CHAPTER 1
Introduction to JAVA

Introduction

Java is a general-purpose, **object-oriented programming language** and a **plateform** developed by **Sun Microsystems of USA in the year 1996**. ***James Gosling*** is known as the father of Java. Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the Oak name to **Java**.

Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

Now it is owned by **Oracle**, and more than **3 billion** devices run Java.

It is used for:

- Mobile applications (specially Android apps)
- Desktop applications
- Web applications
- Web servers and application servers
- Games
- Database connection
- And much, much more!

1.1 Why Use Java?

- Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)
- It is one of the most popular programming language in the world.
- It is easy to learn and simple to use
- It is open-source and free

- It has a huge community support (tens of millions of developers)
- Java is an object oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs
- As Java is close to C++ and C#, it makes it easy for programmers to switch to Java or vice versa

1.2 Java History:

James Gosling 

is a famous Canadian software developer who has been with Sun Microsystems since 1984 and is considered as father of Java programming language. Gosling did the original design of Java and implemented its original compiler and virtual machine.

- In1990, Sun Micro Systems Inc. (US) was conceived a project to develop software (which should be portable and machine independent) for consumer electronic devices that could be controlled by a remote. This project was called **Stealth Project** but later its name was changed to **Green Project**.

- In June 1991, Project Manager **James Gosling** and his team members **Patrick Naughton**, **Mike Sheridan**, **Chris Wrath**, and **Ed Frank** initiated the Java language project. This small team of sun engineers called **Green Team**.

- Gosling thought C and C++ would be used to develop the project. But the problem he faced with them is that they were system dependent languages. The trouble with C and C++ (and most other languages) is that they are designed to be compiled for a specific target and could not be used on various processors, which the electronic devices might use.

- James Gosling with his team started developing a new language, which was completely system independent. This language was initially called **OAK** (Oak is a symbol of strength

and chosen as a national tree of many countries like U.S.A., France, Germany, Romania etc.). Since this name was registered by some other company, later it was changed to Java.

- James Gosling and his team members were consuming a lot of coffee while developing this language. Good quality of coffee was supplied from a place called "Java Island'. Hence they fixed the name of the language as Java. The symbol for Java language is cup and saucer.

- Sun formally announced Java at Sun World conference in 1995. On January 23rd 1996, JDK1.0 version was released. Bill Joy, Arthur van Hoff, Jonathan Payne, Frank Yellin, and Tim Lindholm were key contributors to the maturing of the original prototype.

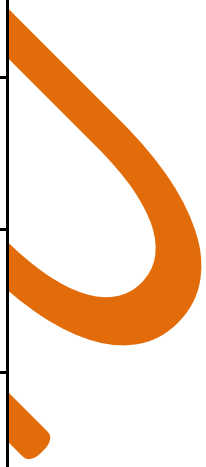
- Latest version of Java is **Java 15** or **JDK 15.1.0** released on 15th Sept'2020. With Java 8 release, java provided supports for **functional programming**, **new JavaScript engine**, **new APIs for date time manipulation**, **new streaming API** etc.

1.3 Java Version History

From the first version released in 1996 to the latest Standard Edition (SE) version 15 released in Sept 2020.

Java SE Version	Version Number	Release Date
JDK 1.0 (Oak)	1.0	January 1996
JDK 1.1	1.1	February 1997
J2SE 1.2 (Playground)	1.2	December 1998

J2SE 1.3 (Kestrel)	1.3	May 2000
J2SE 1.4 (Merlin)	1.4	February 2002
J2SE 5.0 (Tiger)	1.5	September 2004
Java SE 6 (Mustang)	1.6	December 2006
Java SE 7 (Dolphin)	1.7	July 2011
Java SE 8	**1.8**	**March 2014**
Java SE 9	9	September, 21st 2017
Java SE 10	10	March, 20th 2018
Java SE 11	11	September, 25th 2018
Java SE 12	12	March, 19th 2019
Java SE 13	13	September, 17th 2019
Java SE 14	14	March, 17th 2020
Java SE 15	15	September, 15th 2020
Java SE 16	*16*	*Expected on March*



1.4 JAVA FEATURES:

Features of Java

Features of a language are nothing but the set of services or facilities provided by the language vendors to the industry programmers. Some important **features of java** are:

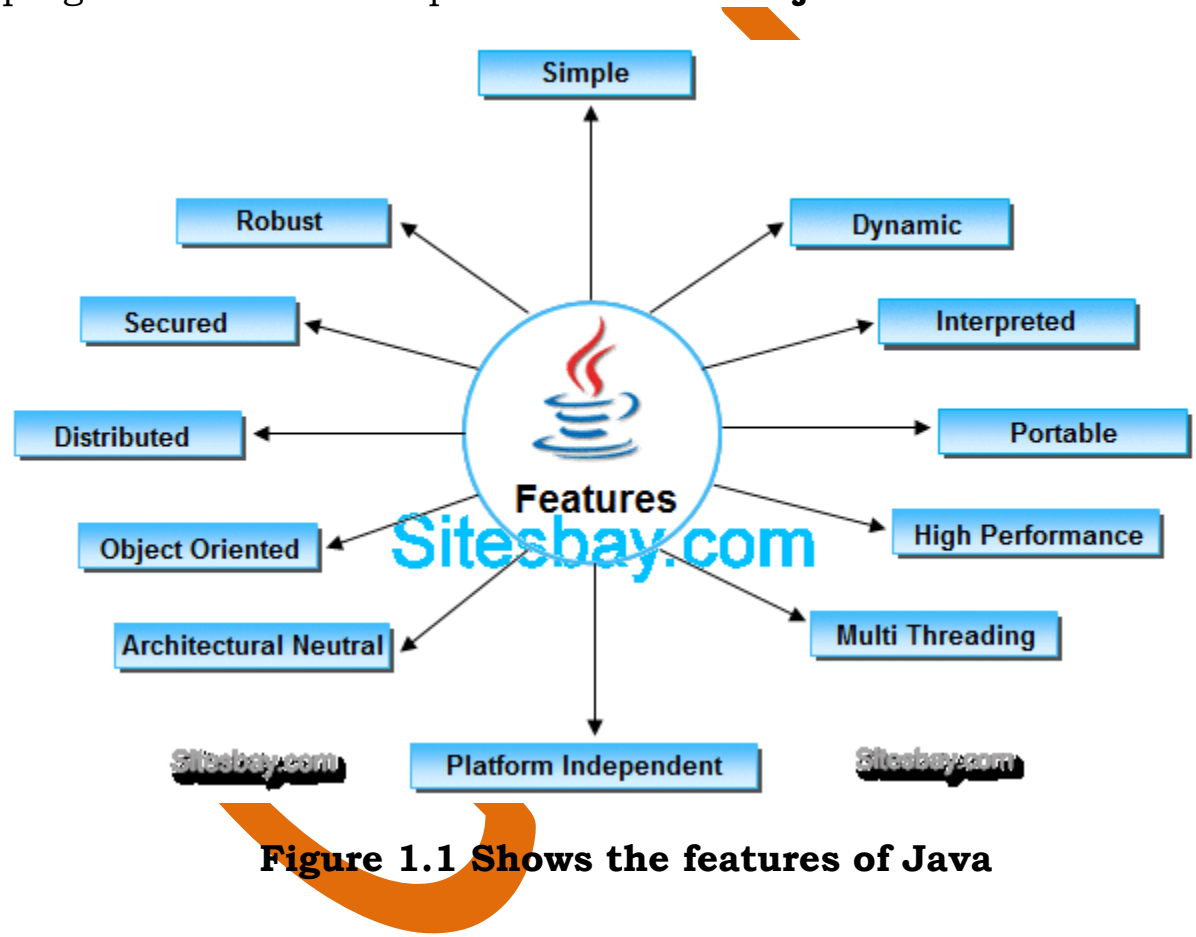
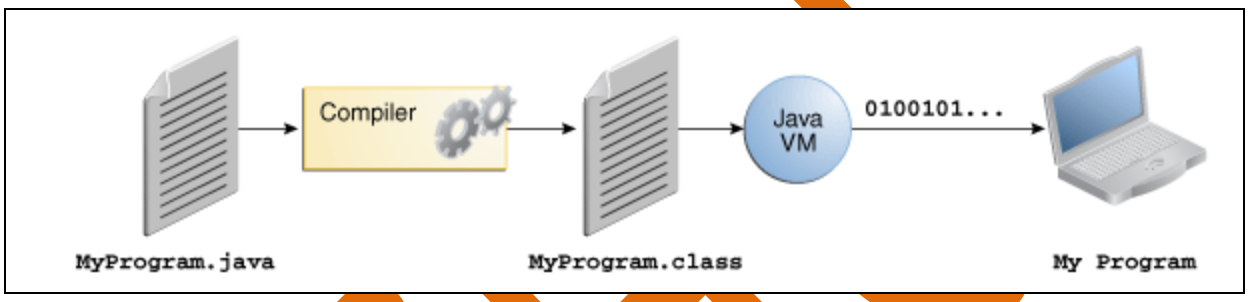


Figure 1.1 Shows the features of Java

- **Simple**: Learning and practicing java is easy because of resemblance with c and C++.
- **Object Oriented Programming Language**: Unlike C++, Java is purely OOP.
- **Secure:** Java achieves this protection by confining a Java program to the Java execution environment and by making it inaccessible to other parts of the computer. It is a more secure language compared to other language; In this language, all

