Lab 1
Installation of Android Studio, running first application and Understanding project folder structure

1. Installation

Initially, to set up an Android development environment in our system we need to install the following components manually by downloading from different sites.

1. Eclipse IDE
2. Eclipse Plugin
3. Android SDK

To make the android development environment setup process simple Google introduced a new android IDE called **Android Studio**. The **Android Studio** will contain all the required components like Eclipse IDE, Eclipse Plugin and Android SDK so we don’t need to download the components separately.

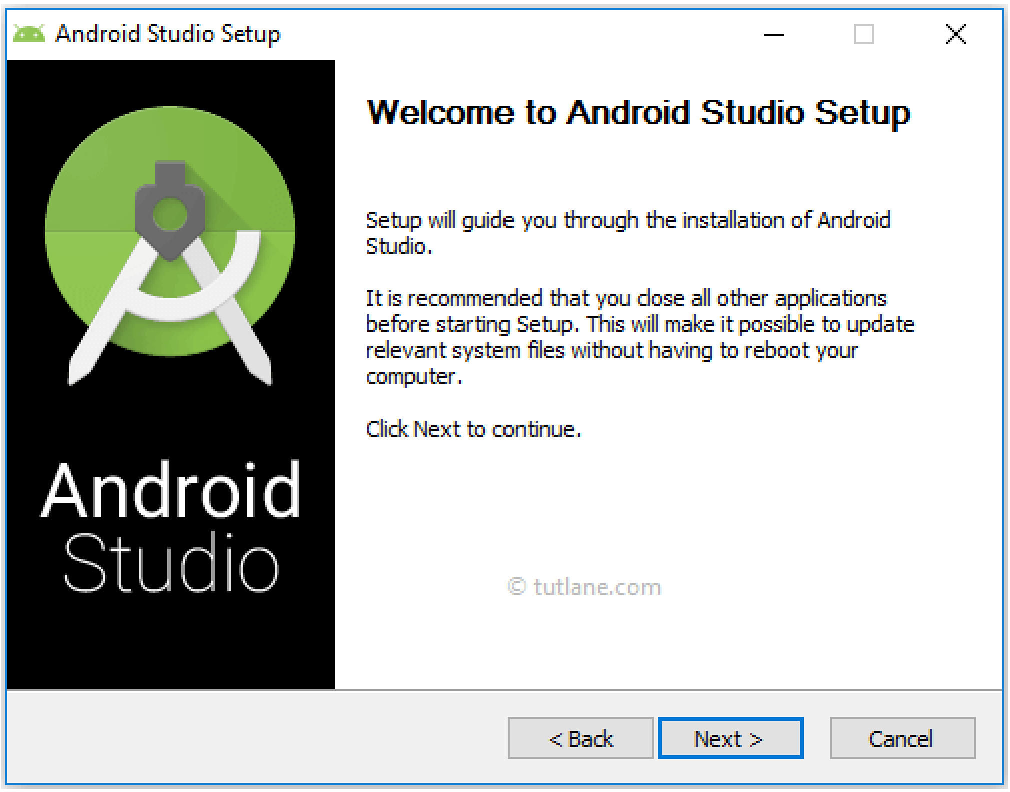
Android Studio is the official IDE for android development and it’s based on **IntelliJ IDEA** software. It’s available for Windows, MAC and LINUX operating systems.

We can download the latest version of Android Studio from the following URL.

https://developer.android.com/studio

Download the latest version of Android Studio from the above URL and launch **Android Studio.exe** file by double-clicking on it.

The initial android studio setup screen will open like as shown below in that click **Next** to continue for further steps of environment setup.


Android Studio Setup

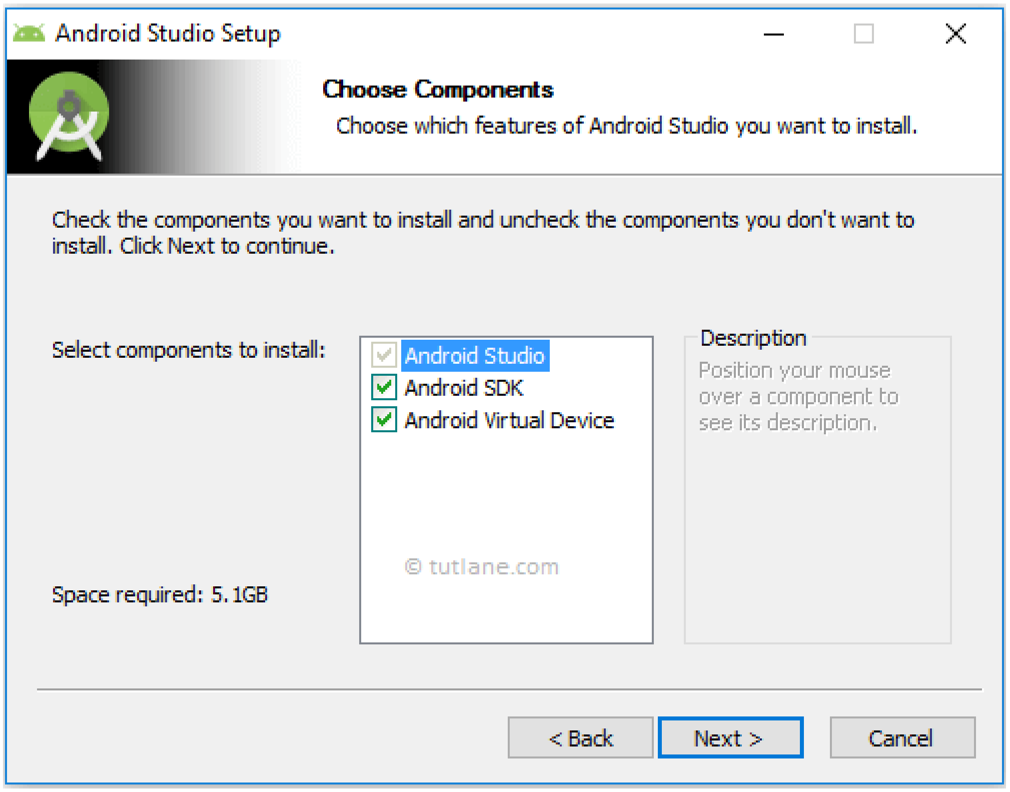
Welcome to Android Studio Setup

Setup will guide you through the installation of Android Studio.

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

Click Next to continue.

Now we need to select the required components to set up an android environment. Here we selected all three components (**Android Studio**, **Android SDK** and **Android Virtual Device**) and click **Next** like as shown below.


Android Studio Setup

Choose Components

Choose which features of Android Studio you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

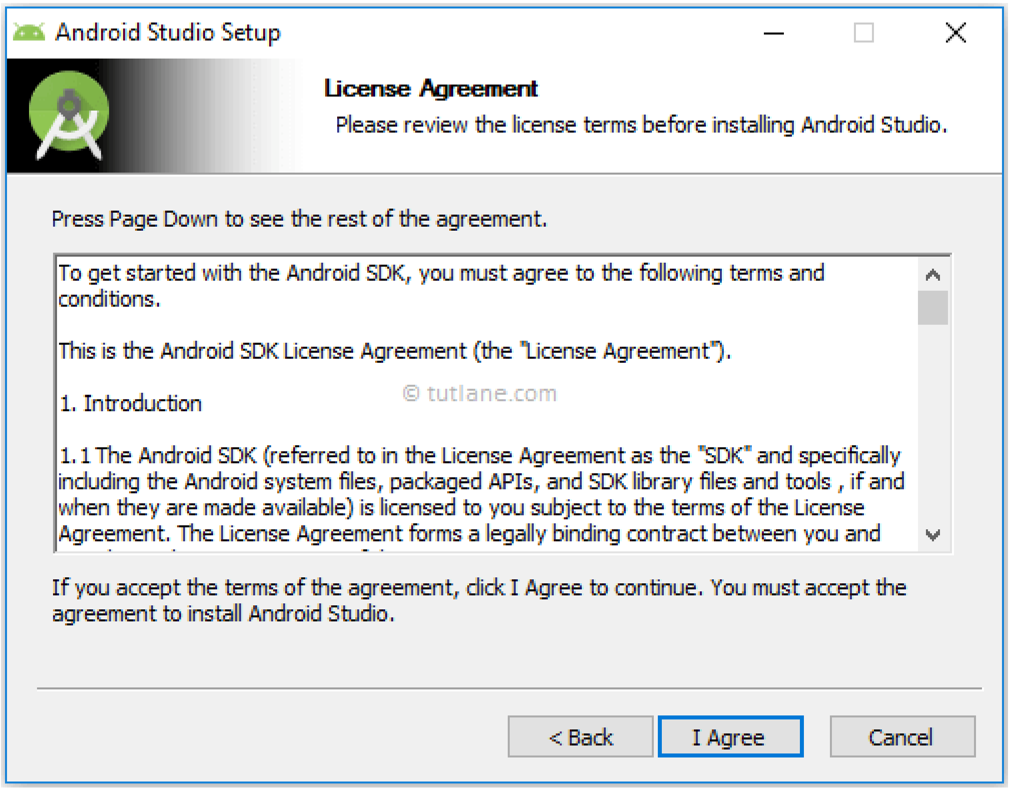
[✓] Android Studio
[✓] Android SDK
[✓] Android Virtual Device

Description

Position your mouse over a component to see its description.

Space required: 5.1GB

Now we need to agree on the License agreements to proceed further, click on **I Agree** button like a shown below.


Android Studio Setup
License Agreement
Please review the license terms before installing Android Studio.

Press Page Down to see the rest of the agreement.

To get started with the Android SDK, you must agree to the following terms and conditions.

This is the Android SDK License Agreement (the "License Agreement").

1. Introduction

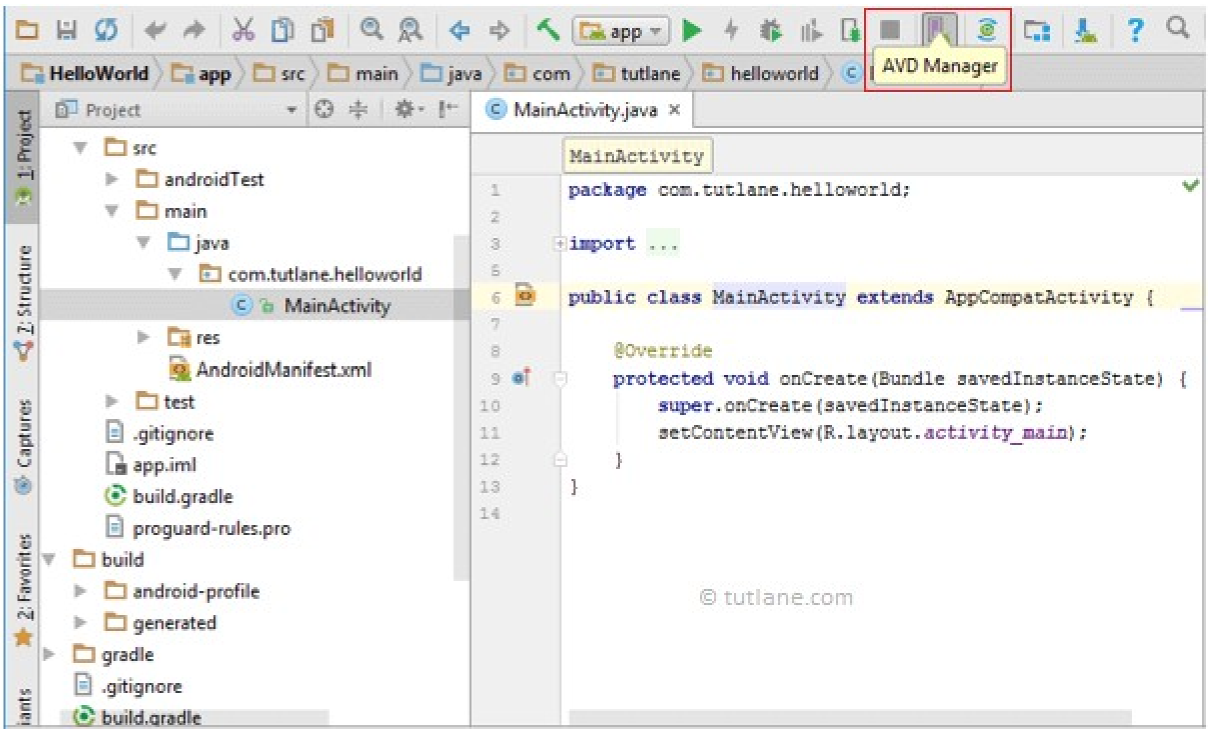
1.1 The Android SDK (referred to in the License Agreement as the "SDK" and specifically including the Android system files, packaged APIs, and SDK library files and tools , if and when they are made available) is licensed to you subject to the terms of the License Agreement. The License Agreement forms a legally binding contract between you and

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Android Studio.

