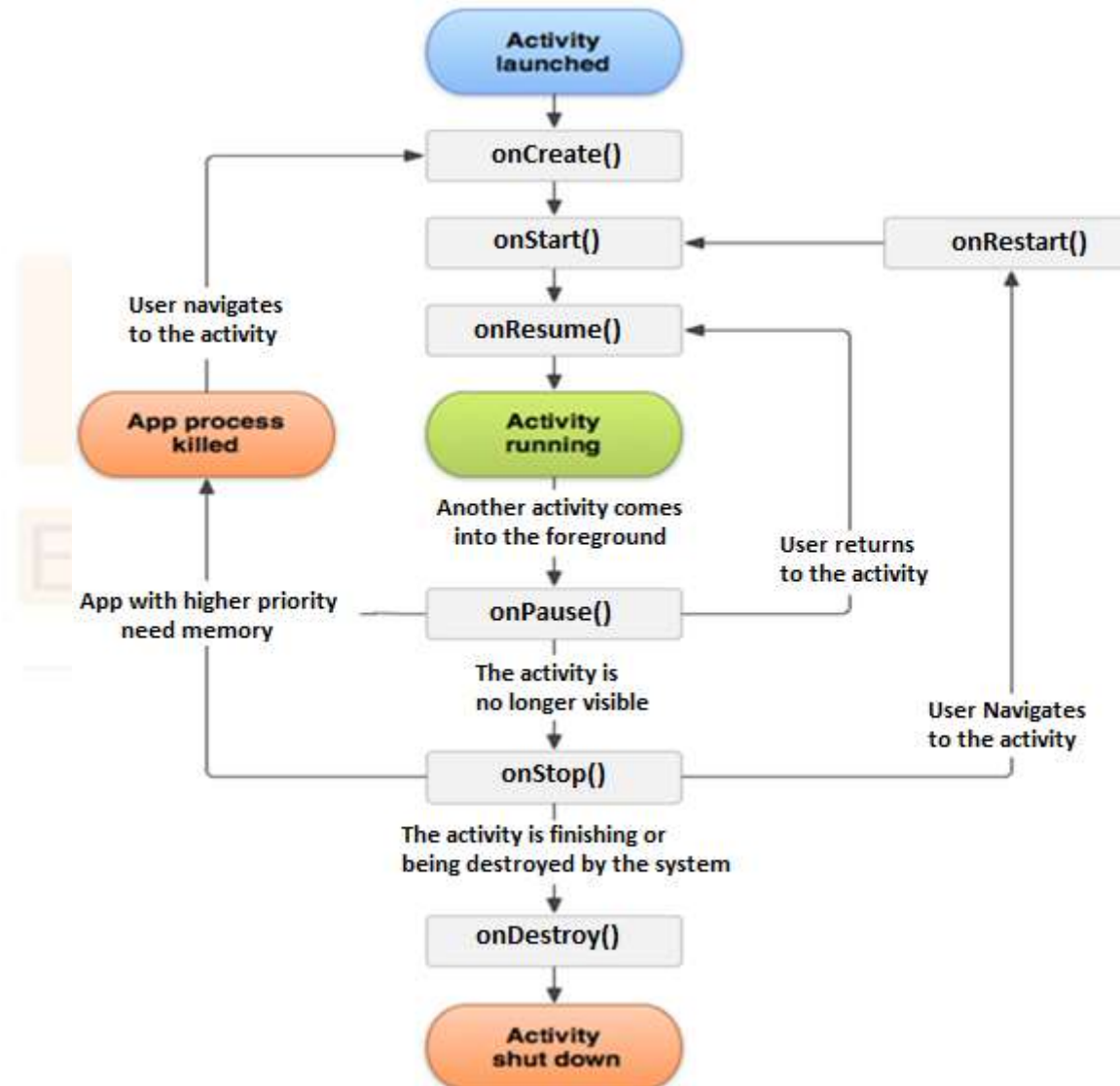
Source Language: Java

Target Source Set: main

The name of the layout to create for the activity

Previous Next Cancel Finish

## Activity life cycle



## Activity life cycle methods

Method	Description
<b>onCreate</b>	called when activity is first created.
<b>onStart</b>	called when activity is becoming visible to the user.
<b>onResume</b>	called when activity will start interacting with the user.
<b>onPause</b>	called when activity is not visible to the user.
<b>onStop</b>	called when activity is no longer visible to the user.
<b>onRestart</b>	called after your activity is stopped, prior to start.
<b>onDestroy</b>	called before the activity is destroyed.

