

Java Fundamentals for Android™ Development

Version 7

By Android ATC Team



www.androidatc.com



- Java Fundamentals topics.
- Lessons target beginners and allow a smooth start with android programming.
- Practical lessons and instructions accompanied by relevant snapshots.

Android ATC

Java Fundamentals for Android™ Development

Course Code: AND-404 version 7

Hands-on Guide to Java Programming

Java Fundamentals for Android™ Development

Course Code: AND-404 Version 7

© 2017 Android ATC

Published by: Android ATC

ISBN: 978-0-9900143-7-9

Price: Free

Information in this book, including URL and other Internet Web site references, is subject to change without notice. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Android ATC.

Android ATC is not responsible for webcasting or any other form of transmission received from any linked site.

Android ATC is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Android ATC of the site or the products contained therein.

Android ATC Company may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. As expressly provided in any written license agreement from Android ATC, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

This “Java fundamentals for Android™ development” e-book is a detailed guide that provides the basics to understand the Java programming concept. It is a combination of theoretical and practical guide that covers skills and knowledge every developer should learn before starting the Android development course.

At the time of writing this e-book, the latest version of Android OS released was 7 (Nougat) and that of Android Studio was 2.2. All exercises in this e-book were built to be compatible with these latest versions. Since the update of both Android OS and Android Studio is a continuous process, it is highly possible that any of these components has already been updated by the time you start your training using this e-book. If this is the case, you might notice some minor difference in the exercises’ steps and the screenshots provided, depending on how a major an update has been.

This neither makes the lessons outdated nor the exercises incorrect. It is only impractical to release a new version of the e-book for every update. This e-book targets trainees who don't have background in object oriented programming.

Android ATC Training team continuously works on providing the most up to date labs and code samples. Nonetheless, we would like to apologize in advance in case any exercise or screenshot is inaccurate.

Warning and Disclaimer:

This e-book is designed to provide information about Java development course for free. Every effort has been made to make this e-book as complete and as accurate as possible.

Exam

No exam is available for this course.

Trademark Acknowledge:

All terms mentioned in this e-book which are known to be trademarks or service marks have been appropriately capitalized. Use of a term in this e-book should not be regarded as affecting the validity of any trademark or service mark.

Android is a trademark of Google Inc. The Android robot is reproduced or modified from work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution License.

Feedback Information:

As Android ATC, our goal is to create in-depth technical books of the highest quality and value. Each book or e-book is crafted with care and precision, undergoing rigorous development that involves the unique expertise of members from professional technical community.

Readers' feedback is natural continuation of this process. If you have any comments regarding how we could improve the quality of this book, or otherwise alter it to better suit your needs, you can contact us through email at: info@androidatc.com. Please make sure to include the book title and ISBN in your message. We greatly appreciate your assistance Android ATC Team.

Table of Contents

Lesson 1: First Step in Java

The History of Java	1-2
How Java Programs work?	1-2
Install Java JDK and JRE.....	1-4
Why did Google choose Java over other programming languages?.....	1-8
Android OS Structure.....	1-8
Install Android Studio	1-9

Lesson 2: Create and Run Java Projects

Creating an Android Project (Java Project) Using Android Studio	2-2
Writing a Java Program	2-7
Java Methods.....	2-9
Running a Java Program.....	2-9
Write a Comment	2-15
Java Variables and Their Data Type.....	2-16

Lesson 3: Control Flow Statements

Introduction.....	3-2
IF – Else Statement	3-2
If...Else and Else...If... Statement.....	3-4
If Else and Logical Operators	3-5
Switch Statement.....	3-7
While Loop	3-8
Do-while Loop	3-10
For Loop	3-11
The Break Statement	3-13
The Continue Statement	3-14

Lesson 4: Methods and Arrays

Introduction.....	4-2
Method Structure	4-3
Call Method by Value	4-6
Call Method by Reference.....	4-8
Arrays.....	4-10
Enter Data to a Java Program	4-13
Object-Oriented Programming (OOP) Concepts	4-15
Java Class	4-18

Lesson 1: First Step in Java

- The History of Java
- How Java Programs work?
- Install Java JDK and JRE
- Why did Google choose Java over other programming languages?
- Android OS Structure
- Install Android Studio

The History of Java

In the early 90s, extending the power of network computing to the activities of everyday life was a radical vision. In 1991, a small group of Sun Microsystems engineers called the "Green Team" believed that the next wave in computing was the union of digital consumer devices and computers. Led by James Gosling, the team worked around the clock and created the programming language that would revolutionize our world – Java.

The Green Team demonstrated their new language with an interactive, handheld home-entertainment controller that was originally targeted at the digital cable television industry. Unfortunately, the concept was much too advanced for the team at the time. But it was just right for the Internet, which was just starting to take off. In 1995, the team announced that the Netscape Navigator Internet browser would incorporate Java technology.

Oracle Corporation is the current owner of the official implementation of the Java SE platform, following their acquisition of Sun Microsystems on January 27, 2010. This implementation is based on the original implementation of Java by Sun.

Today, Java not only permeates the Internet, but also is the invisible force behind many of the applications and devices that power our day-to-day lives. From mobile phones to handheld devices, games and navigation systems to e-business solutions, Java is everywhere. Java is, one of the most popular programming languages in use in the IT industry.

To read more information about Java history you can find them on Oracle web site on the following link: <http://oracle.com.edgesuite.net/timeline/java/>

To find more detentions, practical examples and details about Java you may explore the Oracle web site through the following link:

<https://docs.oracle.com/javase/tutorial/>

How Java Programs work?

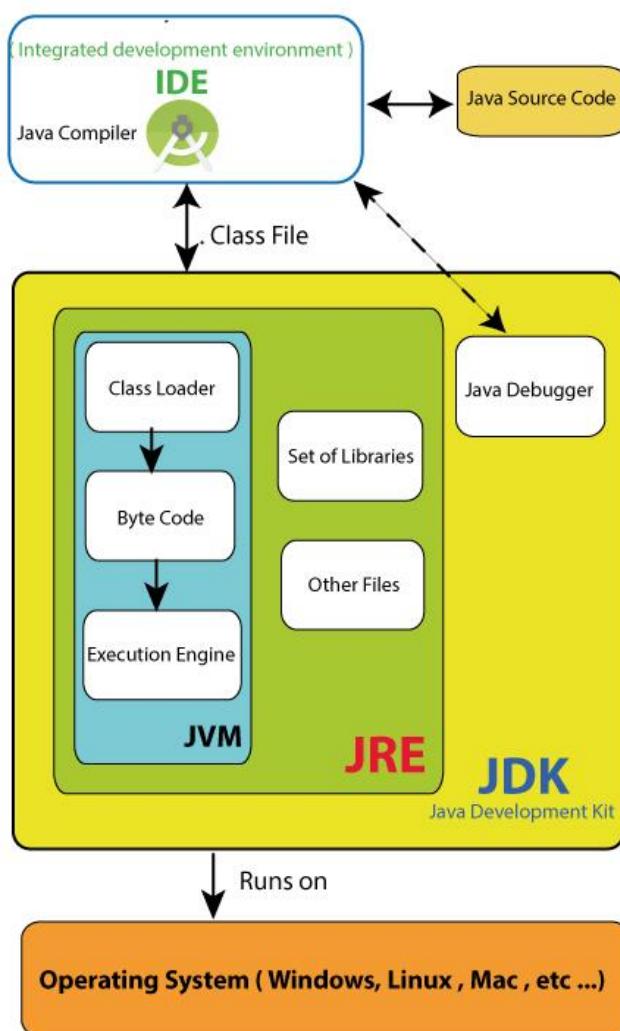
Imagine you want to create a Microsoft word file that includes reports or any other contents, what's the first step you need to take? The answer is you will open the software that you will use to create this file, which means you will open the Microsoft word software, where this software will help you create this word document and anyone who wants to read this file also needs this Microsoft Word software to read the content of the file.

The same thing for Java, you need software to write or open the source code file, this software is called Java Compiler or IDE (Integrated Development Environment). We have a lot of IDE software like Eclipse, NetBeans, Android Studio and others. They are

called Javac. Here in this book, we will use Android Studio as the Java compiler.

The Java source codes which were written in Android Studio (IDE) will be considered as project.class files that have the ability to run and get results, where this class file will be moved to JRE (Java Runtime Environment). JRE in turn has a part called Class Loader, which is responsible for receiving the class file and executes it by the Execution Engine part of the JRE.

The following flow chart will display the work flow path of any Java file from being written to run and produce the output.



Java Program Work Flow

The previous chart displays that there is a main part of creating and running Java program called JDK (Java Development Kit), this software includes the part, which is responsible to write and run the code and then send the result to the operating system.

Install Java JDK and JRE

To start creating the Java program, we must install the Android Studio (IDE) and to install it, we need to install the prerequisites of it which are the JDK and JRE. You may download the JDK and JRE for free from the official website of Oracle, below is the download link:

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

Java SE Development Kit 7u79		
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.		
<input type="radio"/> Accept License Agreement	<input checked="" type="radio"/> Decline License Agreement	
Product / File Description	File Size	Download
Linux x86	130.4 MB	jdk-7u79-linux-i586.rpm
Linux x86	147.6 MB	jdk-7u79-linux-i586.tar.gz
Linux x64	131.69 MB	jdk-7u79-linux-x64.rpm
Linux x64	146.4 MB	jdk-7u79-linux-x64.tar.gz
Mac OS X x64	196.89 MB	jdk-7u79-macosx-x64.dmg
Solaris x86 (SVR4 package)	140.79 MB	jdk-7u79-solaris-i586.tar.Z
Solaris x86	96.66 MB	jdk-7u79-solaris-i586.tar.gz
Solaris x64 (SVR4 package)	24.67 MB	jdk-7u79-solaris-x64.tar.Z
Solaris x64	16.38 MB	jdk-7u79-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	140 MB	jdk-7u79-solaris-sparc.tar.Z
Solaris SPARC	99.4 MB	jdk-7u79-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	24 MB	jdk-7u79-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	18.4 MB	jdk-7u79-solaris-sparcv9.tar.gz
Windows x86	138.31 MB	jdk-7u79-windows-i586.exe
Windows x64	140.06 MB	jdk-7u79-windows-x64.exe

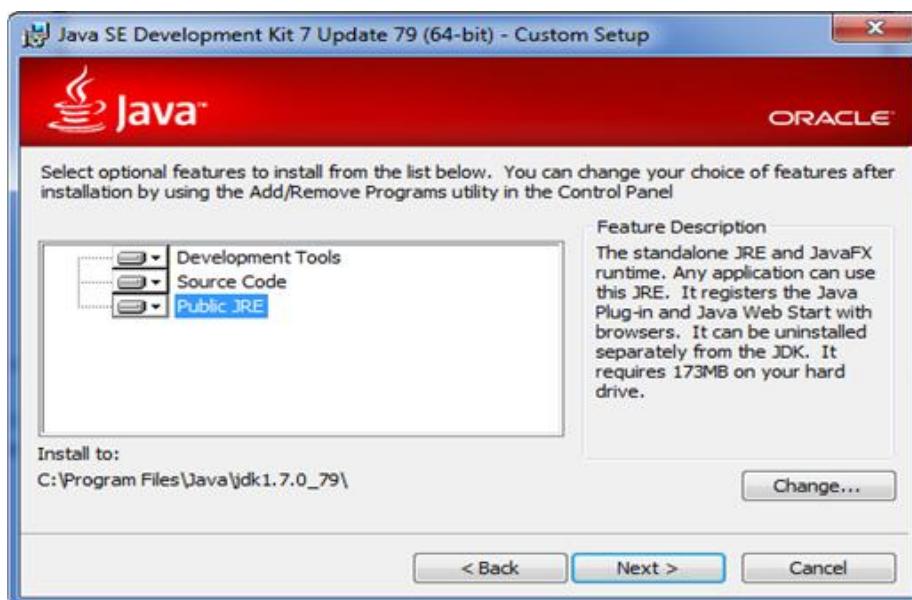
This Java SE kit which you will select includes the JDK & JRE thus your selection will depend on your operating system which you have on your computer.

Select “**Accept License Agreement**” and click the JDK file type which is most suitable for your operating system. The selected file will be downloaded on your system. Once the file is downloaded on your system, double click on it and you will see the following screen:

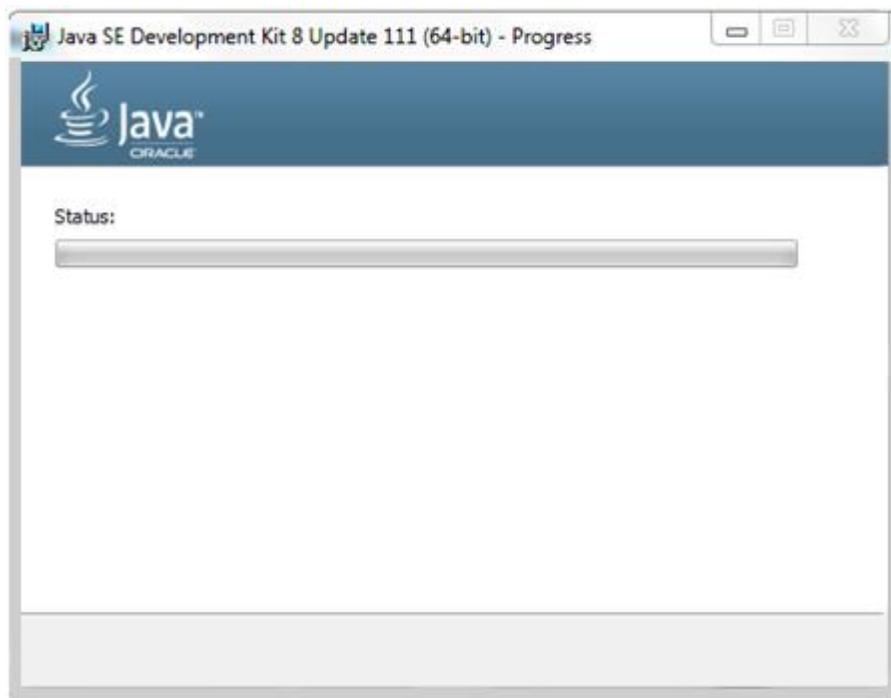
Then, click Next on the following dialog box:



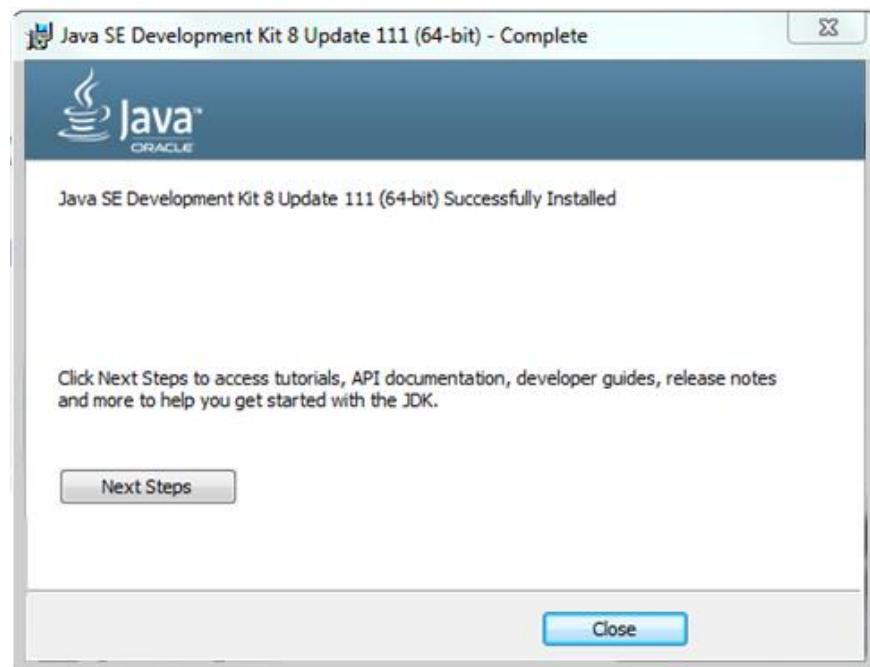
On the following dialog box, click **Next**.



The installation will start like following; wait for it to finish.

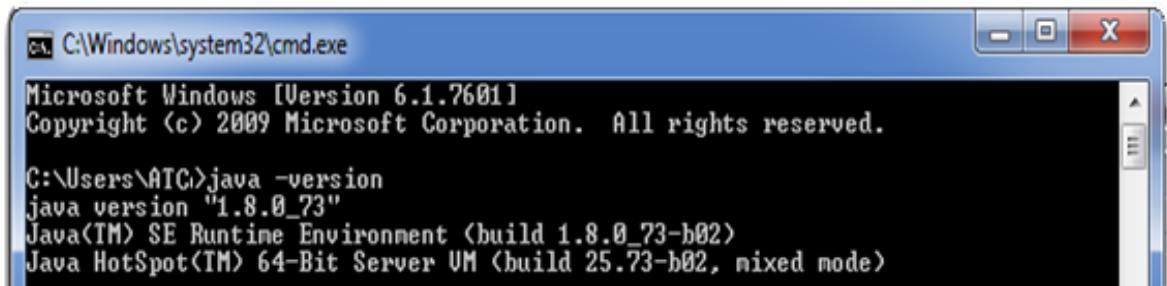


Once the installation is complete, you will see the following screen. Click "Close" to finish the installation procedure.



To verify if you have successfully installed JDK on your Windows machine, follow the steps below:

1. Open a command prompt by clicking Start → Run, then type "cmd", then click OK
2. In the window that opens, type java –version then press **Enter** if you are using Microsoft Windows.
3. You should see the following message in the console if your installation was completes successfully.



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text:
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\ATC>java -version
java version "1.8.0_73"
Java(TM) SE Runtime Environment (build 1.8.0_73-b02)
Java HotSpot(TM) 64-Bit Server VM (build 25.73-b02, mixed mode)

Why did Google choose Java over other programming languages?

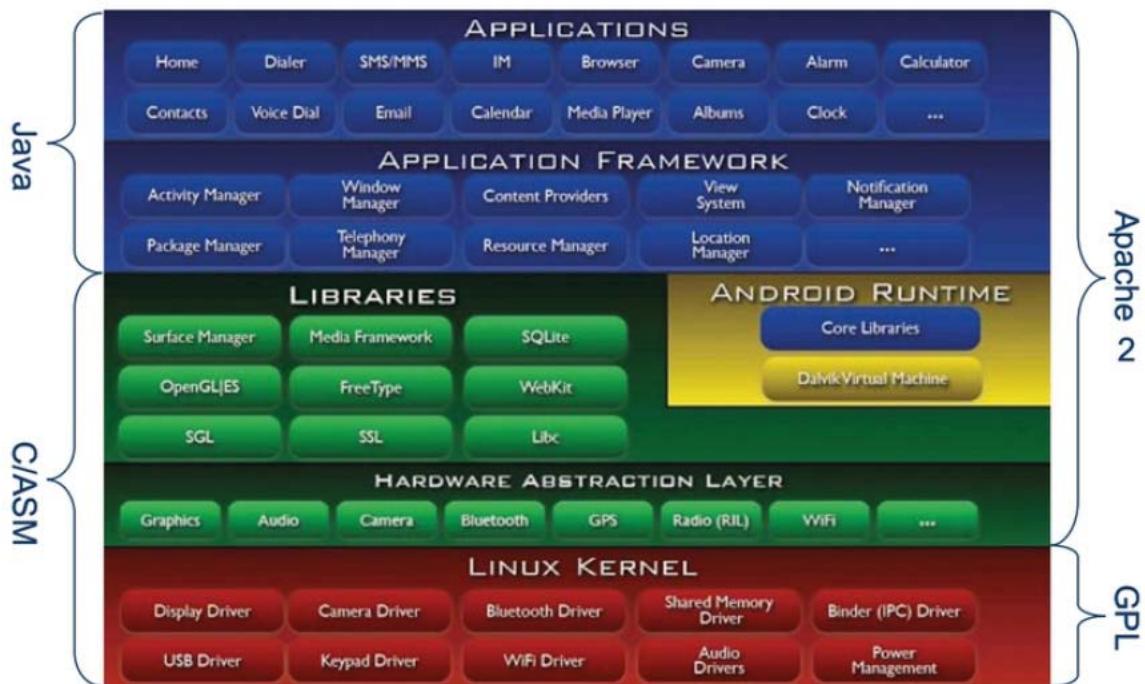
The basic advantages of having Java programming language for Android SDKs (software development kits) are given below:

1. Java is a known programming language; developers know it. Java has yet again emerged as one of the world's most popular programming language. Also, there are many engineers who specialize in Java making a vast developers' community which collaborates with each other.
2. It's harder to shoot yourself with Java than with C/C++ code since it has no pointer arithmetic.
3. It runs in a VM, so no need to recompile it for every phone out there and Java is easy to secure. This is Java's very important feature. Running on a VM (thus no recompiling) is a huge plus. Also, it easily separates processes from each other, preventing a rogue application from destroying your phone or interfering with other applications. Every App has assigned its own address.
4. As said in point number 1 above, since Java is the most popular programming language, a large number of development tools are available for developers. Java has huge open source support, with many libraries and tools that are available to make developers life easier.
5. Several mobile phones already use Java ME, so Java is known in the mobile industry and the engineering industry.
6. Also, Android as an operating system runs on many different hardware platforms including smart TVs, Android wear etc. Given that almost all VMs JIT compile down to native code, raw code speed is often comparable with native speed. A lot of delays attributed to higher-level languages are less to do with the VM overhead than other factors (a complex object runtime, 'safety' checking memory access by doing bounds checking, etc.).
7. Java allows developers to create sandbox applications, and create a better security model so that one bad App can't take down your entire OS.