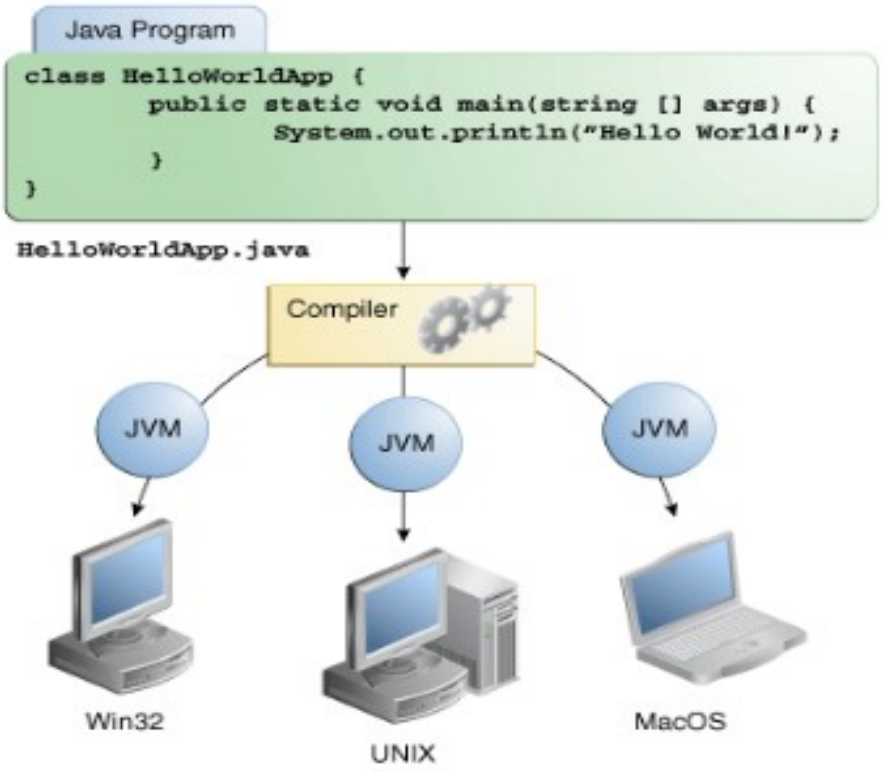
code is covered in byte code after compilation which is not readable by human or machine except JVM.

- **Plate-form Independent & Architectural Neutral:** Java development team work on the philosophy of **write once; run anywhere, anytime, forever**" and as a result the Java Virtual Machine (JVM) was developed.
- Only the JVM can execute the bytecode. Java byte code is not machine dependent; it can run on any machine with any processor and with any OS.



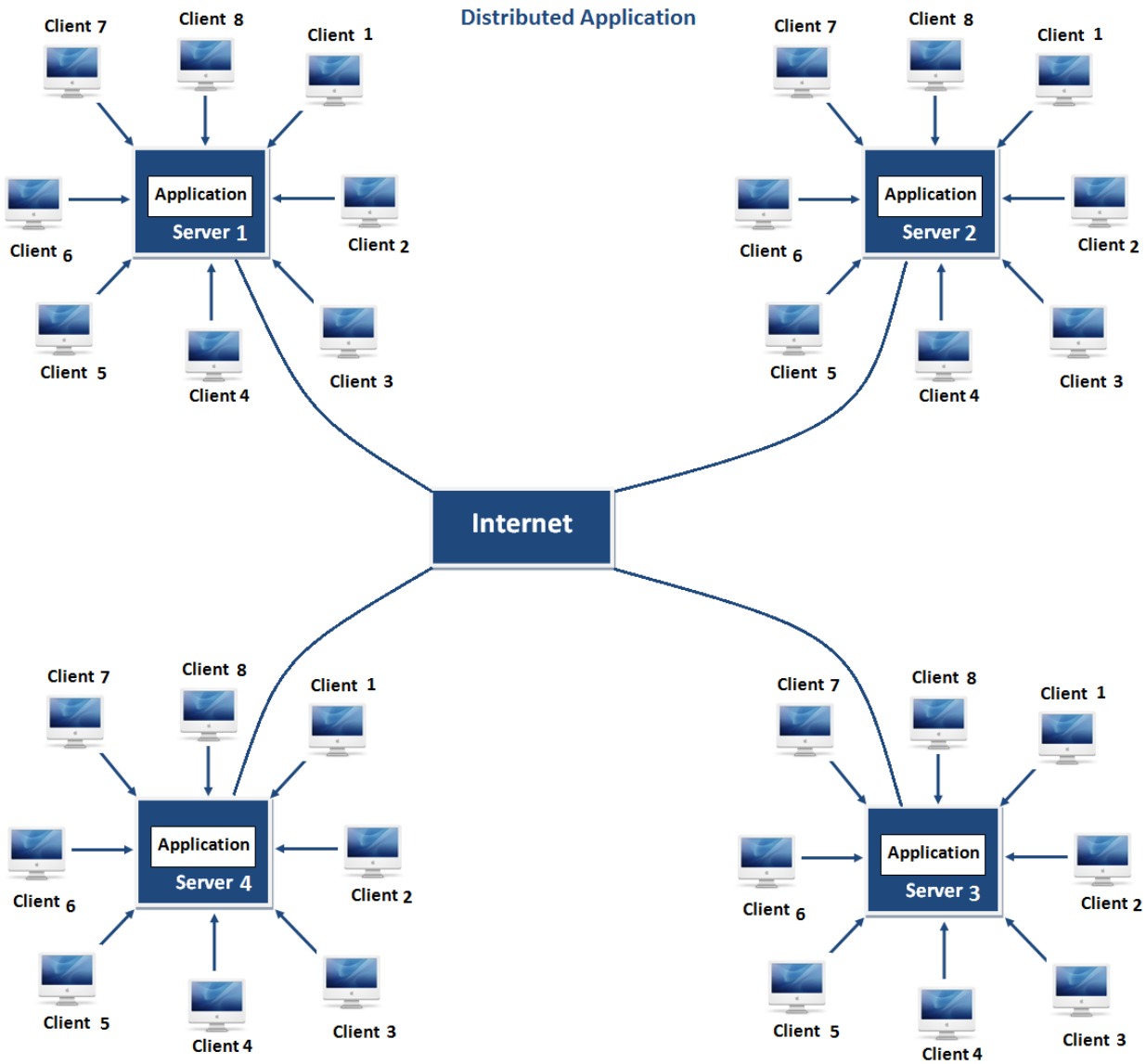
In the Java programming language, all source code is first written in plain text files ending with the .java extension. Those source files are then compiled into .class files by the javac compiler. A .class file does not contain code that is native to your processor; it instead contains *bytecodes* — the machine language of the Java Virtual Machine (Java VM).

Because the Java VM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the Solaris™ Operating System (Solaris OS), Linux, or Mac OS.



- **Portable**: In Java, many types of computers and operating system are in use throughout the world and are connected to the Internet. For downloading programs through different platforms connected to the Internet, some portable, executable code is needed. **Java does not have implementation dependent aspects and it yields or gives same result on any machine.**

- **Distributed**: In distributed application multiple client system depends on multiple server systems so that even problem occurred in one server will never be reflected on any client system. In java we can create distributed applications using RMI (Remote Method Invocation), EJB (Enterprises Java Beans) and has extensive set of library which works in agreement with TCP/IP to designed network based application.



- **Robust (Strong/ Powerful):** Most programs in use today fail for one of the two reasons: memory management or exceptional conditions. Thus, the ability to create robust programs was given a high priority in the design of Java. Java forces the user to find mistake in the early stages of program development. Java checks code at compilation time. However, it also checks the code at run time also. **Java programs will not crash because of its exception handling and its memory management features**.

- **Interpreted:** Java programs are compiled to generate the **byte code**. This byte code can be downloaded and interpreted by

the interpreter. **.class** file will have byte code instructions and JVM which contains an interpreter will execute the byte code.

- **High Performance**:
  It has high performance because of following reasons:
  - **Garbage collector**, collect the unused memory space and improve the performance of the application.
  - It support **multithreading**, because of this time consuming process can be reduced to executing the program.
  - Along with interpreter there will be JIT (Just In Time) compiler which enhances the speed of execution.
- **Multithreaded**: Executing different parts of program simultaneously is called multithreading. This is an essential feature to design server side programs.
- **Dynamic**: We can develop programs in Java which dynamically change on Internet (e.g.: Applets).

1.5 COMPARISON IN JAVA AND C++

1	Java is true Object oriented language, since it is not possible to write a java program without using atleast one class.	C++ is not a purely object-oriented programming language, since it is possible to write C++ programs without using a class or an object.
2	Java does not support operator overloading.	C++ supports operator overloading.
3	It supports labels with loops and statement blocks	It supports goto statement.
4	Java compiled into byte code for the Java Virtual Machine. The source code is independent on operating system.	Source code can be written to be platform independent C++ typically compiled into machine code.
5	Java does not support multiple inheritance of classes but it supports interface.	C++ supports multiple inheritance.

6	Allocation and deallocation of memory will be taken care of by JVM.	Allotting memory and deallocating memory is the responsibility of the programmer.
7	Java does not support global variable. Every variable should declare in class.	C++ support global variable.
8	Java does not use pointer.	C++ uses pointer.

1.6 JAVA ENVIRONMENT

Java environment includes a large number of development tools and thousands of classes and methods. The development tools are part of the system known as ***Java Development Kit*** (JDK) and classes and methods are part of the ***Java standard Library*** (JSL), also known as the ***Application Programming Interface*** (API).

1.6.1 JAVA DEVELOPMENT KIT

The Java Development Kit comes with a collection of tools that are used for developing and running Java programs. They include:

Appletviewer (for running Java applets) : Enable us to run Java Applets(without actually using a Java-compatible browser).

Javac (Java compiler) : Which translates Java sourcecode to bytecode files that interpreter can understand.

Java (Java Interpreter) : Which runs applets and applications by reading and interpreting bytecode.

Javap(Java disassemble) : Which enables us to convert bytecode files into a program description.

Javah (for C header files) : Produce header file for use with native methods.

Javadoc (for creating HTML documents) : Create HTML-format documentation from Java source code files.

Jdb(Java debugger) : Which helps use to find errors in our programs.

1.6.2 APPLICATION PROGRAMMING INTERFACE

The Java Standard Library (or API) includes thousands of classes and methods grouped into several functional packages. Most commonly used packages are:

Language support Package: A collection of classes and methods required for implementing basic features of Java. **java.lang**

Utilities packages: A collection of classes to provide utility functions such as date and time functions. **java.util**

Input/Output package: A collection of classes required for input/output manipulation. **java.io**

Networking package: A collection of classes for communicating with other computers via internet. **java.net**

Applet package: This includes as set of classes that allows us to create Java applets. **java.applet**

1.6.3 JAVA RUNTIME ENVIRONMENT

The Java runtime Environment (JRE) allows the execution of programs developed in Java. It primarily comprises of the following:

Java Virtual Machine: It converts bytecode into machine code.