## Activity Lifecycle Situations

- When open the app  
onCreate() --> onStart() --> onResume()
- When back button pressed and exit the app  
onPause() --> onStop() --> onDestroy()
- When home button pressed  
onPaused() --> onStop()
- After pressed home button when again open app from recent task list or clicked on icon  
onRestart() --> onStart() --> onResume()

## (Continued) Activity Lifecycle Situations

- When open app another app from notification bar or open settings  
onPaused() --> onStop()
- Back button pressed from another app or settings then used can see our app  
onRestart() --> onStart() --> onResume()
- When any dialog open on screen  
onPause()
- After dismiss the dialog or back button from dialog  
onResume()

## (Continued) Activity Lifecycle Situations

- Any phone is ringing and user in the app  
onPause() --> onResume()
- When user pressed phone's answer button  
onPause()
- After call end  
onResume()
- When phone screen off  
onPaused() --> onStop()
- Again screen on  
onRestart() --> onStart() --> onResume()



## Activity Example

```
public class MainActivity extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main1);
        Log.d("lifecycle", "onCreate invoked");
    }

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

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

## (Continued) Activity Example

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

## Activity Example Output

onCreate invoked

onStart invoked

onResume invoked

\*if you minimize your application:

onPause invoked

onStop invoked

\*if you reopen the application:

onStart invoked

onResume invoked

\*if you close your application:

onPause invoked

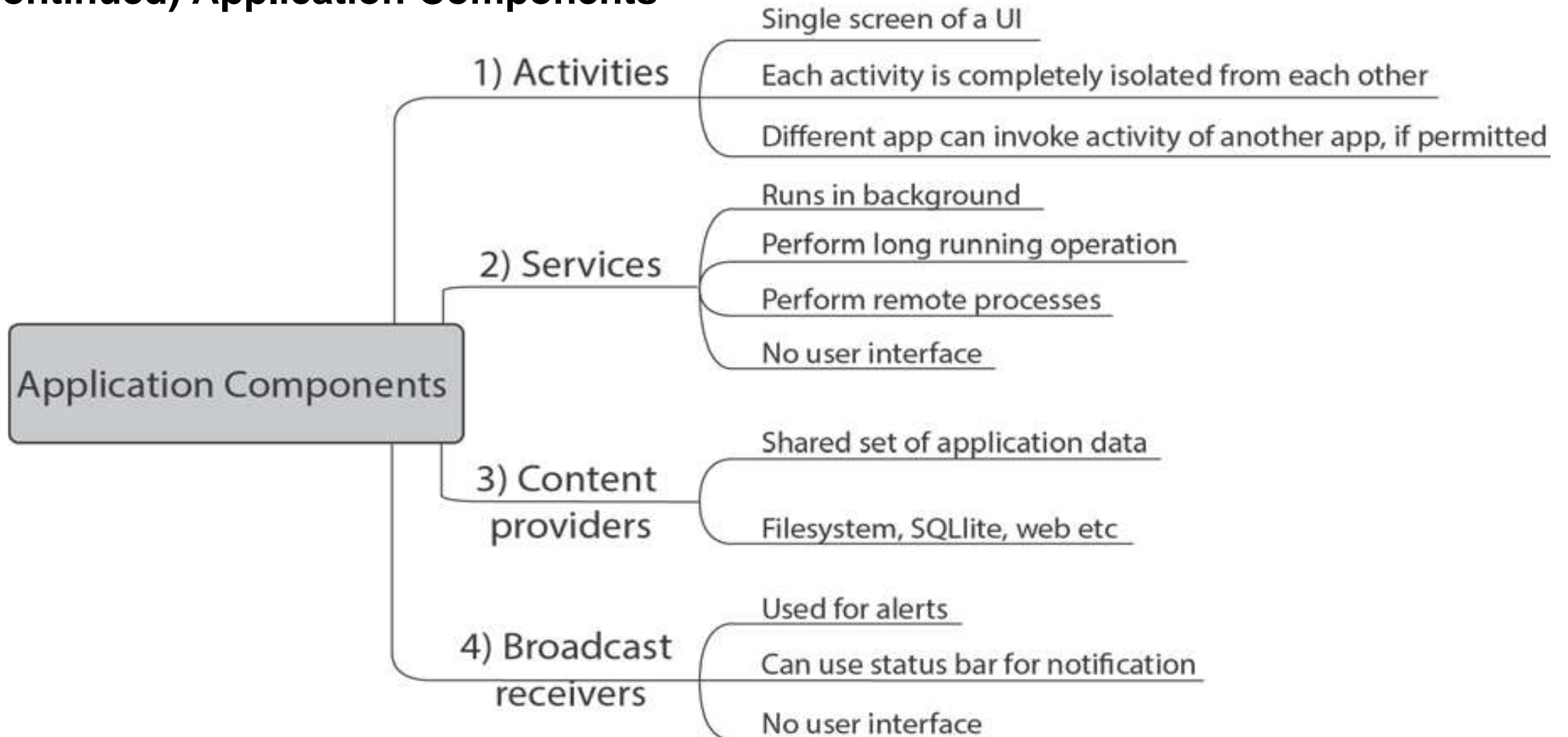
onStop invoked

onDestroy invoked

## Application Components

- **Activity** - Activity is like a frame or window in java that represents GUI. It represents one screen of android.
- **Services** - Service is a Android component that runs in the background. It is used to perform long running operations like playing music, handle network transaction etc.
- **Broadcast Receiver** - Used to communicate between Android OS and application.
- **Content Providers** - Take care of data and database used in the application.
- **Intents** - Are objects which helps in passing values from one activity to another.

## (Continued) Application Components



## Application Components Service

### Service

- Service is one of the Android component which runs in the background without direct interaction with the user .
- A service cannot be viewed by the user as it is not a part of user interface.
- Services are used for long running thread operations, *i.e.*, Internet downloads, data processing, updating content providers etc.
- Service works even if the activity gets destroyed.

### Services are of two forms:

- Started
- Bound

## (Continued) Application Components Service

- **Started Service:**

A service is started when an activity starts by calling `startService()`. Once started, the service runs in the background even if the activity is destroyed. *For example*, `onStart()` is used to start service.

- **Bound Service:**

