JAVA INTRODUCTION

Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. This tutorial gives a complete understanding of Java. This reference will take you through simple and practical approaches while learning Java Programming language.
Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]).
The latest release of the Java Standard Edition is Java SE 8. With the advancement of Java and its widespread popularity, multiple configurations were built to suit various types of platforms. For example: J2EE for Enterprise Applications, J2ME for Mobile Applications.
The new J2 versions were renamed as Java SE, Java EE, and Java ME respectively. Java is guaranteed to be **Write Once, Run Anywhere.**
Java is −

- **Object Oriented** − In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- **Platform Independent** − Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
- **Simple** − Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
- **Secure** − With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **Architecture-neutral** − Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
- **Portable** − Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.
- **Robust** − Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
- **Multithreaded** − With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
- **Interpreted** − Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
- **High Performance** − With the use of Just-In-Time compilers, Java enables high performance.
- **Distributed** − Java is designed for the distributed environment of the internet.
- **Dynamic** − Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

History of Java

James Gosling initiated Java language project in June 1991 for use in one of his many set-top box projects. The language, initially called ‘Oak’ after an oak tree that stood outside Gosling's office, also went by the name ‘Green’ and ended up later being renamed as Java, from a list of random words.
Sun released the first public implementation as Java 1.0 in 1995. It promised **Write Once, Run Anywhere** (WORA), providing no-cost run-times on popular platforms.
On 13 November, 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL).
On 8 May, 2007, Sun finished the process, making all of Java's core code free and open-source, aside from a small portion of code to which Sun did not hold the copyright.

Popular Java Editors

To write your Java programs, you will need a text editor. There are even more sophisticated IDEs available in the market. But for now, you can consider one of the following −

- **Notepad** − On Windows machine, you can use any simple text editor like Notepad (Recommended for this tutorial), TextPad.
- **Netbeans** − A Java IDE that is open-source and free which can be downloaded from https://www.netbeans.org/index.html.
- **Eclipse** − A Java IDE developed by the eclipse open-source community and can be downloaded from https://www.eclipse.org/.

Application

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in, javatpoint.com, etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games, etc.

Types of Java Applications

There are mainly 4 types of applications that can be created using Java programming:

1) Standalone Application

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications, etc. is called enterprise application. It has advantages of the high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

4) Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

Java Platforms / Editions

There are 4 platforms or editions of Java:

1) Java SE (Java Standard Edition)

It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

2) Java EE (Java Enterprise Edition)

It is an enterprise platform which is mainly used to develop web and enterprise applications. It is built on the top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc.

3) Java ME (Java Micro Edition)

It is a micro platform which is mainly used to develop mobile applications.

4) JavaFX

It is used to develop rich internet applications. It uses a light-weight user interface API.

First Java Program

Let us look at a simple code that will print the words ***Hello World***.

Example

Live Demo

```
public class MyFirstJavaProgram {
   /* This is my first java program.
    * This will print 'Hello World' as the output
    */
   public static void main(String []args) {
      System.out.println("Hello World"); // prints Hello World
   }
}
```

Let's look at how to save the file, compile, and run the program. Please follow the subsequent steps −

- Open notepad and add the code as above.
- Save the file as: MyFirstJavaProgram.java.
- Open a command prompt window and go to the directory where you saved the class. Assume it's C:\.
- Type 'javac MyFirstJavaProgram.java' and press enter to compile your code. If there are no errors in your code, the command prompt will take you to the next line (Assumption : The path variable is set).
- Now, type ' java MyFirstJavaProgram ' to run your program.
- You will be able to see ' Hello World ' printed on the window.