Runtime class libraries: These are the set of core class libraries that are required for the execution of Java program

User Interface toolkits: AWT and Swings are the examples of toolkit the support varied input methods for the users to interact with the application program.

1.7 JAVA VIRTUAL MACHINE

Java Virtual Machine (JVM) is the heart of entire Java program execution process. First of all, the .java program is converted into a **.**class file consisting of byte code instructions by the java compiler at the time of compilation. Remember, this java compiler is outside the JVM. This **.**class file is given to the JVM. JVM mainly performs following operations.

- Allocating sufficient memory space for the class properties.
- Provides runtime environment in which java bytecode can be executed

- Converting byte code instruction into machine level instruction.

JVM is separately available for every Operating System while installing java software so that **JVM is platform dependent**.

Note: Java is platform Independent but JVM is platform dependent because every Operating system have different-different JVM which is install along with JDK Software.

Following figure shows the architecture of Java Virtual Machine.

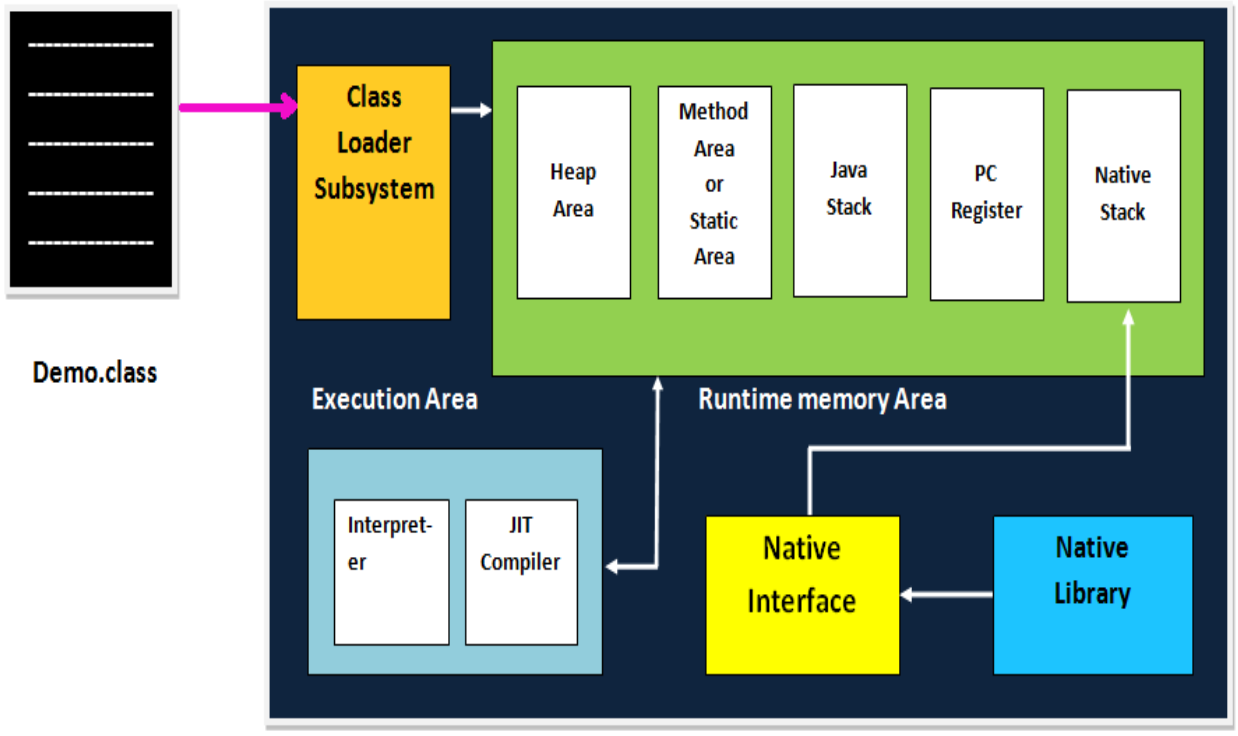


Figure :The internal architecture of the Java virtual machine.

In JVM, there is a module (or program) called class loader sub system, which performs the following instructions:

- First of all, it loads the .class file into memory.

- Then it verifies whether all byte code instructions are proper or not. If it finds any instruction suspicious, the execution is rejected immediately.
- If the byte instructions are proper, then it allocates necessary memory to execute the program. This memory is divided into 5 parts, called ***run time memory areas***, which contain the data and results while running the program. These areas are as follows:

1. **Heap Area** – All the **Objects** and its corresponding **instance variables** and **arrays** will be stored here.
2. **Method Area** – All the **Class level data** will be stored here including **static variables Static method and static block**.
3. **Java Stack Area** – In which all the non-static variable of class will be stored and whose address referred by object reference.
4. For every thread, a separate **runtime stack** will be created. For every **method call**, one entry will be made in the stack memory which is called as **Stack Frame**. All **local variables** will be created in the stack memory. Stack area is thread safe since it is not a shared resource. Stack Frame is divided into three sub-entities such as
    a. **Local Variable Array** – Related to the method how many **local variables**are involved and the corresponding values will be stored here.
    b. **Operand stack** – If any intermediate operation is required to perform, **operand stack** act as runtime workspace to perform the operation.
    c. **Frame data** – All symbols corresponding to the method is stored here. In the case of any **exception**, the catch block information will be maintained in the frame data.
5. **PC Registers** – Each thread will have separate **PC Registers,** to hold address of **current executing instruction** once the instruction is executed the PC register will be **updated** with the next instruction. It holds the address of next executable instruction that means that use the priority for the method in the execution process?
6. **Native Method Stacks:** When a Java application invokes a native method, that application does not only use *Java Stacks* but also uses the *native method stack* for the execution

of native methods (for example C/C++ functions). To execute the native methods, generally native method libraries (for example C/C++ header files) are required. The libraries required for the execution of native methods are available to the Java Virtual Machine through **Java Native Interface**.

7. **Execution Engine** Which contains **Interpreter** and **JIT compiler** whenever any java program is executing at the first time interpreter will comes into picture and it converts one by one byte code instruction into machine level instruction JIT compiler (just in time compiler) will comes into picture from the second time onward if the same java program is executing and it gives the machine level instruction to the process which are available in the buffer memory. JVM will identify the Hot spots in the .class files and it will give it to JIT compiler where the normal instructions and statements of Java program are executed by the Java interpreter.

Note: The main aim of JIT compiler is to speed up the execution of java program.

Generally, any language (like C/C++, pascal, cobol, etc.) will use either an interpreter or a compiler to translate the source code into a machine code. But in JVM, we got interpreter and JIT compiler both working at the same time on byte code to translate it into machine code. Now, the main question is why both are needed and how both work simultaneously? To understand this, let us take some sample code. Assume these are byte code instructions:

print a;

print b;

Repeat the following 10 time by changing i value from 1 to 10.

When, the interpreter starts execution from 1st instruction, it converts **print a**; into machine code and gives it to the microprocessor, For this, say the interpreter has taken 2

nanoseconds time. The processor takes it, executes it, and the value of **a** is displayed.

Now the interpreter comes back into memory and reads the 2nd instruction **print b**;. To convert this into machine code, it takes another 2 nanoseconds. Then the instruction is given to the processor and it execute it.

Next, the interpreter comes to the 3rd instruction, which is a looping statement **print a**; This should be done 10 times and this is known to the interpreter. Hence, the first time it converts **print a**; into machine code, it takes 2 nanosecond. After giving the instruction to the processor, it comes back to the memory and reads the **print a** instruction the 2nd time converts it to machine code. This will take another 2 nanoseconds. This will given to the processor to execute. Like this, the interpreter will convert the **print a** instruction for 10 times, consuming a total of 10 x 2 = 20 nanoseconds. This procedure is not efficient in terms of time. That is the reasons why JVM does not allocate this code to the interpreter. It allows this code to the JIT compiler.

Let us see how JIT compiler will execute the looping instruction. First of all, the JIT compiler reads the **print a** instruction and converts that into machine code. For this, say, it is taking 2 nanoseconds. Then the JIT compiler allots a block of memory and pushes this machine code instruction into that memory. For this, say, it is taking another 2 nanoseconds. This means JIT compiler has taken a total of 4 nanoseconds. Now, the processor will fetch this instruction from memory and execute it 10 times. Just observe that the JIT compiler has taken only 4 nanoseconds for execution; whereas to execute the same loop, the interpreter needs 20 nanoseconds. Thus, JIT compiler increase the speed of execution.

Byte Code: It is the unique characteristic property of the Java programming language. It is something like a normal text file. Therefore, it cannot be affected by virus. It is an intermediate between a human readable source and machine readable source. Byte codes are plateform-independent. Therefore, JVM is plateform dependent to make Java programming plateform independent.

1.7 Process of Compiling and Running a Java Application Program

All Java source code is written in text editor and saved with the .Java extension. Source code file are compiled into .class file by the java compiler. A .class file contains byte codes (plate-form independent intermediate code between a human readable source and machine readable source).

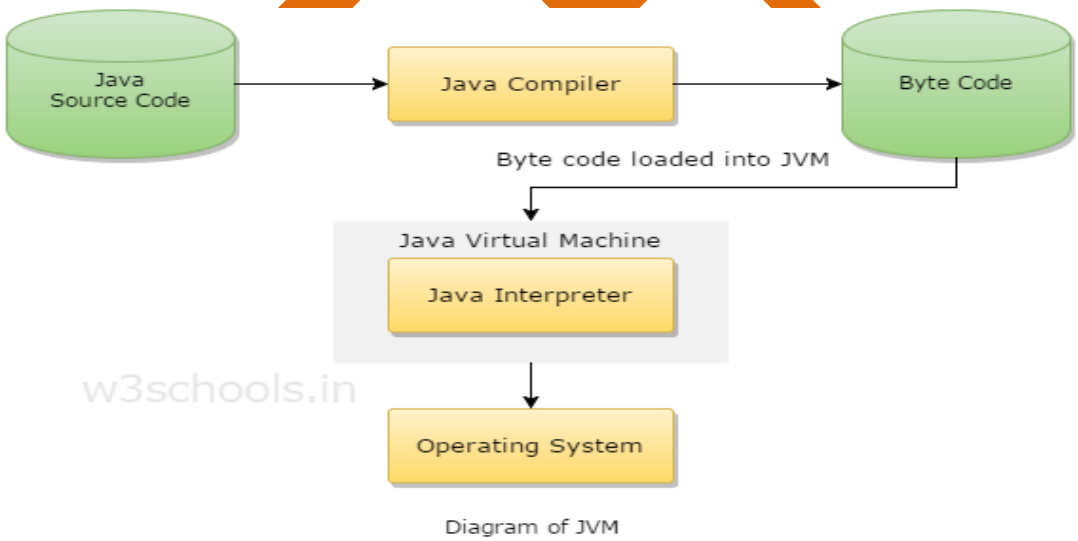


Figure Abstract view of Java program execution.