Android OS Structure

Before you start coding, you will learn Android OS structure on which your app will run. Android uses Linux 2.6 kernel as the hardware abstraction, below is the Android OS structure.

The below picture shows: Java code working on the application layer of the Android operating system structure.

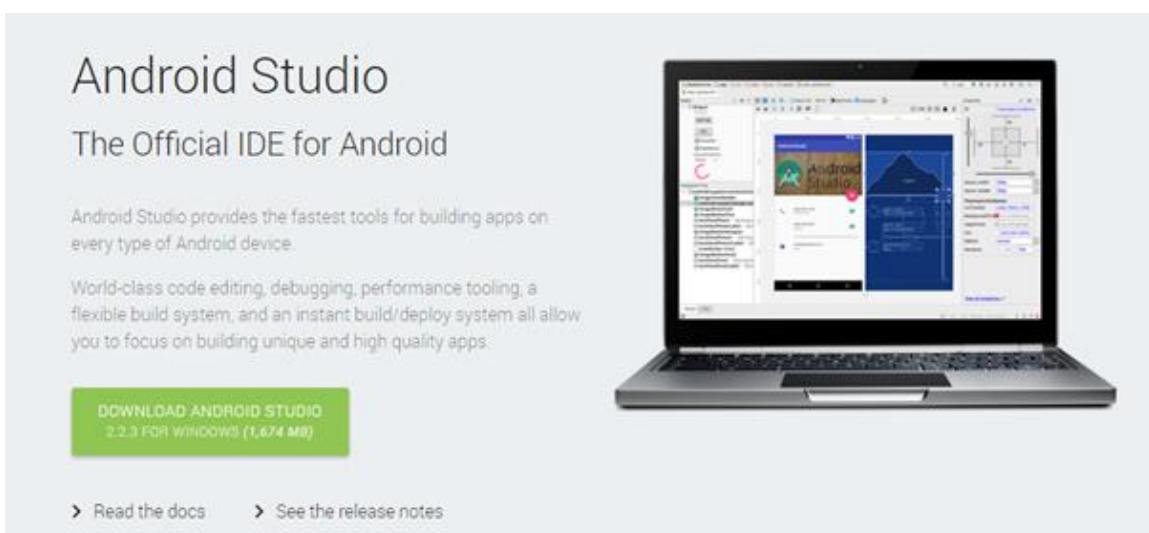


Install Android Studio

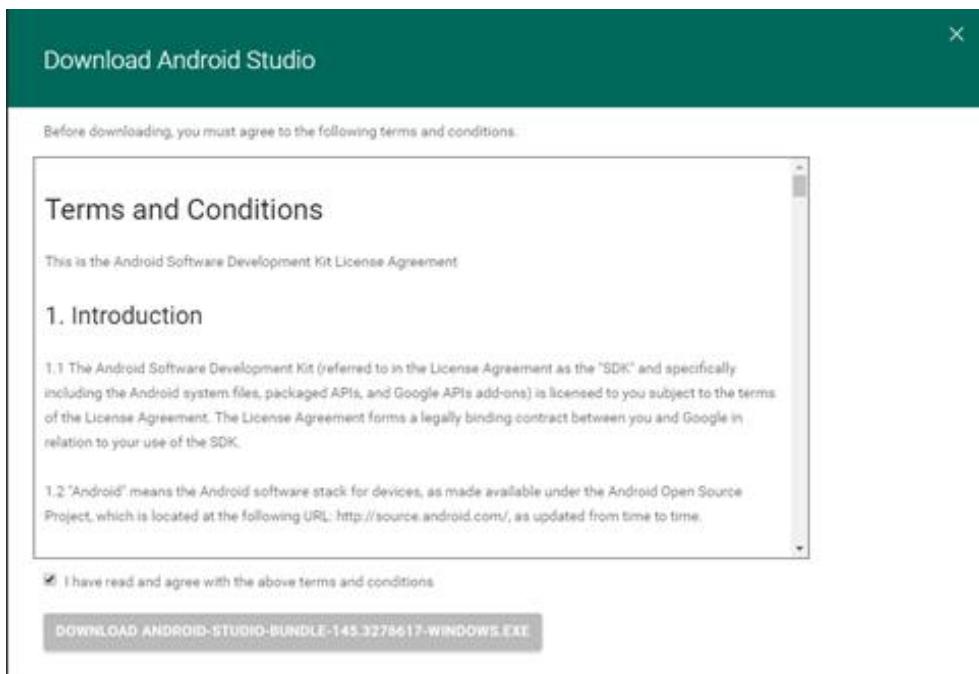
Following are the steps to install Android Studio on your system

1. Open the following link to download Android Studio

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



2. Click the “Download Android Studio” button.



3. Accept the terms and conditions then click on “Download Android Studio for Windows” button. This will start the download of an executable file called android-studio-bundle-xxx.xxxxxxx.exe, where xxx.xxxxxxx refers the build number.



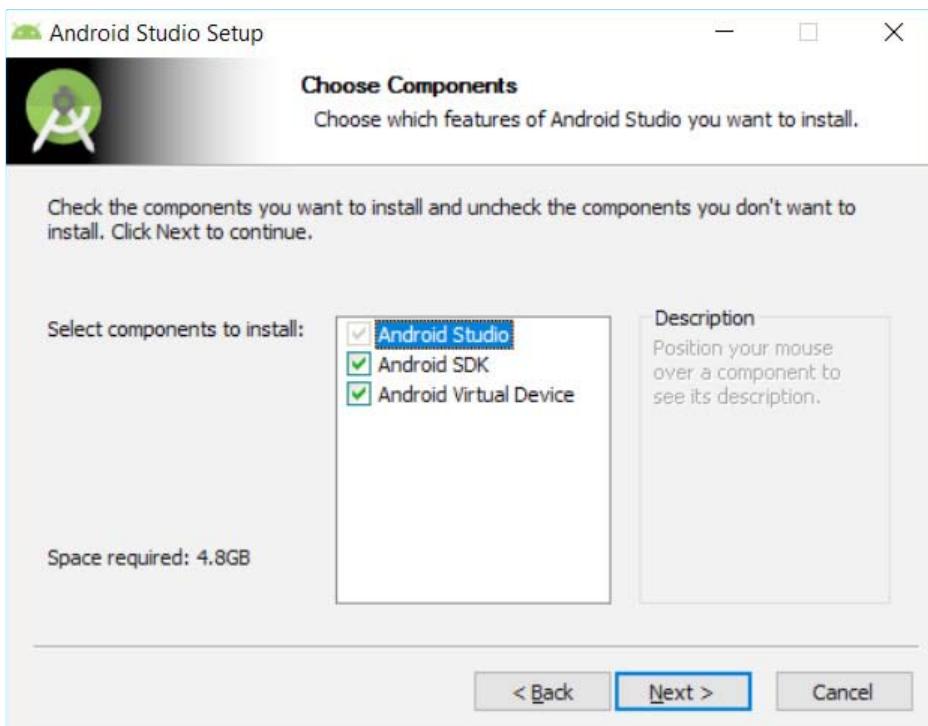
Note that the button label and file name would change if you are using a Mac machine because the download webpage will automatically detect your system.

4. Run the Windows installer file to start the installation wizard.

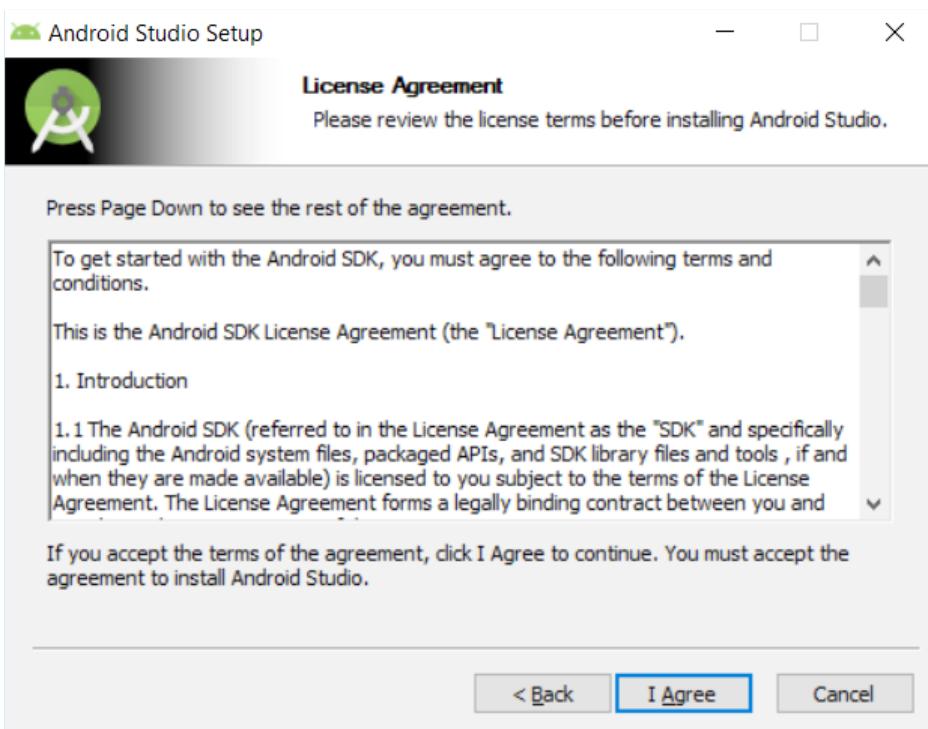


Make sure that you have Java Development Kit (JDK) 6 or higher installed on your machine to run Android studio . Android Studio now comes bundled with OpenJDK 8. Existing projects still use the JDK specified in File > Project Structure > SDK Location. You can switch to use the new bundled JDK by clicking File > Project Structure > SDK Location and checking the Use embedded JDK checkbox.

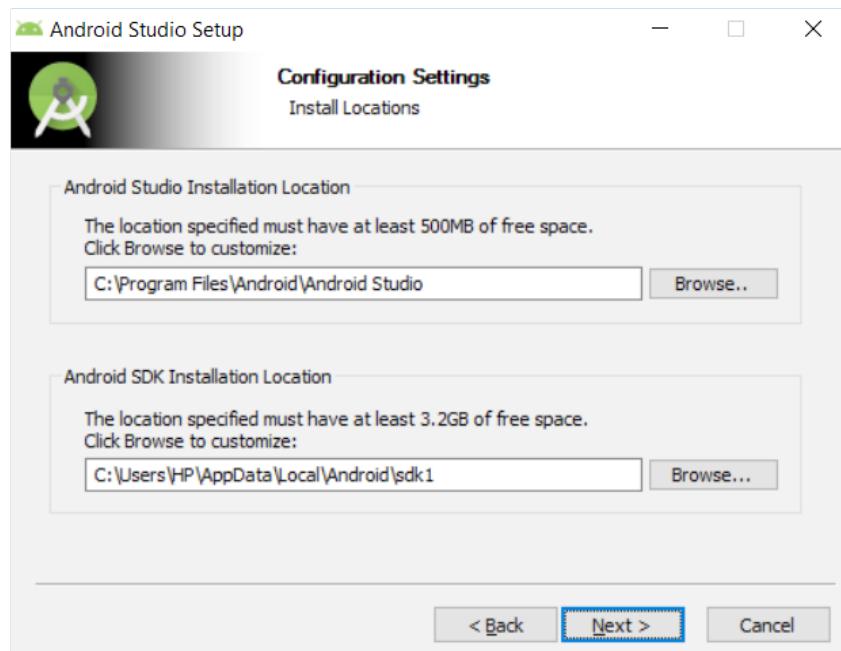
5. Select the components you have to install and click, Next.



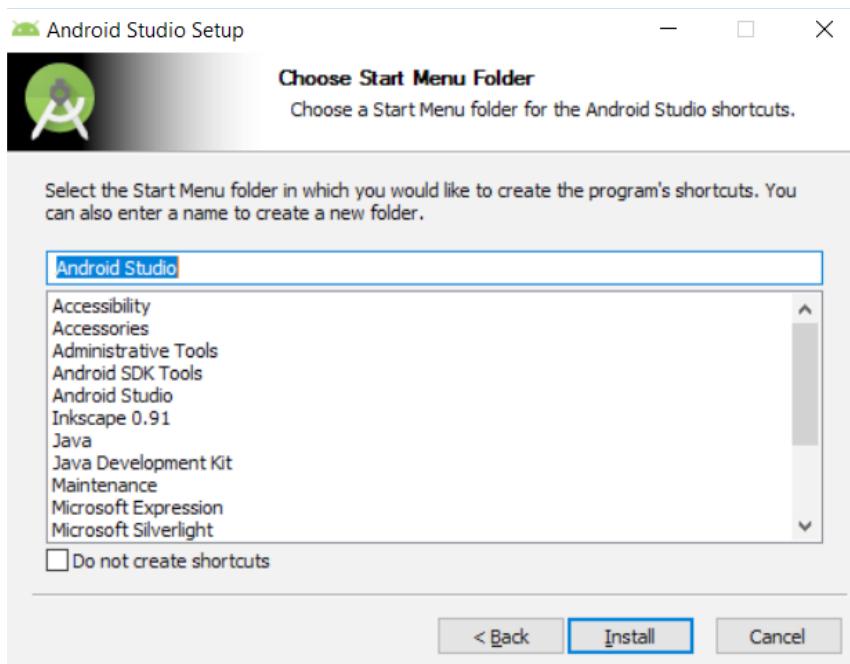
6. Click "I Agree" to move to next step.



7. Next, you should specify the location of the Android SDK folder if you already have the SDK installed on your machine. This is an important step to follow if you want to use the already installed Android SDK tools, platforms, system images...etc. instead of using the single platform and system image that come with Android Studio installation bundle.



8. Next, click "Install" to start the installation process.



9. Once installation is complete. Open the Android Studio and it will show following loading screen.



Now, Android Studio is installed on your machine and ready to be used for Android Development.

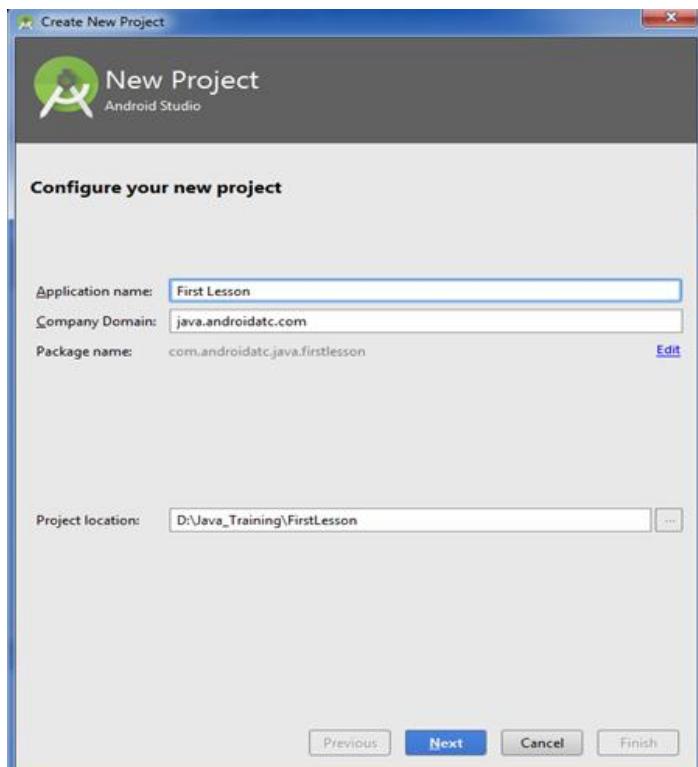
Lesson 2: Create and Run Java Projects

- Creating an Android Project (Java Project) Using Android Studio
- Writing a Java Program
- Java Methods
- Running a Java Program
- Write a Comment
- Java Variables and Their Data Type

Creating an Android Project (Java Project) Using Android Studio

It's important to know that Android studio isn't built for Java development; it is specialized for building Android applications, so you will not be able to create "New Java Project" in Android Studio. Instead, you will be creating an Android project containing a Java library by which you will be testing your Java Code.

1. Open Android Studio
2. To create an Android project Click : "Start a new Android Studio project" ,to get the following dialog box:

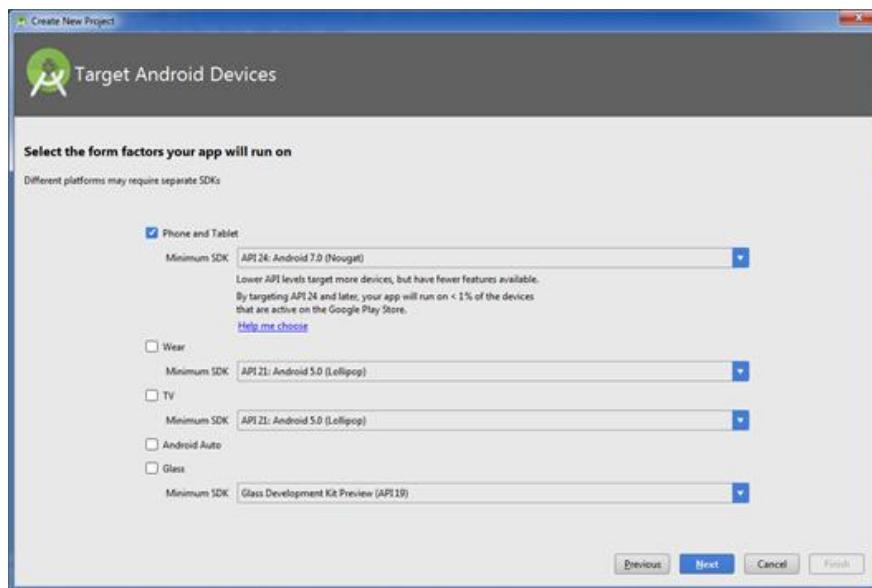


3. Write the **Application Name**: First Lesson, **Company Name** here consists of three words separated by dot and in this example project write **java.androidatc.com** and the Project location you should select where the project will be saved, then click Next.

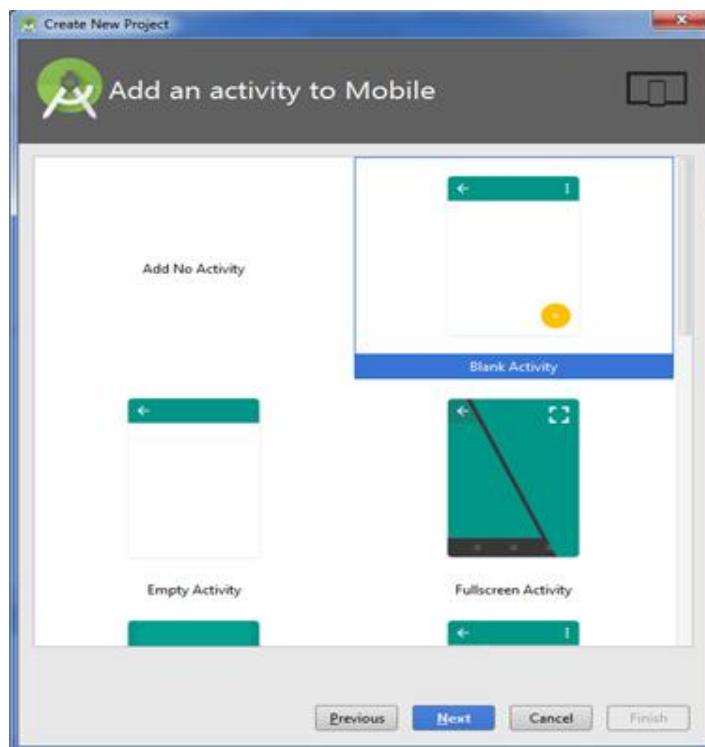


The package name uniquely identifies the app on the device; it is also unique in the Google Play store. This means that once you have published an app with this package name, you can never change it; doing so would cause your app to be treated as a brand new app, and existing users of your app will not see the newly packaged app as an update.

4. On the next dialog box, there are some details about the description of the Android device which this application will run on. You will learn about this dialog box in the Android Application Development course. Keep the default configuration and click **Next**.