A service is bound when an activity binds by calling `bindService()`. A bound service offers a client-server interface to bind that activity to interact with the service, send requests, fetch details etc.

*Example:* `onBind()` is used to bind service.



## Application Components Content Providers

### Content Providers

Content Providers is one of the Android component which shares data between the applications.

They encapsulate the data, and provide mechanisms for defining data security. **Content Resolver** class handles the data which are passed. All the data sent from the application are stored temporarily in the content provider and later those data are moved to a database and stored as files etc.

## (Continued) Application Components Content Providers

- Content providers let you centralize content in one place and has many different applications access it, as needed.
- A content provider behaves very much like a database, where you can query it, edit its content, as well as add or delete content using insert(), update(), delete(), and query() methods.
- In most cases this data is stored in an SQLite database.
  - *For example,* Consider the case, when you make all entries in registration page and click the Register button in an application. The data entered are passed to the Content providers and later stored in the database. In MyProfile page the data is been fetched from the database and passed to the content providers which on the other hand is displayed in the app.

## (Continued) Application Components Content Providers

- **INSERT():**

To insert the values to the SQLite Table you have to use **getContentResolver().insert()** method.

- **UPDATE:**

To update the values to the SQLite Table you have to use **getContentResolver().update()** method.

- **DELETE:**

To delete the values to the SQLite Table you have to use **getContentResolver().delete()** method.

- **QUERY:**

Query is used when you need to retrieve only specific list of rows from the database based on your need or requirement. The requirement is passed in arguments and required rows are fetched using cursor. Cursor is like a pointer which is used when more rows are fetched. It helps to identify each rows with their id.

## Application Components Broadcast Receivers

Broadcast Receiver is an Android component which allows you to register application events and respond to the changes happening in the system. There are two ways a Broadcast can be originated.

- **From the system** - a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured.
- **From the application only** - Application can initiate its own broadcast. *Example:* when some data is downloaded from internet, it broadcasts to another application letting it know that data is downloaded and ready for use.
- **Receiver Lifecycle:** A Broadcast Receiver object is only valid for the duration of the call to **onReceive (Context, Intent)**. Once your code returns from this function, the system considers the object to be finished and no longer active.

## (Continued) Application Components Broadcast Receivers

### Syntax for Broadcast Receiver:

The Broadcast Receiver class extends BroadcastReceiver and override onReceive Method. You will write the code under onReceive method() to perform your process once receiving the notification from the System or any broadcast intent.