JVM works on different operating systems. The .class files (bytecode) can run on various operating systems. JVM converts bytecode into computer-readable machine code that can be executed.

Compiling: is the process of translating source code written in a particular programming language into computer-readable machine code that can be executed.

C:\> javac Sample.java

This command will produce a file 'Sample.class', which is used for running the program with the command 'java'.

Running: is the process of executing program on a computer.

C:\> java Sample

1.8 Java divided into three categories, they are

- J2SE (Java 2 Standard Edition) J2SE is used for developing client side applications.
- J2EE (Java 2 Enterprise Edition) J2EE is used for developing server side applications.
- J2ME (Java 2 Micro or Mobile Edition) J2ME is used for developing mobile or wireless application by making use of a predefined protocol called WAP(wireless Access / Application protocol).

1.9 How to Set Path and Classpath in Java

Setting path and classpath in Java is very simple but before do this process you need to know about path variable and classpath variable. Here I will show you **how to set path and classpath in Java** in very simple and easy way.

Path Variable

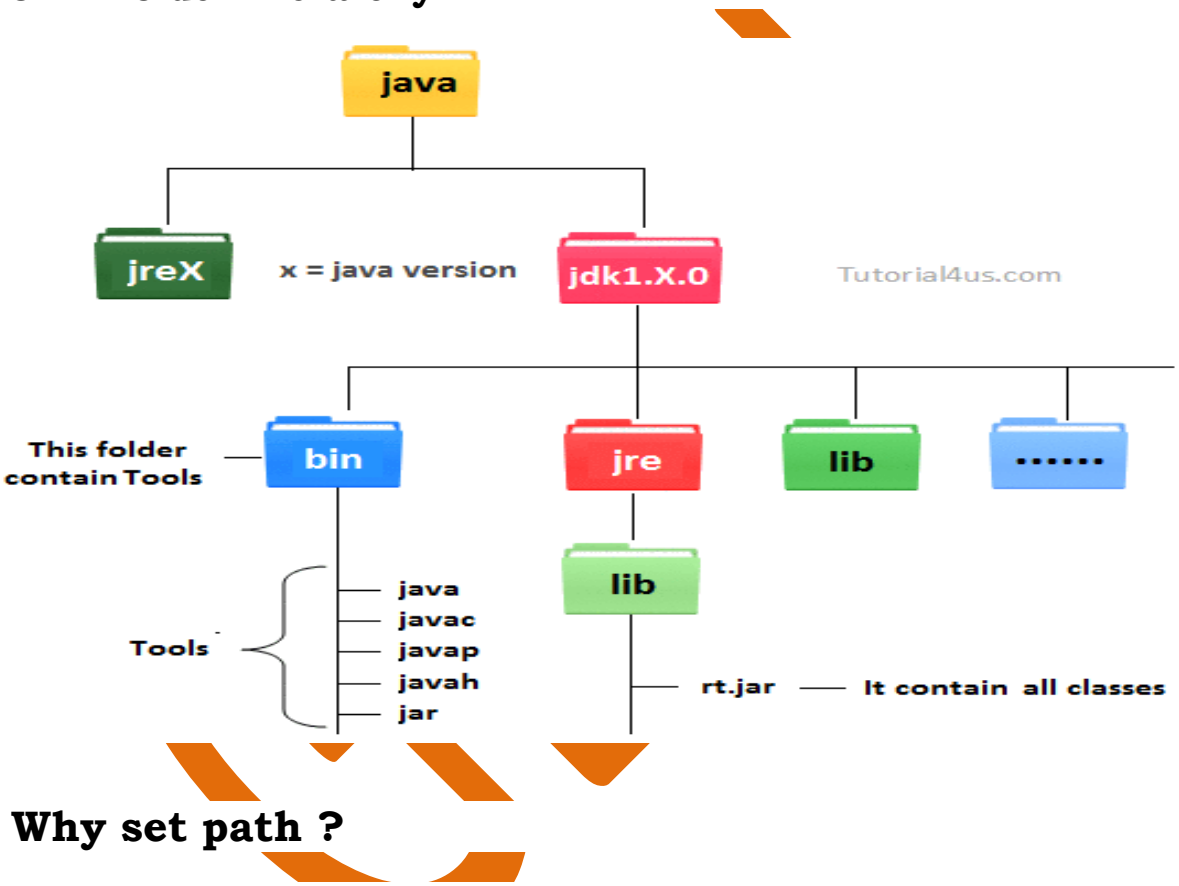
Path variable is set for providing path for all Java tools like java, javac, javap, javah, jar, appletviewer which are used in java

programming. All these tools are available in **bin** folders so we set path upto bin folders.

Classpath Variable

Classpath variable is set for providing a path for predefined Java classes which is used in our application. All classes are available in **lib/rt.jar** so we set classpath upto lib/rt.jar.

JDK Folder Hierarchy



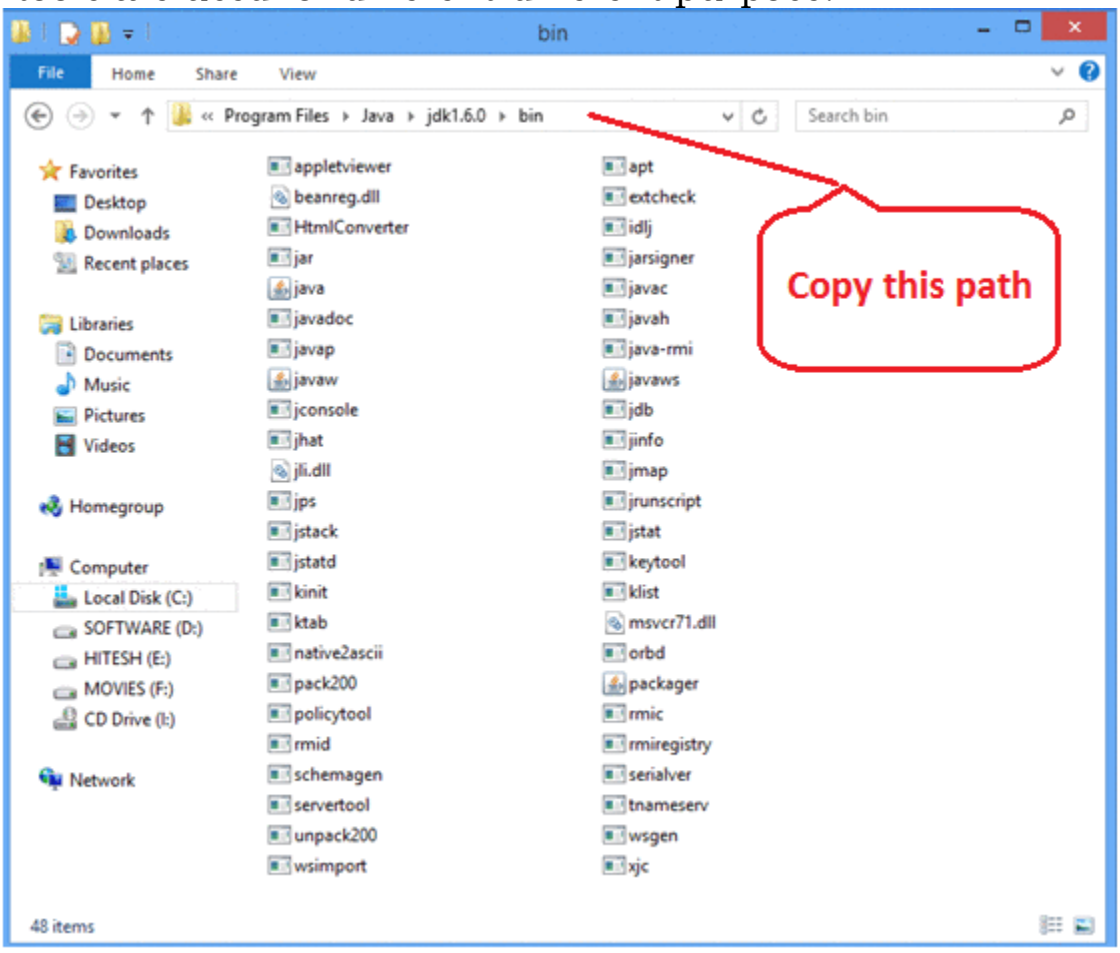
Why set path ?

The following programming error is generally for all Java programmers when they compile any Java program.

'javac' is not recognized as an internal or external command, operable program or batch file.