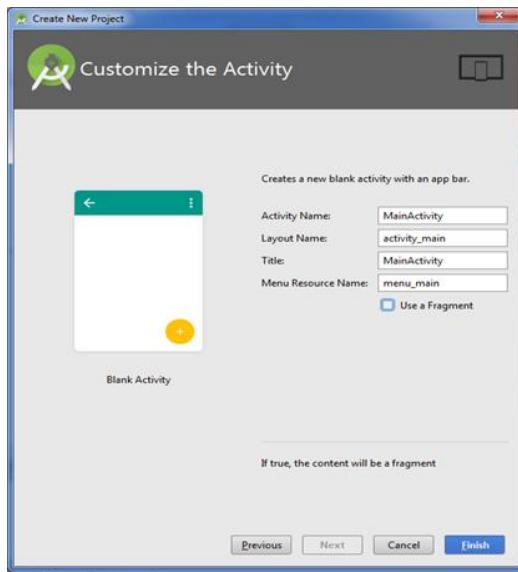


5. In the following dialog box, keep the default configuration “Blank Activity” and click **Next**.

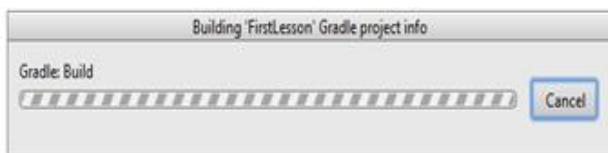


Create and Run Java Projects

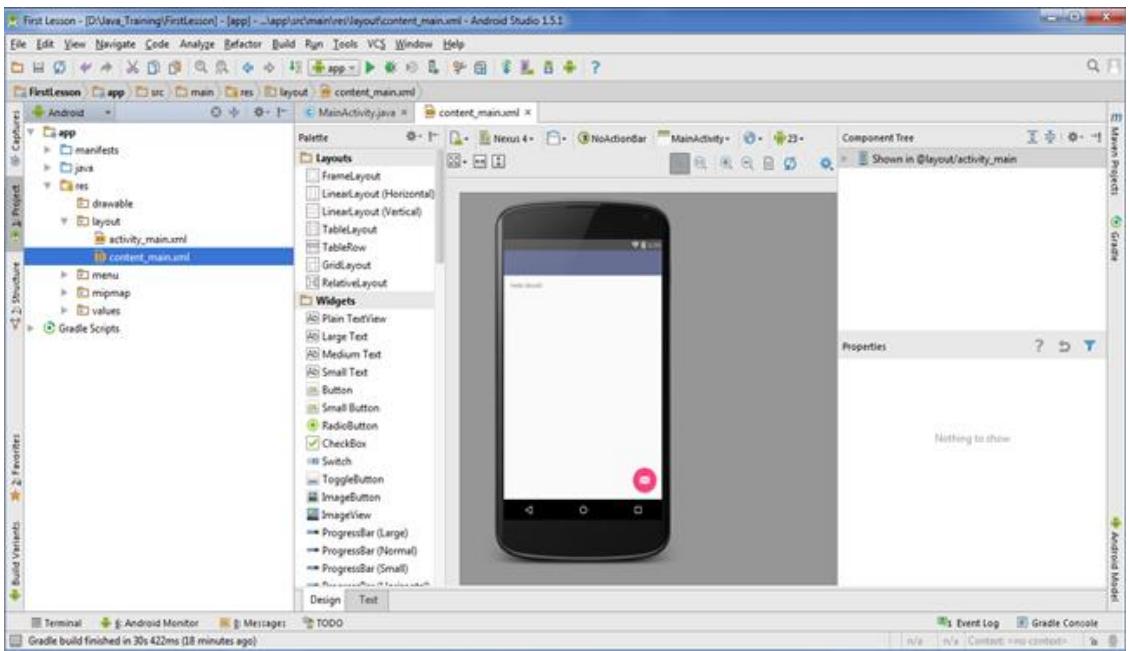
6. In the following dialog box, keep the default configuration -this configuration will be explained in details in the Android Application Development course- click **Finish**.



7. After clicking the Finish button, creating the application process starts and you will see the following process:



8. You will get the following screen:



You can check the directory where you have created the application to find the following files that have been created as it is illustrated in the following image:

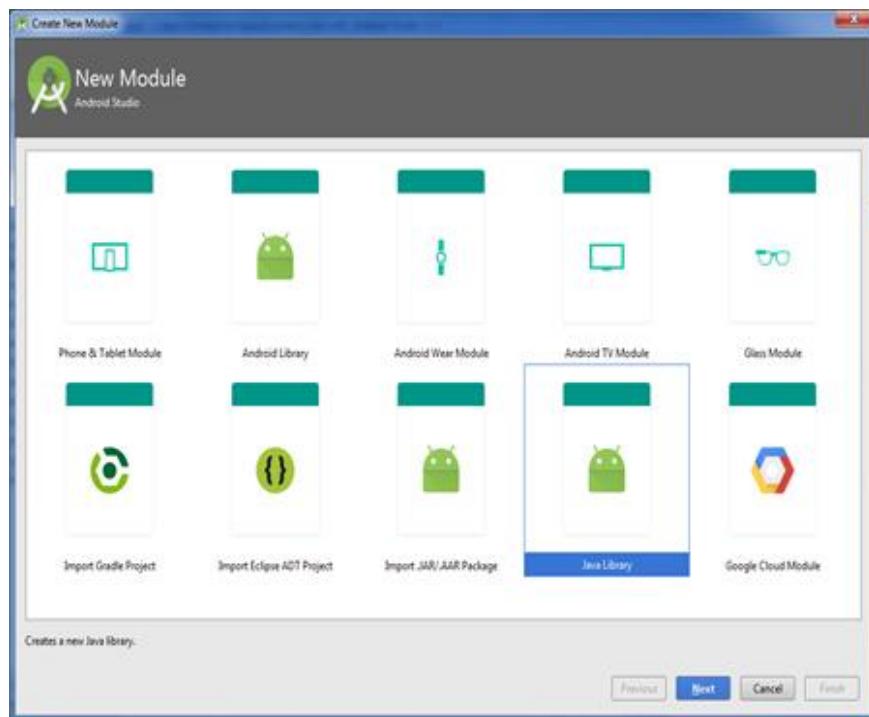
The files Path is: **D:\Java_Training\FirstLesson**

A screenshot of a Windows File Explorer window. The path is displayed as Computer > New Volume (D:) > Java_Training > FirstLesson. The folder contains the following items:

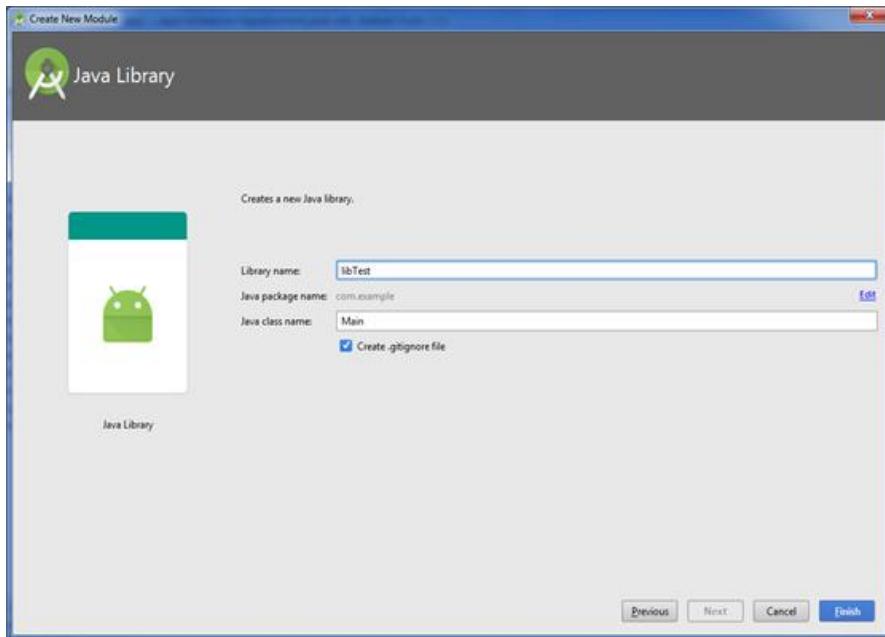
Name	Date modified	Type	Size
.gradle	8/19/2016 12:31 AM	File folder	
.idea	8/19/2016 12:31 AM	File folder	
app	8/19/2016 12:31 AM	File folder	
build	8/19/2016 6:02 AM	File folder	
gradle	8/19/2016 12:30 AM	File folder	
libtest	8/19/2016 1:23 AM	File folder	
.gitignore	8/19/2016 12:30 AM	GITIGNORE File	1 KB
build.gradle	8/19/2016 12:30 AM	GRADLE File	1 KB
FirstLesson.iml	8/19/2016 12:31 AM	IML File	1 KB
gradle.properties	8/19/2016 12:30 AM	PROPERTIES File	1 KB
gradlew	8/19/2016 12:30 AM	File	5 KB
gradlew.bat	8/19/2016 12:30 AM	Windows Batch File	3 KB
local.properties	8/19/2016 12:30 AM	PROPERTIES File	1 KB
settings.gradle	8/19/2016 1:23 AM	GRADLE File	1 KB

9. To create Java Library, click on File menu → **New** → **New Module...**

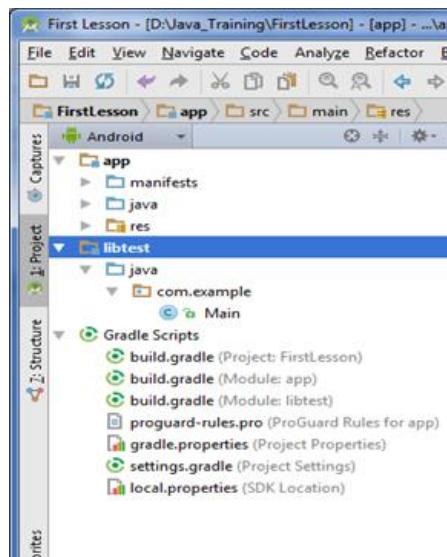
10. In the image below, we will get the below dialog box. Select "**Java Library**" and click **Next**.



11. In the following dialog box, enter the “**Library name**”: *libTest* and the “**Java class name**”: *Main* and then click **Finish**.



You will get the following illustrator on the left side of the Android Studio:



In the left side, there are three main elements where the solution explorer resides:

1. **App**
2. **Libtest (Library Name)**
3. **Gradle Scripts**

Note: You will be dealing with only the Libtest module, where you will be writing the Java Code.

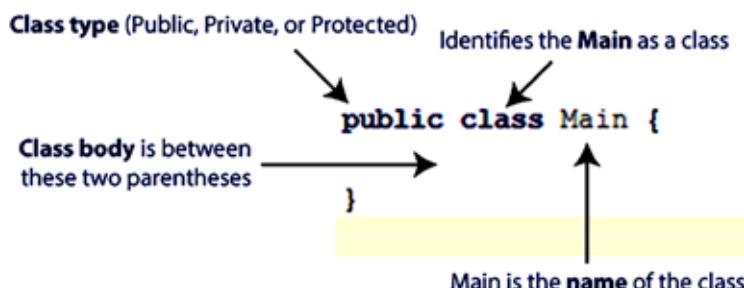
Writing a Java Program

Java program or Java project consists of group of classes; each class will achieve part of a Java program.

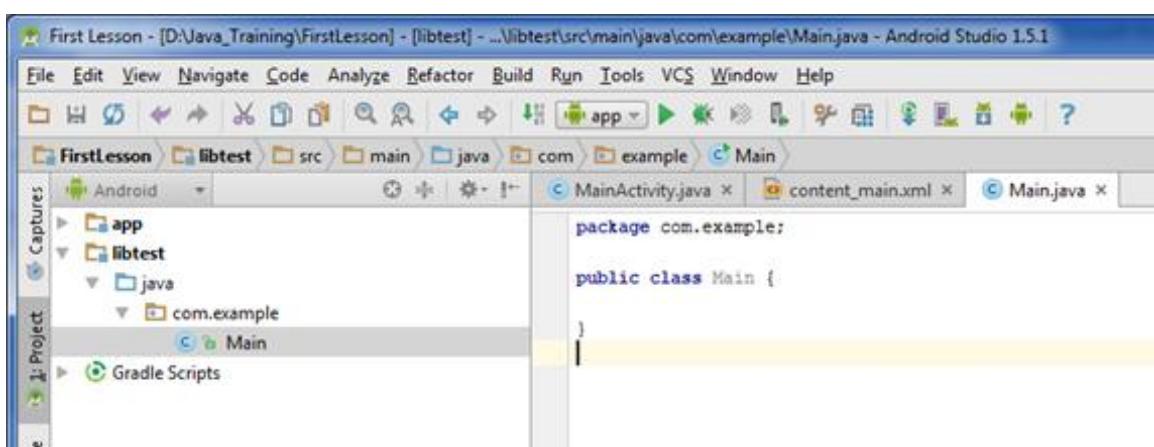
At the start of each project you should start with write class, to create a class you must consider the following three items:

1. Class type (Public, Private, or Protected).
2. Write "class" keyword to declare the class.
3. Next to "class" keyword is the name of your java class.

The following image may display what I mean more:



The following screenshot shows the Main class you have created.



Each Java project consists of group of classes, and there are three types of classes:

1. **Public** is the default class type and it exposes the class to other classes outside the package which means any class can refer to the field or call the method of the public class
2. **Private** hides from other classes within the package, which means that only the current class will have access to the field or method.

3. **Protected** is a version of public restricted only to subclasses, which means that only the current class and subclasses (and sometimes also same-package classes) of this class will have access to the field or method of the protected class.



Before starting Java with Android Studio keep in mind the following:

- 1- The red marks in Android Studio refer to compile-time errors in your Java code. A compile-time error (also known as a compiler error) is an error that prevents the computer from translating your code.
- 2- Java is case-sensitive, which means that system.out.println isn't the same as System.Out.Println

Java class starts with the following function:

```
Public static void main (String args[])
```

The following table displays what's the meaning of each part:

Public	means that main () can be called from anywhere.
Static	means that main () doesn't belong to a specific object
Void	means that main() returns no value
Main	main is the name of a function. Main () is special because it is the start of the program.
String	means the data type
Args	args is the argument passed to the function. "args" is not special; you could name it anything else and the program would work the same.

The Java program will include the following at the start:

```
Public class Main {  
  
    public static void main (String args[])  
    {  
  
    }}
```

Java Methods

If you run the above code it will not give you any result, you have to add the Java configuration, which will help building a Java program by adding methods.

A Java method is a collection of statements that are grouped together to perform an operation.

When you call the System.out.println() method, for example, the system actually executes several statements in order to display a message on the console.

Print out on screen method:

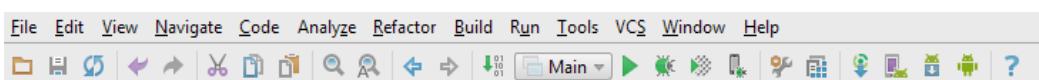
The following example will show how you can use the System.out.println() method to print out whatever is written between the two parentheses:

```
public class Main {  
  
    public static void main (String args[])  
    {  
        System.out.println("Hello Android ATC");  
    }  
}
```

When you will run the above class you will get Hello Android ATC as output on the screen.

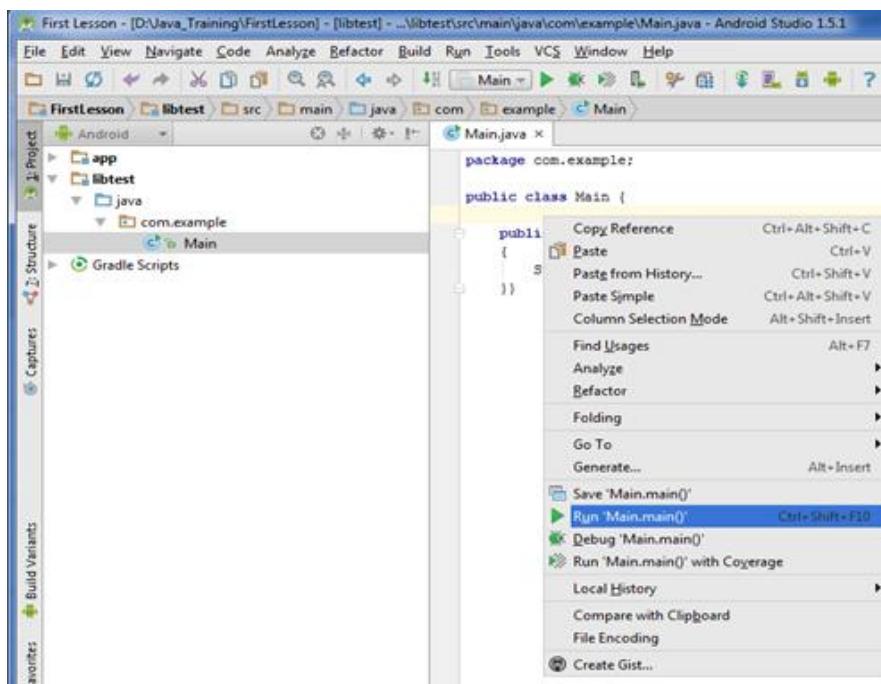
Running a Java Program

You can run a Java class by clicking the run button ► on the Android Studio tool bar.



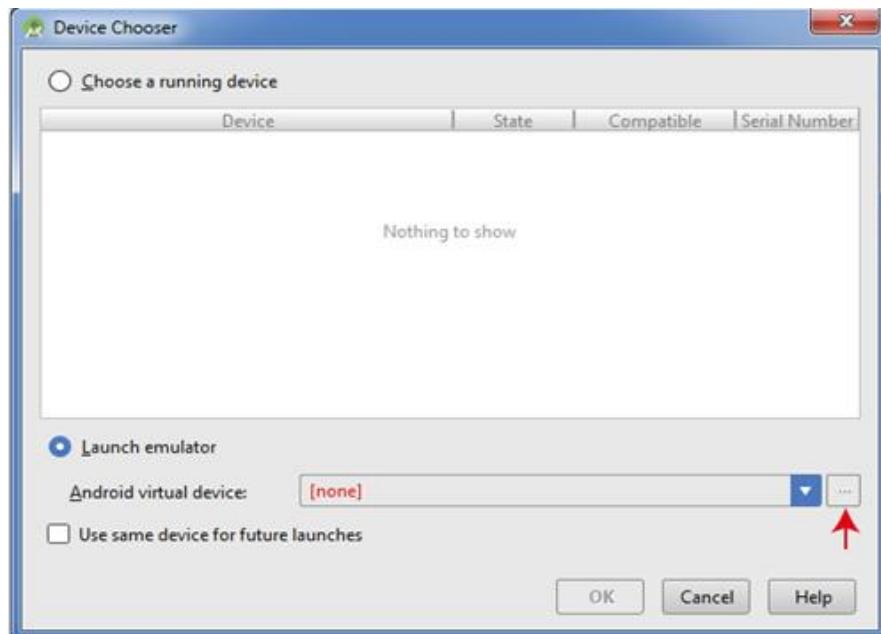
Or by press **Shift + F10** keys.

Also, you can run the class by right clicking anywhere on the class and then select Run on the shortcut menu as the following figure illustrates:



When you click the Run option first time only, you will get the following dialog box which is asking you about the device details that the Java code will run on. The Java code in Android Studio can be tested on phone, tablet, wear or TV.

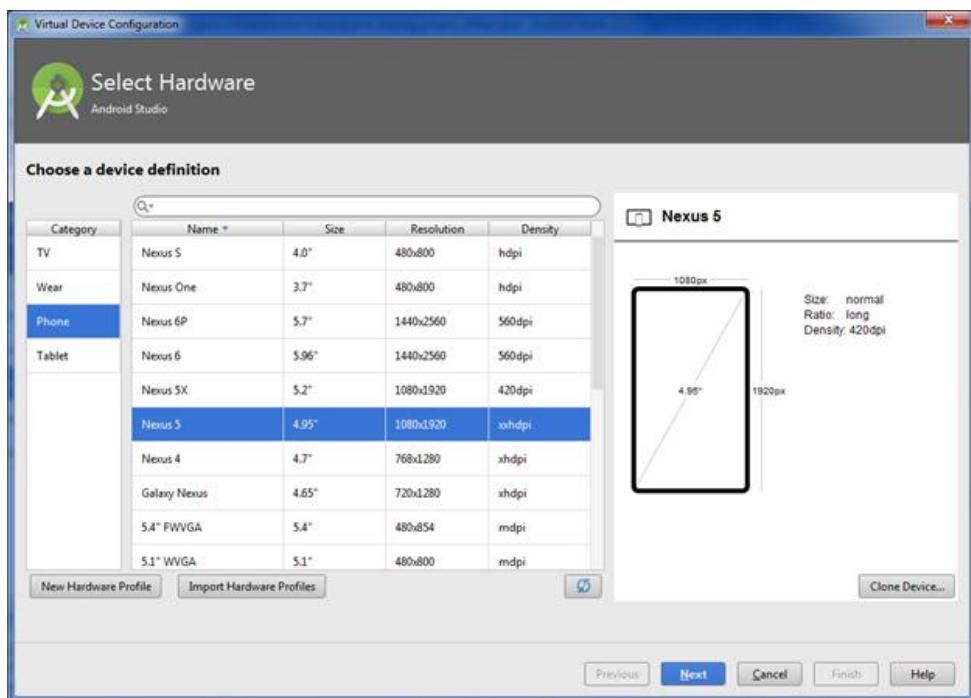
Click the wizard button which is beside the **Android virtual device**



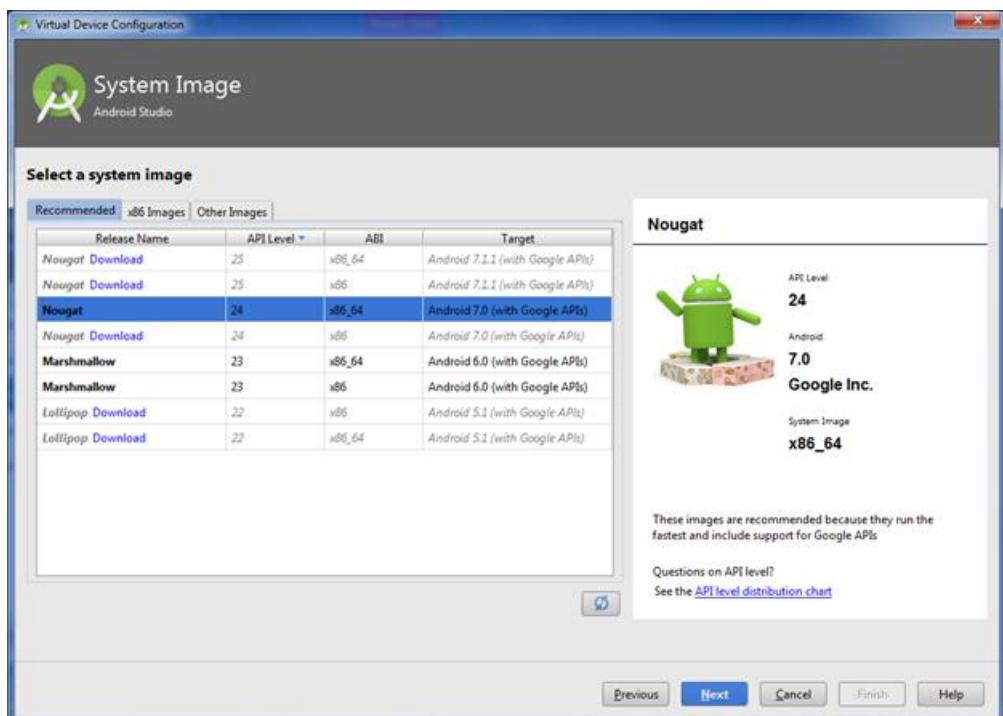
Then, click “**Create Virtual Device...**” button.