```
public class MyReceiver extends BroadcastReceiver
{
    public MyReceiver() {
    }
    @Override
    public void onReceive(Context context, Intent intent)
    {
        // This method is called when this BroadcastReceiver receives an Intent broadcast.
    }
}
```

## Application Components Intent

An Intent is basically a message to say what you did or want something to happen. Depending on the intent, apps or the OS might be listening and will react accordingly.

**For example**, you are planning to go out for a movie around 4:00 PM and you want to invite your friend for the movie. You will be sending an SMS to him mentioning that you have booked the tickets and ask him to get ready before the show time. This message is called as Intents in Android term.

## Application Components Defining Intent

Intent defines the intention of sharing some message or information between Android components like activities, content providers, services, broadcast receivers etc. Intents allows you to interact with components from the same applications as well as with components contributed by other applications.

**For example,** An activity can start an external activity for taking a video. Intents are objects of the **android.content.Intent** type. Your code can send them to the Android system defining the components you are targeting. **For example,** via the **startActivity()** method you can define that the intent should be used to start an activity.



## Application Components Defining Intent

### **startActivity():**

To start an activity, use the method **startActivity(intent)**. Activities which are started by other Android activities are called sub-activities. In other words, all the activities started by `startActivity()` are sub-activities.

### **syntax to declare an Intent object and start another activity:**

```
Intent i = new Intent(currentActivity.this, externalActivity.class);  
startActivity(i);
```

`currentActivity.this` in the argument bracket represents the current activity class name  
`externalActivity.class` is the external activity class name which has to be started.

## (Continued) Application Components Defining Intent

### Implicit and Explicit invocation

Intents are of two types:

1. Explicit
2. Implicit



## (Continued) Application Components Defining Intent

### Explicit Intent type

In Explicit Intent type, the target component which we are going to launch is known and declared in the code. This Intent type is mainly used to have interaction with one activity and another activity internally in an application.

**For example**, your application has two activities, Activity A and Activity B. You want to start Activity B from Activity A once the required process is completed. In this case, you will be using explicit Intent type, declaring Activity B in targeting class place.

### Code:

```
Intent i = new Intent(ActivityA.this, ActivityB.class);----->Declaring class name  
startActivity(i); ----->// Starts TargetActivity
```

This code starts processing Activity B class from Activity A class.

## (Continued) Application Components Defining Intent

### Implicit Intent type:

In Implicit Intent type, the target components are not known and target class field is left blank. Implicit type is used when you need to interact with the activity of another application.

**For example**, in Whatsapp you have option to set your profile picture. When you click on it, Gallery gets opened to select picture of your choice. Here the activity of Whatsapp (first application) interacts with Gallery application (second application).

### Code:

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://google.com"));
startActivity(intent);
```

In Implicit type, the action to be performed and data are passed in Intent. The above code is written to open and view the website Google.com from your application. **Intent.ACTION\_VIEW** is the action to be performed and **http://google.com** is the data which is to be viewed. The data is always parsed into uri (uniform resource indicator) format as system identifies easily.