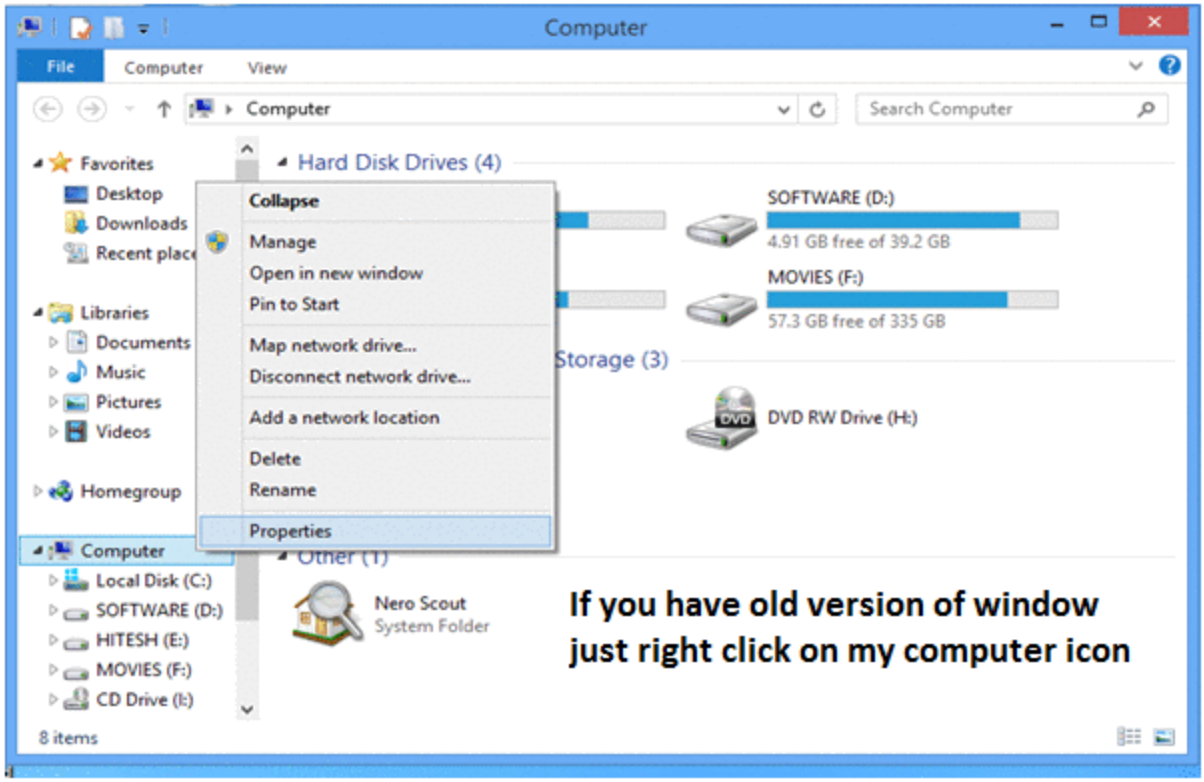
When you get this type of error, then your operating system cannot find the Java compiler **(javac)**. To solve this error you need to set the PATH variable.

Javac is a tool which is available in bin folder so you must set the PATH upto bin folder. In a **bin** folder all tools are available like **javap, javah, jar, javac, java, appletviewer** etc. All these tools are used for different-different purpose.

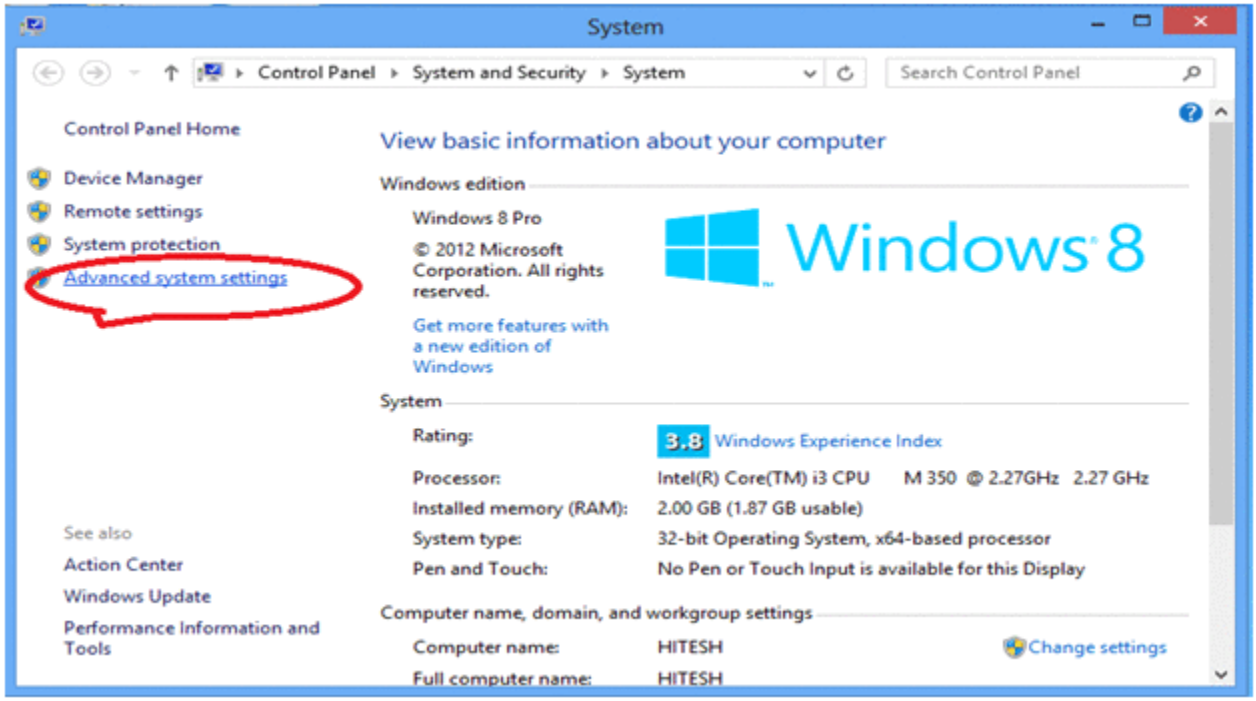


set the path and classpath

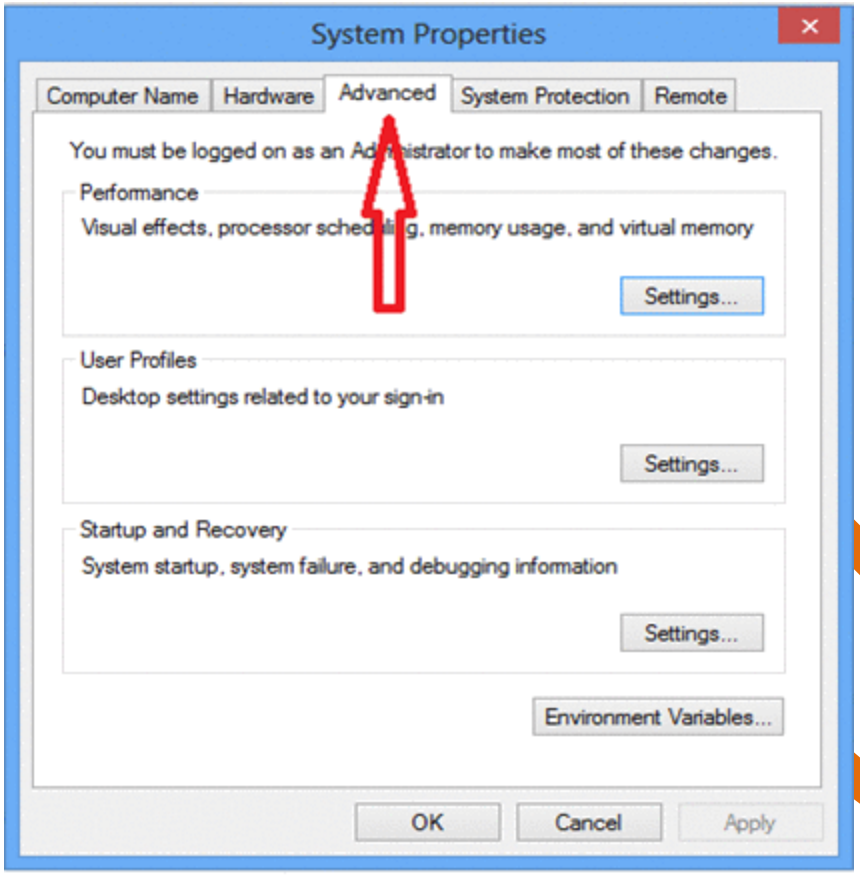
Go on my computer icon and right click, after that click on properties option.