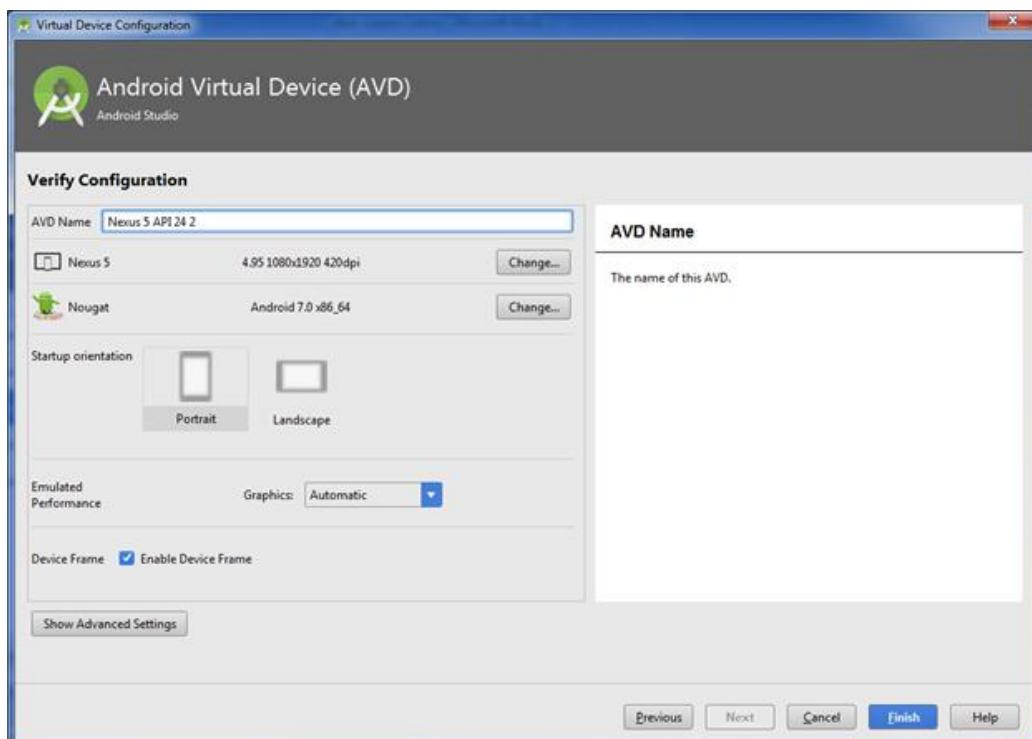
Select “**Nexus 5**” under “**Phone**” category as displayed on the hardware device, then click **Next**.



In the following screen, click **Next**.



Keep the default configuration and click **Finish** in the following screen:



You will get the following screen:



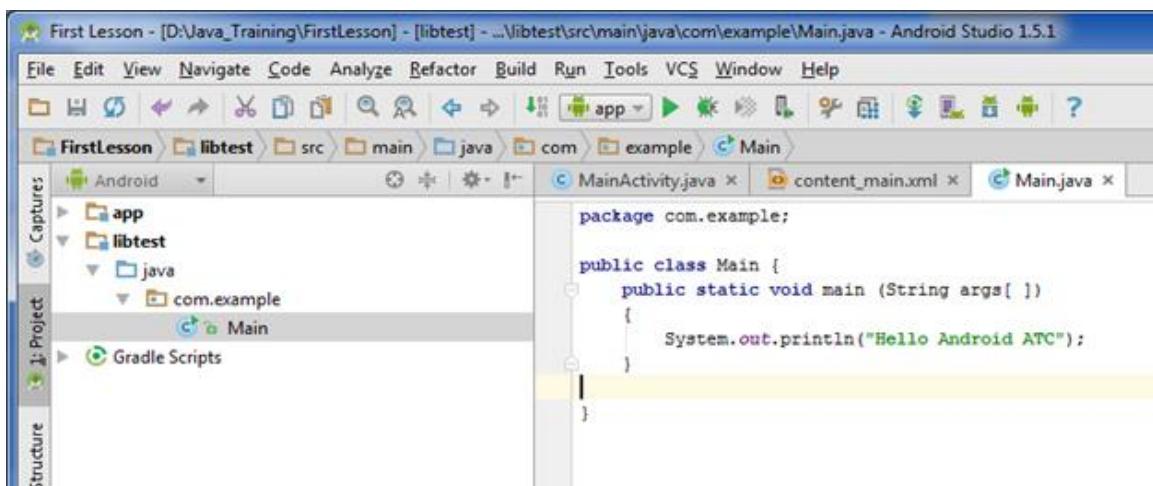
Type	Name	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
□	Nexus 5 API 23	1080 × 1920: xxhdpi	24	Android 7.0	x86_64	2 GB	▶️ 🔍 ⚙️
□	Nexus 5 API 24	1080 × 1920: xxhdpi	24	Android 7.0 (Google APIs)	x86_64	650 MB	▶️ 🔍 ⚙️
■	Nexus 5 API 24.2	1080 × 1920: xxhdpi	24	Android 7.0 (Google APIs)	x86_64	650 MB	▶️ 🔍 ⚙️
□	Nexus 6P API 24	1440 × 2560: 560dpi	24	Android 7.0 (Google APIs)	x86_64	1 GB	▶️ 🔍 ⚙️
□	Nexus 6P API 24.2	1440 × 2560: 560dpi	24	Android 7.0 (Google APIs)	x86_64	650 MB	▶️ 🔍 ⚙️
□	Nexus 6 API 24	1440 × 2560: 560dpi	24	Android 7.0 (Google APIs)	x86_64	1 GB	▶️ 🔍 ⚙️

+ Create Virtual Device... ⚡ ?

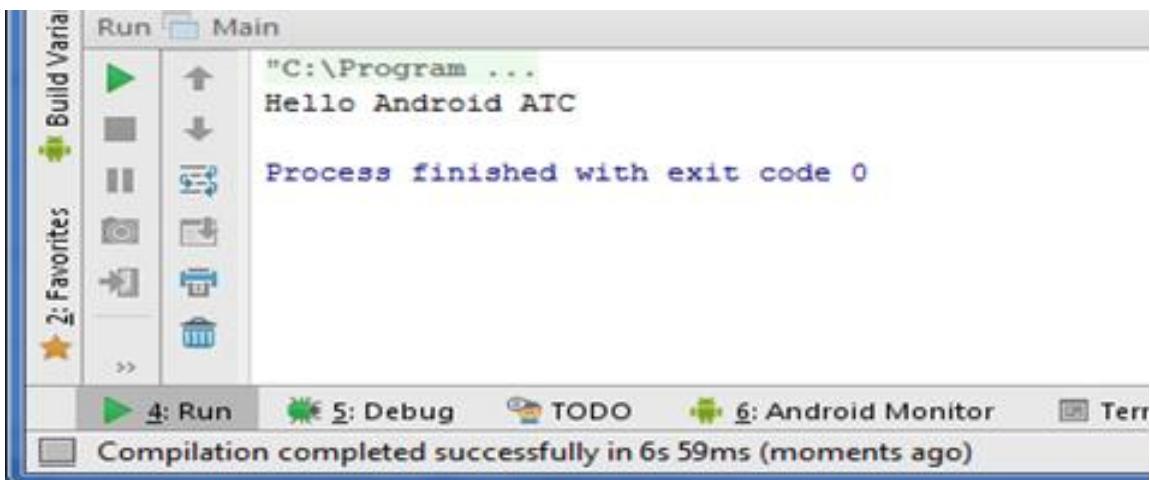
You will get the following:



Now when you will run the below class code again, it will ask again about the virtual device, this time select "**Nexus 5 API 24**" as Android virtual device and click "Ok" to see the output of your code.



You will see the following output, where you will get the “Hello Android ATC” text printed. This text can be anything you wrote above between parentheses of `System.out.println`.

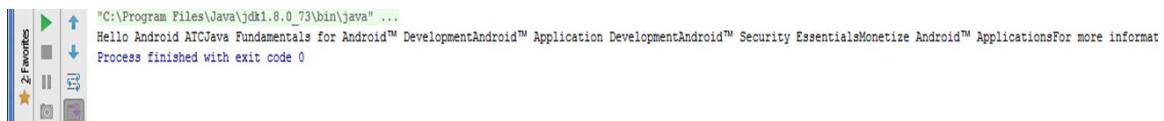


Add more text using the `System.out.print` as shown below:

```
public class Main {

    public static void main ( String[] args ) {
        System.out.print("Hello Android ATC");
        System.out.print("Java Fundamentals for Android™ Development");
        System.out.print("Android™ Application Development");
        System.out.print("Android™ Security Essentials");
        System.out.print("Monetize Android™ Applications");
        System.out.print("For more information check www.androidatc.com");
    }
}
```

When you run the Java program you will get the following result:



The screenshot shows the Android Studio interface with the 'Run' tab selected. The output window displays the command used ("C:\Program Files\Java\jdk1.8.0_73\bin\java" ...), followed by the program's output: "Hello Android ATC", "Java Fundamentals for Android™ Development", "Android™ Application Development", "Android™ Security Essentials", "Monetize Android™ Applications", and "For more information check www.androidatc.com". The message "Process finished with exit code 0" is at the bottom.

All the output of the `System.out.print` methods will be on the same line; however, if you replace `System.out.print` method with `System.out.println` in the previous code, we will get the following result:



The screenshot shows the Android Studio interface with the 'Run' tab selected. The output window displays the command used ("C:\Program Files\Java\jdk1.8.0_73\bin\java" .). The program's output is on separate lines: "Hello Android ATC", "Java Fundamentals for Android™ Development", "Android™ Application Development", "Android™ Security Essentials", "Monetize Android™ Applications", and "For more information check www.androidatc.com". The message "Process finished with exit code 0" is at the bottom.

Write a Comment

We have two ways to write the Java comment; you can define **line comments** or **block comments**.

Line comments:

Any line starting with double forward slash the Java compiler will consider it as comment, which means that this part will not run or appear to those who use this application, it will remain internal. Comments are used to write short description about different parts of the Java program.

Example:

```
public class Main {  
  
    public static void main (String args[])  
    {  
        // This my first line comment, it consists of one line !  
  
        System.out.println("Hello Android ATC");  
    }  
}
```

Block comments:

It is like a line comment but it includes more than one line and it starts with /* and ends with */. This is used to add multiple lines as comments in the code without adding // in start of each line.

Example:

```
public class Main {  
  
    public static void main (String args[])  
    {  
        /*  
        This is a block comment  
        It consists of more than one line  
        I can write here info about this part of Java program  
        */  
  
        System.out.println("Hello Android ATC");  
    }  
}
```

You can comment or uncomment any line or multiple lines of code in addition to adding description about code.

Java Variables and Their Data Type

A Java variable is a piece of memory that can contain a data value. Variables are typically used to store information which your Java program needs to do its job. A variable thus has a data type. All variables in Java must be declared before they can be used

For example:

```
Int x=1;
```

Doing so tells your program that a field (variable) named "x" exists, holds numerical data, having an initial value of "1" and the data type of this field is integer. A variable's data type determines the values it may contain, plus the operations that may be performed on it. We have two categories of data types as follow:

1. Primitive data types.
2. Composite data types.

A primitive data type uses a small amount of memory to represent a single item of data. It is preserved by the programming language and reserved keywords are used for naming the primitive data types. In addition to int, the Java programming language supports seven other data types. The following table displays the Primitive data types:

Data Type	Description	Default Value
byte	The byte data type is an 8-bit signed integer.	0
short	The short data type is a 16-bit signed integer.	0
int	The integer data type is a 32-bit signed integer. It has a maximum value of 2,147,483,647.	0
long	The long data type is a 64-bit signed integer.	0L
float	The float data type is a single-precision 32-bit floating point.	0.0f
double	The double data type is a double-precision 64-bit floating point.	0.0d
Boolean	The Boolean data type has only two possible values: true and false	'\u0000'
char	The char data type is a single 16-bit Unicode character.	null

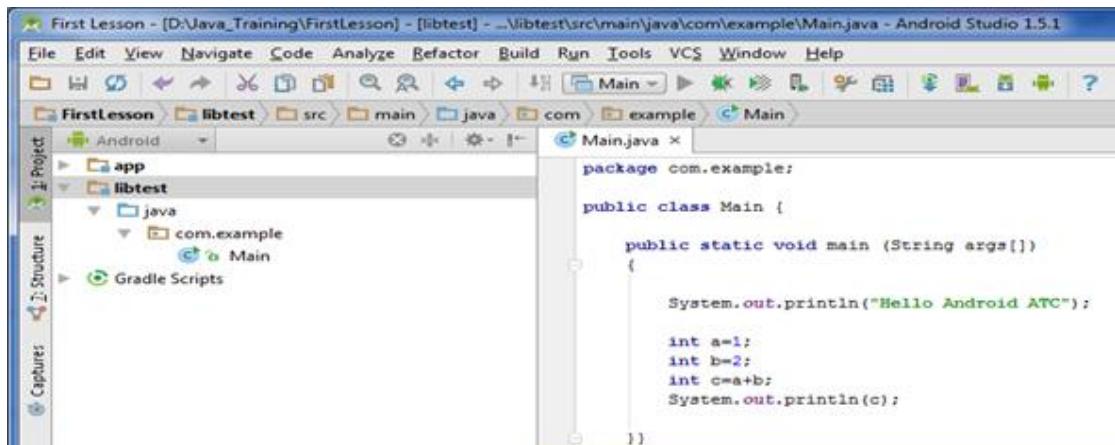
Composite data types will be explained in the next lessons.

Java has the following rules and conventions for naming variables:

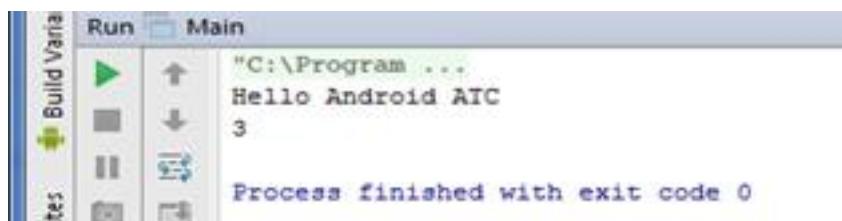
- Variable names are case-sensitive and white space is not permitted.
- Beginning with a letter, the dollar sign "\$", or the underscore character "_" is allowed
- Subsequent characters may be letters, digits, dollar signs, or underscore characters.
- By convention, you should name your variables using "camel case", i.e. if the name consists of only one word, it is all lowercase letters. If it consists of more than one word, the first letter of each subsequent word is capitalized.
- Also by convention, constants are all capitalized and contain underscore.

The following is an example which will show how to declare two integer variables (a & b) and declare the another variable c which will be the sum of a & b

```
public class Main {  
    public static void main (String args[])  
    {  
        System.out.println("Hello Android ATC");  
  
        int a=1;  
        int b=2;  
        int c=a+b;  
        System.out.println(c);  
    }  
}
```



When you will run this Java program, you will get the following result:



We can modify the previous code by modifying the print method as follows:

```
public static void main (String args[])
{
    System.out.println("Hello Android ATC");

    int a=1;
    int b=2;
    int c=a+b;
    System.out.println("a+b= " + c );

}
```

The output of run the previous code will be as follows:



The following code is another way to write the same previous Java code and it will give the same previous run result:

```
public class Main {

    public static void main (String args[])
    {
        System.out.println("Hello Android ATC");

        int a;
        a=1;
        int b;
        b=2;
        int c;
        c=a+b;
        System.out.println("a+b= " + c );

    }
}
```

Unlike the previous examples, in the code above you declared the variables and then initialized them in the next line, whereas in the previous examples you declared and initialized the variables in the same line.

One of the main points to know while dealing with variables and objects is to know their life cycle.

The life cycle of an object starts with the declaration, in this phase you will be allocating the necessary space, second phase is to initialize your variable. After that, the object will reside in the memory until you destroy your object or when it reaches the end of its life cycle. Your program consists of many scopes and your object will remain accessible within its scope.

Assignment statement and Assignment Expressions

An assignment statement in Java uses the assignment operator (=) to assign the result of an expression to a variable. In its simplest form, you can find below how the assignment is done in Java:

Variable = expression;

Example:

```
int a = (b * c) / 4;
```

A compound assignment operator is an operator that performs a calculation and an assignment at the same time. All Java binary arithmetic operators (that is, the ones that work on two operands) have equivalent compound assignment operators:

- + = Addition and assignment
- = Subtraction and assignment
- * = Multiplication and assignment
- / = Division and assignment
- % = Remainder and assignment

For example, the statement

a += 10; is equivalent to a = a + 10;

More about these operators will be explained later.

The following are more examples about the data types Java:

Boolean data type:

A boolean data type can hold only two values- true or false.

Example:

```
Public class Main {  
    public static void main ( String[] args ) {  
        boolean a=true;  
        boolean b=false;  
  
        System.out.println(a);  
    }  
}
```

The run result of this code is as follows:



char data type:

The char data type is used to store a single character such as 'a', 'Z', '9', '&', '\$' and so on. Characters may be letters, digits or special symbols. A character can also be a non-graphic symbol such as a new line or a tab. Characters are always enclosed in single quotes.

If you wish to store the value '\$' in a variable x, you would write the following statement:

```
char x = '$';
```

Example:

```
public class Main {  
    public static void main ( String[] args ) {  
        char x='$';  
        char y='A';  
        char z='a';  
        char m=6;  
        char q=20;  
        int w=m+q;  
        System.out.println(x);  
        System.out.println(y);  
        System.out.println(z);  
        System.out.println(m);  
        System.out.println(q);  
        System.out.println(w);  
    }  
}
```

The following is the output of the code written above.

The screenshot shows the Android Studio interface during a run session. The title bar indicates "Run: AVD: Nexus_5_API_23" and the selected configuration "Main". The main pane displays the command-line output of the Java application. The output shows the characters '\$', 'A', 'a', the integer '26', and the message "Process finished with exit code 0". To the left of the output pane is a vertical toolbar with various run-related icons. At the bottom of the window, there is a navigation bar with tabs for "Messages", "Terminal", "Android Monitor", "Run" (which is currently active), and "Tools".

```
"C:\Program Files\Java\jdk1.8.0_73\bin\java" ...  
$  
A  
a  
26  
Process finished with exit code 0
```

String data type:

String data type is used to store words or sentences.

Example:

```
public class Main {  
    public static void main ( String[] args ) {  
        String x="Hello Android ATC :";  
        String y="Java Fundamentals for Android ";  
        System.out.println(x + y);  
    }  
}
```

The following is the output of the code written above.



Float data type:

The type float specifies a single-precision value that uses 32 bits of storage. Single precision is faster on some processors and takes half as much space as double precision, but will become imprecise when the values are either very large or very small. Variables of type float are useful when you need a fractional component, but don't require a large degree of precision. For example, float can be useful when representing dollars and cents.

The following image shows that when you declare variable x as float without fractional components you will not get any compile time error, however; if you declare another variable with fractional component you will get red underline as shown in the image below, that can be solved by adding character "f" at the end of the number.

```
package com.androidatc.training.java_training_project_2;

public class Main {

    public static void main ( String[] args ) {
        float x=4;
        float y=2.01;
    }
}
```

The next screenshot displays that red underline has been removed after adding the character "f" at the end of the float number:

```
package com.androidatc.training.java_training_project_2;

public class Main {

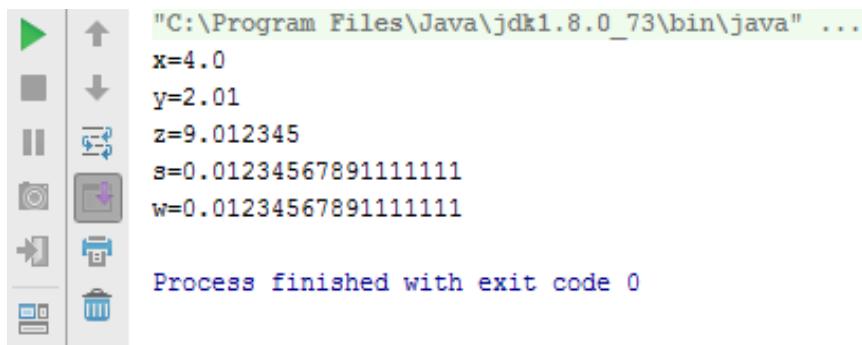
    public static void main ( String[] args ) {
        float x=4;
        float y=2.01f;
    }
}
```

Double data type:

Double precision, as denoted by the double keyword, uses 64 bits to store a value. Double precision is actually faster than single precision on some modern processors that have been optimized for high-speed mathematical calculations. All transcendental math functions, such as sin(), cos(), and sqrt(), return double values. When you need to maintain accuracy over many iterative calculations, or are manipulating large-valued numbers, double is the best choice.