## Resources Files

Resources are the additional files and static content that your code uses, such as bitmaps, layout definitions, user interface strings, animation instructions, and more.

You should always maintain these resources independent from your source code and provide alternative resources to support specific device configurations.

For example, you can create a layout file that is applied to your UI by default, but also include variations of that layout that is applied to specific screen sizes or screen orientations.

## Providing Resources

You should always externalize app resources such as images and strings from your code, so that you can maintain them independently.

You should also provide alternative resources for specific device configurations, by grouping them in specially-named resource directories. At runtime, Android uses the appropriate resource based on the current configuration.

For example, you might want to provide a different UI layout depending on the screen size or different strings depending on the language setting.

Once you externalize your app resources, you can access them by using resource IDs that are generated in your project's `R.class`.

## (Continued) Providing Resources

You should place each type of resource in a specific subdirectory of your project's **res/** directory. For example, here's the file hierarchy for a simple project:

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

As you can see in this example, the **res/** directory contains all the resources (in subdirectories) such as, an image resource, two layout resources, **mipmap/** directories for launcher icons, and a string resource file.



# Introduction to Android and Android Studio

## Resource directories

Directory	Resource Type
<b>animator/</b>	XML files that define property animations.
<b>anim/</b>	XML files that define tween animations. (Property animations can also be saved in this directory, but the animator/ directory is preferred for property animations to distinguish between the two types.)
<b>color/</b>	XML files that define a state list of colors. See Color State List Resource
<b>drawable/</b>	Bitmap files (.png, .9.png, .jpg, .gif) or XML files that are compiled into the following drawable resource subtypes: <ul style="list-style-type: none"><li>•Bitmap files</li><li>•Nine-Patches (re-sizable bitmaps)</li><li>•State lists</li><li>•Shapes</li><li>•Animation drawables</li><li>•Other drawables</li></ul> See Drawable Resources.
<b>mipmap/</b>	Drawable files for different launcher icon densities. For more information on managing launcher icons with mipmap/ folders, see Managing Projects Overview.
<b>layout/</b>	XML files that define a user interface layout. See Layout Resource.

## (Continued) Resource directories

Directory	Resource Type
values/	<p>XML files that contain simple values, such as strings, integers, and colors. Whereas XML resource files in other res/ subdirectories define a single resource based on the XML filename, files in the values/ directory describe multiple resources. For a file in this directory, each child of the &lt;resources&gt; element defines a single resource. For example, a &lt;string&gt; element creates an R.string resource and a &lt;color&gt; element creates an R.color resource.</p> <p>Because each resource is defined with its own XML element, you can name the file whatever you want and place different resource types in one file. However, for clarity, you might want to place unique resource types in different files. For example, here are some filename conventions for resources you can create in this directory:</p> <ul style="list-style-type: none"><li>•arrays.xml for resource arrays (typed arrays).</li><li>•colors.xml for color values</li><li>•dimens.xml for dimension values.</li><li>•strings.xml for string values.</li><li>•styles.xml for styles.</li></ul> <p>See String Resources, Style Resource, and More Resource Types.</p>