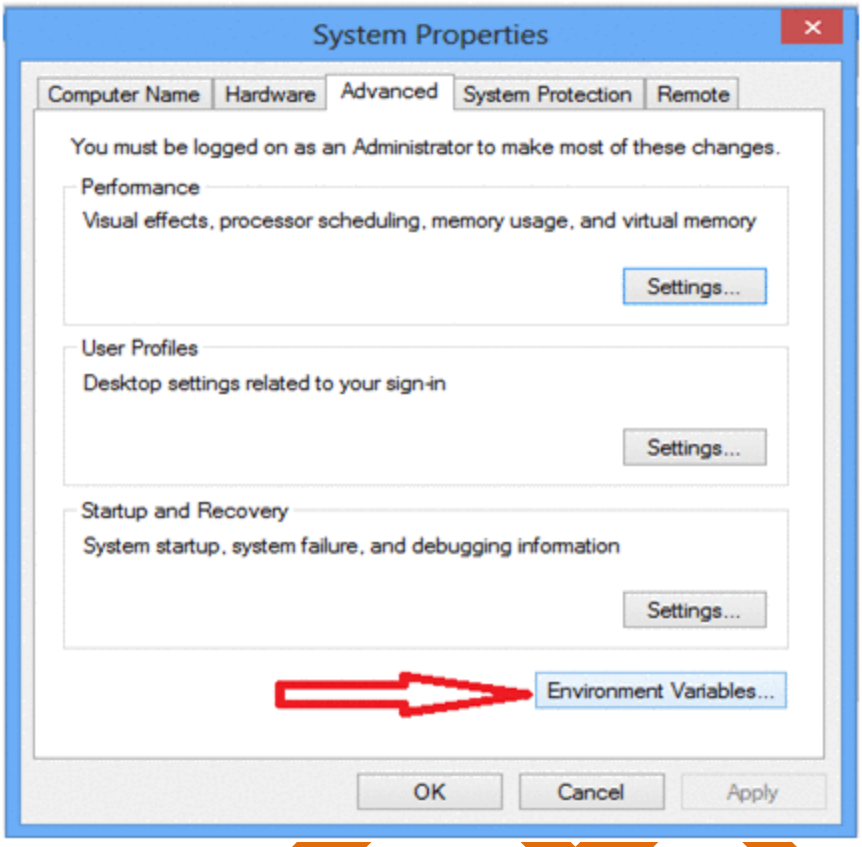
Now click on advance setting



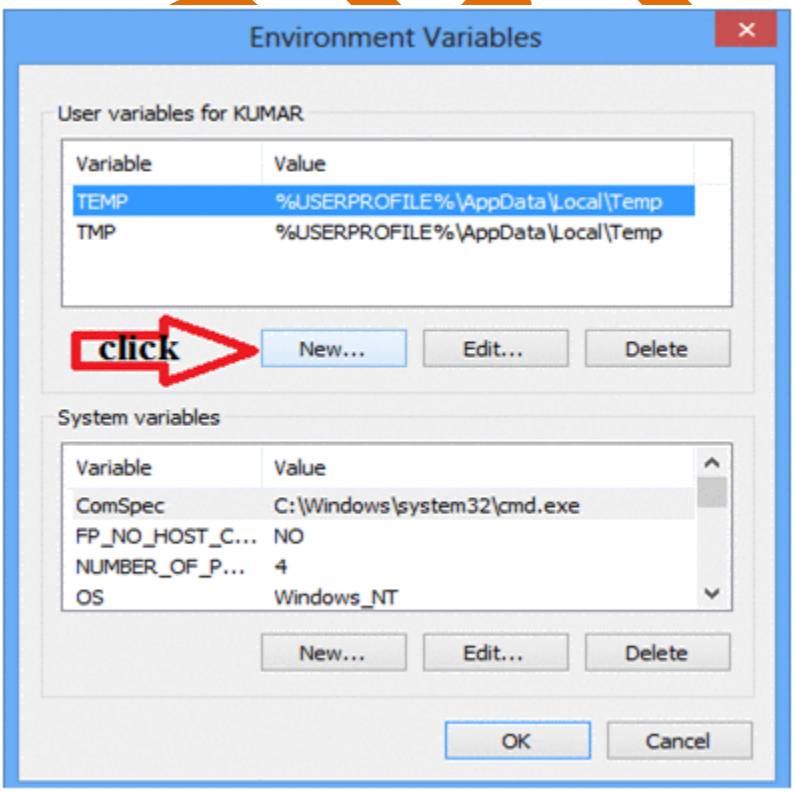
Click on advance



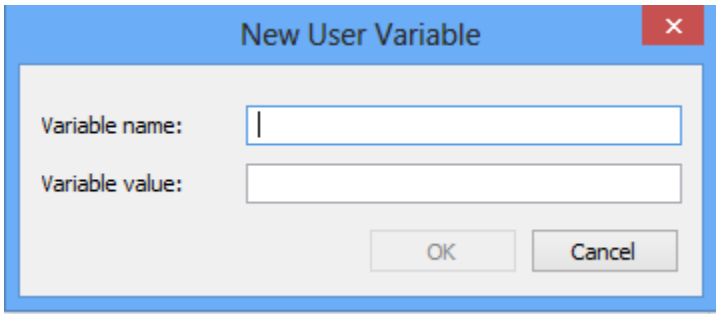
Click on Advance variables



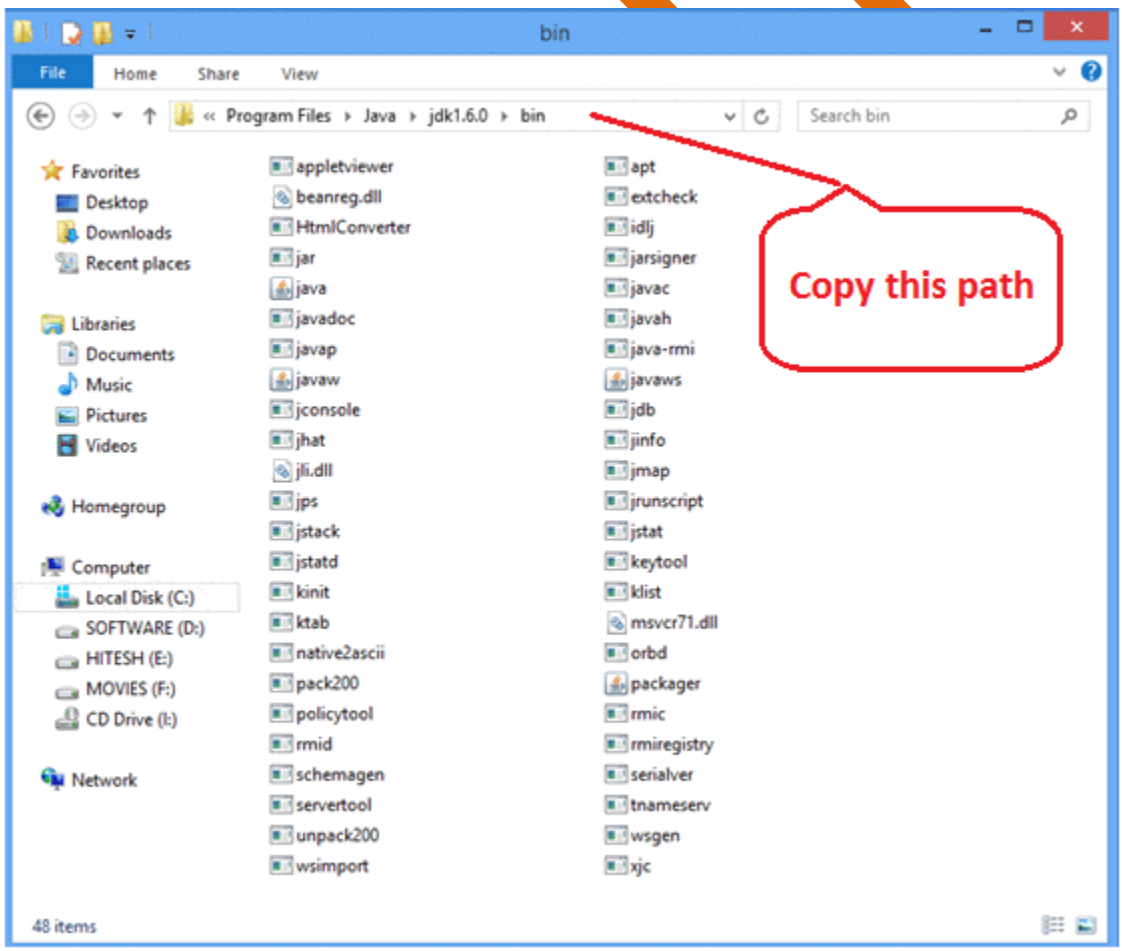
Click on new button which is below the first box.



Now one dilog box is appear, now ignore this but do not close.

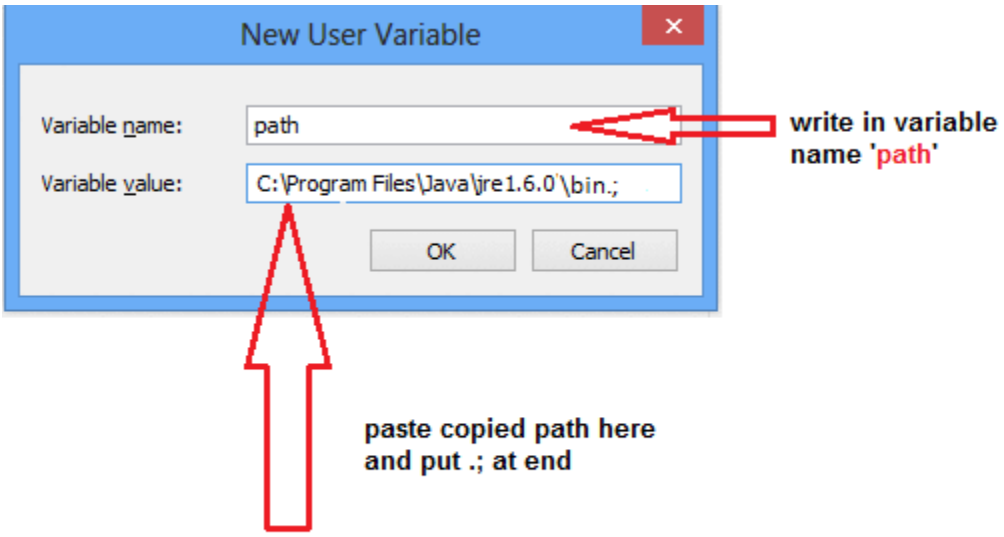


Now open my computer open c:/ > Programs Files > java > java1.6.0 > bin copy this path



Now come back on previous open dilogbox and write variable name '**path**' and for variable value paste all copied path upto the