The following example displays how you can declare some variables as float and double data type, and how the Java program will consider the first 32 bits of the float number and first 64 bits of the double number in its output.

The following is the output of the code written above.



Numeric type Conversions

Casting between primitive types enables you to convert the value of one type to another primitive type. It most commonly occurs with the numeric types, and there's one primitive type (Boolean) that can never be used in a cast. Boolean values must be either true or false and cannot be used in a casting operation.

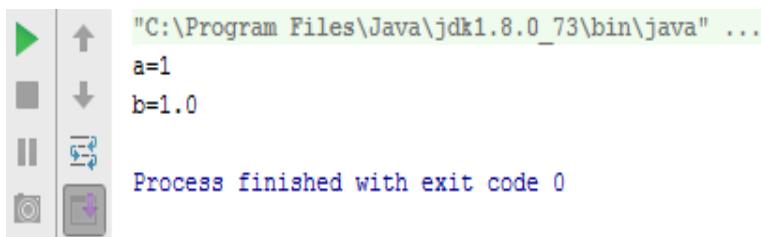
In many casts between primitive types, the destination can hold larger values than the source, so the value is converted easily. An example would be casting a byte into an int. Because a byte holds values from -128 to 127 and an int holds from -2.1 billion to 2.1 billion, there's more than enough room to cast a byte into an int.

You can often automatically use a byte or a char as an int; you can use an int as a long, an int as a float, or anything as a double. In most cases, because the larger type provides more precision than the smaller, no loss of information occurs as a result. The exception is casting integers to floating-point values; casting an int or a long to a float, or a long to a double, can cause some loss of precision.

The following example displays how you can work with different data types in the same Java program without writing any extra command to convert part of them to another data type:

```
public class Main {  
    public static void main ( String[] args ) {  
        int a=1;  
        float b=2;  
        b=a;  
        System.out.println("a=" + a);  
        System.out.println("b=" + b);  
    }  
}
```

The output of this code is the following:

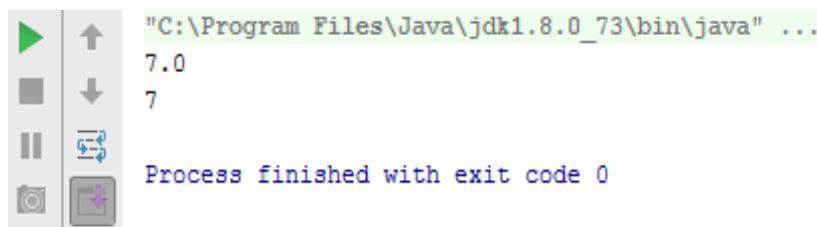


```
"C:\Program Files\Java\jdk1.8.0_73\bin\java" ...  
a=1  
b=1.0  
Process finished with exit code 0
```

We can use a Java method to establish relationship between two different data types as is illustrated in the following example:

```
public class Main {  
    public static void main ( String[] args ) {  
        Integer a = 7;  
        float b = a.floatValue();  
        System.out.println(b);  
        System.out.println(a);  
    }  
}
```

The output of this code is as follows:



```
"C:\Program Files\Java\jdk1.8.0_73\bin\java" ...
7.0
7
Process finished with exit code 0
```

Converting Numbers to Strings:

Sometimes you need to convert a number to a string because you need to operate on the value in its string form.

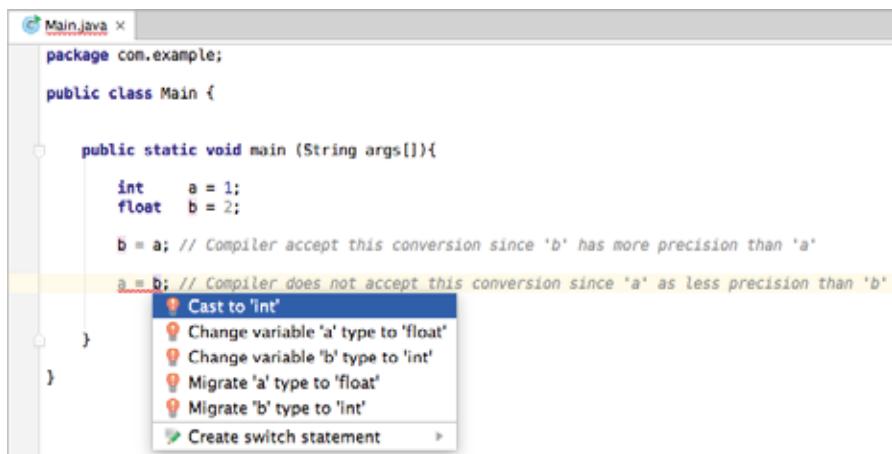
The following is an easy way to convert an integer variable to a string:

```
int i;
String s2 = String.valueOf(i);
```

Also, the class method, `toString()`, converts its primitive type to a string. For example:

```
int i;
double d;
String s3 = Integer.toString(i);
String s4 = Double.toString(d);
```

The following screenshot displays how the Android Studio as compiler will allow us to convert the value of one type to another primitive type:



Lesson 3: Control Flow Statements

- Introduction
- IF – Else Statement
- If...Else and Else...If... Statement
- If Else and Logical Operators
- Switch Statement
- While Loop
- While and Do-while Loop
- For Loop
- The Break Statement
- The Continue Statement

Introduction

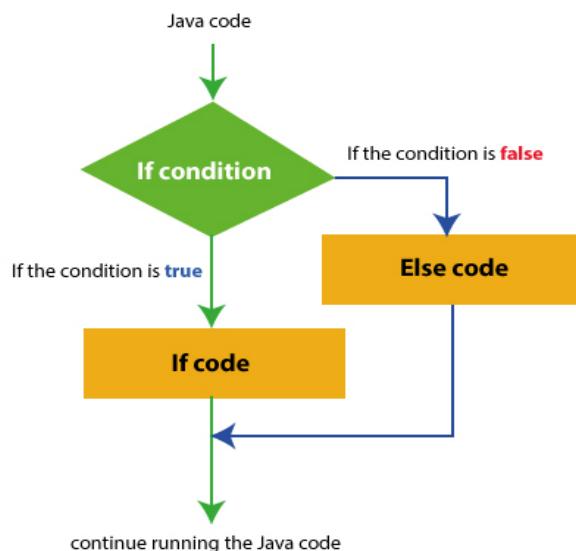
The statements inside your source files are generally executed from top to bottom, in the order that they appear. **Control flow statements**; however, break up the flow of execution by employing decision making, looping, branching, and enabling your program to conditionally execute particular blocks of code. This section describes the decision-making statements (if-then, if-then-else, switch), the looping statements (for, while, do-while), and the branching statements (break, continue, return) supported by the Java programming language. This lesson will explain how these control flow statements can control the work of Java programs.

The following table includes some Boolean operators that will help in control flow of the Java program:

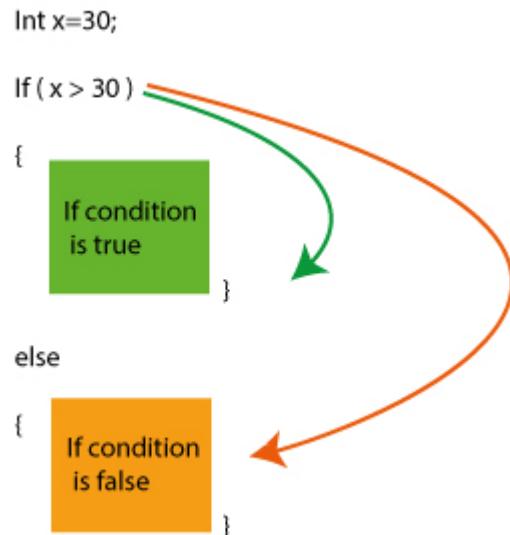
Operator	Name
==	Equal to
!=	Not Equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

If – Else Statement

The if- else statement is the most basic of all the control flow statements. It tells your program to execute a certain section of code only if a particular test evaluates to true.



The If statement will include Boolean condition like If ($x > 5$), here in this example if x is greater than 5 the program will execute a special part of Java program; otherwise, if x is not greater than 5 the Java program will execute the part which belongs to else statement.



In the following example $x=30$, assume that If statement includes a condition $x < 20$, if this condition is true the program will execute the program part, which belongs to the If statement; which means it will print "This is if statement". However, here $x=30$ meaning that the If condition $x < 20$ is Not true; therefore, the Java workflow will move to execute the part which belongs to else statement. So, in this case, it will print "This is else statement".

```
package com.example;

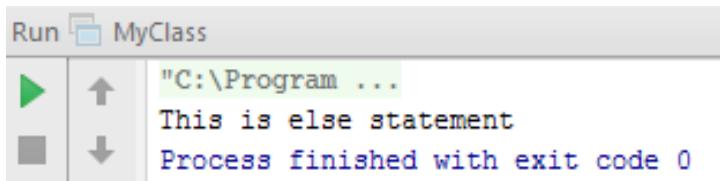
public class MyClass {
    public static void main(String args[]){
        int x = 30;

        if( x < 20 ){

            System.out.print("This is if statement");
        }

        else{
            System.out.print("This is else statement");
        }
    }
}
```

The result of this code is as follows:



```
Run MyClass
[C:\Program ...
This is else statement
Process finished with exit code 0
```

If...Else and Else...If... Statement:

If statement is also called “if-then” statement. This statement is the most basic of all the control flow statements. It tells your program to execute a certain section of code only if a particular test evaluates to true.

An If-statement can be followed by an optional else-if-else statement, which is very useful to test various conditions using a single if-else-if statement.

The following example displays how if-else and else-if statements work:

```
package com.example;

public class MyClass {
    public static void main(String args[]){
        int x = 30;

        if( x == 10 ){System.out.println("Value of X is 10");}
//X is not equal 10 because it is equal 30

        else if( x > 20 ){System.out.println("Value of X is > 20");}
//Yes, x greater than 20 because it is equal to 30

        else if( x < 15 ){System.out.println("Value of X is < 15");}
//X is not less than 15 because it is equal to 30

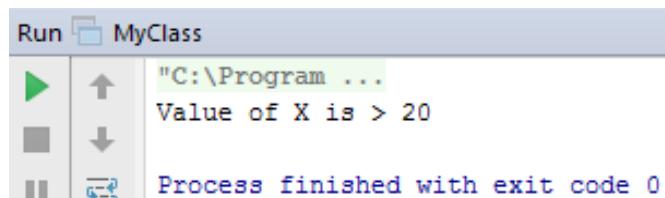
        else if( x < 13 ){System.out.println("Value of X is < 13");}
//X is not less than 13 because it is equal to 30

        else if (x > 17){System.out.println("Value of X is > 17");}
//Yes, x greater than 17 because it is equal to 30

        else {System.out.println("x is not equal to 10);}

// This else statement belong to first if statement "if(x == 10)"
    }
}
```

The result of this program will be as follows:



```
Run MyClass
C:\Program ...
Value of X is > 20
Process finished with exit code 0
```

In the previous Java application we have two else if statements that have true results, but because the Java compiler reads the code from top to bottom, line by line. When any of if or else if statements have true value the compiler will move with the response action outside the last else and complete any remain part of the Java code.

Here in this program, when the Java compiler found true value in the `else if(x > 20)`, it operates the action for this else-if-statement “`System.out.println("Value of X is > 20")`” and then it avoids, another else if statements even if it has true value, and then it moved to whatever is the last else statement{`System.out.println("x is not equal to 10")`}which belongs to the first if statement.

If Else and Logical Operators

We can use logical operators like AND “`&&`” and OR “`| |`” If you have two conditions that must be true or one of them is enough to be true respectively. Here, by using the logical operators IF statement you can test more than one condition at the same time.

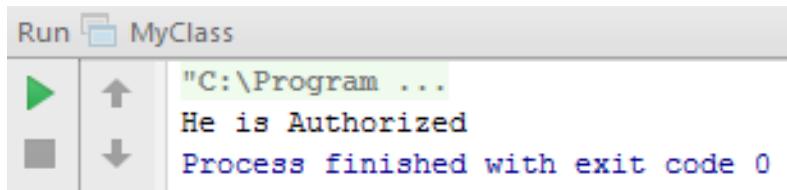
The following example using “`&&`” logical operator to guarantee the two conditions must be true:

```
package com.example;
public class MyClass {
    public static void main(String args[]){
        int Age = 18;
        int DOB = 1998;

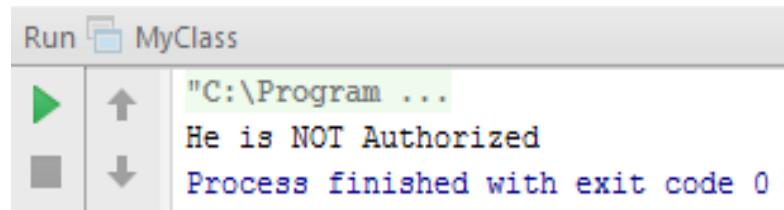
        if( Age >= 18 && DOB>=1998 ){System.out.print("He is Authorized");}
        else{
            System.out.print("He is NOT Authorized");
        }

    }
}
```

Here in the above code, the two conditions are true, therefore the result will be as follows:



If you change the age value to be 16 in "int Age = 16;" then run the application again you will get the following result:



We can replace the AND operator "&&" with OR operator "||" in the same Java application, where the application will be as follows":

```
package com.example;

public class MyClass {

    public static void main(String args[]){
        int Age = 16;
        int DOB = 1998;

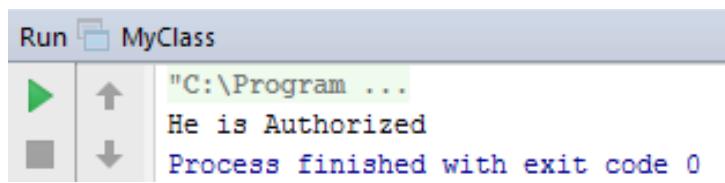
        if( Age >= 18 || DOB>=1998 ) {

            System.out.print("He is Authorized");
        }

        else{
            System.out.print("He is NOT Authorized");
        }

    }
}
```

Here since we are using OR operator "||" and one of the two conditions is true the result will be as follows:



Switch Statement:

The switch statement can have a number of possible execution paths. A switch works with the byte, short, char, and int data types. It also works with the String class, and a few special classes that wrap certain types: Character, Byte, Short, and Integer.

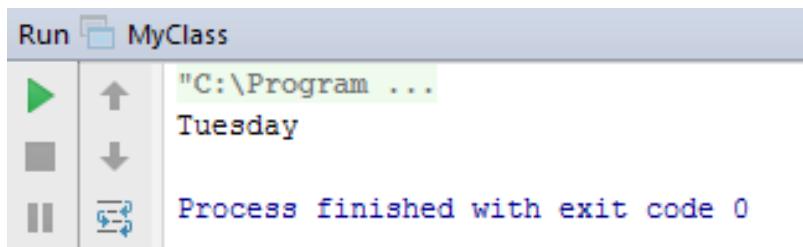
The following code example, declares an int named day whose value represents a day. The code displays the name of the day, based on the value of day, using the switch statement.

```
package com.example;

public class MyClass {
    public static void main(String args[]){

        int day = 3;
        String DayString;
        switch (day) {
            case 1: DayString = "Sunday";
            break;
            case 2: DayString = "Monday";
            break;
            case 3: DayString = "Tuesday";
            break;
            case 4: DayString = "Wednesday";
            break;
            case 5: DayString = "Thursday";
            break;
            case 6: DayString = "Friday";
            break;
            case 7: DayString = "Saturday";
            break;
            default: DayString = "Invalid Day";
            break;
        }
        System.out.println(DayString);
    }
}
```

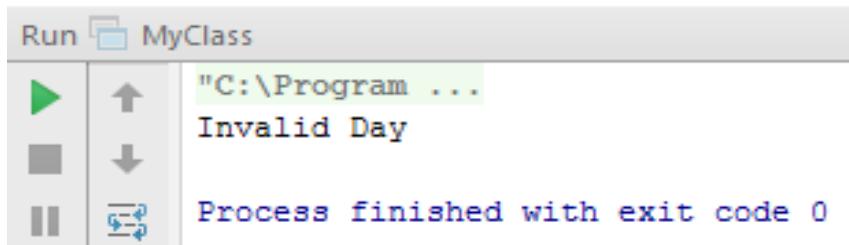
The output of run this code is as follows:



```
Run MyClass
[C:\Program ...
Tuesday
Process finished with exit code 0
```

The body of a switch statement is known as a switch block. A statement in the switch block can be labeled with one or more case or default labels. The switch statement evaluates its expression, and then executes all statements that follow the matching case label.

If we have changed the int day = 3; to int day = 9; since we don't have case 9, which means that the default part at the last line of switch statement will be working. And the run result will be as follows:

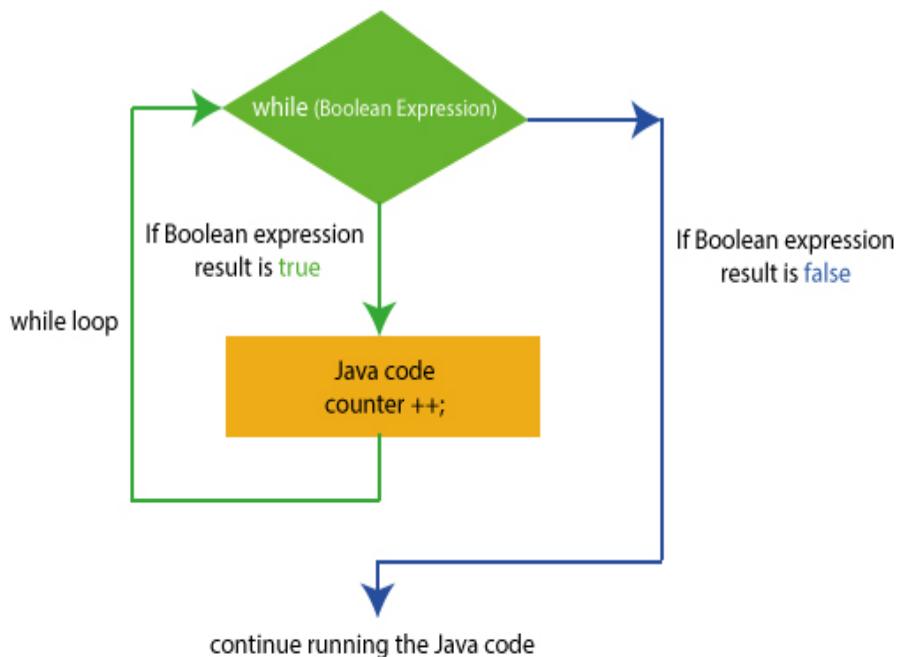


```
Run MyClass
[C:\Program ...
Invalid Day
Process finished with exit code 0
```

While Loop:

A while statement depends on a Boolean expression, which must return a Boolean value. If the expression evaluates to true, the while statement executes the statement(s) in the while block. The while statement continues testing the expression and executing its block until the expression evaluates to false.

```
while (Boolean Expression)
{
    while block (Java Statements)
}
```



The following example shows using the while statement to print the values from 0 through 10:

```
package com.example;

public class MyClass {
    public static void main(String args[]){
        // While Loop example by “Android ATC”

        int count =0;
        while (count <11)

        {
            System.out.println(“count = “ + count);
            count++;
        }

        System.out.println(“Finish”);
    }
}
```

The result for this code is as follows:

```
"C:\Program Files\Java\jdk1.8.0_73\bin\java" ...
count = 0
count = 1
count = 2
count = 3
count = 4
count = 5
count = 6
count = 7
count = 8
count = 9
count = 10
Finish

Process finished with exit code 0
```

0: Messages 1: Terminal 2: Android Monitor 3: Run 4: TODO

Do-while Loop

The difference between do-while and while is that do-while evaluates its expression at the bottom of the loop instead of the top. Therefore, the statements within the do block are always executed at least once.

```
do {
    statement(s)
} while (expression);
```

The following example using the do-while statement to print the values from 0 through 10:

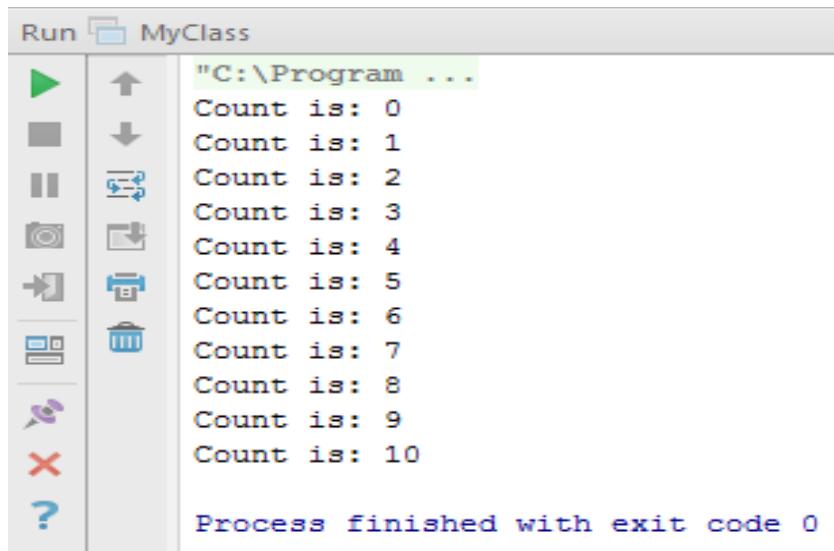
```
package com.example;

public class MyClass {
    public static void main(String args[]){
        // do - While Loop example by "Android ATC"

        int count = 0;
        do {
            System.out.println("Counter is: " + count);
            count++;
        }
        while (count < 11);

    }
}
```

The result for this code is as follows:



A screenshot of a Java IDE showing the output of a program named 'MyClass'. The output window displays the following text:
"C:\Program ...
Count is: 0
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Count is: 5
Count is: 6
Count is: 7
Count is: 8
Count is: 9
Count is: 10
Process finished with exit code 0

For Loop

A for statement provides a compact way to iterate over a range of values. Programmers often refer to it as the “for loop” because of the way in which it repeatedly loops until a particular condition is satisfied. The general form of the for statement can be expressed as follows:

```
for (initialization; termination; increment)
{
    statement(s)
}
```

When using this version of the for statement, keep in mind that:

- The initialization expression initializes the loop; it's executed once, as the loop begins.
- When the termination expression evaluates to false, the loop terminates.
- The increment expression is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment or decrement a value.