## (Continued) Resource directories

Directory	Resource Type
<b>menu/</b>	<b>XML files that define app menus, such as an Options Menu, Context Menu, or Sub Menu. See Menu Resource.</b>
<b>raw/</b>	<b>Arbitrary files to save in their raw form. To open these resources with a raw InputStream, call <code>Resources.openRawResource()</code> with the resource ID, which is <code>R.raw.filename</code>. However, if you need access to original file names and file hierarchy, you might consider saving some resources in the <code>assets/</code> directory (instead of <code>res/raw/</code>). Files in <code>assets/</code> aren't given a resource ID, so you can read them only using <code>AssetManager</code>.</b>
<b>xml/</b>	<b>Arbitrary XML files that can be read at runtime by calling <code>Resources.getXML()</code>. Various XML configuration files must be saved here, such as a searchable configuration.</b>
<b>font/</b>	<b>Font files with extensions such as <code>.ttf</code>, <code>.otf</code>, or <code>.ttc</code>, or XML files that include a <code>&lt;font-family&gt;</code> element. For more information about fonts as resources, go to <a href="#">Fonts in XML</a>.</b>

## Accessing Resources

Once you provide a resource in your application you can apply it by referring its resource ID. All resource IDs are defined in your project's **R** class, which the **aapt** tool automatically generates.

When your application is compiled, **aapt** generates the **R class**, which contains resource IDs for all the resources in your **res/** directory.

For each type of resource, there is an **R** subclass (for example, **R.drawable** for all drawable resources), and for each resource of that type, there is a static integer (for example, **R.drawable.icon**). This integer is the resource ID that you can use to retrieve your resource.

## (Continued) Accessing Resources

Although the R class is where resource IDs are specified, you should never need to look there to discover a resource ID. A resource ID is always composed of:

**Resource type:** Each resource is grouped into a "type," such as string, drawable, and layout.

**Resource name:** Which is either the filename, excluding the extension; or the value in the XML android:name attribute, if the resource is a simple value (such as a string).

## Two ways to access a resource

There are two ways you can access a resource:

- In code:** Using a static integer from a sub-class of your R class, such as: `R.string.hello` string is the resource type and hello is the resource name. There are many Android APIs that can access your resources when you provide a resource ID in this format.
- In XML:** Using a special XML syntax that also corresponds to the resource ID defined in your R class, such as: `@string/hello` string is the resource type and hello is the resource name. You can use this syntax in an XML resource any place where a value is expected that you provide in a resource.

## Accessing Resources in Code

You can use a resource in code by passing the resource ID as a method parameter. For example, you can set an `ImageView` to use the **res/drawable/myimage.png** resource using **`setImageResource()`**:

```
ImageView imageView = (ImageView) findViewById(R.id.myimageview);  
imageView.setImageResource(R.drawable.myimage);
```

You can also retrieve individual resources using methods in `Resources`, which you can get an instance of with **`getResources()`**.



## Accessing Resources Syntax

Here's the syntax to reference a resource in code:

**[<package\_name>.]R.<resource\_type>.<resource\_name>**

- **<package\_name>** is the name of the package in which the resource is located (not required when referring resources from your own package).
- **<resource\_type>** is the R subclass for the resource type.
- **<resource\_name>** is either the resource filename without the extension or the android:name attribute value in the XML element (for simple values).

## Accessing Resources from XML

You can define values for some XML attributes and elements using a reference to an existing resource. You will often do this when creating layout files, to supply strings and images for your widgets.

For example, if you add a **Button** to your layout, you should use a string resource for the button text:

```
<Button  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="@string/submit" />
```

# Self Assessment Questions

---

1. Who is known as the 'Father of Android'?

- a. Andy Rubin
- b. Larry Page
- c. Sundar Pitchai
- d. Brian page

Answer: **Andy Rubin**

# Self Assessment Questions

2. Which one of the given options is the bottom most or last located layer in Android Architecture?

- a. Java API Framework
- b. Linex Kernel
- c. Hardware Abstraction Layer(HAL)
- d. Dalvik Virtual Machine(DVM)

Answer: **Linex Kernel**

# Self Assessment Questions

3. Which one of the given options is used to interpret .dex files?

- a. Java Virtual Machine (JVM)
- b. Jack
- c. Dalvik Virtual machine (DVM)
- d. File Interpreter

Answer: **Dalvik Virtual machine (DVM)**

# Self Assessment Questions

---

4. \_\_\_\_\_ is the first android device launched in \_\_\_\_\_.