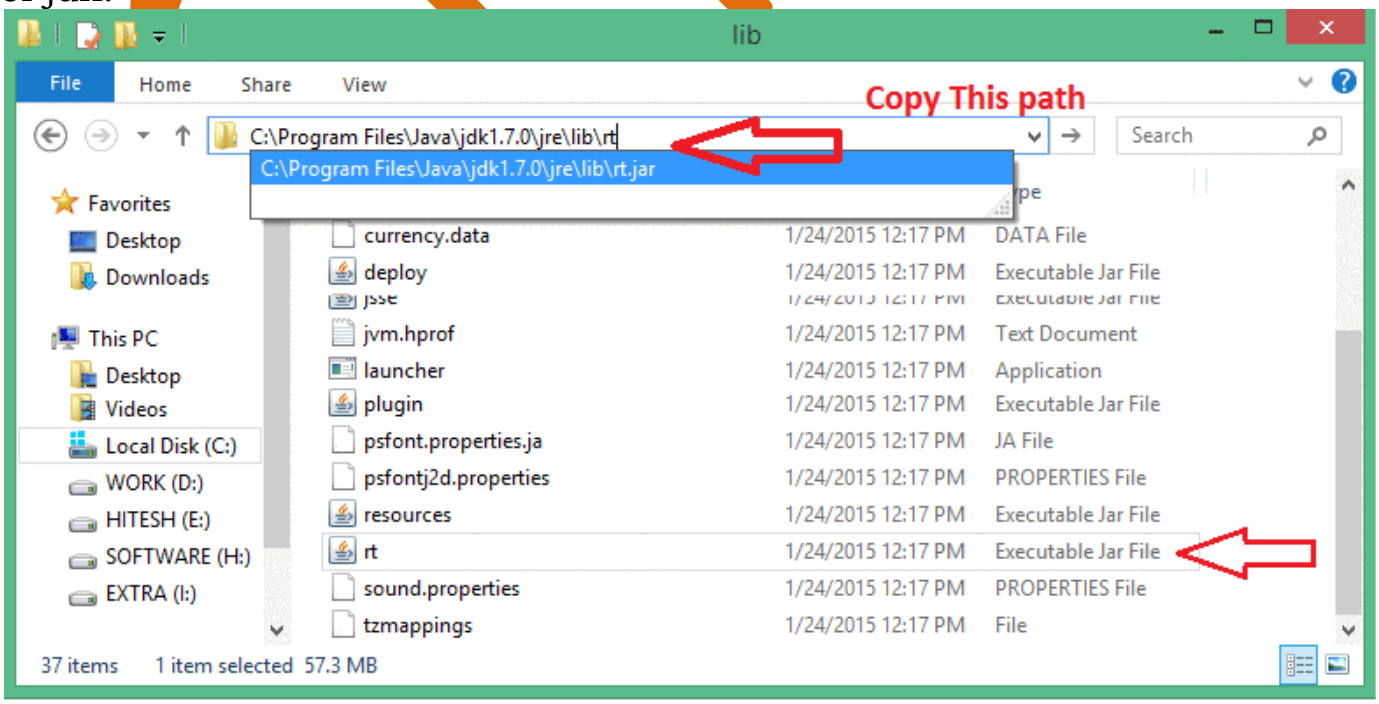
bin folder. Put .; at the end. It (.) selects all the tools from the bin folder.



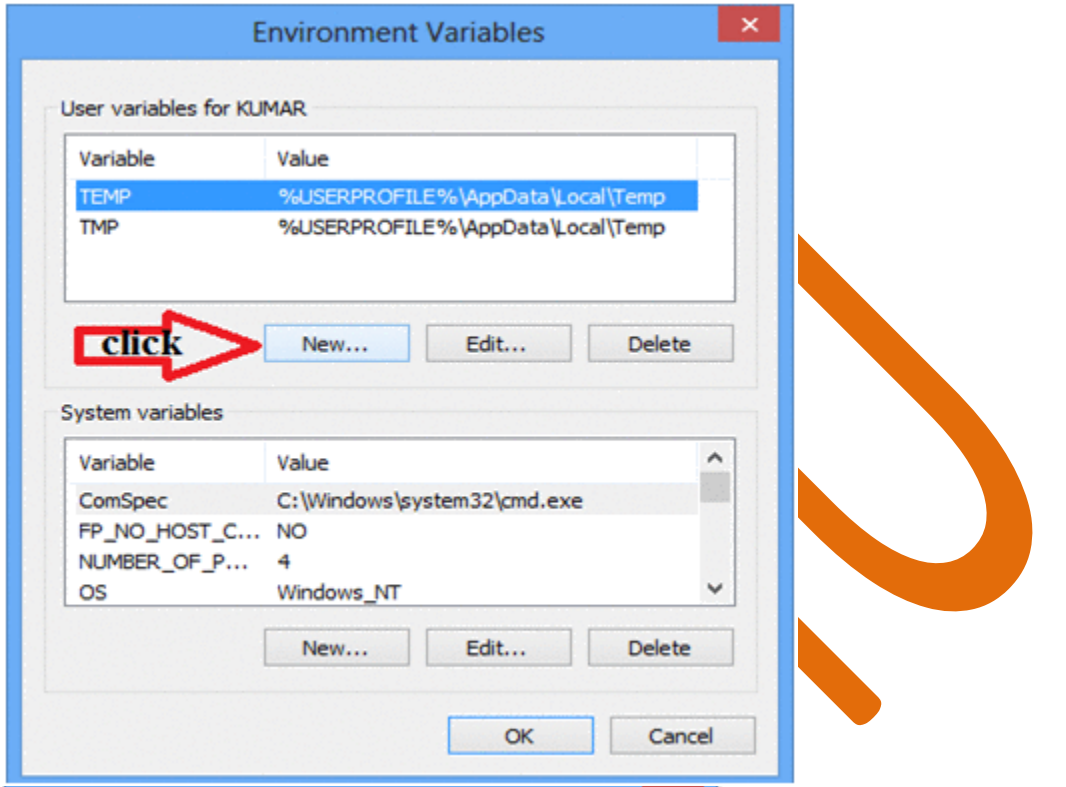
C\Program Files\java\jre\1.6.0\bin.;

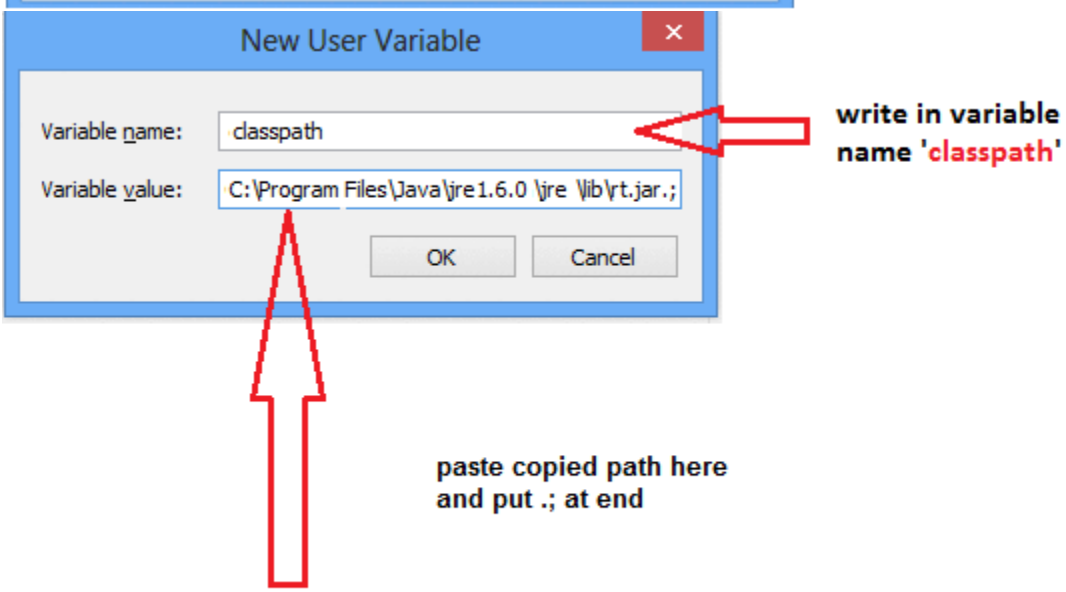
Now open my computer open c:/ > Programs Files > java > java1.6.0 > jre > lib > rt.jar copy this path

Note: rt.jar is available in lib folder this jar files contains all classes of jdk.



Now again come back on Environment variable dilogbox and click on new. Now one box is open and write path variable as '**classpath**' and for variable value paste all copied paths upto rt.jar. Put .; at the end. It (.) selects all the classes from lib folder.





C\Program Files\java\jre1.6.0\jre\lib\rt.jar.;

Note: Finally after set classpath restart your system, or you can re-open command prompt.

1.10 Structure of the Java Program:

As all other programming languages, Java also has a structure.

- There could be only one public class per source code file but it can have multiple non public class.
- In case there is any public class present in source code file, name of the file should be the class name.
- Sequence of difference statement in source code file would be ***package >> import >> Class declaration.***
- No Sequence rule is applied for *Comments. Comments* can be there in any part of the source code file at any location.
- File with no public class can have any name there is no rule applied for the same.
- ***Import*** and ***package*** statements applied to all the classes in same source code file.

1.11 Rules Applied on Java Source Code File

Classes are written in Java source file. A source file can contain more than one Java class. Below are the rules related to Java source code file?

- The first line of the C/C++ program contains include statement. For example, <stdio.h> is the header file that contains functions, like printf (), scanf () etc. So if we want to use any of these functions, we should include this header file in C/ C++ program.
- Similarly in Java first we need to import the required packages. By default **java.lang.*** is imported. Java has several such packages in its library.

- A package is a kind of directory that contains a group of related classes and interfaces. A class or interface contains methods.
- Since Java is purely an Object Oriented Programming language, we cannot write a Java program without having at least one class or object.
- So, it is mandatory to write a class in Java program. We should use class keyword for this purpose and then write class name.
- In C/C++, program starts executing from main method similarly in Java, program starts executing from main method. The return type of main method is void because program starts executing from main method and it returns nothing.

1.12 Naming Conventions

Naming conventions specify the rules to be followed by a Java programmer while writing the names of packages, classes, methods etc.

- Package names are written in small letters.

  e.g.: java.io, java.lang, java.awt etc

- Each word of class name and interface name starts with a capital

  e.g.: Sample, AddTwoNumbers

- Method names start with small letters then each word start with a capital

  e.g.: sum (), sumTwoNumbers (), minValue ()

- Variable names also follow the same above method rule

  e.g.: sum, count, totalCount

- Constants should be written using all capital letters

  e.g.: PI, COUNT

- Keywords are reserved words and are written in small letters.

  e.g.: int, short, float, public, void

First Java Program

Sample Program:

```
//A Simple Java Program
import java.lang.System;
import java.lang.String;
class Sample
{
        public static void main(String args[])
                {
                        System.out.print ("Hello world");
                }
}
```

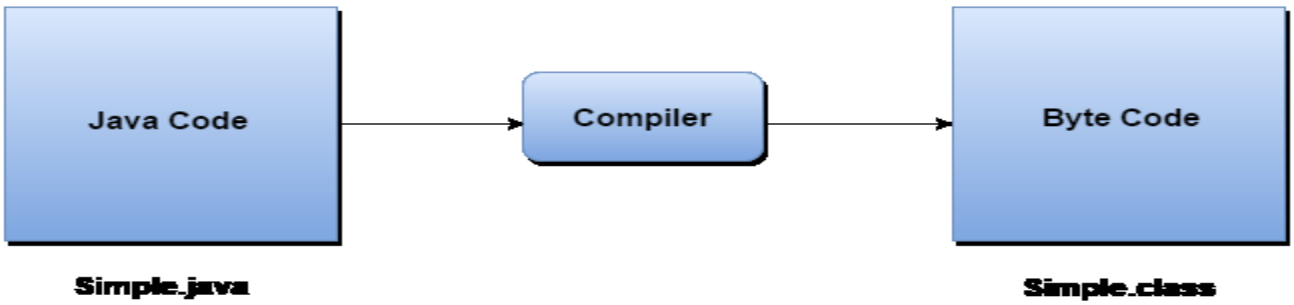
- Since Java is purely an Object Oriented Programming language, without creating an object to a class it is not possible to access methods and members of a class.
- But main method is also a method inside a class, since program execution starts from main method we need to call main method without creating an object.
- Static methods are the methods, which can be called and executed without creating objects.
- Since we want to call main() method without using an object, we should declare main() method as static. JVM calls main() method using its Classname.main() at the time of running the program.
- JVM is a program written by Java Soft people (Java development team) and main() is the method written by us. Since, main () method should be available to the JVM, it should be declared as public. If we don't declare main () method as public, then it doesn't make itself available to JVM and JVM cannot execute it.