The following example is using the for loop statement to print the values from 0 through 10:

```
package com.example;

public class MyClass {

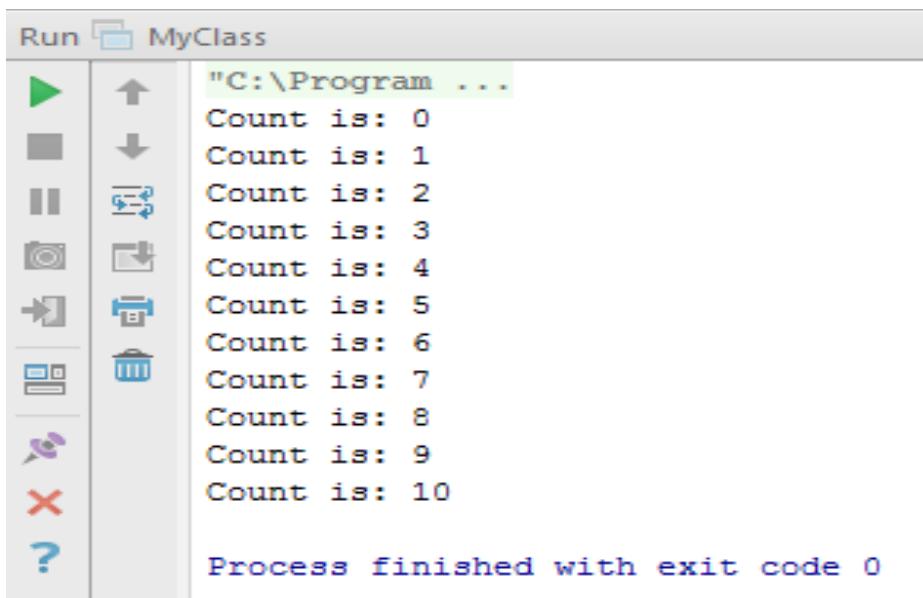
    public static void main(String args[]) {

        // For Loop example by "Android ATC"

        for (int count = 0; count < 11; count++)
        {
            System.out.println("Count is: " + count);

        }
    }
}
```

The runt result for this code is as follows:



The screenshot shows the 'Run' window in Android Studio with the title 'MyClass'. The window displays the command-line output of the Java application. The output consists of eleven lines, each showing the value of the 'count' variable from 0 to 10, followed by the message 'Process finished with exit code 0' at the bottom.

Output
"C:\Program ...
Count is: 0
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Count is: 5
Count is: 6
Count is: 7
Count is: 8
Count is: 9
Count is: 10
Process finished with exit code 0

The Break Statement

The break statement allows you to exit a loop from any point within its body, bypassing its normal termination expression. When the break statement is encountered inside a loop, the loop is immediately terminated, and program control resumes at the next statement following the loop. It can be used in any type of loop like While, do-While, and for loop.

The following example displays using the break statement, where when the count value will be equal to 5, the break statement will work and the while loop is immediately terminated, and program control resumes at the next statement following the while loop:

```
package com.example;

public class MyClass {
    public static void main(String args[]){

        // Break example by "Android ATC"

        int count =0;
        while (count <11)
        {
            System.out.println("count = " + count);

            count++;

            if (count==5) break;

        }
    }
}
```

The runt result for this code is as follows:

```
Run MyClass
[C:\Program ...]
count = 0
count = 1
count = 2
count = 3
count = 4
Process finished with exit code 0
```

The Continue Statement

The continue statement will cause control to go directly to the test condition and then continue the looping process. In the case of the for-loop, the increment part of the loop continues. One good use of continue is to restart or skip a statement sequence when an error occurs.

In the following example, the result of the next Java application will print the count value from 0 till 7 except 4:

```
package com.example;

public class MyClass {
    public static void main(String args[]) {

        // For Loop example by "Android ATC"

        for (int count = 0; count < 8; count++) {

            if(count==4)continue;

            System.out.println("Count is: " + count);

        }
    }
}
```

The runt result for this code is as follows:

```
"C:\Program ...  
Count is: 0  
Count is: 1  
Count is: 2  
Count is: 3  
Count is: 5  
Count is: 6  
Count is: 7  
Process finished with exit code 0
```

Lesson 4: Methods, Arrays and Java Class

- Introduction
- Method Structure
- Call Method by Value
- Call Method by Reference
- Arrays
- Enter Data to a Java Program
- Object Oriented Programming (OOP) concepts
- Java Class

Introduction

A Java method is a collection of statements that are grouped together to perform an operation. When you call the `System.out.println()` method, for example, the system actually executes several statements in order to display a message on the console.

Each method has its own name and when this name is encountered in a program, the execution of the program branches to the body of that method. When the method is finished execution, it returns to the area of the program code from where it was called, and the program continues to the next line of code.

The following example displays using the method in a simple way:

```
package com.example;

// Java Methods Created by Android ATC

public class MyClass {

    public static void main(String args[]) {
        ATC();
        ATC();
        ATC(); }

    static void ATC()
    {
        System.out.println("Welcome to Android ATC");

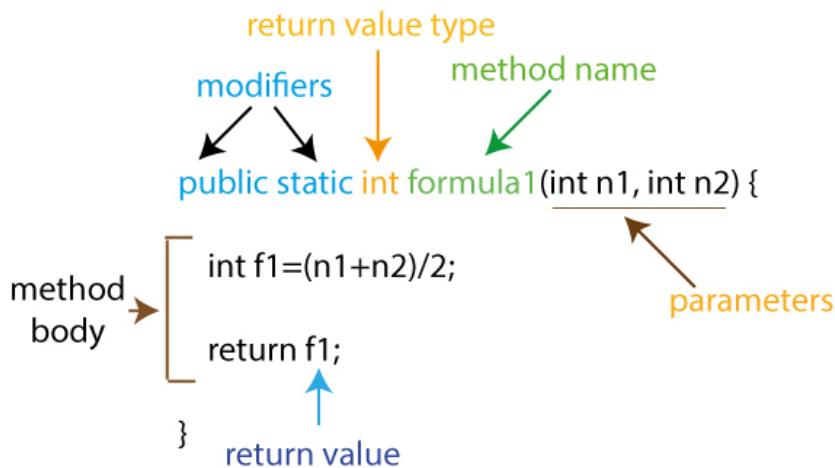
        return;
    }
}
```

In this example, the method name was ATC and that method was supposed to print "Welcome to Android ATC" text each time it was called. In this example, it was called **three times**; therefore the result of this code is the following snapshot:



Method Structure

Below is the structure of Java method:



Public (modifiers): It will be explained in the following lessons.

Static (modifiers): It is a keyword, which is supposed to set the Method to a static method (by default all methods are not static). The static keyword can be used while declaring a function, variables or even Classes. It will be explained in the following lessons.

Return Type: It can be any primitive or user data type. In case this function did not return any value, you can use the keyword “**void**” to indicate that this function does not return anything.

Method parameters: Every function can have zero, one or more parameters, which can be used in your method body.

Return value: It represents the value returned by your method.

In the following example, the method name is “`formula1`” which includes two integer numbers `n1` and `n2` as parameters. These numbers will be used in the “`fromula1`” method to achieve the following calculation “ $(n1+n2)/2$ ”. We can call this method by using its name “`formula1`” to use it to make the same calculation for different values. Think of a method as a subprogram that acts on data and often returns a value.

When we will run the following Java code:

```
package com.example;

// Java Methods Created by Android ATC

public class MyClass {

    public static void main(String args[]) {
        int a = 10;
        int b = 6;
        int c = equation1(a, b);
        System.out.println("Equation Value = " + c);

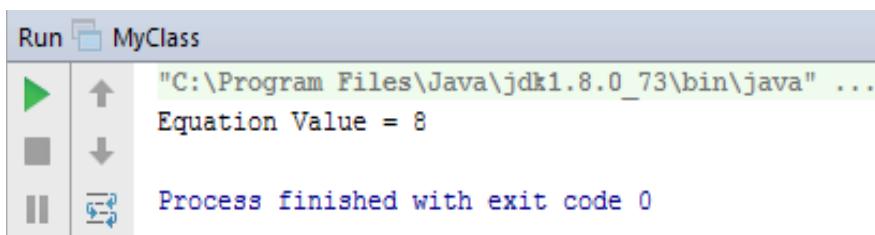
        /** returns the result of the below equation*/

        public static int equation1(int n1, int n2) {

            int t=(n1+n2)/2;

            return t;
        }
    }
}
```

We will get the following result:



Return command, which is highlighted in yellow in the above program; the execution returns to the area of the program code from where it was called along with the calculated result returned by the equation1 method for values "a" and "b". Then, the program continues on to the lines of code after the equation method.

If we repeated the same Java code with some of the following changes:

```
package com.example;

// Java Methods Created by Android ATC

public class MyClass {

    public static void main(String args[]) {
        int a = 10;
        int b = 6;
        int c = equation1(a, b);
        System.out.println("Equation Value = " + c);

        /** returns the result of the below equation*/

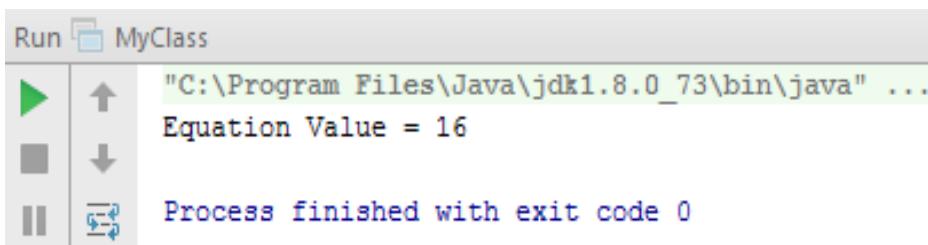
        public static int equation1(int n1, int n2) {

            int t=(n1+n2)/2;
            int s=n1+n2;

            return s;
        }
    }
}
```

Here, we added a new formula " $s=n1+n2$ ". We have in the equations1 method two values, $t=8$ and $s=16$. Which value will be returned by equations1 method if we call it? The answer is: the value of 16 will be retuned by equation1 method and that value is being assigned to variable c so c will now hold the value of 16.

The following will be the result of the previous Java code:



Call Method by Value

If you check the following Java code: Can you predict what is the value of a variable "a" if we run the following code?

```
public class lesson4 {

    public static void main(String args[]) {
        int a=10;

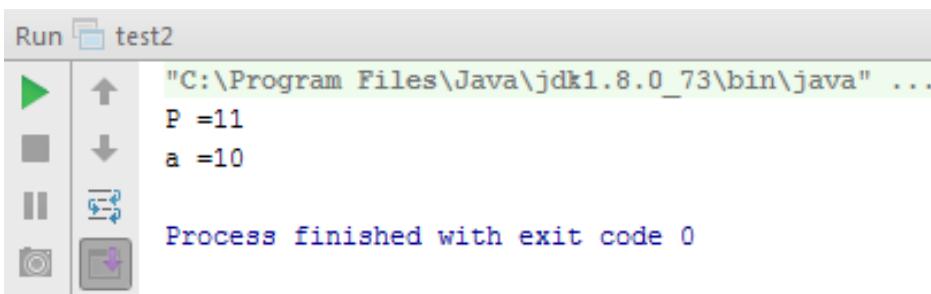
        increase(a);
        System.out.println("a =" +a);

    }

    /** returns the result of the below equation*/

    public static int increase(int p) {
        p=p+1;
        System.out.println("P =" +p);
        return p;
    }
}
```

The answer is a=10, the following is the output of the java code:

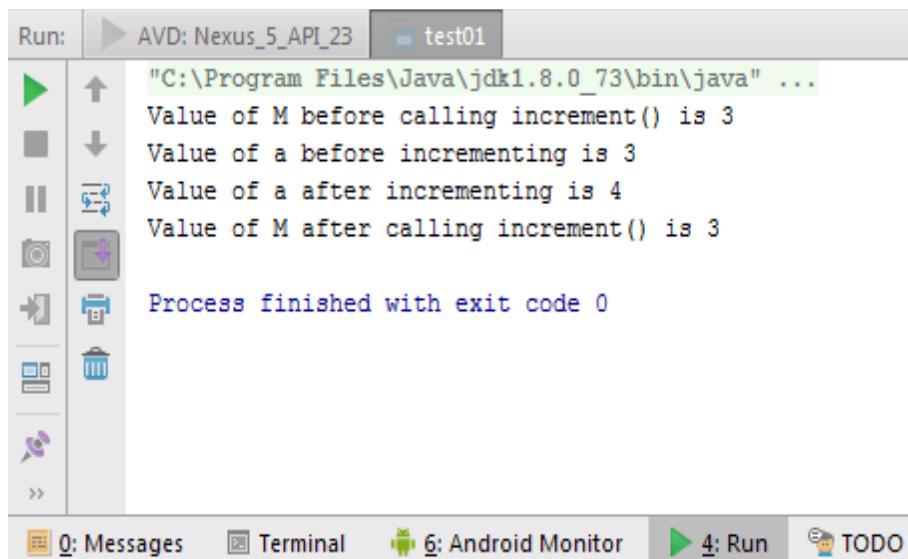


Here, when the variable a was declared as an integer variable with value equal to 10, this value has been reserved on the memory as a=10, and when the program called the method "increase", this method copied the value of the variable "a" to the variable "p", then through the formula ($p=p+1$) the value of the variable "p" becomes 11. However, the value of "a" is still equal to 10. The variable "a" passed as arguments still hold its original value. This is what is called **call or pass by value**.

Let us look for another simple program and examine its output:

```
public class lesson4 {  
  
    public static void main(String args[]) {  
  
        int M =3;  
        System.out.println ("Value of M before calling increment() is "+M);  
        increment(M);  
        System.out.println ("Value of M after calling increment() is "+M);  
    }  
  
    public static void increment ( int a ) {  
        System.out.println ("Value of a before incrementing is "+a);  
        a= a+1;  
        System.out.println ("Value of a after incrementing is "+a);  
    }  
}
```

The following will be the run result of the previous Java code:



Call Method by Reference

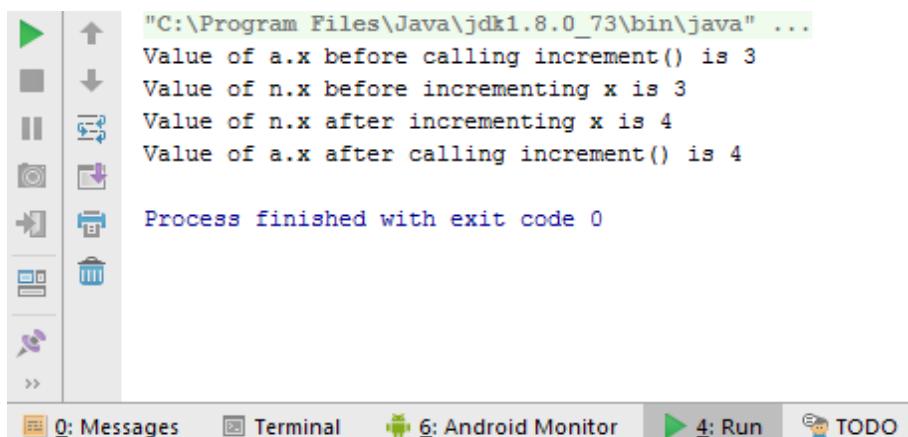
A method gives a copy of the value storage by the argument and not the original argument. In contrast, to pass by reference means pass a reference to the original variable (Object), i.e. passing the address in memory of the original variable (the original value).

```
class Number {
    int x;
}
public class test01 {
    public static void main(String args[]) {

        Number a = new Number();
        a.x=3;
        System.out.println("Value of a.x before calling increment() is "+a.x);
        increment(a);
        System.out.println("Value of a.x after calling increment() is "+a.x);
    }

    public static void increment(Number n) {
        System.out.println("Value of n.x before incrementing x is "+n.x);
        n.x=n.x+1;
        System.out.println("Value of n.x after incrementing x is "+n.x);
    }
}
```

The run result of this code is as follows:



```
"C:\Program Files\Java\jdk1.8.0_73\bin\java" ...
Value of a.x before calling increment() is 3
Value of n.x before incrementing x is 3
Value of n.x after incrementing x is 4
Value of a.x after calling increment() is 4

Process finished with exit code 0
```

The changes made to the variable `x` that was a part of the object in the `increment()` method had an effect on the original variable (the object which contained that integer variable) that

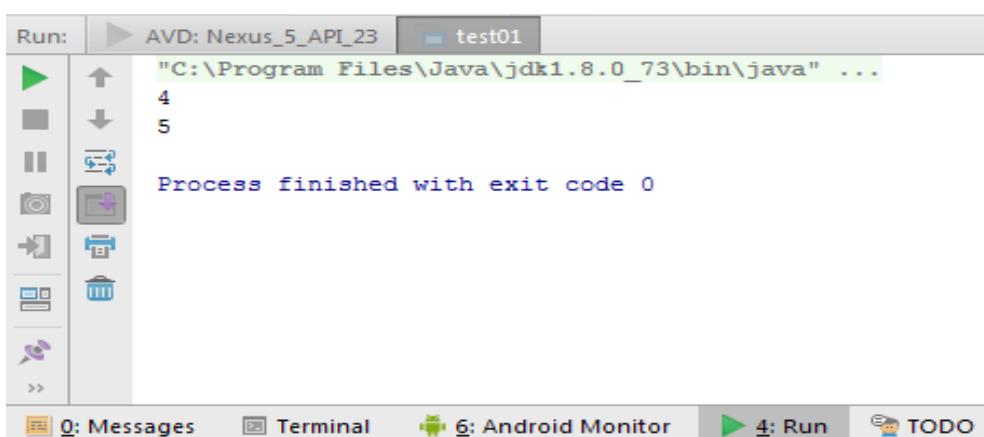
was passed as an argument. The difference lies in the type of the variable that was passed as an argument. int is a primitive data type while Number is a reference data type. Primitive data types in Java are passed by value while reference data types are passed by reference.

The concept of call by reference can be better understood if one tries to look into what a reference actually is and how a variable of a class type is represented. When we declare a reference type variable, the compiler allocates only space where the memory address of the object can be stored. The space for the object itself isn't allocated. The space for the object is allocated at the time of object creation using the new keyword. A variable of reference type differs from a variable of a primitive type in the way that a primitive type variable holds the actual data while a reference type variable holds the address of the object which it refers to and not the actual object.

Let us look at another simple program and examine its output:

```
class Number {  
    int x;  
}  
public class test01 {  
  
    public static void main ( String[] args ) {  
        Number a = new Number();  
        a.x=4;  
        System.out.println(a.x);  
        Number b=a;  
        b.x=5;  
        System.out.println(b.x);  
    }  
}
```

The run result of this code is as follows:



Arrays

So far, what you have studied each variable stored one data item. If we wish to store a large number of data items for the same variable, we need to use the array. An array is used to store a group of values, all of which have the same data type. The length of an array is established when the array is created. After creation, its length is fixed.

Much like C or C++, Java arrays are indexed numerically on a 0-based system. This means the first element in the array is at index 0; the second is at index 1, and so on.

Example:

```
public class ArrayAndroidATC {

    public static void main ( String[] args ) {
        int[] x;

        // allocates memory for 10 integers
        x = new int[10];

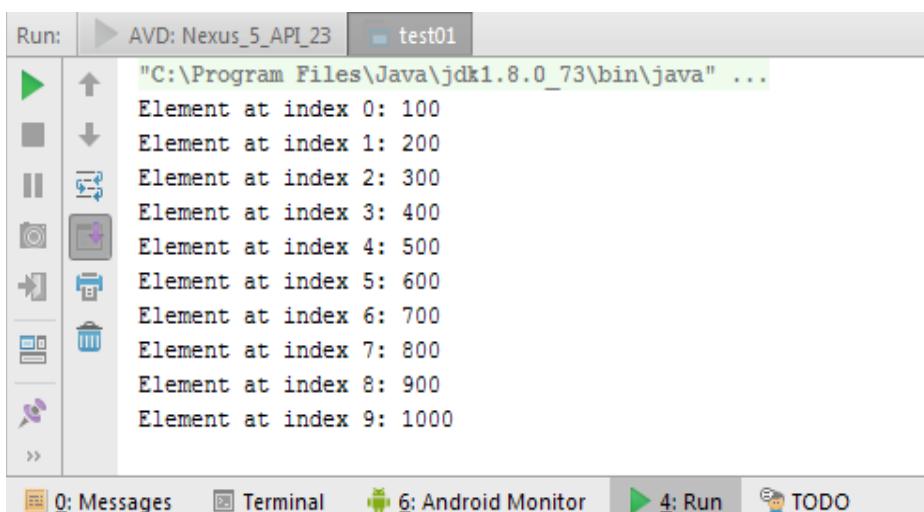
        // initialize first element
        x[0] = 100;

        // and so forth
        x[1] = 200;
        x[2] = 300;
        x[3] = 400;
        x[4] = 500;
        x[5] = 600;
        x[6] = 700;
        x[7] = 800;
        x[8] = 900;
        x[9] = 1000;

        System.out.println("Element at index 0: "+ x[0]);
        System.out.println("Element at index 1: "+ x[1]);
        System.out.println("Element at index 2: "+ x[2]);
        System.out.println("Element at index 3: "+ x[3]);
        System.out.println("Element at index 4: "+ x[4]);
        System.out.println("Element at index 5: "+ x[5]);
        System.out.println("Element at index 6: "+ x[6]);
        System.out.println("Element at index 7: "+ x[7]);
        System.out.println("Element at index 8: "+ x[8]);
        System.out.println("Element at index 9: "+ x[9]);

    }
}
```

The run result of this code is as follows:



Declaring a Variable to Refer to an Array:

The preceding program declares an array (named testArray) with the following line of code:

```
// declares an array of integers
int[] TestArray;
```

Like declarations for variables of other types, an array declaration has two components: the array's type and the array's name. An array's type is written as **type[]**, where type is the data type of the contained elements; the brackets are special symbols indicating that this variable holds an array. The size of the array is not part of its type (which is why the brackets are empty). An array's name can be anything you want. As with variables of other types, the declaration does not actually create an array; it simply tells the compiler that this variable will hold an array of the specified type.