Now we need to specify the local machine drive location to install Android Studio and Android SDK. After selecting the location path to install the required components, click **Next** like as shown below.

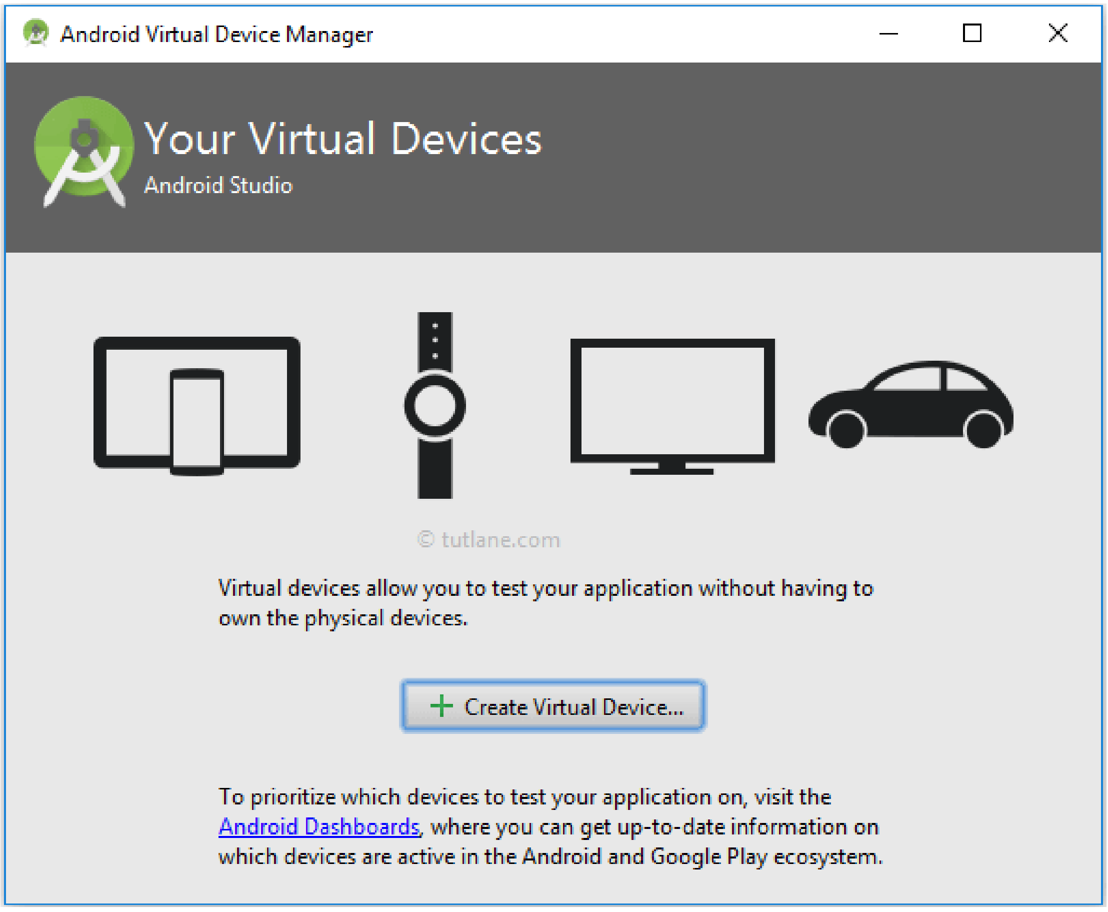
2. Create Android Virtual Device

To test our android application we should have an Android Virtual Device (AVD). We can create virtual device by click on **AVD Manager** like as shown below.



```
package com.tutlane.helloworld;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}

```

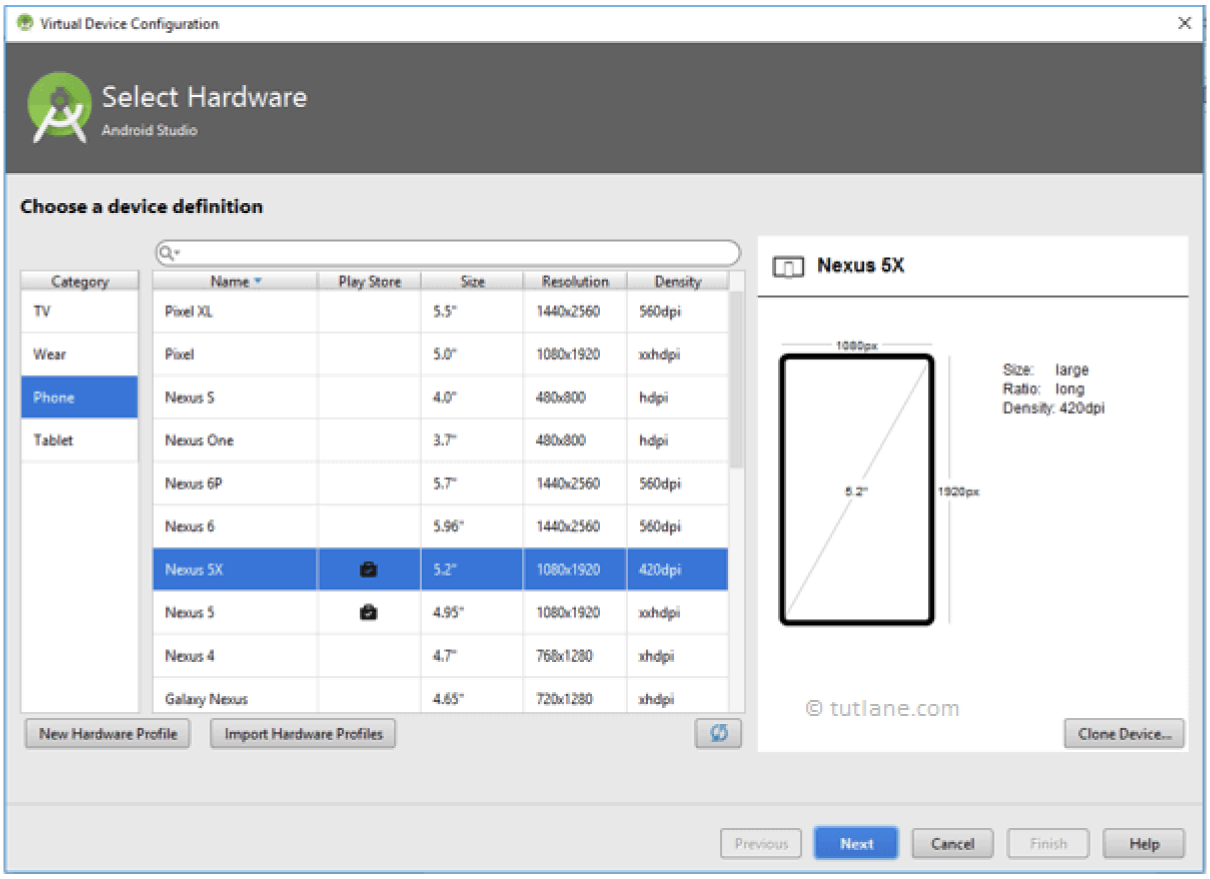
When we click on **AVD Manager**, a new window will open in that click on **Create Virtual Device** like as shown below.



Virtual devices allow you to test your application without having to own the physical devices.

To prioritize which devices to test your application on, visit the Android Dashboards, where you can get up-to-date information on which devices are active in the Android and Google Play ecosystem.

Now select the required device type and Click **Next** to create a virtual device like as shown below.

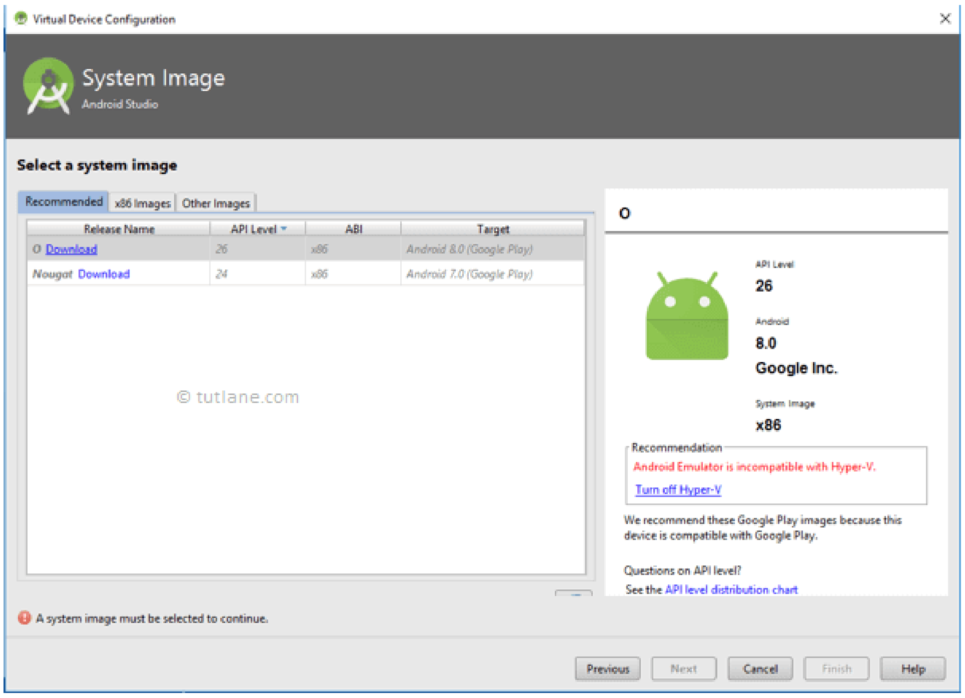


Category
TV
Wear
Phone
Tablet

Name	Play Store	Size	Resolution	Density
Pixel XL		5.5"	1440x2560	560dpi
Pixel		5.0"	1080x1920	xxhdpi
Nexus S		4.0"	480x800	hdpi
Nexus One		3.7"	480x800	hdpi
Nexus 6P		5.7"	1440x2560	560dpi
Nexus 6		5.96"	1440x2560	560dpi
Nexus 5X	✓	5.2"	1080x1920	420dpi
Nexus 5	✓	4.95"	1080x1920	xxhdpi
Nexus 4		4.7"	768x1280	xhdpi
Galaxy Nexus		4.65"	720x1280	xhdpi

Size: large
Ratio: long
Density: 420dpi

Now we need to download and select the system image and click **Next** like as shown below.



Release Name	API Level	ABI	Target
O Download	26	x86	Android 8.0 (Google Play)
Nougat Download	24	x86	Android 7.0 (Google Play)

API Level
26
Android
8.0
Google Inc.
System Image
x86

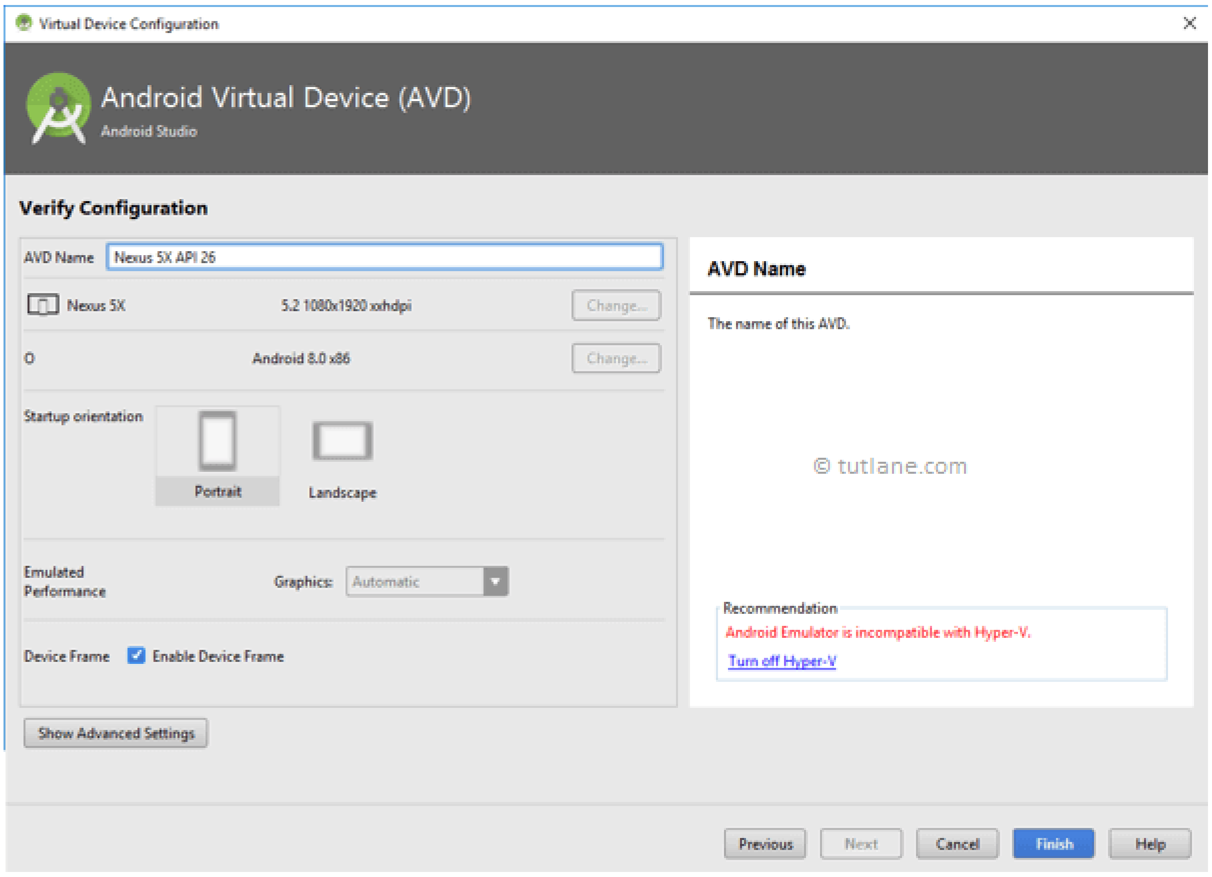
Recommendation
Android Emulator is incompatible with Hyper-V.
Turn off Hyper-V

We recommend these Google Play images because this device is compatible with Google Play.