- JVM always looks for main() method with String type array as parameter otherwise JVM cannot recognize the main() method, so we must provide String type array as parameter to main () method. Main() method having signature with default variable arguments.

- A class code starts with a {and ends with a}.
- A class or an object contains variables and methods (functions). We can create any number of variables and methods inside the class.
- This is our first program, so we had written only one method called main().
- Our aim of writing this program is just to display a string "Hello world".
- In Java, print() or write() method is used to display something on the monitor.
- A method should be called by using **objectname.methodname()**. So, to call **print()** method, create an object to **PrintStream** class then call **objectname.print()** method.
- An alternative is given to create an object to **PrintStream Class** i.e. **System.out**.
- Here, **System** is the class name and **out** is a static variable in System class **out** is called a field in System class.
- When we call this field a **PrintStream class object will be created internally**. So, we can call print() method as: System.out.print ("Hello world");
- println() is also a method belonging to PrintStream class. It throws the cursor to the next line after displaying the result.
- In the above Sample program System and String are the classes present in java.lang package.

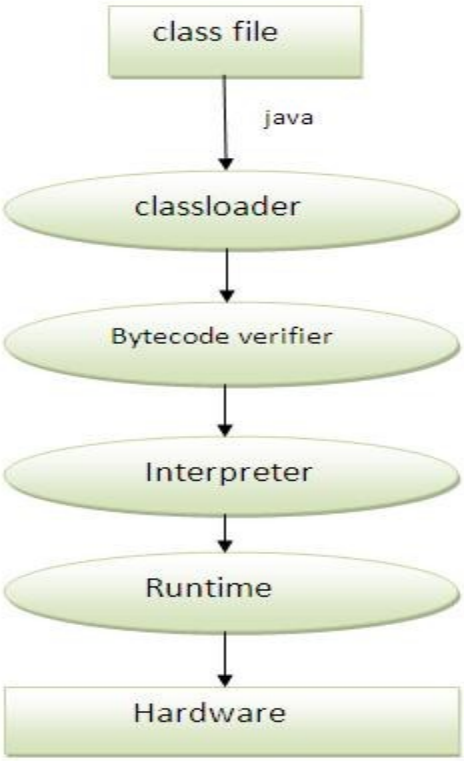
What happens at compile time?

At compile time, java file is compiled by Java Compiler (It does not interact with OS) and converts the java code into bytecode.



What happens at runtime?

At runtime, following steps are performed:



Classloader: is the subsystem of JVM that is used to load class files.

Bytecode Verifier: checks the code fragments for illegal code that can violate access right to objects.

Interpreter: read bytecode stream then execute the instructions.

1.13. Recommended Reading

The Java Platform

A *platform* is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Microsoft Windows, Linux, Solaris OS, and Mac OS.

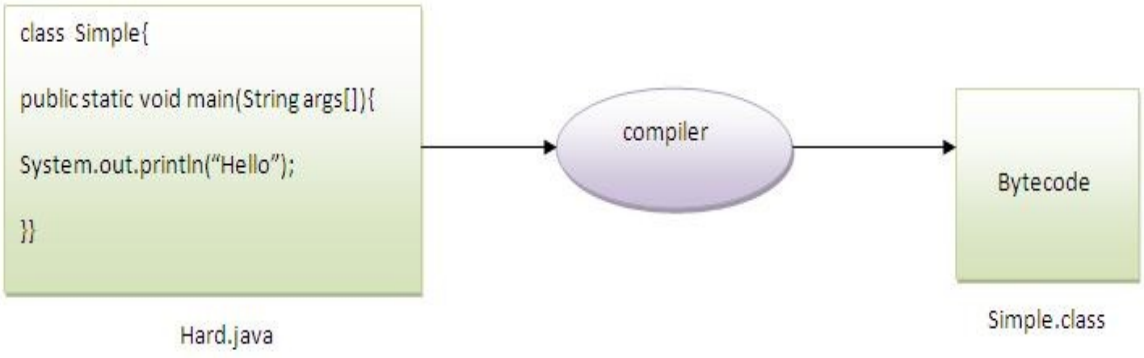
Most platforms can be described as a combination of the operating system and underlying hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

The Java platform has two components:

- The *Java Virtual Machine*
- The *Java Application Programming Interface* (API)

Can you save a java source file by other name than the class name?

Yes, if the class is not public. It is explained in the figure given below:

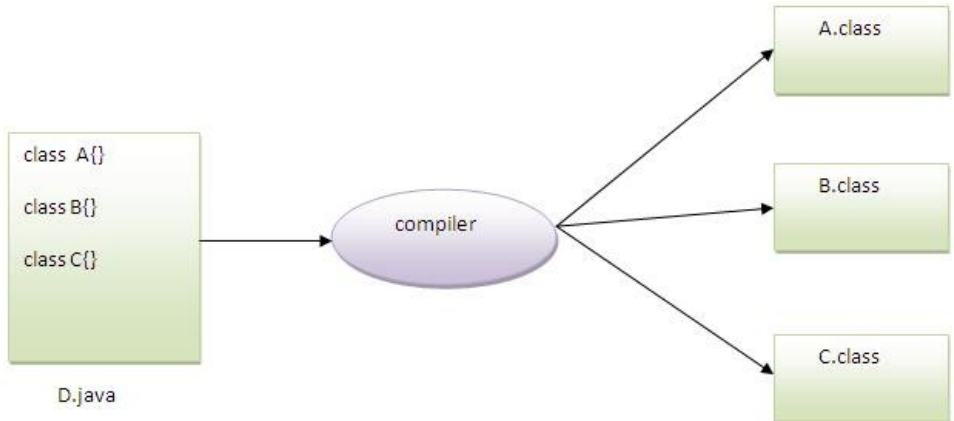


To compile: javac Simple.java

To execute: java Simple

Q) Can you have multiple classes in a java source file?

Yes, like the figure given below illustrates:



Difference between JDK, JRE, and JVM

JVM

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode. JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each OS is different from each other. However, Java is platform independent. There are three notions of the JVM: *specification*, *implementation*, and *instance*.

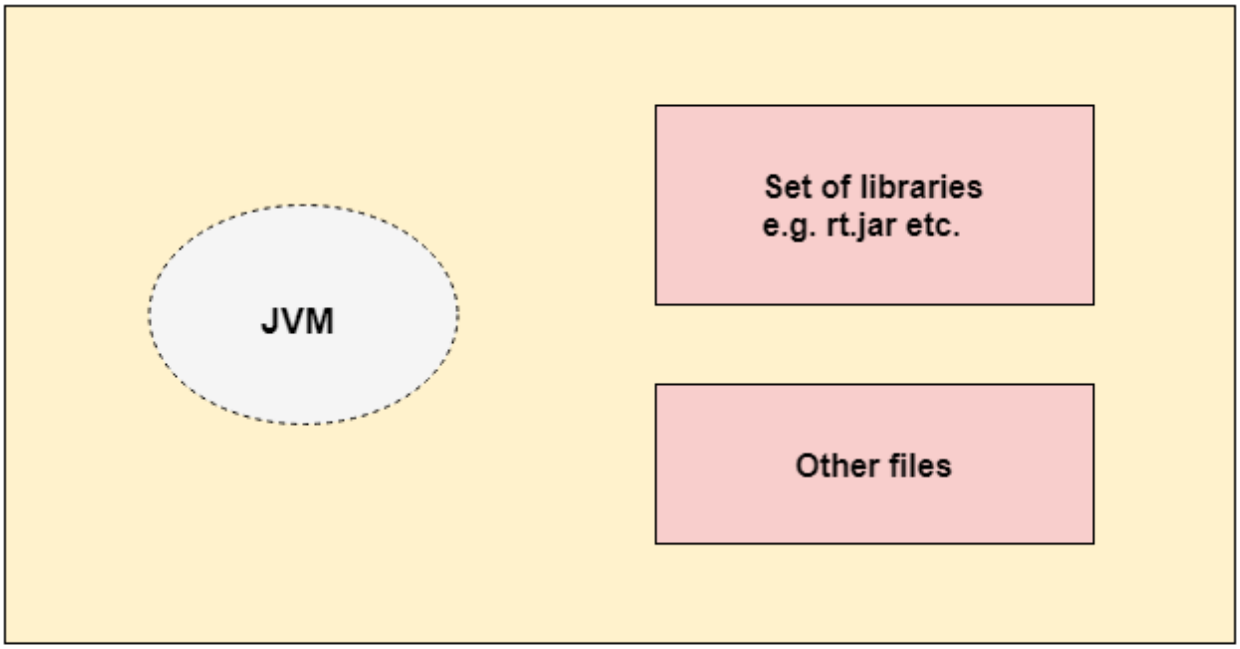
The JVM performs the following main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JRE

JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

The implementation of JVM is also actively released by other companies besides Sun Micro Systems.



JRE

JDK

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.

JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

- Standard Edition Java Platform
- Enterprise Edition Java Platform
- Micro Edition Java Platform

The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.