Similarly, you can declare arrays of other types:

```
byte[] testArrayOfBytes;
short[] testArrayOfShorts;
long[] testArrayOfLongs;
float[] testArrayOfFloats;
double[] testArrayOfDoubles;
boolean[] testArrayOfBooleans;
char[] testArrayOfChars;
String[] testArrayOfStrings;
```

Note: The following line uses an array initialize to initialize the array **a[]**, the length of the array in the below statement. It is automatically calculated:

```
int[] a= { 3, 34, 7, 9};
```

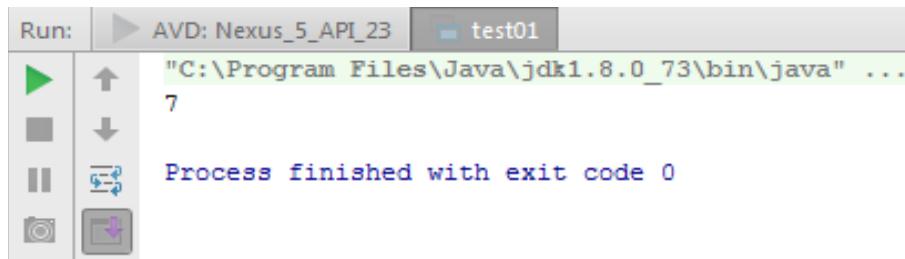
Can you expect the run result of this code, which is as follows?

```
public class AndroidArray {

    public static void main ( String[] args ) {
        int[ ] a= { 3, 34, 7, 9};

        System.out.println(a[2]);
    }
}
```

The run result of this code is as follows:



The following is an example about how to loop through the elements of an array.

```
public class ArrayLoop {

    public static void main ( String[] args ) {
        int[] a = {3, 4, 7, 9};
        for (int x : a) {
            System.out.println(x);
        }
    }
}
```

The result of this code is as follows:

```
"C:\Program Files\Java\jdk1.8.0_73\bin\java" ...
3
4
7
9
Process finished with exit code 0
```

Enter data to a Java program:

We can enter any data to Java program through the keyboard or other input devices using **Scanner** class.

The following displays how we can use scanner class to get some data from user for a java program.

First of all, you have to create an instance from the scanner:

```
Scanner scanner = new Scanner(System.in);
```

After writing this line of code, Android studio will notify you that the scanner cannot be resolved; this is because the library necessary for using the Scanner object does not exist in your Java file.

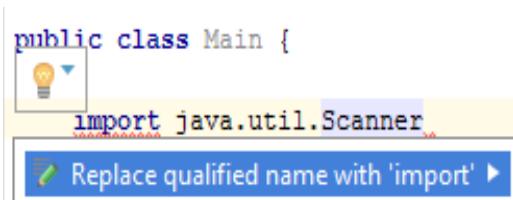
There are two ways to include this library:

1. Write the following code before your class declaration:

```
import java.util.Scanner;
```

2. Place the cursor on the Scanner object, and click the shortcut below to include the necessary library automatically.

- On Windows click Ctrl+1.
- On Mac OS X click Cmd+1.
- Move the mouse pointer to the lamp sign as it is illustrated in the below figure and select “Replace qualified name with “import”



You will get the below code at the start of the code:

```
import java.util.Scanner;

public class Main {
```

Then, we will add: `Scanner scanner = new Scanner(System.in);` to the java code.

The following example prints the exam result depending on the score which is entered to it, where it will print "Pass: You are Android Certified Application Developer "if the score is greater or equal to 70 and "Fail: You May Repeat the Exam after 24 Hours" if the score is less than 70.

```
import java.util.Scanner;

public class Main {

    public static void main (String args[]){

        float score;

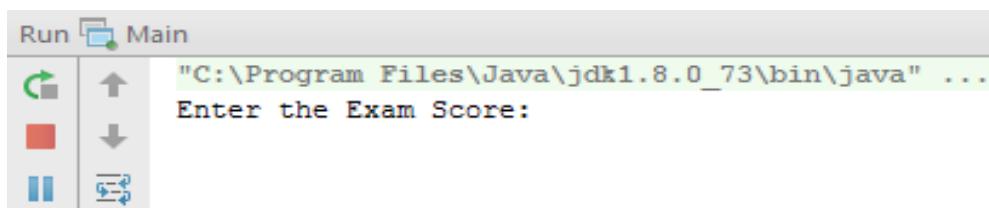
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the Exam Score:");

        score = scanner.nextInt();

        if(score>=70){
            System.out.println("Pass: You are an Android Certified Application Developer ");
        }
        else {
            System.out.println("Fail: You May Repeat the Exam after 24 Hours");
        }
    }
}
```

The run result of this code is as follows:



```
"C:\Program Files\Java\jdk1.8.0_73\bin\java" ...
Enter the Exam Score:
```

Type 80 then press "Enter" key, you will get the following result:



```
"C:\Program Files\Java\jdk1.8.0_73\bin\java" ...
Enter the Exam Score:
80
Pass: You are an Android Certified Application Developer
Process finished with exit code 0
```

Object-Oriented Programming (OOP) Concepts

If you've never used an object-oriented programming language before, you'll need to learn a few basic concepts before you can begin writing any code. This lesson will introduce you to objects, classes, inheritance, interfaces, and packages. Each discussion focuses on how these concepts relate to the real world, while simultaneously providing an introduction to the syntax of the Java programming language. The basics are necessary to learn before starting Android application development.

What Is an Object?

Objects are the key to understanding object-oriented technology. Look around right now and you'll find many examples of real-world objects: your car, your desk, your computer, your Car.

Real-world objects share two characteristics: They all have state and behavior. Computers have state (type, color, speed, capacity) and behavior (processing, playing media, browsing). Car also has state (current gear, current pedal cadence, and current speed) and behavior (changing gear, changing pedal cadence, applying brakes). Identifying the state and behavior for real-world objects is a great way to begin thinking in terms of object-oriented programming.

What Is a Class?

A class is a blueprint or prototype from which objects are created. This section defines a class that models the state and behavior of a real-world object. It intentionally focuses on the basics, showing how even a simple class can cleanly model state and behavior. In this lesson you will discuss Java class in details. In the previous lessons you have already studied classes practically.

What Is Inheritance?

Inheritance provides a powerful and natural mechanism for organizing and structuring your software. This section explains how classes inherit state and behavior from their superclasses or parent classes, and how to derive one class from another using the simple syntax provided by the Java programming language. The class, which inherits the state and behavior from another class, is known as child or derived class and the class from which the state and behavior are inherited is called parent or base class.

The real day example of inheritance can be an *Employee* class and the *Developer*, *Manager*, *Designer* classes. In this example *Employee* is the parent class and all the other classes i-e *Developer*, *Manager*, *Designer* are the child classes because they are already employees and hold the attributes of the *Employee* class.

Now you will study the same example of inheritance with the help of Java code. Make a simple Employee class with a single property called salary like the following:

```
class Employee{  
    float salary = 5000;  
}
```

Now add another class “Developer” which will be a child class. To make any class a child class of any parent class the “extend” keyword is used as shown below:

```
class Developer extends Employee {  
    int bonus = 1000;  
    public static void main(String args[]){  
        Programmer p=new Programmer();  
        System.out.println("Programmer salary is:"+p.salary);  
        System.out.println("Bonus of Programmer is:"+p.bonus);  
    } }
```

In the code above, Employee is the parent class and Developer is the child class. Inside Developer class, we have main() function. You can see in the code above that the variable inside the parent class is accessible from within the child class. p.salary prints the value of the salary variable which is inside the parent class and it also prints the value of bonus variable which is inside the child class itself. In this way, the child class has its own variables and functions as well as it can access the variables and functions of the parent class.

Following will be the output of the code above.

Programmer salary is: 50000.0

Bonus of programmer is: 10000

More practical examples of inheritance will be explained later in this lesson under the section of Class.

What Is an Interface?

An interface is a contact between a class and the outside world. When a class implements an interface, it promises to provide the behavior published by that interface. In the interface, you only declare the methods and definitions of those methods that are not written in the interface. Any class that implements that interface includes the definition of the methods only declared inside the interface. In this way, you can achieve full abstraction and multiple inheritances in Java. Generally, without interface, multiple inheritances is not allowed in Java.

In order for any class to use the interface, “implements” keyword is used. The following is the example of interface using Java code.

```
interface displayable{
void display();
}

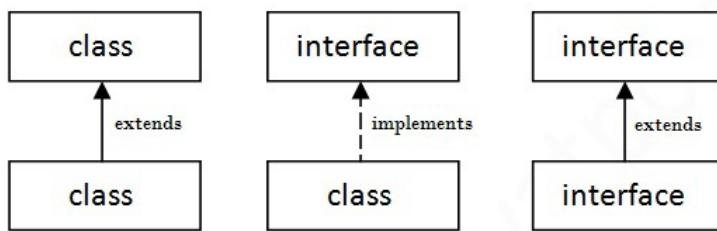
class InterfaceDemo Implements displayable{
public void display (){
System.out.println("Hello World");
}

public static void main(String args[]){
InterfaceDemo obj = new InterfaceDemo ();
obj. display ();
}
}
```

As you can see in the code above, the “displayable” is the interface and it declares a method name “display” and has no definition of this method inside it. Then, the Java class “InterfaceDemo” implements the “displayable” interface and includes the definition of the method “display” which was only declared in the interface. Following will be the output of this code.

Hello World

The following diagram explains the relationships between classes and interface :



What Is a Package?

A package is a namespace for organizing classes and interfaces in a logical manner. Placing your code into packages makes large software projects easier to manage. Packages can be categorized in two forms

1. Built-in package
2. User-defined package

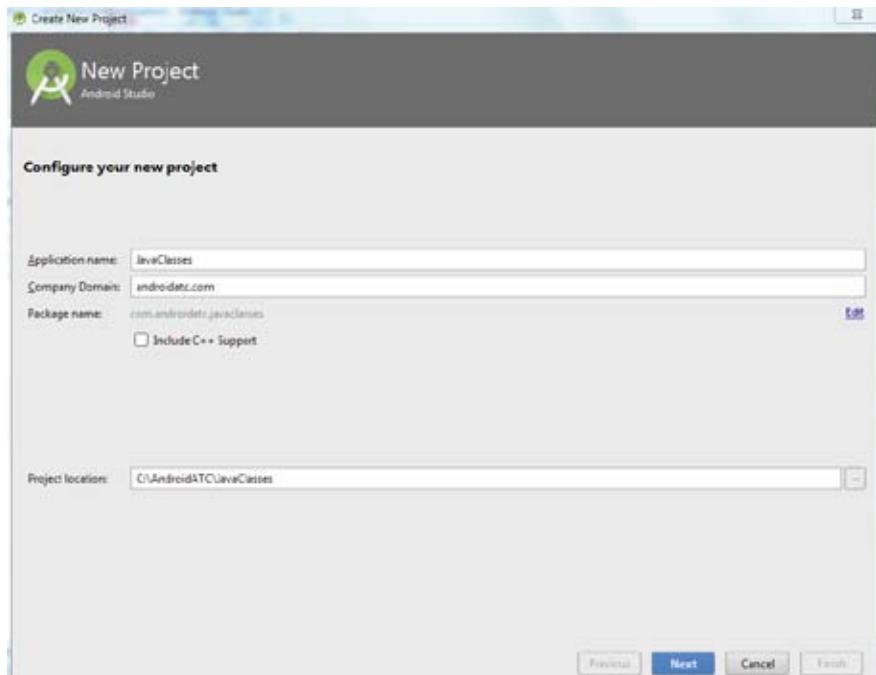
Built-in packages are provided by programming languages like Java and can be used within your code to reuse the already provided code, thus speeding up the development. Few of the built-in packages are java, lang, awt, javax, swing, net, io, util, sql... etc.

On the other hand the user-defined package includes the classes, interfaces... etc created by you.

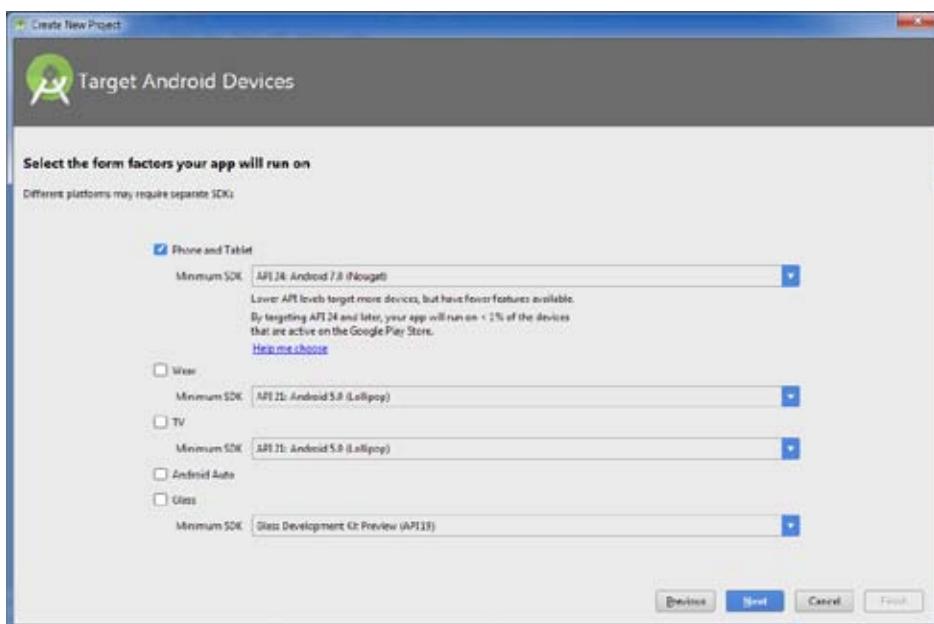
Java Class

A class is a group of objects having common properties. In the real world, you'll often find many individual objects all of the same kind. In this section, you will study a practical example of class and also practice other concepts like inheritance you have studied previously. The following is a practical exercise:

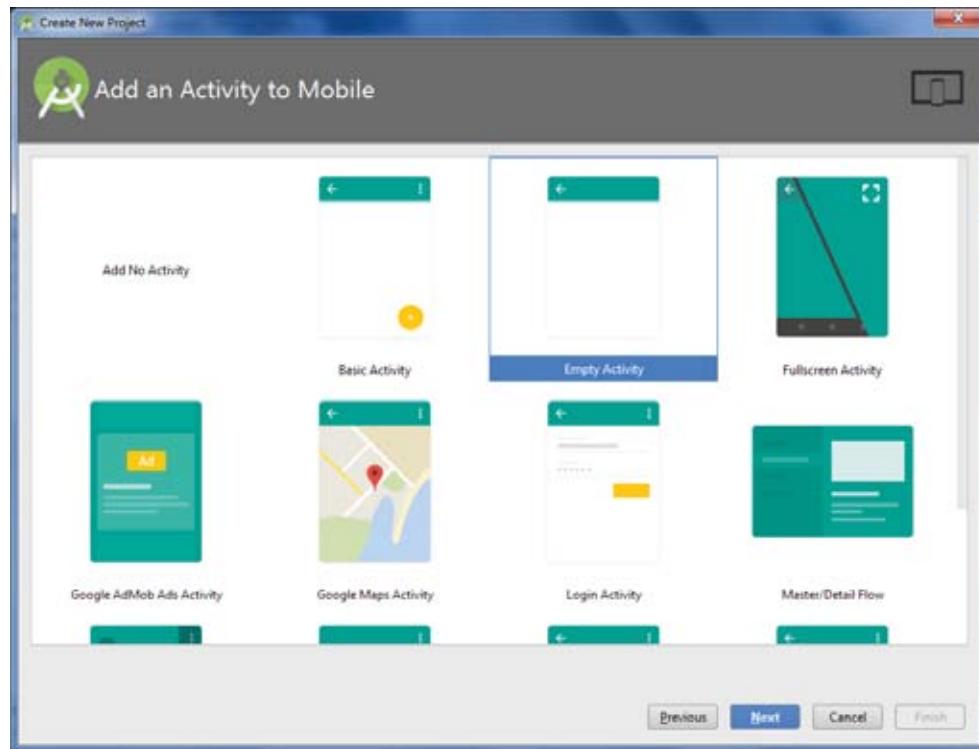
1. Open Android Studio
2. To create an Android project Click : “Start a new Android Studio project”, to get the following dialog box:



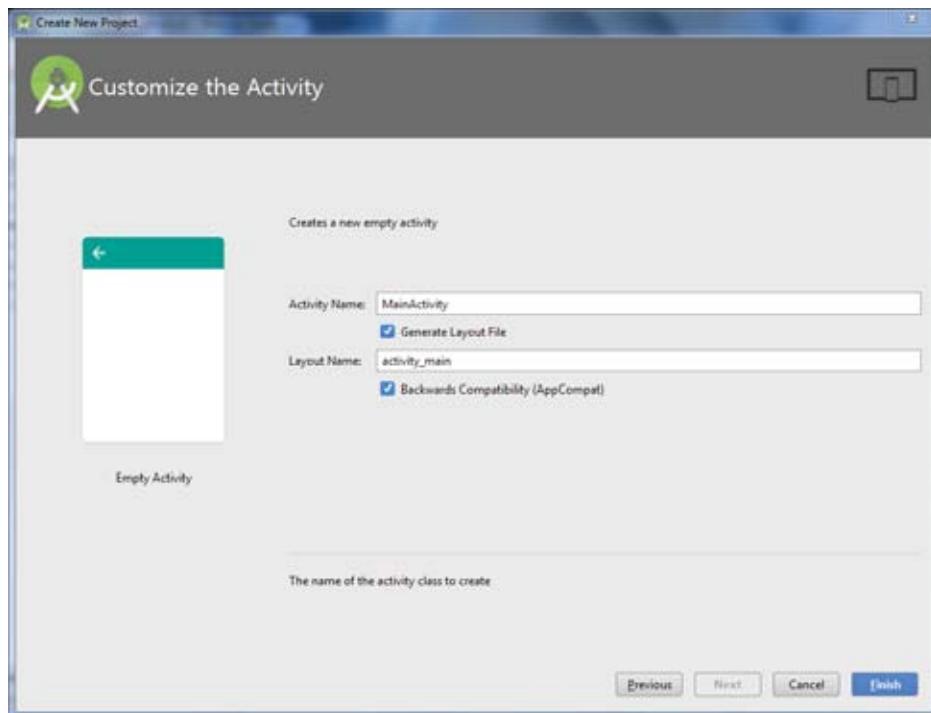
3. Write the **Application Name**: JavaClasses, **Company Domain** which in this case consists of three words separated by dot. In this example project, write **androiddata.com** and the Project location you should select where the project will be saved, then click **Next**.
4. In the next dialog box, there are some details about the description of the Android device, on which this application will run on. You will learn about this dialog box in the Android Application Development course. Keep the default configuration and click **Next**.



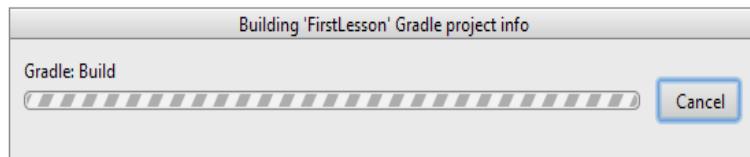
5. In the following dialog box, keep the default configuration “**Empty Activity**” and click **Next**.