Questions on API level?
See the API level distribution chart

A system image must be selected to continue.

Now verify the configuration of **android virtual device** (AVD) and click **Finish** like as shown below.


Virtual Device Configuration

Android Virtual Device (AVD)
Android Studio

Verify Configuration

AVD Name: Nexus 5X API 26

Nexus 5X 5.2 1080x1920 xxhdpi Change...

O Android 8.0 x86 Change...

Startup orientation: Portrait Landscape

Emulated Performance Graphics: Automatic

Device Frame ✓ Enable Device Frame

Show Advanced Settings

AVD Name

The name of this AVD.

© tutlane.com

Recommendation
Android Emulator is incompatible with Hyper-V.
Turn off Hyper-V

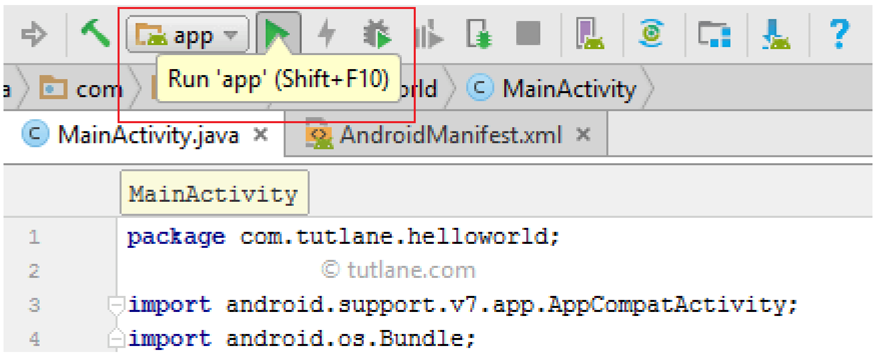
Previous Next Cancel Finish Help

This is how we need to add android virtual device (AVD) in an android studio to test our android applications.

Once we are done with the setup of android virtual device in an android studio, create a sample application in the android studio and run the app using AVD manager.