- a. HTC dream, 2008
- b. HTC hero, 2008
- c. HTC desire, 2008
- d. HTC fire, 2008

Answer: **HTC dream, 2008**

# Self Assessment Questions

---

5. AVD is also known as \_\_\_\_\_.

- a. Android Virtual Debug
- b. Android Versatile Device
- c. Android Virtual Device
- d. Android Vender Debug

Answer: **Android Virtual Device**



# Self Assessment Questions

---

6. Android is based on \_\_\_\_\_ OS.

- a. Linux kernel
- b. Window NT
- c. Zen os
- d. Digital camera OS

Answer: **Linux Kernel**

# Self Assessment Questions

---

7. ART is also known as \_\_\_\_\_.

- a. Android Real Time
- b. Android Run Time
- c. Android Rate Time
- d. Android Run Test

Answer: **Android Run Time**

# Self Assessment Questions

---

8. SDK is also known as \_\_\_\_\_.

- a. SDK - Software Development Kit
- b. SDK - Software Debug Kit
- c. SDK - Software Device Kit
- d. SDK - System Development Kit

Answer: **SDK - Software Development Kit**

# Self Assessment Questions

---

9. Which one of the given Android component deals with UI (User Interface) and user interaction?
- a. Content Provider
  - b. Intents
  - c. Activity
  - d. Broadcast Receiver

Answer: **Activity**

# Self Assessment Questions

10. Which one of the given Android component takes care of data and database used in the application?

- a. Services
- b. Intents
- c. Activity
- d. Content Provider

Answer: **Content Provider**

# Self Assessment Questions

11. Which one of the given Android component helps in passing values from one activity to another?

- a. Services
- b. Intents
- c. Activity
- d. Broadcast Receiver

Answer: **Intents**

# Self Assessment Questions

---

12. Which one of the given options is used to start the life cycle of an Android?

- a. onDestroy
- b. onCreate
- c. onPause
- d. onRestart

Answer: **onCreate.**



# Self Assessment Questions

13. The user leaves an activity and opens another activity. This method is called as \_\_\_\_\_.

- a. onDestroy
- b. onCreate
- c. onPause
- d. onRestart

Answer: **onPause.**

# Self Assessment Questions

14. Which one of the given options contains information of your package, including components of the application such as activities, services, broadcast receivers, content providers etc?

- a. Android Manifest File
- b. Android component file
- c. Activity File
- d. Broadcast Receiver

Answer: **Android Manifest File**

# Self Assessment Questions

15. \_\_\_\_\_ restriction that limits access to a part of the code or to data on the device.

- a. Restriction
- b. Intents
- c. Activity
- d. Permission

Answer: **Permission**

# Self Assessment Questions

16. Manifest XML starts with the \_\_\_\_\_ keyword.

- a. <!Manifest!>
- b. <Manifest>
- c. !Manifest
- d. ManifestXML

Answer: **<Manifest>**

# Self Assessment Questions

17. Which one of the given options are correct?

The types of intents are:

- i. Explicit
  - ii. Implicit
- 
- a. Only i
  - b. Only ii
  - c. Both a & b

Answer: **Both a & b**

# Self Assessment Questions

18. Which one of the given Android component runs in the background without direct interaction with the user?

- a. Activity
- b. Filter
- c. Services
- d. Intent

Answer: **Services**

# Self Assessment Questions

19. The \_\_\_\_\_ belongs to Content Provider which handles the data that are passed between application.

- a. Content Retriever
- b. Content Resolver
- c. Content Filter
- d. Content Explorer

Answer: **Content Resolver**



# Self Assessment Questions

20. Which one of the given Android component allows you to register application events and respond to the changes happening in the system?