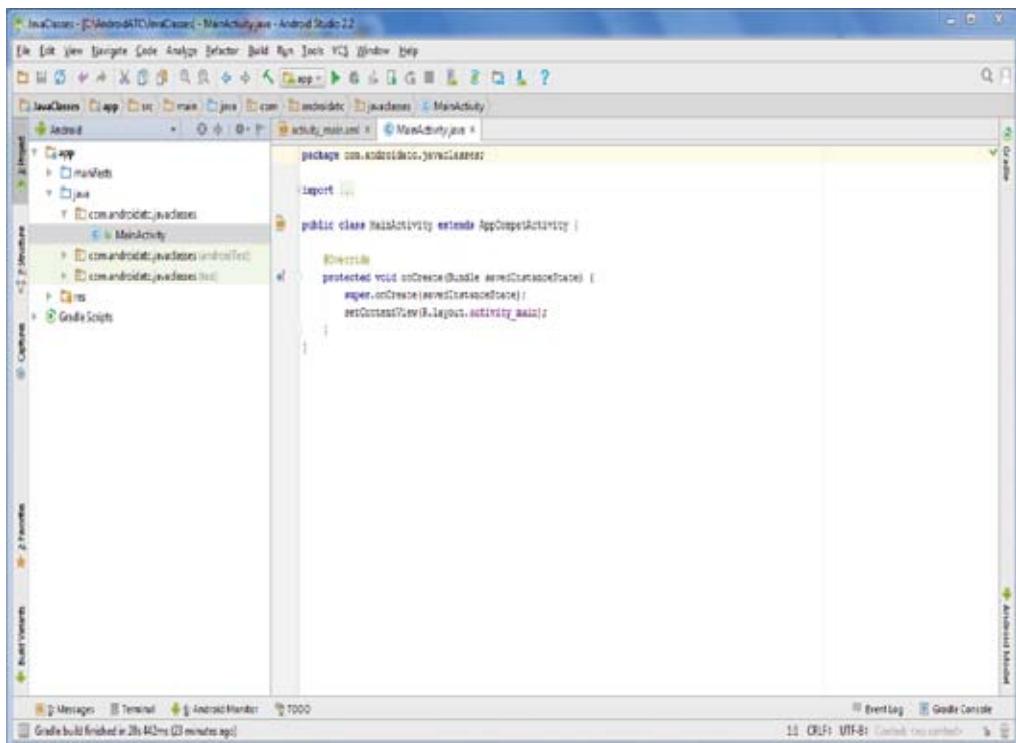
6. In the following dialog box, keep the default configuration -this configuration will be explained in details in the Android Application Development course- click **Finish**.



7. After clicking the Finish button, creating the application process starts and you will see the following :

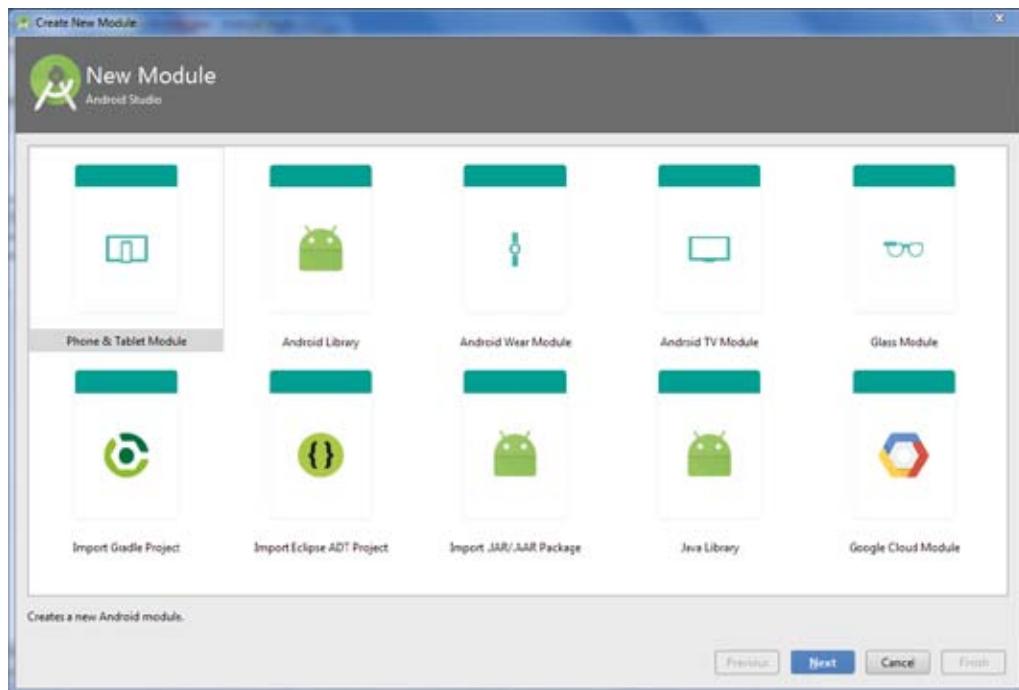


8. Then the following screen shows:

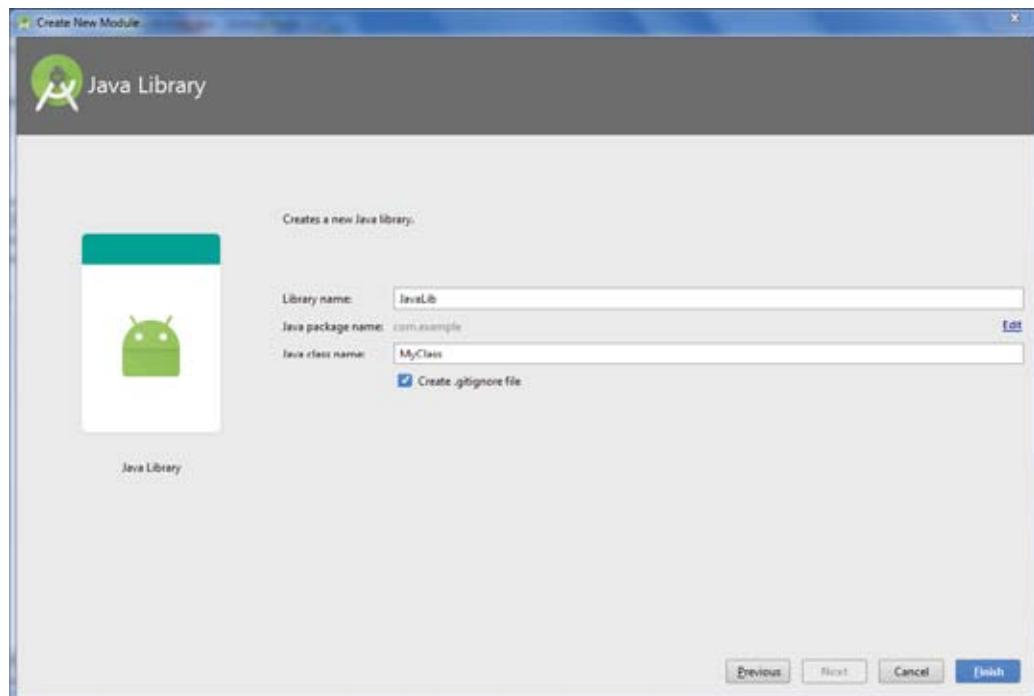


9. To create Java Library, click on File menu → **New** → **New Module...**

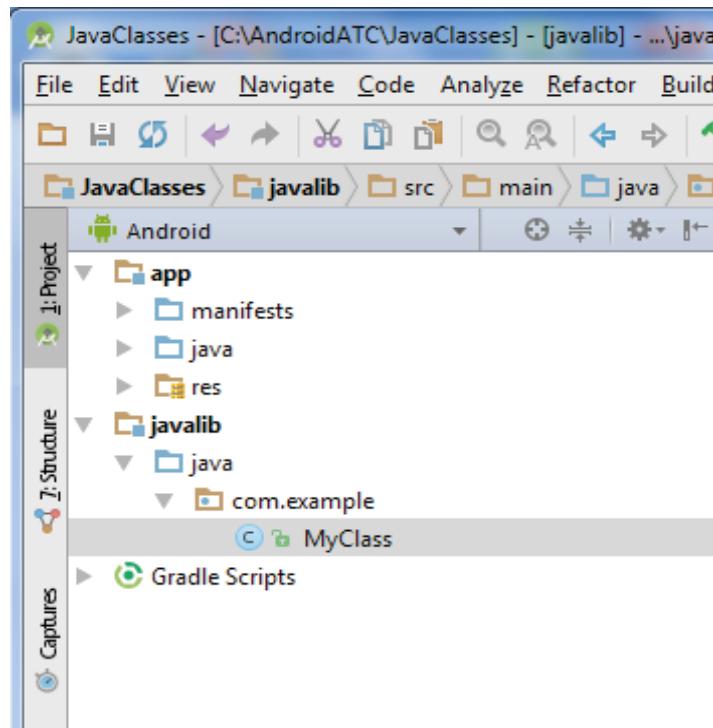
10. In the dialog box below, select “**Java Library**” and click **Next**.



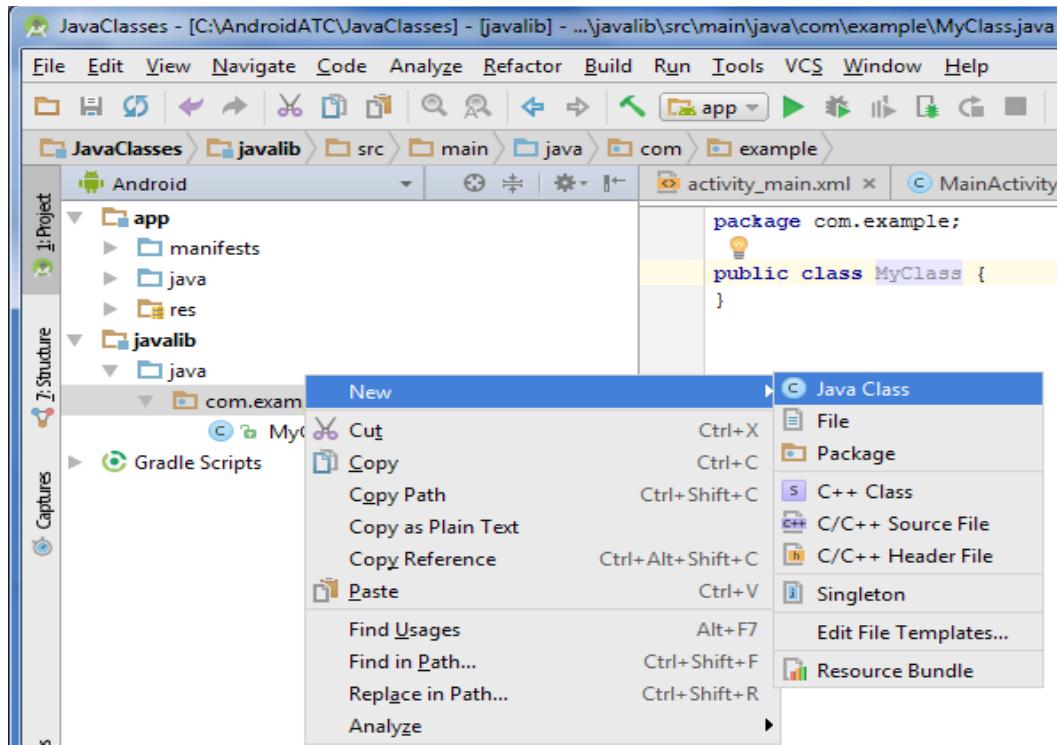
11. In the following dialog box, enter the “**Library name**”: **JavaLib** and the “**Java class name**”:**MyClass** then click **Finish**.



You will get the following illustration on the left side of the Android Studio:



12. Now, right click on package name, then “**New**” and then “**Java Class**” as shown in the image below.



13. Enter the name “**AndroidATC**” as the name of class in the window which opens after the previous step. This will add a new class named “**AndroidATC**” under the package. Open this newly created class and add functions inside this class named “getCompanyInfo()” and “getContactInfo()”. The first function getCompanyInfo() will return the brief introduction of Android ATC and the second function getContactInfo() will return contact information of Android ATC. The class will now look like the following:

```
package com.example;
public class AndroidATC {
    public String getCompanyInfo(){
        return "Android Advanced Training Consultants (Android ATC) provides
courses and assessment exams " +
               "to certify the competencies of current and prospective employees.";
    }

    public String getContactInfo(){
        return "Phone = +1-214-393-9225 \n Skype = androidatc \n Email = info@
androidatc.com";
    }
}
```

Note the following in the code above.

- The package “**com.example**” name is included in the very beginning of the class.
- Both functions are returning the Strings using the “**return**” keyword.
- Both functions have “**String**” as the return type, which means that whenever these functions are called, they will return a “String” type value which are the company information and the contact information respectively.

14. Now similarly, add another class named “AndroidATCTrainings” as you did in the steps above and open the file when you have made it. Now add an “Array” of type “String” named “courses”. This will contain the courses being offered at AndroidATC. The array will look like the following:

```
public String[] courses = {"Java Fundamentals for Android",
                           "Android Application Development",
                           "Android Security Essentials",
                           "Monetize Android Applications" };
```

In this way, you declared (reserved memory) as well as initialized (assigned values) an array of type String with the names of courses. Similarly add another array that will contain the certifications being offered at Android ATC. The new array will look like the following:

```
public String[] certifications = {  
    "Android certified Application Developer",  
    "Android certified Application Engineer",  
    "Android certified Trainer"  };
```

15. Now, add a function getCourse() in this class that will take a course number as the argument and return the course at that number from the array of courses. This will look like the following:

```
public String getCourse(int courseNumber){  
    return courses[courseNumber];  
}
```

16. Then add another function getCertification() in this class which will take the certification number as argument and return the certification at that index from the array of certifications. This will look the following.

```
public String getCertification(int certificationCode){  
    return certifications[certificationCode];  
}
```

17. As you studied in the section of inheritance, any class that acts as a child class uses “extend” keyword to inherit the properties from parent class. So now, you will make AndroidATCTrainings class as the child class of AndroidATC class by modifying the starting line of AndroidATCTrainings class like the following:

```
public class AndroidATCTrainings extends AndroidATC{  
}
```

18. This will turn AndroidATCTrainings class as the child class of AndroidATC. Since AndroidATC has the functions that return the common information that is the same for all training centers, this characterizes a parent class. Any class can use this common information by making AndroidATCTrainings as its parent class. The following is a complete code of AndroidATCTrainings class.

```
public class AndroidATCTrainings extends AndroidATC{  
  
    public String[] courses = {  
“Java Fundamentals for Android”,  
“Android Application Development”,  
“Android Security Essentials”,  
“Monetize Android Applications”};  
  
    public String[] certifications = {  
“Android certified Application Developer”,  
“Android certified Application Engineer”,  
“Android certified Trainer” };  
  
    public String getCourse(int courseNumber){  
        return courses[courseNumber];  
    }  
  
    public String getCertification(int certificationCode){  
        return certifications[certificationCode];  
    }  
}
```

19. Now, open your “MyClass” which was created when you created the library for your project in the beginning. Add the “main” function inside this class.

```
public static void main(String[] args) {  
  
}
```

Now add two integer variables “courseNumber” and “certificationNumber”. These variables

will be passed to getCourse() and getCertification() as arguments for getting the course and certification information respectively.

```
int courseNumber = 3;  
int certifictionNumber = 2;
```

20. Now inside the main function, make an object of AndroidATCTrainings class like the following:

```
AndroidATCTrainings androidATCTrainings = new AndroidATCTrainings();
```

21. Now, you can call the functions from AndroidATCTrainings class as well as AndroidATC class since AndroidATC is the parent class of AndroidATCTrainings class and the object of AndroidATCTrainings class has access to the public, protected functions of AndroidATC class. Start calling the functions AndroidATCTrainings class using its object you created in previous step.

```
androidATCTrainings.getCourse(courseNumber)
```

22. In this line, you are passing “courseNumber” as argument in getCourse() function, which will return the course at index 3 inside the “courses” array which is “Monetize Android Applications”. Similarly you can get the certification by passing index in getCertification() function like the following:

```
androidATCTrainings.getCertification(certifictionNumber)
```

This will return “Android certified Trainer” since this is at index 2 in the array of “certifications” in AndroidATCTrainings class. In case the values of courseNumber and cerificationNumber are greater than the respective array size, your program will display error. So, it better to put the checks for valid value before passing to the function like the following:

```
if(courseNumber>=0 && courseNumber<= androidATCTrainings.courses.length-1) {  
}
```

It goes for certificationNumber too. In case of an invalid number being passed, you can add the else case that will ask the user to assign a valid number to courseNumber or certificationNumber.

23. Now, you will call the functions of AndroidATC class (parent class) using the same object of AndroidATCTrainings class (child class) as below:

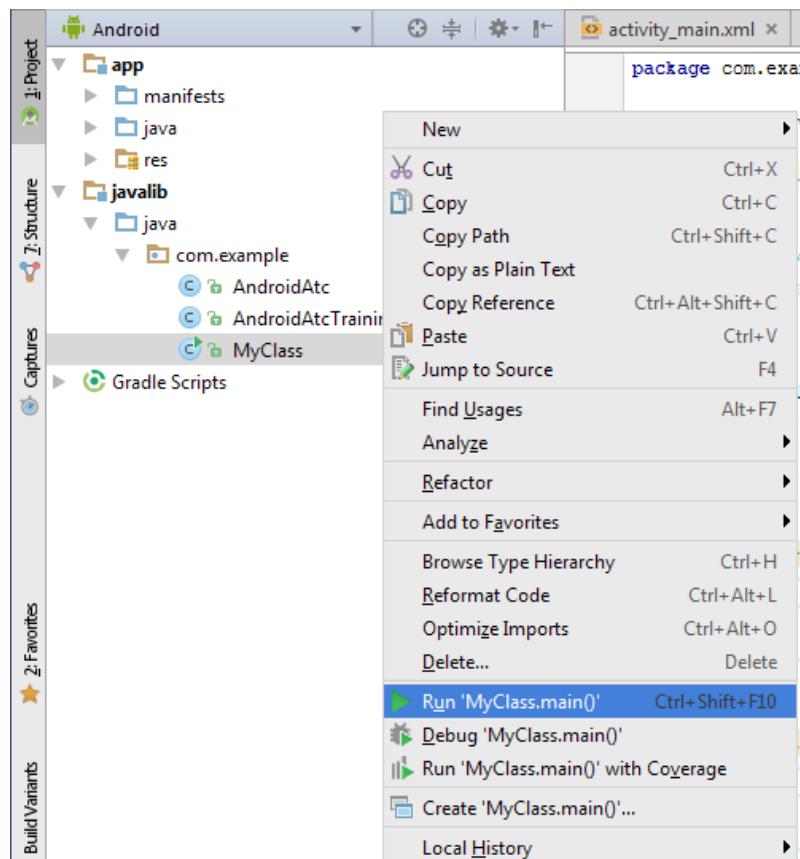
```
androidATCTrainings.getCompanyInfo()  
androidATCTrainings.getContactInfo()
```

You can use "System.out.println()" function to print all these values in the console.

24. Following is the complete code of MyClass.

```
package com.example;  
public class MyClass {  
  
    public static void main(String[] args) {  
        int courseNumber = -3;  
        int certifictionNumber = -2;  
  
        AndroidATCTrainings androidATCTrainings = new AndroidATCTrainings();  
        System.out.println("Company Information\n");  
        System.out.println(androidATCTrainings.getCompanyInfo());  
        System.out.println("Contact Information\n");  
        System.out.println(androidATCTrainings.getContactInfo());  
  
        if(courseNumber>=0 && courseNumber<= androidATCTrainings.courses.length-1)  
        {  
            System.out.println("Course Information\n");  
            System.out.println(androidATCTrainings.getCourse(courseNumber));  
        }else{  
            System.out.println("Enter the valid course number");  
        }  
        if(certifictionNumber>=0 && certifictionNumber <= androidAtcTrainings.certifications.length-1) {  
            System.out.println("\nCertification Information\n");  
            System.out.println(androidATCTrainings.getCertification(certifictionNumber));  
        }else{  
            System.out.println("Enter the valid certification number");  
        }  
    }  
}
```

25. Now right click on MyClass and select "Run MyClass.main()" to run your code.



26. You will see the following output printed in the console.

```
Run MyClass
"C:\Program Files\Android\Android Studio\jre\bin\java" ...
Company Information
Android Advanced Training Consultants (Android ATC) provides courses and assessment exams to certify the competencies of current and prospective employees.
Contact Information
Phone = +1-214-393-9225
Skype = androidatc
Email = info@androidatc.com
Course Information
Monetize Android Applications
Certification Information
Android certified Trainer
Process finished with exit code 0
```

Now, if you give invalid (greater or lesser the array size) values to courseNumber and certificationNumber, the following messages will be displayed:

```
Run MyClass
"C:\Program Files\Android\Android Studio\jre\bin\java" ...
Company Information
Android Advanced Training Consultants (Android ATC) provides courses and assessment exams to certify the competencies of current and prospective employees.

Contact Information
Phone = +1-214-393-9225
Skype = androidatc
Email = info@androidatc.com

Enter the valid course number
Enter the valid certification number

Process finished with exit code 0
```

The following are important points to remember in this practical example

1. AndroidATC is the parent class.
2. AndroidATCTrainings is the child class.
3. MyClass is the class having the main function.
4. The object of AndroidATCTrainings class can use public and protected functions of AndroidATC class. This is practical demonstration of inheritance.
5. A function takes argument to be used inside functions and can return values (Strings in this case)

Now, that you have completed the Java Fundamentals for Android Development course, you are ready to take the Android application development course.

To study this course you should select one of the following two choices:

Alternative 1: "Instructor-led training course", Android ATC courses are available at more than 130 computer training centers located worldwide, to register for this course, kindly go to: http://www.androidatc.com/_training_center.php and select your country on the list, to find the nearest Android ATC authorized training center, kindly contact them directly and ask them about the courses schedule and training price. This list is continuously updated.

Alternative 2: You may choose to study the course on your own by ordering the self-study guide from the Android ATC Web site, kindly go to "www.androidatc.com" then select the "SELF STUDY" tab and place your order online for the "Android Application Development" book, where this book is only available in hard copy format. The book should be delivered to you in about four working days. Kindly note: There is no software format (PDF) available for this book.

For more information, don't hesitate to contact us by email "support@androidatc.com".