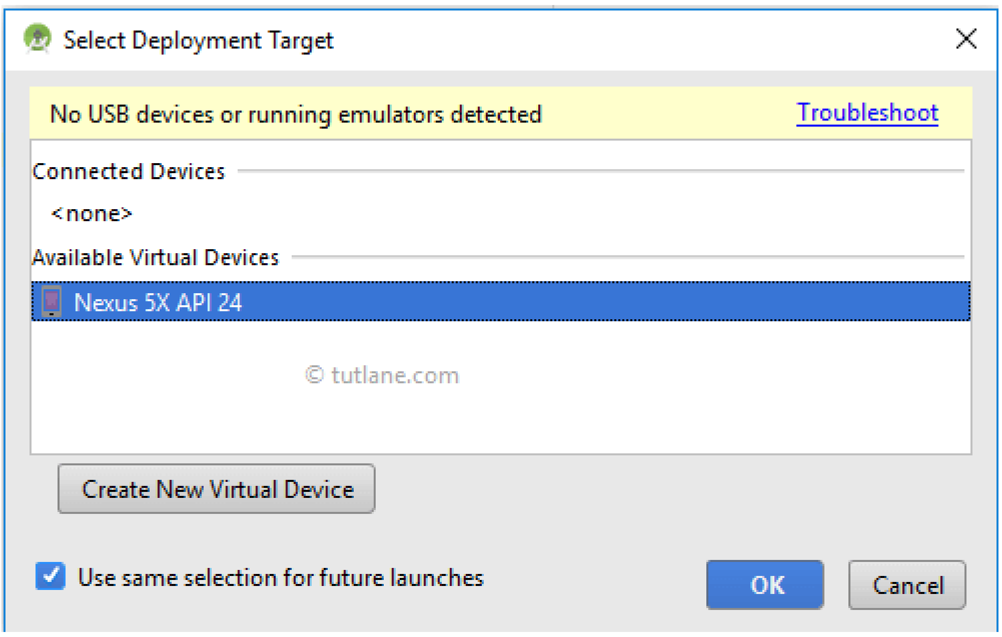
Run Android Application

To run android applications we need to click on **Run** button or press `Shift` + `F10` like as shown below



```
package com.tutlane.helloworld;

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

After clicking on play button new window will open in that select **Android Virtual Device** (AVD) and click **OK** like as shown below.



Select Deployment Target

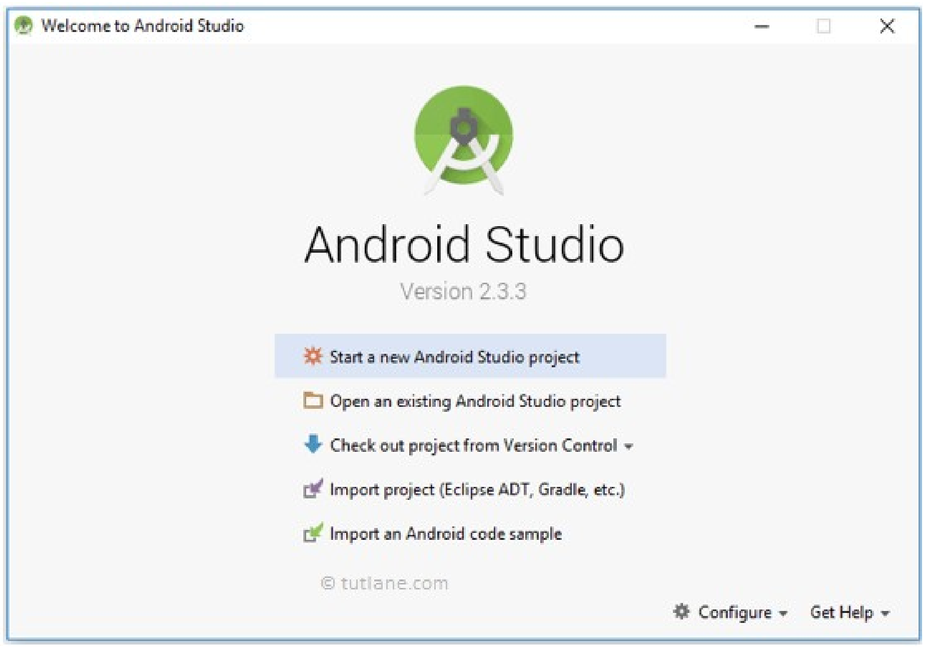
No USB devices or running emulators detected

Troubleshoot
Connected Devices
<none>
Available Virtual Devices
Nexus 5X API 24
© tutlane.com
Create New Virtual Device
Use same selection for future launches
OK
Cancel

This is how we can setup an android virtual device (AVD) emulator in android studio to replicate the functionality of real android devices.

3. Running First Application

Once we are done with Android Studio Installation, open android studio and that will be like as shown below.


Welcome to Android Studio
Android Studio
Version 2.3.3
Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Import project (Eclipse ADT, Gradle, etc.)
Import an Android code sample
© tutlane.com
Configure
Get Help

Here we're going to choose the **New Project** option because we haven't created any other project and we need to create a new one. So, we will select the New Project from the given options.

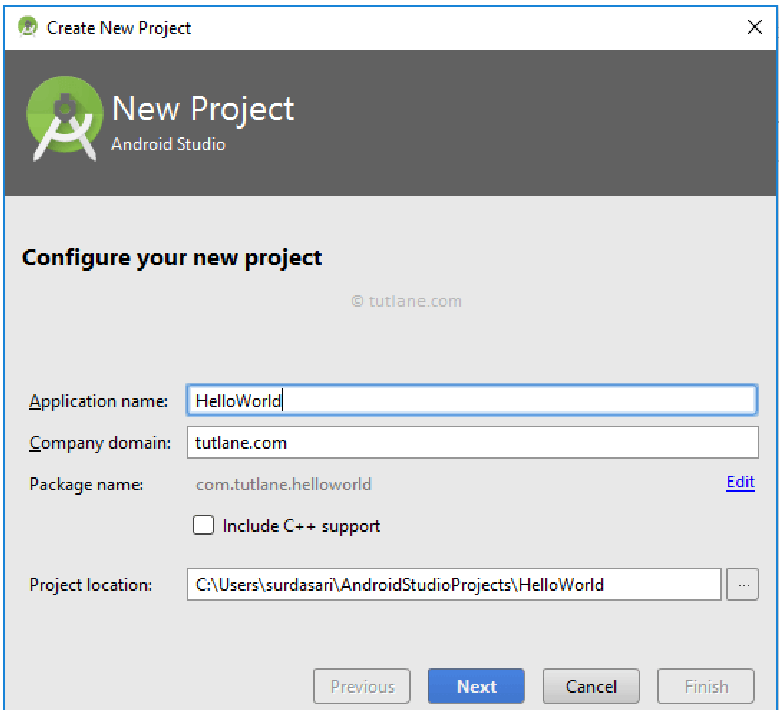
However, we can choose **Import Project** if we'd like to import a project from any other way, for example, Eclipse project into Android Studio. Android Studio will convert the Eclipse project to an Android Studio project, adding the necessary configuration files for us.

If we select **Open Project** from the list of options, we can open projects created with either Android Studio or IntelliJ IDEA.

Check out from Version Control, we can check out a copy of a project that's under version control. This is a great way to quickly get up to speed with an existing project.

To get us started, choose **New Project** from the list of options. This will show us a list of options to configure our new project.

As we click on "**New Project**" from the above option, then the next screen will be open like this, where we have to mention our **Project's name**, **Company domain** and **Project location** (we called it the main path where this application will be saved) because the **Package name** will be created automatically as we create the project in Android Studio.



Configure your new project

Application name: HelloWorld

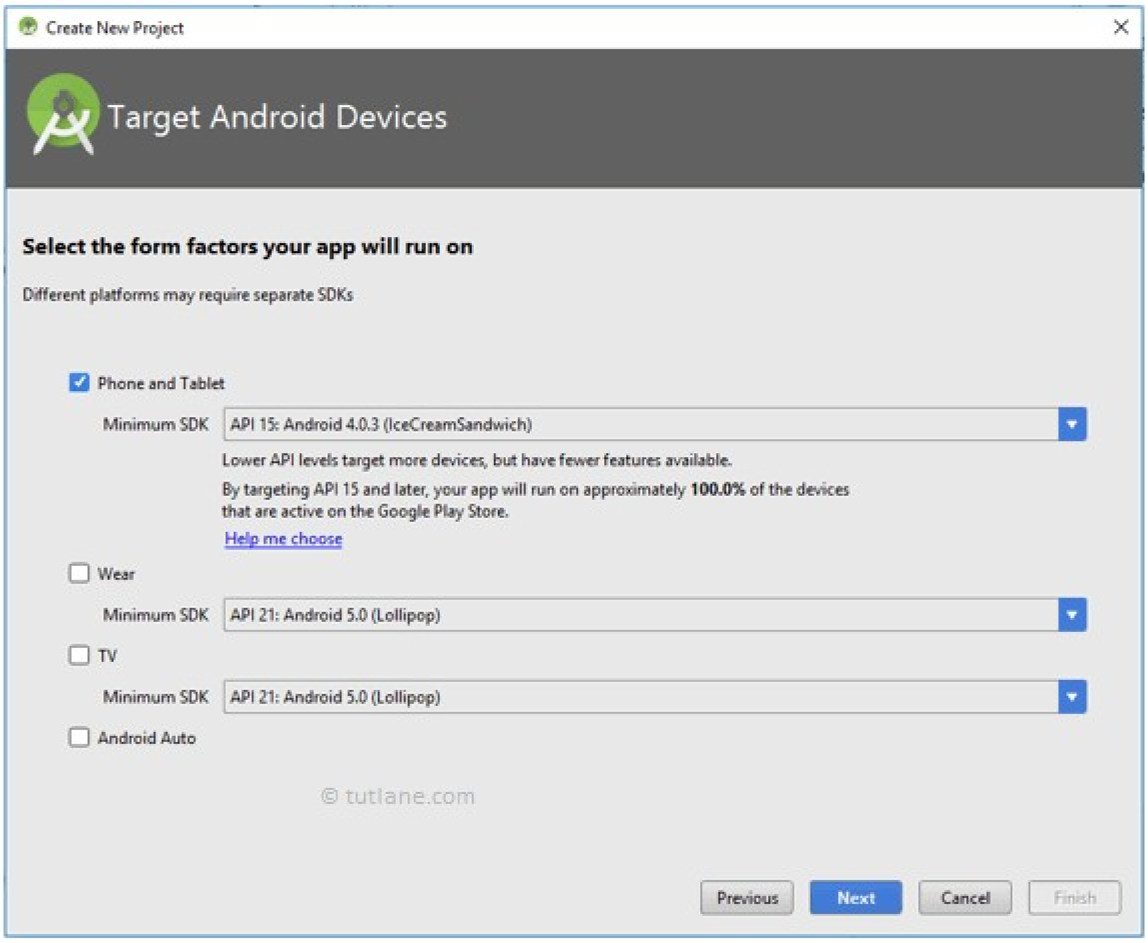
Company domain: tutlane.com

Package name: com.tutlane.helloworld Edit

☐ Include C++ support

Project location: C:\Users\surdasari\AndroidStudioProjects\HelloWorld

After entering all the details if we click on the “**Next**” button another screen will appear where we have select the different platforms and SDK targets like as shown below based on our requirements.



Select the form factors your app will run on

Different platforms may require separate SDKs

☑ Phone and Tablet

Minimum SDK: API 15: Android 4.0.3 (IceCreamSandwich)

Lower API levels target more devices, but have fewer features available.

By targeting API 15 and later, your app will run on approximately **100.0%** of the devices that are active on the Google Play Store.

Help me choose

☐ Wear

Minimum SDK: API 21: Android 5.0 (Lollipop)

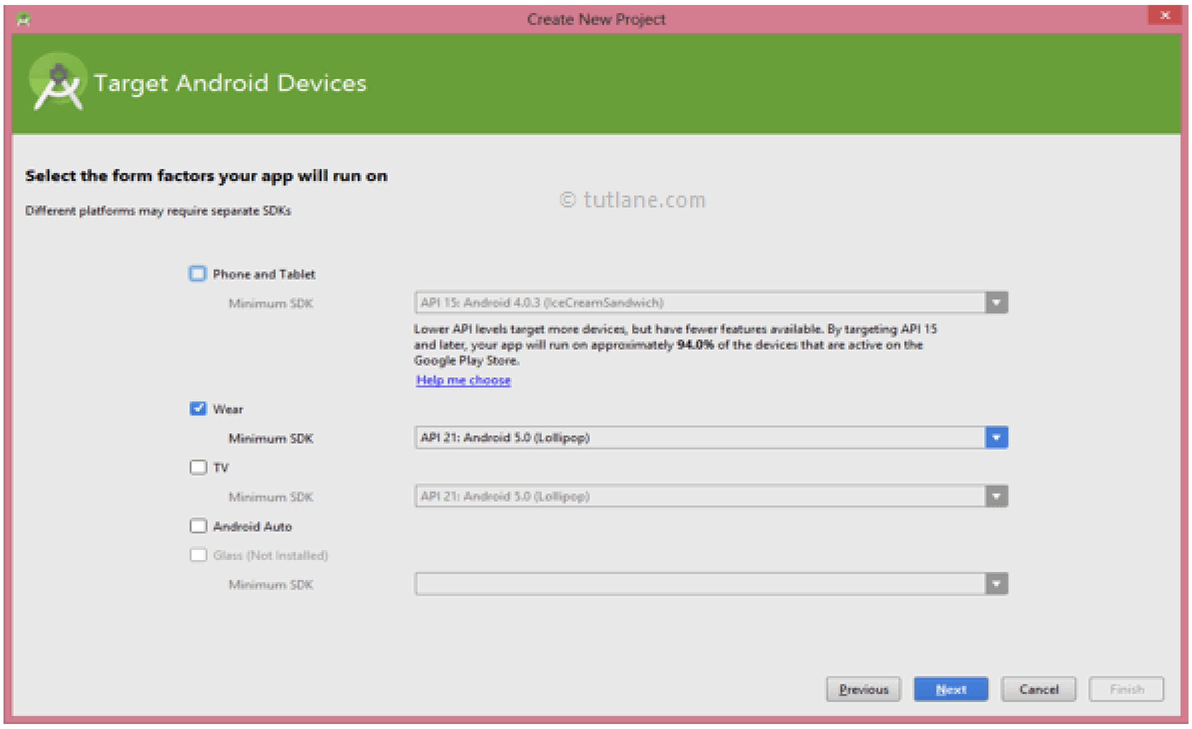
☐ TV

Minimum SDK: API 21: Android 5.0 (Lollipop)

☐ Android Auto

Here we need to select the type of Platform which we are going to use for the Application development like if we select "**Phone and Tablet**", then it will show it's different **API** and **SDK** version and similar to others.

If we choose "**Wear**", then it will show it's API and SDK versions like as shown below.


Create New Project

Target Android Devices

Select the form factors your app will run on

Different platforms may require separate SDKs

© tutlane.com

☐ Phone and Tablet
Minimum SDK: API 15: Android 4.0.3 (IceCreamSandwich)