- a. Services
- b. Content Provider
- c. Activity
- d. Broadcast Receiver

Answer: **Broadcast Receiver**

# Assignment

## General Instructions:

*Please answer the below set of questions. These set of questions are meant for testing unit 1.*

- *The answers should be clear, legible and well presented.*
  - *Illustrate your answers with suitable examples wherever necessary.*
  - *Please quote sources (if any) of data, images, facts etc.*
- 
1. What is Android?
  2. What are the versions released so far in Android?
  3. Explain about Android architecture and various blocks in it..
  4. What are the key components of Android?
  5. What is an Activity in Android?

# Assignment

---

7. What is an Intent in Android and explain about its types?
8. Explain the life cycle of Activity?
9. Explain about the Android Services?
10. Explain about the Content Providers?
11. Explain how Broadcast Intent and Receiver are used?

## Summary

- ✓ Classify the Android Architecture.
- ✓ Classify the Versions/Evolution, Dalvik VM,
- ✓ Basic Android Application, Application Folder Structure.
- ✓ Classify the Manifest file, R.java file, Activity, Activity life cycle.
- ✓ Application Components, Resource Files.

# Introduction to Android and Android Studio

## Document Links

Topics	URL	Notes
Introduction, History, Features, Android Architecture, Versions/Evolution, Dalvik VM, Installing Android Studio (latest Version), Android Studio Environment, First Android Application, Application Folder Structure, Manifest file, R.java file, Activity, Activity life cycle, Application Components, Resource Files	<a href="https://www.tutorialspoint.com/android/index.htm">https://www.tutorialspoint.com/android/index.htm</a> <a href="https://developer.android.com/index.html">https://developer.android.com/index.html</a>	This link explains About all concepts of Introduction, History, Features, Android Architecture, Versions/Evolution, Dalvik VM, Installing Android Studio (latest Version), Android Studio Environment, First Android Application, Application Folder Structure, Manifest file, R.java file, Activity, Activity life cycle, Application Components, Resource Files

# Introduction to Android and Android Studio

## Video Links

Topics	URL	Notes
Introduction, History, Features, Android Architecture, Versions/Evolution, Dalvik VM, Installing Android Studio (latest Version), Android Studio Environment, First Android Application, Application Folder Structure, Manifest file, R.java file, Activity, Activity life cycle, Application Components, Resource Files	<a href="https://www.tutorialspoint.com/android_online_training/index.asp">https://www.tutorialspoint.com/android_online_training/index.asp</a>	This link explains About all concepts of Introduction, History, Features, Android Architecture, Versions/Evolution, Dalvik VM, Installing Android Studio (latest Version), Android Studio Environment, First Android Application, Application Folder Structure, Manifest file, R.java file, Activity, Activity life cycle, Application Components, Resource Files

# Introduction to Android and Android Studio

## E-Book Links

Topics	URL	Page Number
Introduction, History, Features, Android Architecture, Versions/Evolution, Dalvik VM, Installing Android Studio (latest Version), Android Studio Environment, First Android Application, Application Folder Structure, Manifest file, R.java file, Activity, Activity life cycle, Application Components, Resource Files	<a href="http://yuliana.lecturer.pens.ac.id/Android/Buku/professional_android_4_application_development.pdf">http://yuliana.lecturer.pens.ac.id/Android/Buku/professional_android_4_application_development.pdf</a>	Page Number 1 to 53
	<a href="https://www.manning.com/books/android-in-action-third-edition">https://www.manning.com/books/android-in-action-third-edition</a>	Chapter 1, Chapter 2
	<a href="ftp://ftp.micronet-rostov.ru/linux-support/books/programming/Mobile-Apps/[Wiley.%20Wrox]%20-%20Beginning%20Android%20Application%20Development%20-%20[Wei-Meng%20Lee].pdf">ftp://ftp.micronet-rostov.ru/linux-support/books/programming/Mobile-Apps/[Wiley.%20Wrox]%20-%20Beginning%20Android%20Application%20Development%20-%20[Wei-Meng%20Lee].pdf</a>	Page Number 1 to 27