Output

```
C:\> javac MyFirstJavaProgram.java
C:\> java MyFirstJavaProgram
Hello World
```

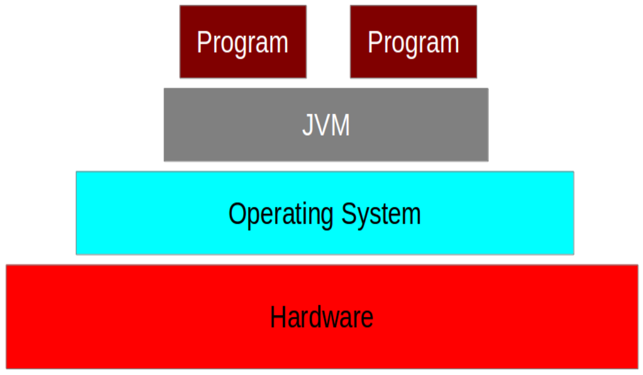
Basic Syntax

About Java programs, it is very important to keep in mind the following points.

- **Case Sensitivity** − Java is case sensitive, which means identifier **Hello** and **hello** would have different meaning in Java.
- **Class Names** − For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.
  Example: *class MyFirstJavaClass*
- **Method Names** − All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.
  Example: *public void myMethodName()*
- **Program File Name** − Name of the program file should exactly match the class name.
  When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match, your program will not compile).
  Example: Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as *'MyFirstJavaProgram.java'*
- **public static void main(String args[])** − Java program processing starts from the main() method which is a mandatory part of every Java program.

The Java virtual machine (JVM) is a software implementation of a computer that executes programs like a real machine.

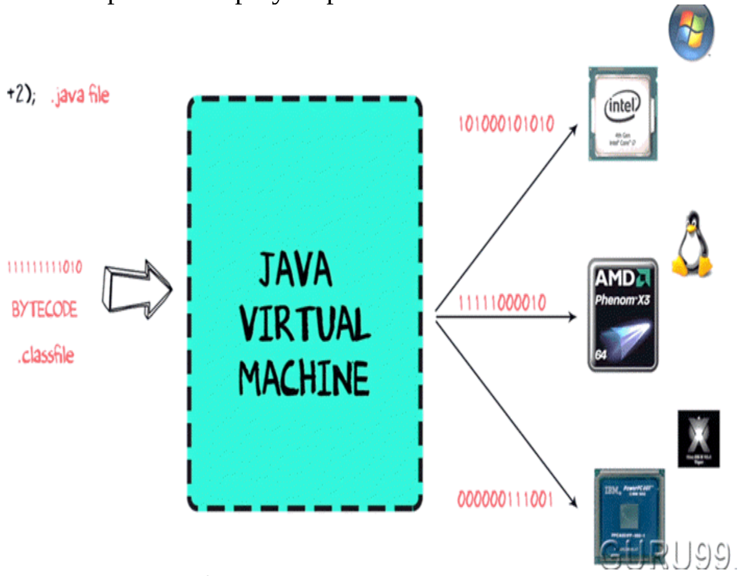
The Java virtual machine is written specifically for a specific operating system, e.g., for Linux a special implementation is required as well as for Windows.



Java programs are compiled by the Java compiler into *bytecode*. The Java virtual machine interprets this *bytecode* and executes the Java program.

How Java Virtual Machine works?

By using **Java Virtual Machine**, this problem can be solved. But how it works on different processors and O.S. Let's understand this process step by step.



Step 1) The code to display addition of two numbers is System.out.println(1+2), and saved as .java file.

Step 2) Using the java compiler the code is converted into an intermediate code called the **bytecode.** The output is a **.class file.**

Step 3) This code is not understood by any platform, but only a virtual platform called the **Java Virtual Machine.**

Step 4) This Virtual Machine resides in the RAM of your operating system. When the Virtual Machine is fed with this bytecode, it identifies the platform it is working on and converts the bytecode into the native machine code.

In fact, while working on your PC or browsing the web whenever you see either of these icons be assured the java virtual machine is loaded into your RAM. But what makes java lucrative is that code once compiled can run not only on all PC platforms but also mobiles or other electronic gadgets supporting java.

Hence,

"Java is a programming language as well as a Platform"

How is Java Platform Independent?