Lower API levels target more devices, but have fewer features available. By targeting API 15 and later, your app will run on approximately **94.0%** of the devices that are active on the Google Play Store.

Help me choose

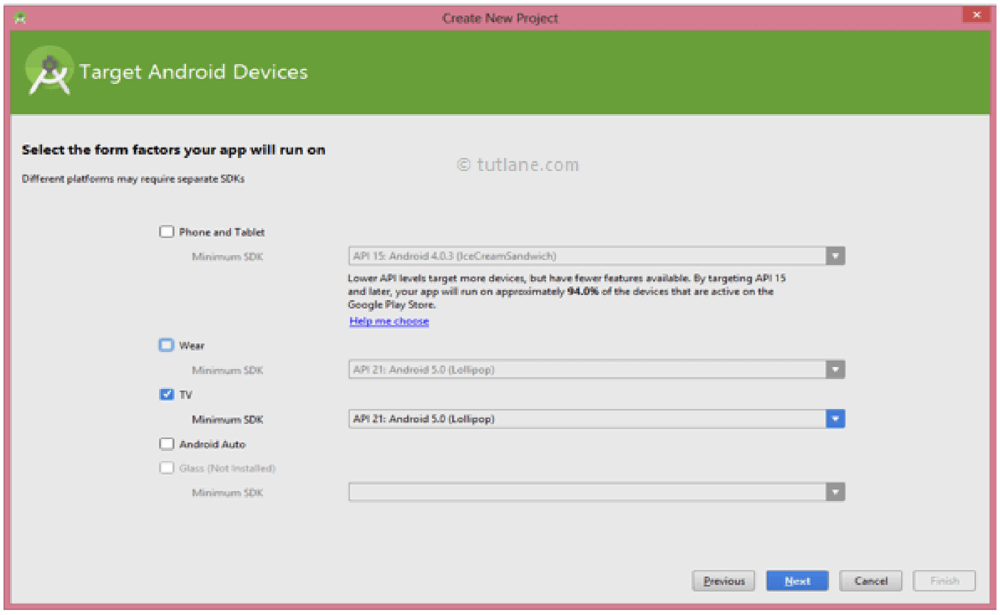
☑ Wear
Minimum SDK: API 21: Android 5.0 (Lollipop)

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

☐ Android Auto

☐ Glass (Not Installed)
Minimum SDK:

In case if we choose "**TV**", then it will show it's API and SDK versions like as shown below.


Create New Project

Target Android Devices

Select the form factors your app will run on

Different platforms may require separate SDKs

© tutlane.com

☐ Phone and Tablet
Minimum SDK: API 15: Android 4.0.3 (IceCreamSandwich)

Lower API levels target more devices, but have fewer features available. By targeting API 15 and later, your app will run on approximately **94.0%** of the devices that are active on the Google Play Store.

Help me choose

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

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

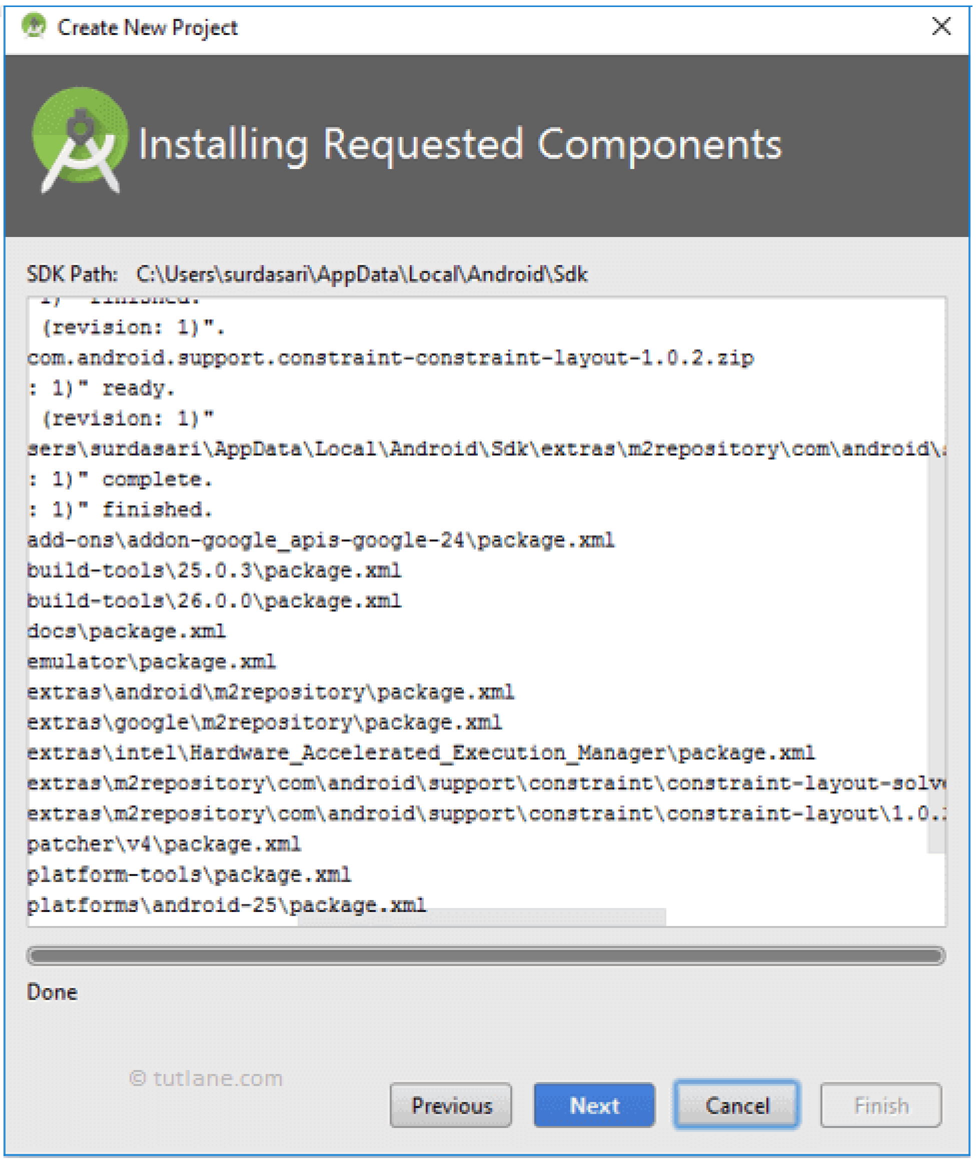
☐ Android Auto

☐ Glass (Not Installed)
Minimum SDK:

Wear: We use this option for **Android Watches** which we can wear to our hand and use the same functionality as we do with the Android devices. You can call, set the alarm, capture images, and many more things easily.

TV: We use this option for **SmartIPTV** which is very common these days. We can see our favorite channels like we see in our Home Televisions and make the changes in the channel easily.

Here we are going to implement an app for phones and tablets, so we selected a **Phone and Tablet** option and click "**Next**" and it will install required components like as shown below.

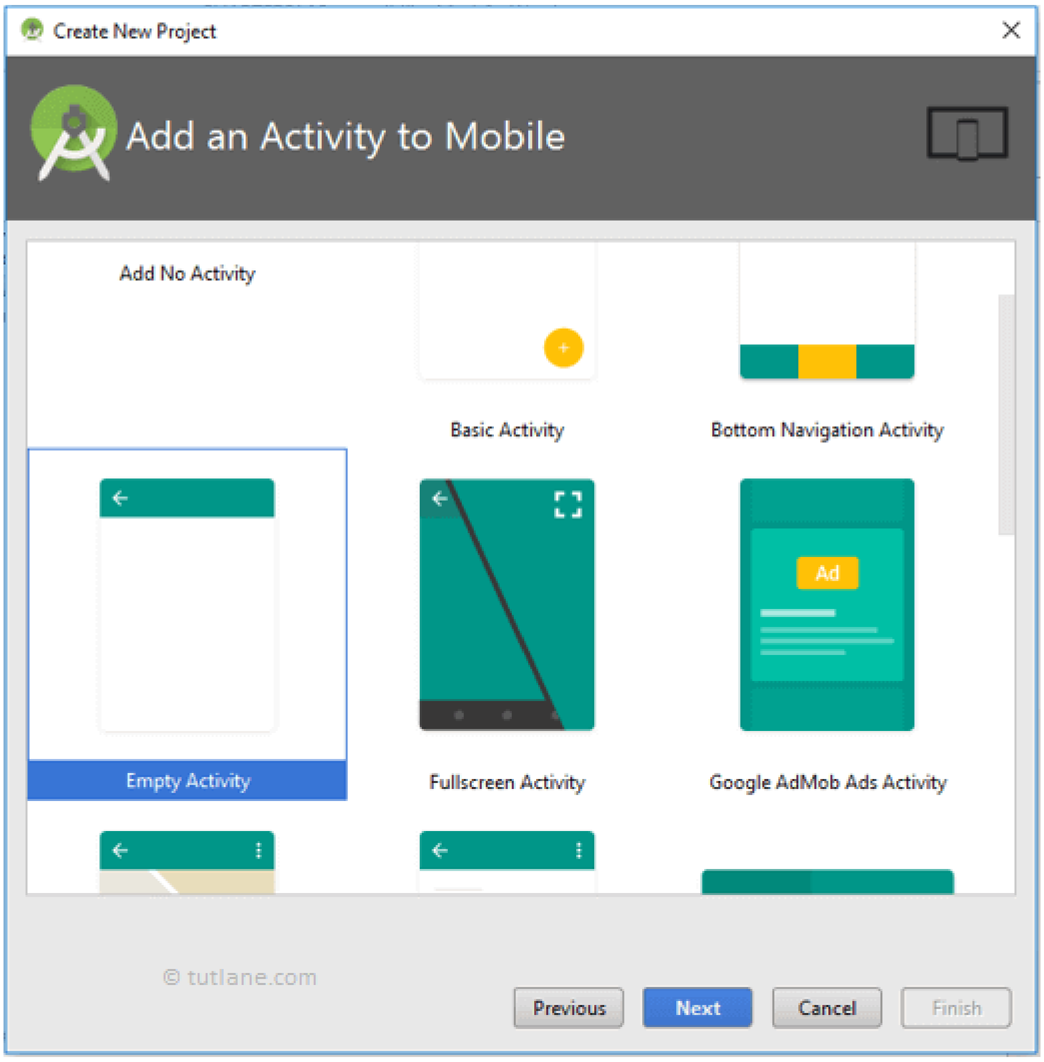


SDK Path: C:\Users\surdasari\AppData\Local\Android\Sdk

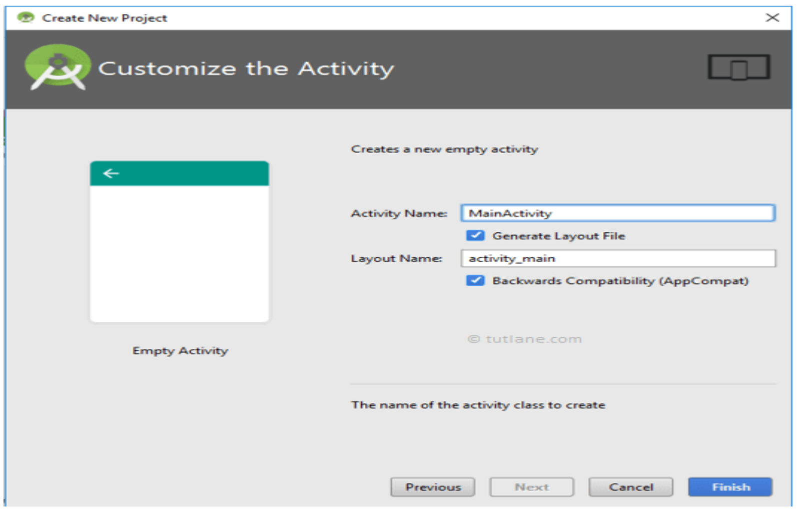
```
 (revision: 1)".
com.android.support.constraint-constraint-layout-1.0.2.zip
: 1)" ready.
 (revision: 1)"
sers\surdasari\AppData\Local\Android\Sdk\extras\m2repository\com\android\
: 1)" complete.
: 1)" finished.
add-ons\addon-google_apis-google-24\package.xml
build-tools\25.0.3\package.xml
build-tools\26.0.0\package.xml
docs\package.xml
emulator\package.xml
extras\android\m2repository\package.xml
extras\google\m2repository\package.xml
extras\intel\Hardware_Accelerated_Execution_Manager\package.xml
extras\m2repository\com\android\support\constraint\constraint-layout-solv
extras\m2repository\com\android\support\constraint\constraint-layout\1.0.
patcher\v4\package.xml
platform-tools\package.xml
platforms\android-25\package.xml
```

Done

Now click **Next** to select the particular Activity for our requirement. If we will select the "**Empty Activity**", then it will show the empty activity in our layout. In case if we choose other options, then it will show the activity which we have chosen. Here we are selecting **Empty Activity** like as shown below.



After choosing the "**Activity**" for our application, then click on the "**Next**" button and it will take you to the next screen like as shown below.



Creates a new empty activity

Activity Name: MainActivity

[✓] Generate Layout File

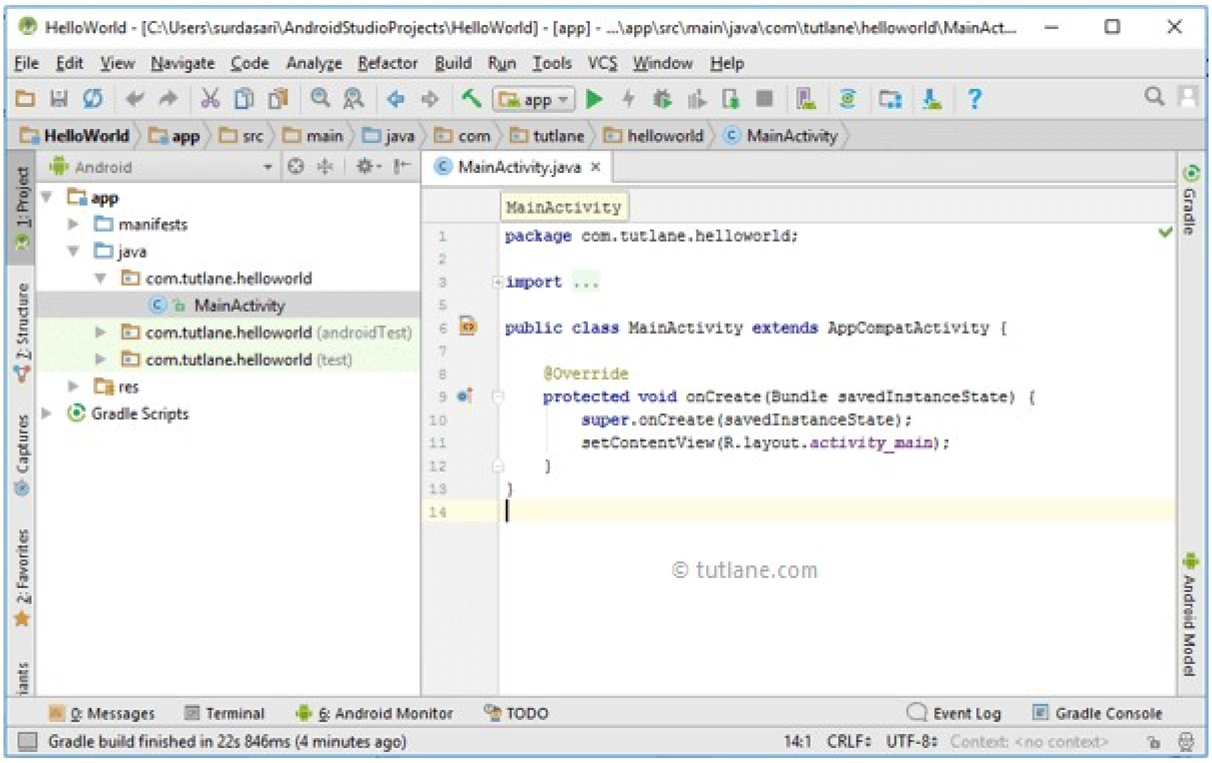
Layout Name: activity_main

[✓] Backwards Compatibility (AppCompat)

The name of the activity class to create

Here we can see that the Activity i.e. **EmptyActivity** which we selected in the previous section and the java file name i.e. "**MainActivity**". Now we are ready for the final step, just click on the "**Finish**" button and it will take you to the Main page where we have to do the coding and create new layouts over there.

After clicking **Finish**, we will be presented with Android Studio's user interface with the project explorer on the left and the workspace on the right like as shown below.



HelloWorld - [C:\Users\surdasari\AndroidStudioProjects\HelloWorld] - [app] - ...\app\src\main\java\com\tutlane\helloworld\MainAct...

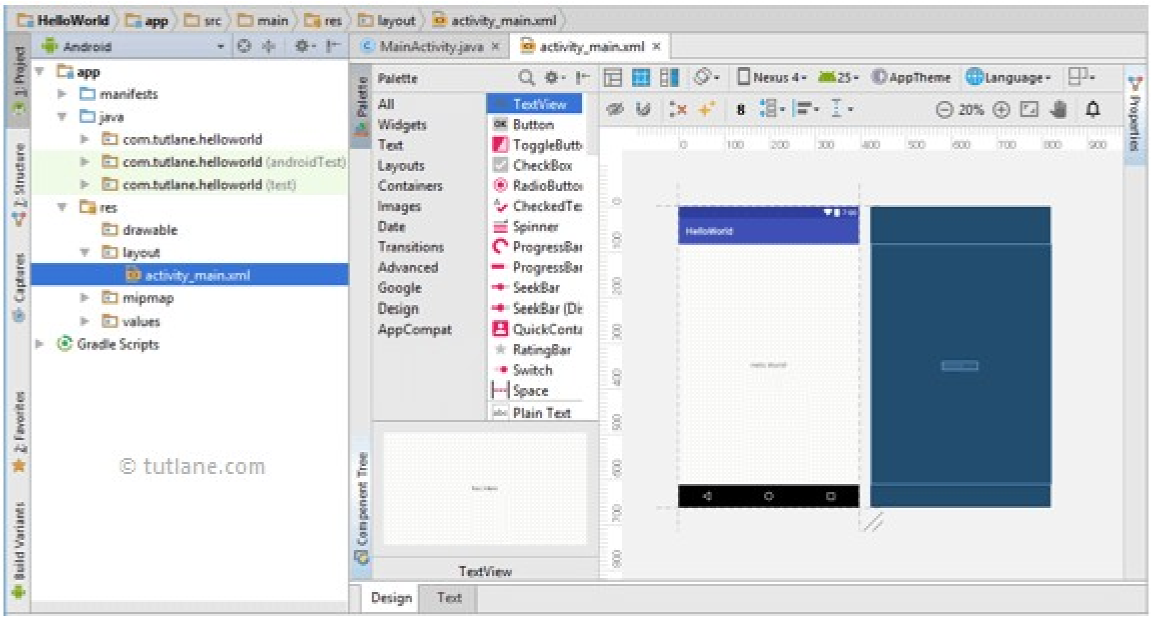
```
package com.tutlane.helloworld;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Gradle build finished in 22s 846ms (4 minutes ago)

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
HelloWorld app src main java com tutlane helloworld MainActivity
Android
app
manifests
java
com.tutlane.helloworld
MainActivity
com.tutlane.helloworld (androidTest)
com.tutlane.helloworld (test)
res
Gradle Scripts
MainActivity.java
© tutlane.com
0: Messages
Terminal
6: Android Monitor
TODO
Event Log
Gradle Console
14:1 CRLF: UTF-8: Context: <no context>

Android Layout File (activity_main.xml)

The UI of our application will be designed in this file and it will contain **Design** and **Text** modes. It will exist in the **layouts** folder and the structure of **activity_main.xml** file in **Design** mode like as shown below.



We can make required design modifications in **activity_main.xml** file either using **Design** or **Text** modes. If we switch to **Text** mode **activity_main.xml** file will contain code like as shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context="com.tutlane.helloworld.MainActivity">
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Hello World!"
      app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintLeft_toLeftOf="parent"
      app:layout_constraintRight_toRightOf="parent"
      app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

Android Main Activity File (MainActivity.java)

The main activity file in android application is **MainActivity.java** and it will exists in **java** folder. The **MainActivity.java** file will contain the java code to handle all the activities related to our app.

Following is the default code of **MainActivity.java** file which is generated by our **HelloWorld** application.

```java
package com.tutlane.helloworld;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

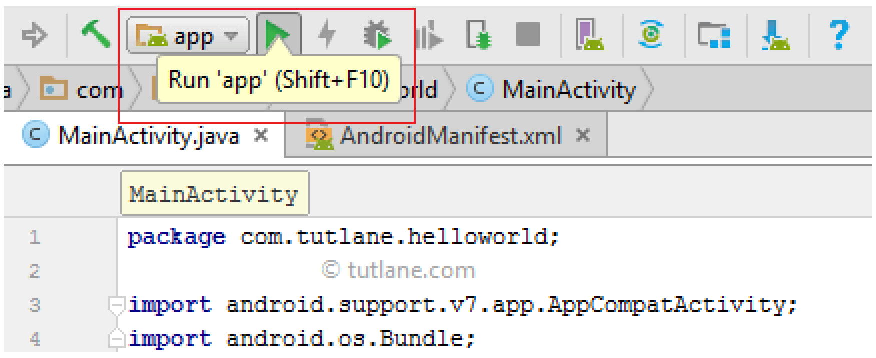
Generally, our application will contain multiple **activities** and we need to define all those **activities** in the **AndroidManifest.xml** file. In our manifest file, we need to mention the main activity for our app using the **MAIN** action and **LAUNCHER** category attributes in **intent filters** (<intent-filter>). In case if we didn't mention the MAIN action or LAUNCHER category for the main activity, our app icon will not appear in the home screen's list of apps.

Following is the default code of the **AndroidManifest.xml** file which is generated by our **HelloWorld** application.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.tutlane.helloworld" >
    <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" >
        <activity android:name=".MainActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

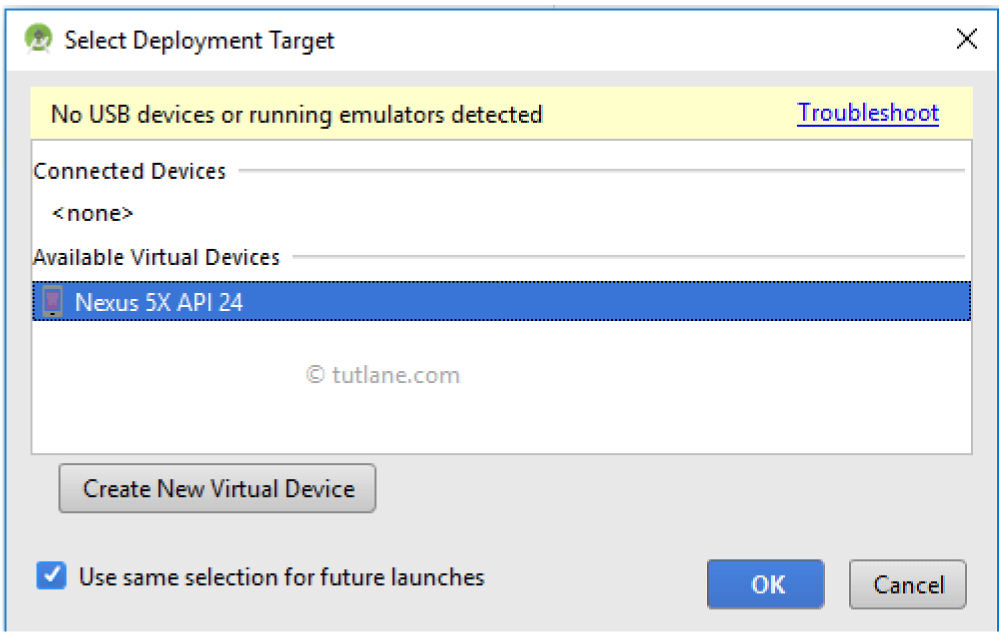
4. Run Android Hello World App

To run android applications we need to click on **Run** button or press `Shift` + `F10` like as shown below



```
package com.tutlane.helloworld;

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

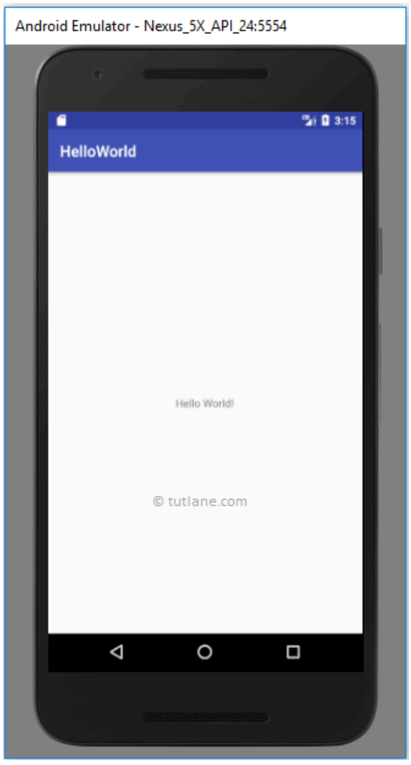
After clicking on the play button new window will open in that select **Android Virtual Device** (AVD) and click **OK** like as shown below.


Select Deployment Target

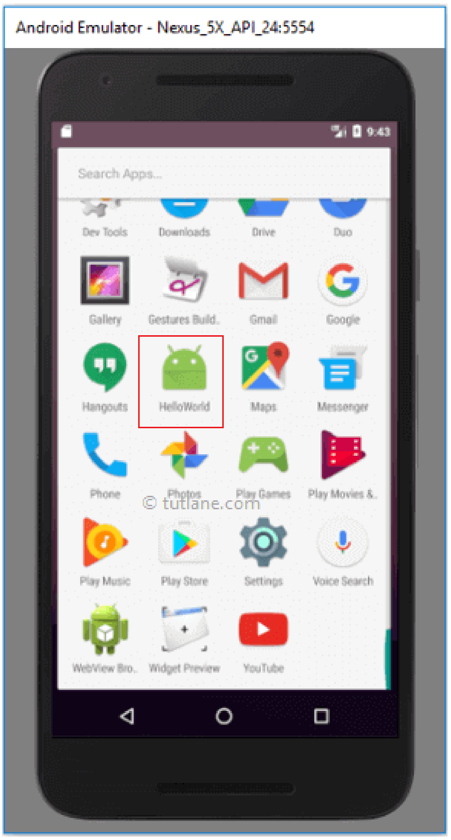
No USB devices or running emulators detected

Troubleshoot
Connected Devices
<none>
Available Virtual Devices
Nexus 5X API 24
Create New Virtual Device
Use same selection for future launches
OK
Cancel

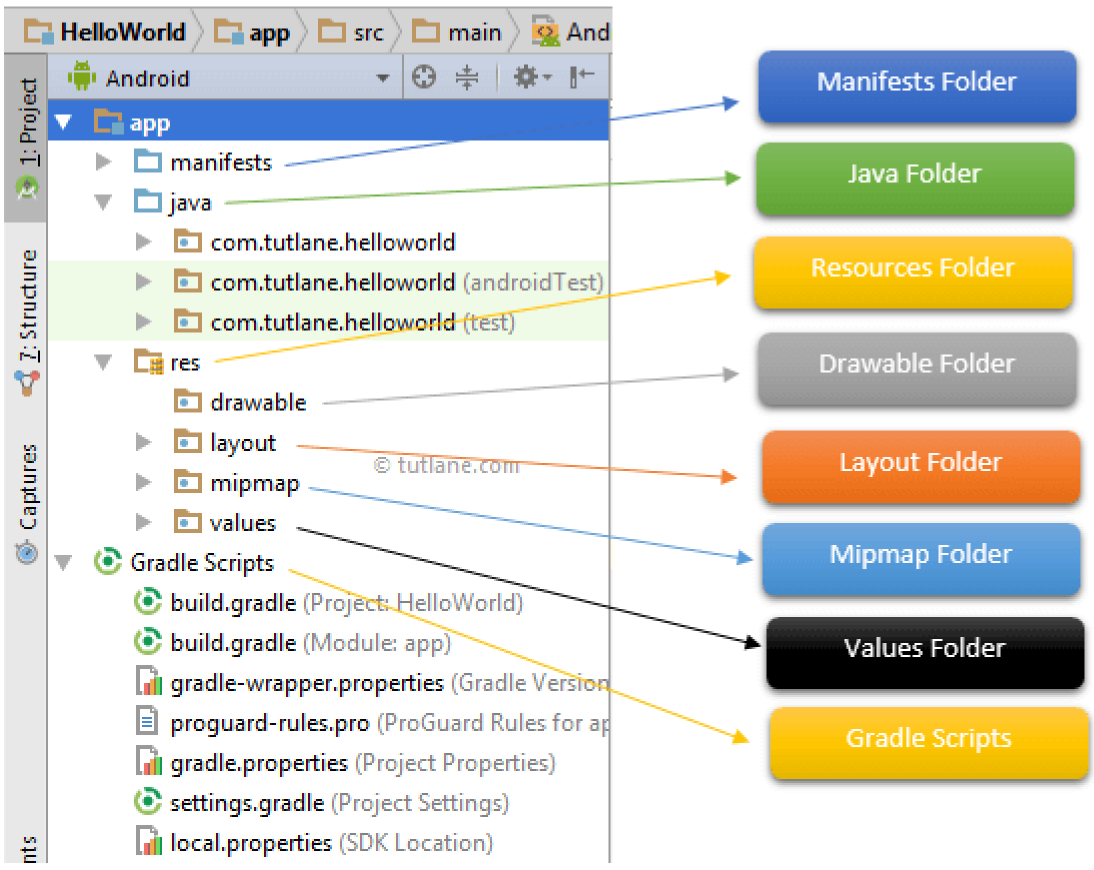
Now our android hello world application will show the result like as shown below



In our **AndroidManifest.xml** file, we mentioned the MAIN action and LAUNCHER category attributes for our main activity file due to that our app icon will create in Home screen list of apps like as shown below.



5. Understanding Android App Folder Structure

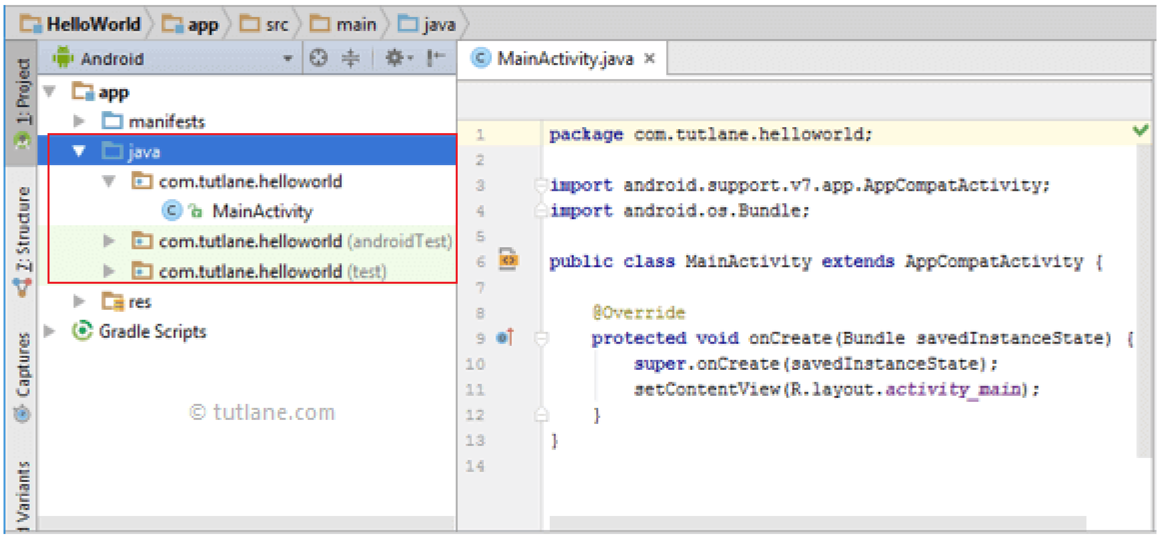

HelloWorld
app
src
main
And
Android
1: Project
7: Structure
Captures
app
manifests
java
com.tutlane.helloworld
com.tutlane.helloworld (androidTest)
com.tutlane.helloworld (test)
res
drawable
layout
mipmap
values
Gradle Scripts
build.gradle (Project: HelloWorld)
build.gradle (Module: app)
gradle-wrapper.properties (Gradle Version
proguard-rules.pro (ProGuard Rules for ap
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
© tutlane.com
Manifests Folder
Java Folder
Resources Folder
Drawable Folder
Layout Folder
Mipmap Folder
Values Folder
Gradle Scripts

The Android project structure on the disk might be differs from the above representation. To see the actual file structure of the project, select **Project** from the **Project** dropdown instead of **Android**.