Like C compiler, Java compiler does not produce native executable code for a particular machine. Instead, Java produces a unique format called bytecode. It executes according to the rules laid out in the virtual machine specification.

Bytecode is understandable to any JVM installed on any OS. In short, the java source code can run on all operating systems.

1.4. Java Runtime Environment vs. Java Development Kit

A Java distribution typically comes in two flavors, the *Java Runtime Environment* (JRE) and the *Java Development Kit* (JDK).

The JRE consists of the JVM and the Java class libraries. Those contain the necessary functionality to start Java programs.

The JDK additionally contains the development tools necessary to create Java programs. The JDK therefore consists of a Java compiler, the Java virtual machine and the Java class libraries.

1.5. Development Process with Java

Java source files are written as plain text documents. The programmer typically writes Java source code in an *Integrated Development Environment* (IDE) for programming. An IDE supports the programmer in the task of writing code, e.g., it provides auto-formating of the source code, highlighting of the important keywords, etc.

At some point the programmer (or the IDE) calls the Java compiler (javac). The Java compiler creates the *bytecode* instructions. These instructions are stored in .class files and can be executed by the Java Virtual Machine.

1.6. Garbage collector

The JVM automatically re-collects the memory which is not referred to by other objects. The Java *garbage collector* checks all object references and finds the objects which can be automatically released.

While the garbage collector relieves the programmer from the need to explicitly manage memory, the programmer still need to ensure that he does not keep unneeded object references, otherwise the garbage collector cannot release the associated memory. Keeping unneeded object references are typically called *memory leaks*.

1.7. Classpath

The *classpath* defines where the Java compiler and Java runtime look for .class files to load. These instructions can be used in the Java program.

For example, if you want to use an external Java library you have to add this library to your classpath to use it in your program.

Open a shell for command line access.

If you don’t know how to do this, google for "How to open a shell under [your_OS]".

Switch to the *javadir* directory with the command cd javadir, for example, in the above example via the cd \home\vogella\javastarter command. Use the ls command (dir under Microsoft Windows) to verify that the source file is in the directory.

Compile your Java source file into a class file with the following command.

javac HelloWorld.java

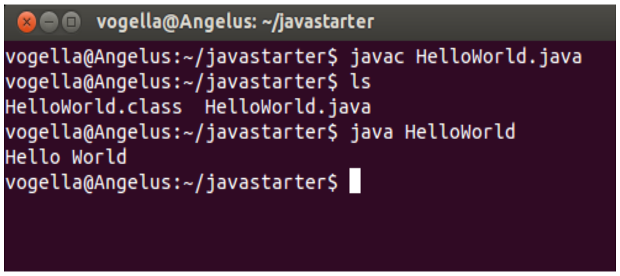
Afterwards list again the content of the directory with the ls or dir command. The directory contains now a file HelloWorld.class. If you see this file, you have successfully compiled your first Java source code into bytecode.

By default, the compiler puts each class file in the same directory as its source file. You can specify a separate destination directory with the -d compiler flag.

You can now start your compiled Java program. Ensure that you are still in the *jardir* directory and enter the following command to start your Java program.

java HelloWorld

The system should write "Hello World" on the command line.



```
vogella@Angelus: ~/javastarter
vogella@Angelus:~/javastarter$ javac HelloWorld.java
vogella@Angelus:~/javastarter$ ls
HelloWorld.class  HelloWorld.java
vogella@Angelus:~/javastarter$ java HelloWorld
Hello World
vogella@Angelus:~/javastarter$
```

3.3. Using the classpath

You can use the classpath to run the program from another place in your directory.

Switch to the command line, e.g., under Windows Start ▸ Run cmd. Switch to any directory you want. Type:

java HelloWorld

If you are not in the directory in which the compiled class is stored, then the system will show an error message: *Exception in thread "main" java.lang.NoClassDefFoundError: test/TestClass*

To use the class, type the following command. Replace "mydirectory" with the directory which contains the test directory. You should again see the "HelloWorld" output.

java -classpath "mydirectory" HelloWorld