The android app project will contain different types of app modules, source code files, and resource files. We will explore all the folders and files in the android app.

Java Folder

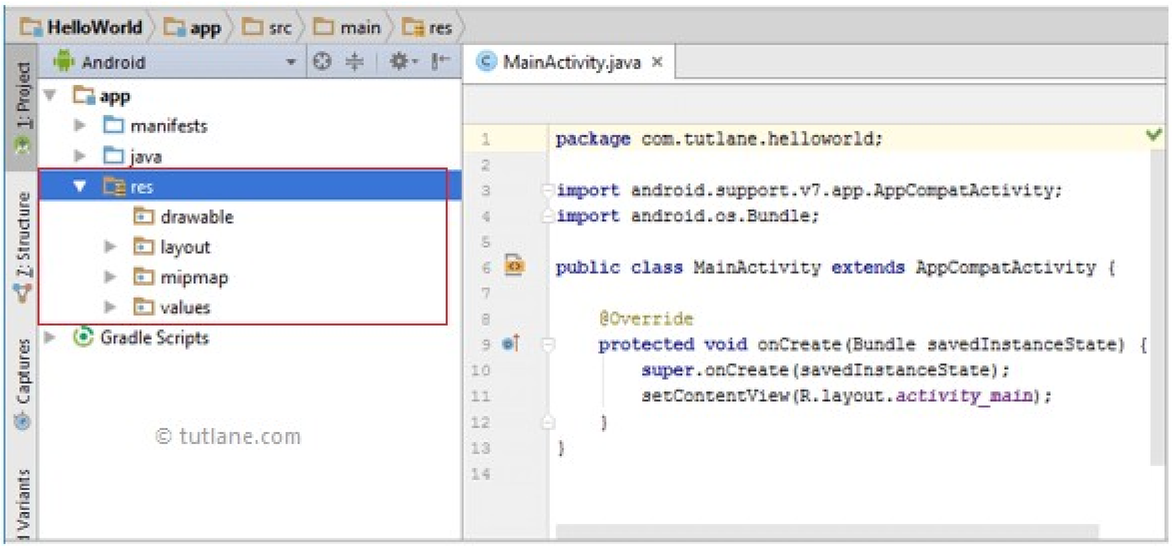
This folder will contain all the java source code (**.java**) files which we'll create during the application development, including JUnit test code. Whenever we create any new project/application, by default the class file **MainActivity.java** will create automatically under the package name "**com.tutlane.helloworld**" like as shown below.



```java
package com.tutlane.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

res (Resources) Folder

It’s an important folder that will contain all non-code resources, such as bitmap images, UI strings, XML layouts like as shown below.



```java
package com.tutlane.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

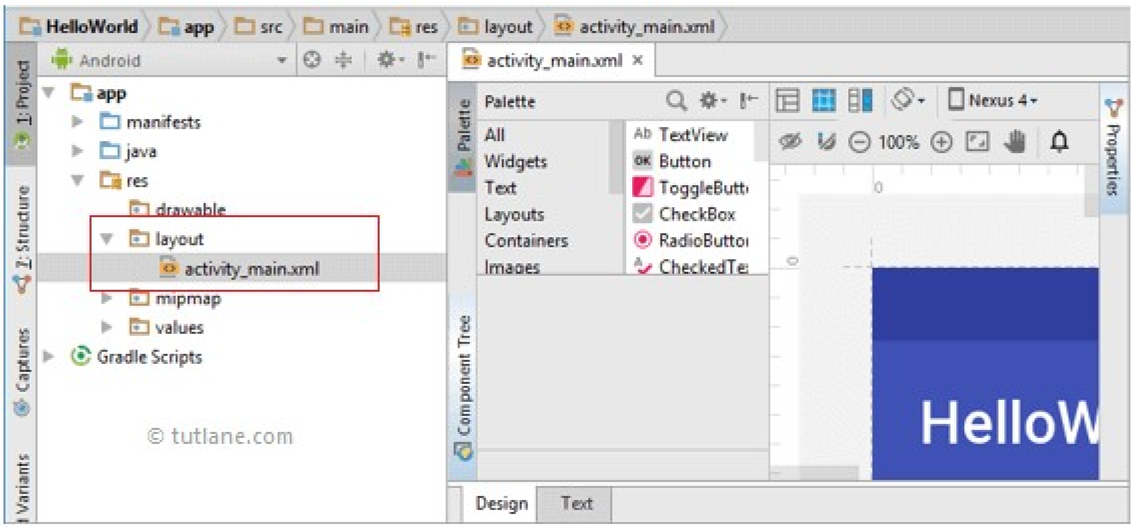
The res (**Resources**) will contain a different type of folders that are

Drawable Folder (res/drawable)

It will contain the different types of images as per the requirement of application. It’s a best practice to add all the images in a **drawable** folder other than app/launcher icons for the application development.

Layout Folder (res/layout)

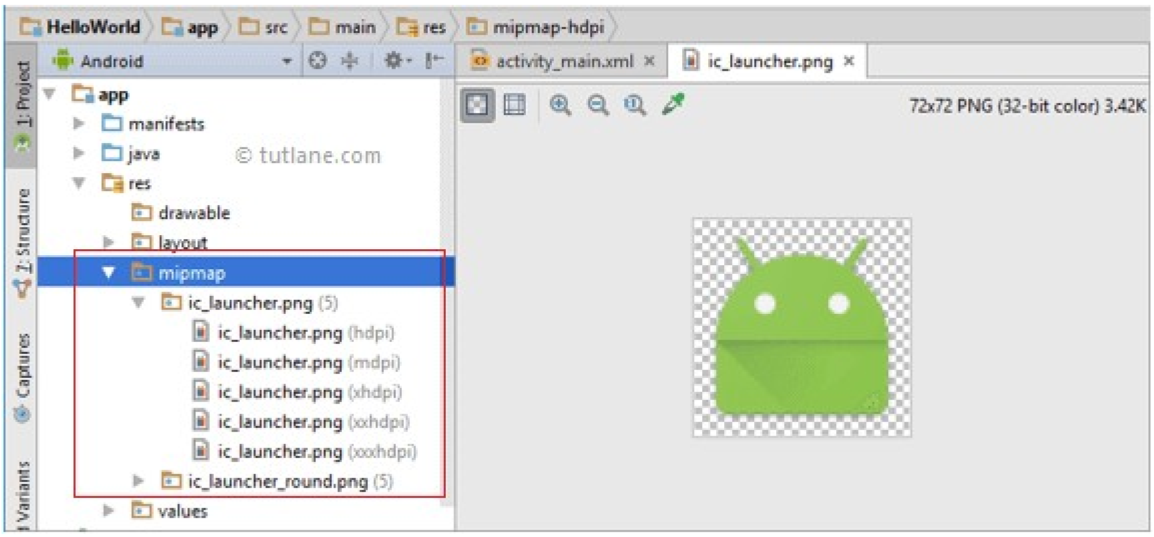
This folder will contain all XML layout files which we used to define the user interface of our application. Following is the structure of the **layout** folder in the android application.



Mipmap Folder (res/mipmap)

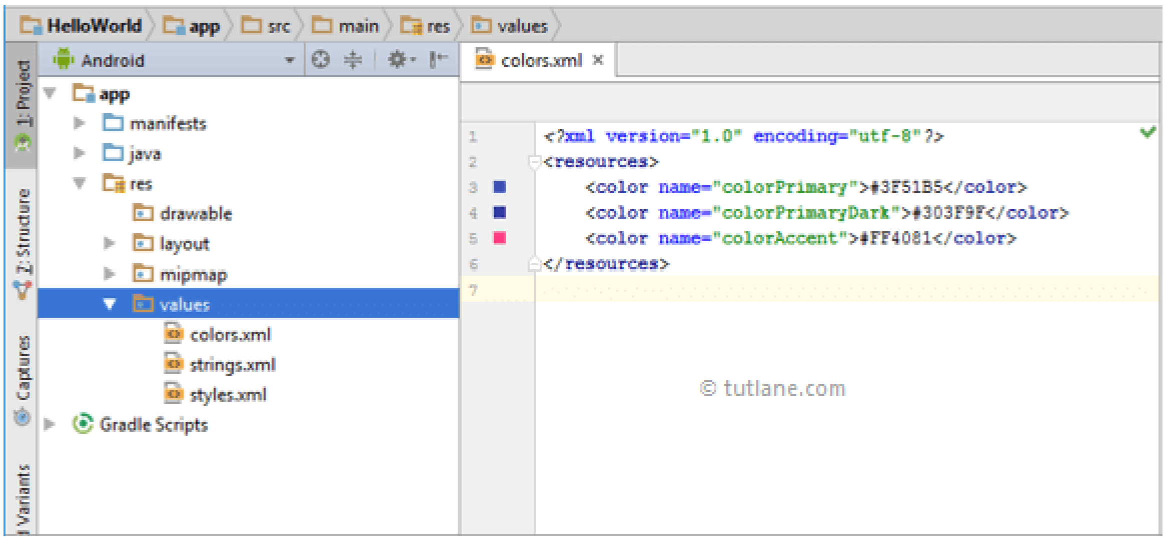
This folder will contain app / launcher icons that are used to show on the home screen. It will contain different density type of icons such as hdpi, mdpi, xhdpi, xxhdpi, xxxhdpi, to use different icons based on the size of the device.

Following is the structure of the **mipmap** folder in the android application.



Values Folder (res/values)

This folder will contain various XML files, such as strings, colors, style definitions and a static array of strings or integers. Following is the structure of the **values** folder in android application.

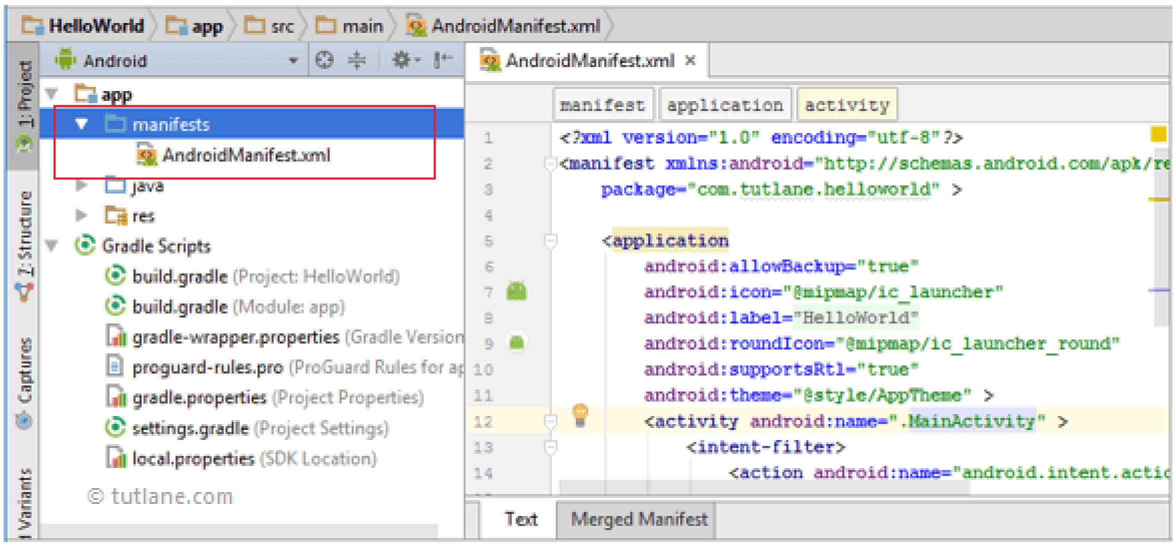


```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

Manifests Folder

This folder will contain a manifest file (**AndroidManifest.xml**) for our android application. This manifest file will contain information about our application such as android version, access permissions, metadata, etc. of our application and its components. The manifest file will act as an intermediate between android OS and our application.

Following is the structure of the **manifests** folder in the android application.

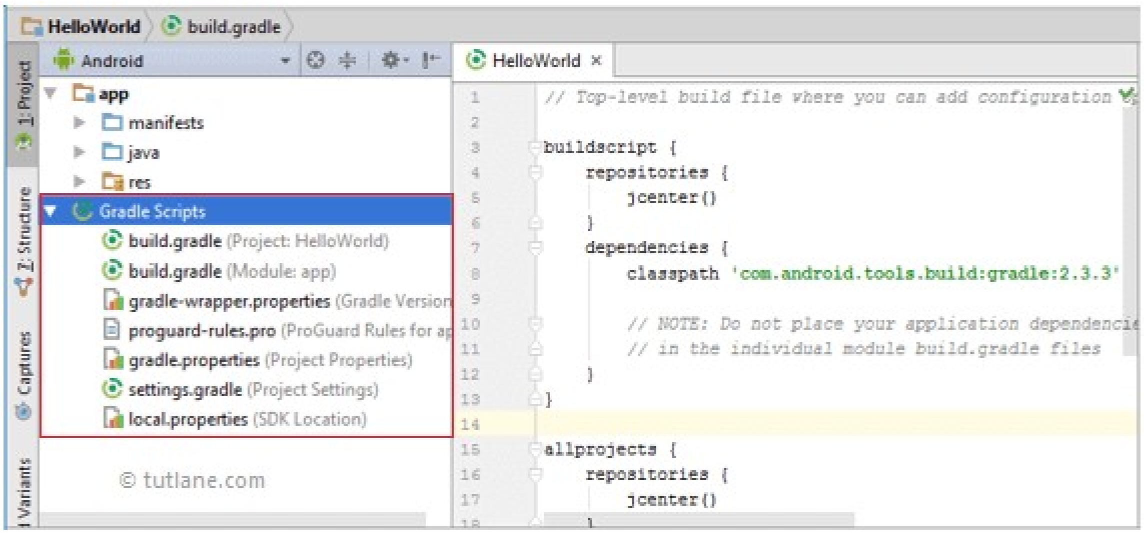

HelloWorld
app
src
main
AndroidManifest.xml
Android
AndroidManifest.xml
app
manifests
AndroidManifest.xml
java
res
Gradle Scripts
build.gradle (Project: HelloWorld)
build.gradle (Module: app)
gradle-wrapper.properties (Gradle Version
proguard-rules.pro (ProGuard Rules for ap
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
© tutlane.com
manifest
application
activity
Text
Merged Manifest
1: Project
7: Structure
Captures
Variants

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/re
    package="com.tutlane.helloworld" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="HelloWorld"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
        <activity android:name=".MainActivity" >
            <intent-filter>
                <action android:name="android.intent.actic
```

Gradle Scripts

In android, Gradle means automated build system and by using this we can define a build configuration that applies to all modules in our application. In Gradle **build.gradle (Project)**, and **build.gradle (Module)** files are useful to build configurations that apply to all our app modules or specific to one app module.

Following is the structure of **Gradle Scripts** in the android application.

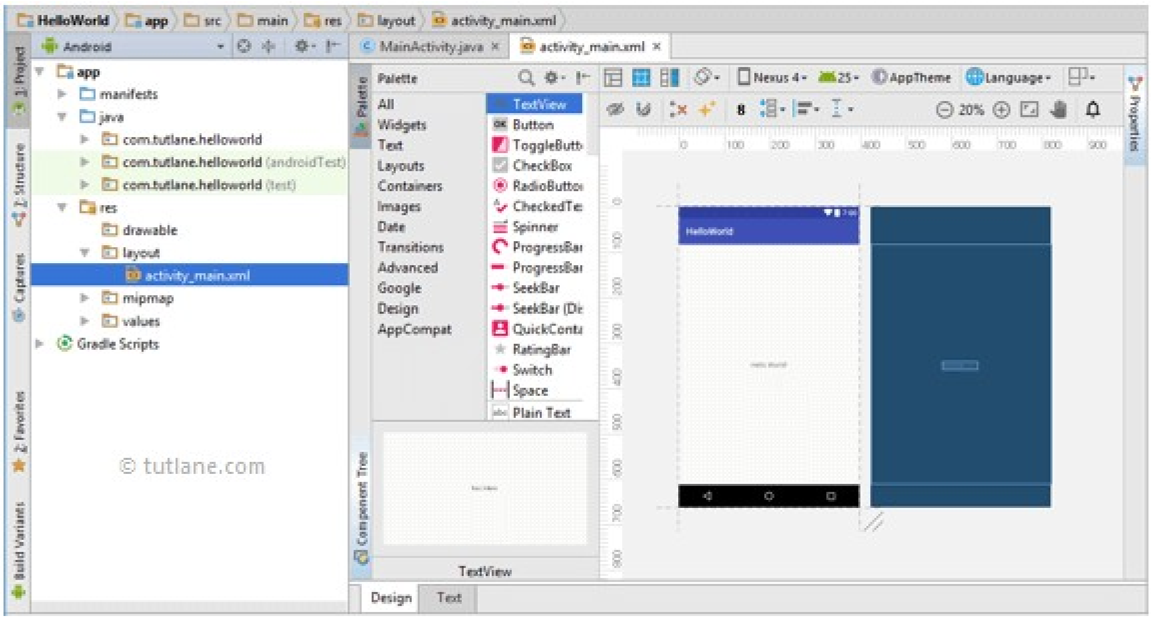

HelloWorld
build.gradle
1: Project
2: Structure
Captures
Variants
Android
app
manifests
java
res
Gradle Scripts
build.gradle (Project: HelloWorld)
build.gradle (Module: app)
gradle-wrapper.properties (Gradle Version
proguard-rules.pro (ProGuard Rules for ap
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
© tutlane.com
HelloWorld ×

```
// Top-level build file where you can add configuration

buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.3.3'

        // NOTE: Do not place your application dependencie
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        jcenter()
        }
```

Following are the important files which we need to implement an app in android studio.

Android Layout File (activity_main.xml)

The UI of our application will be designed in this file and it will contain **Design** and **Text** modes. It will exists in the **layouts** folder and the structure of **activity_main.xml** file in **Design** mode like as shown below.



We can make required design modifications in **activity_main.xml** file either using **Design** or **Text** modes. If we switch to **Text** mode **activity_main.xml** file will contain a code like as shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.tutlane.helloworld.MainActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

Android Main Activity File (MainActivity.java)

The main activity file in the android application is **MainActivity.java** and it will exist in the **java** folder. The **MainActivity.java** file will contain the java code to handle all the activities related to our app.

Following is the default code of **MainActivity.java** file which is generated by our .HelloWorld application

```
package com.tutlane.helloworld;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Android Manifest File (AndroidManifest.xml)

Generally, our application will contain multiple activities and we need to define all those activities in the **AndroidManifest.xml** file. In our manifest file, we need to mention the main activity for our app using the **MAIN** action and **LAUNCHER** category attributes in **intent filters**

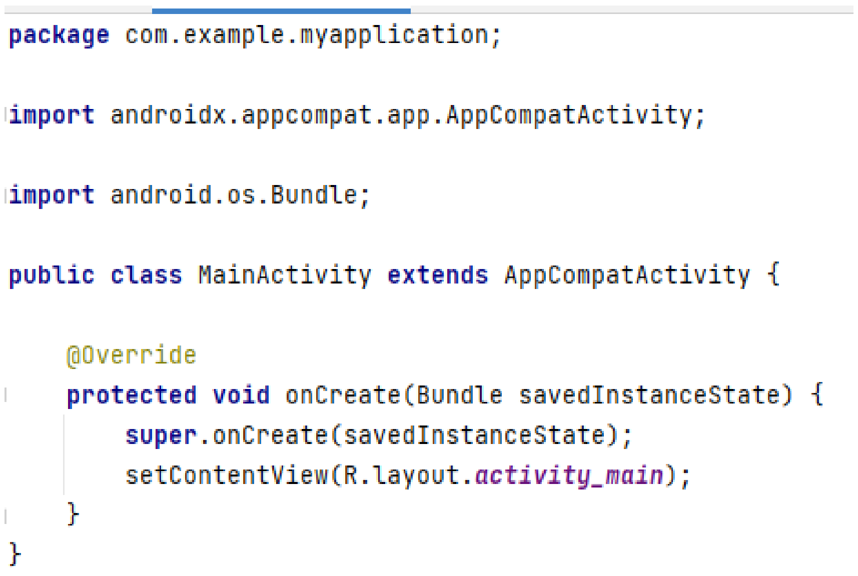
(<intent-filter>). In case if we didn’t mention MAIN action or LAUNCHER category for the main activity, our app icon will not appear in the home screen’s list of apps.

Following is the default code of **AndroidManifest.xml** file which is generated by our **HelloWorld** application.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.tutlane.helloworld" >

    <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" >
        <activity android:name=".MainActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

These are the main folders and files required to implement an application in android studio. If you want to see the actual file structure of the project, select **Project** from the **Project** dropdown instead of **Android**.

6. Understanding ManiActivity.java



```
package com.example.myapplication;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

1.`import androidx.appcompat.app.AppCompatActivity`
All activities in Android inherit from an `android.app.Activity` base class. In our case, `MainActivity` does not inherit directly from that class. Instead, it extends `androidx.appcompat.app.AppCompatActivity`. That, in turn, inherits from `android.app.Activity`, so `MainActivity` has `Activity` in its inheritance hierarchy.

2. `import android.os.Bundle`
Bundle is a utility class that lets you store a set of name-value pairs. You will always find this import along with the import for Activity class because both onCreate() and onFreeze() methods take Bundle as a parameter. Into a Bundle object, you can put integers, longs, strings, arrays, etc along with the keys to identify them. When needed, these values can be obtained by using those keys.

3. `protected void onCreate(Bundle savedInstanceState)`
`MainActivity` has one Java method or Kotlin function: `onCreate()`. This overrides an `onCreate()` method that we inherit. Our job in `onCreate()` is to set up the basic UI that is to be shown by this activity.

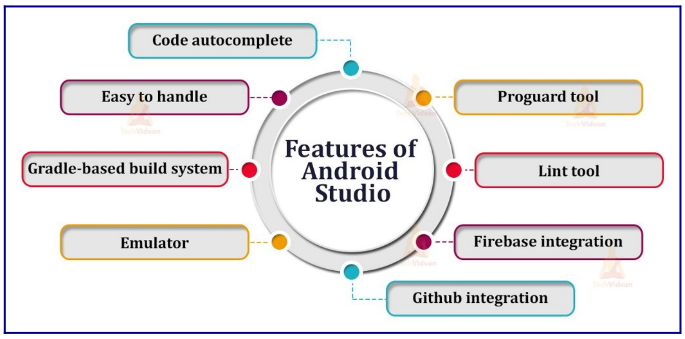
1. When an Activity first call or launched then onCreate(Bundle savedInstanceState) method is responsible to create the activity.
2. When ever orientation(i.e. from horizontal to vertical or vertical to horizontal) of activity gets changed or when an Activity gets forcefully terminated by any Operating System then savedInstanceState i.e. object of Bundle Class will save the state of an Activity.
3. After Orientation changed then onCreate(Bundle savedInstanceState) will call and recreate the activity and load all data from savedInstanceState.
4. Basically Bundle class is used to stored the data of activity whenever above condition occur in app.

4. `super.onCreate(savedInstanceState)`
When we override a method, we have the option of completely replacing the method in our class, or of extending the existing parent class' method. By calling `super.onCreate(savedInstanceState);`, you tell the Dalvik VM to run your code **`in addition`** to the existing code in the onCreate() of the parent class.

5. `setContentView(R.layout.`*`activity_main`*`)`
This is a Java method called **`setContentView`**. It sets the XML file you want as your main layout when the app starts. In between round brackets, you need the name and location of your layout file. The letter R in the round brackets is short for **`res`**. This is the resource folder where those drawable, layout, mipmap, and values folders are. The **`layout.activity_main`** part points to the activity_main XML file, which is in the layout folder of res. If you wanted a different XML file to load when your app starts, you'd point to a different file:

setContentView(R.layout.some_other_xml_file);

7. Features of Android Studio



Android studio provides several remarkable features to offer a hassle-free development process. Some of those features are as follows:

1. Code Autocomplete

You get features like code to autocomplete and auto error suggestions due to IntelliJ IDEA.

2. Easy to handle

Android studio is built in such a way that you can quickly locate the tools and settings which you require in your development.

3. Gradle-based build system

Android studio supports a Gradle-based build system. It helps us to automate, build and test our application. It also provides features like debugging, and APK build.

4. Emulator

No need to worry if you don't have an android phone nearby to test your applications. Android studio provides you to select your emulator where you want to run your applications. The emulator is very versatile because you can choose any version of android to make a virtual device(or emulator).

5. Proguard Tool

It's a tool provided by java, which creates some rules for your app while developing. It looks in your app structure and also helps to minify your code by removing all the unnecessary parts. Thereby it also provides security for your applications.

6. Lint Tool

This tool in Android Studio helps you locate any syntactical or programmatical mistakes in your code. It also helps to structure and format your code to make it look neat.

7. Firebase Integration

You can even directly connect your app to Firebase using the firebase tool in android studio. Firebase provides many features like real-time database, authentication, cloud messaging, etc., which gives more power to your applications.

8. Github Integration

You can integrate your Github repository with your application and directly push files through the Android studio itself.