Advanced Java Programming

Unit 1: Programming in Java

8 Hrs.

Introduction to Java

James Gosling

- James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of sun engineers called Green Team.
- Originally designed for small, embedded systems in electronic appliances like set-top boxes.
- Firstly, it was called "Greentalk" by James Gosling and file extension was .gt.
- After that, it was called Oak and was developed as a part of the Green project.
- Java is a programming language and a platform.
- Java is a high level, robust, secured and object-oriented programming language.

JSE, JEE & JME

Java SE, i.e. **J**ava **S**tandard **E**dition. It is normally used for developing desktop applications. It forms the core/base API.

Java EE, i.e. **J**ava **E**nterprise **E**dition. This was originally known as Java 2 Platform, Enterprise Edition or J2EE. The name was eventually changed to Java Platform, Enterprise Edition or Java EE in version 5. Java EE is mainly used for applications which run on servers, such as web sites.

Java ME, i.e. **J**ava **M**icro **E**dition. It is mainly used for applications which run on resource constrained devices (small scale devices) like cell phones, most commonly games.

Java Architecture

1. Compilation and interpretation in Java

Java combines both the approaches of compilation and interpretation. First, java compiler compiles the source code into bytecode. At the run time, Java Virtual Machine (JVM) interprets this bytecode and generates machine code which will be directly executed by the machine in which java program runs. So java is both compiled and interpreted language.

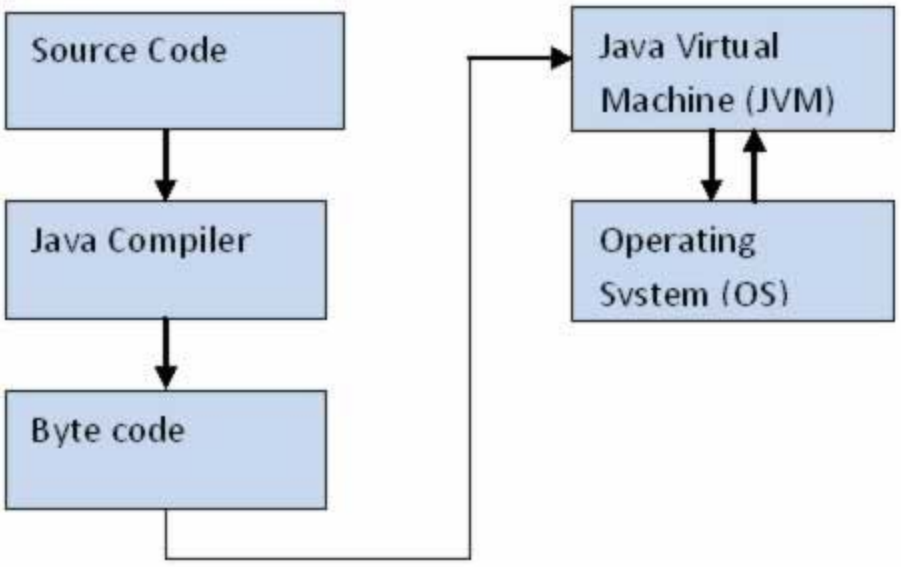


Figure 1: Java Architecture

2. Java Virtual Machine (JVM)

JVM is a component which provides an environment for running Java programs. JVM interprets the bytecode into machine code which will be executed the machine in which the Java program runs.

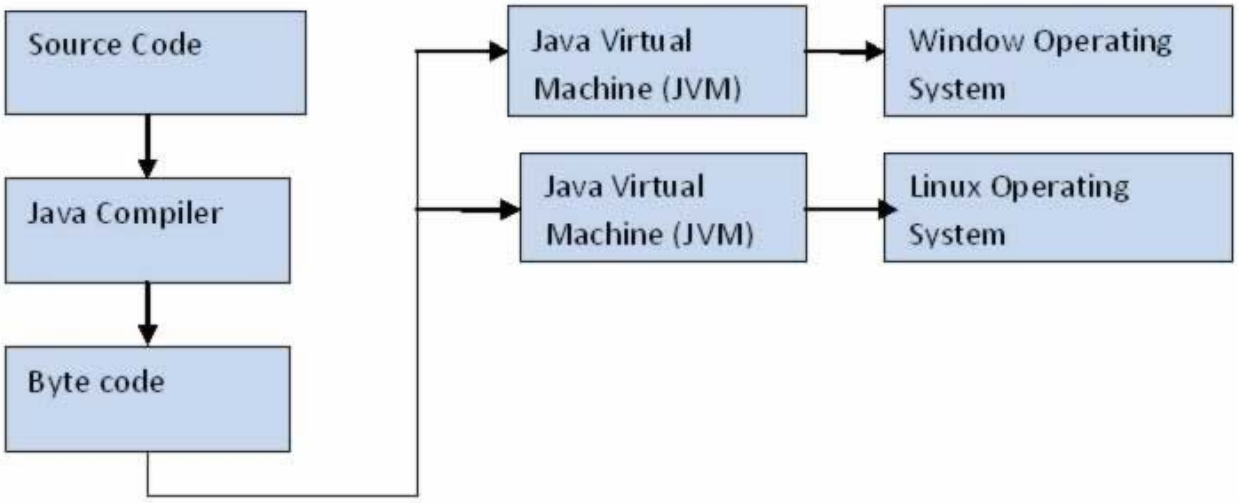
3. Why Java is Platform Independent?

Platform independence is one of the main advantages of Java. In other words, java is portable because the same java program can be executed in multiple platforms without making any changes in the source code. You just need to write the java code for one platform and the same program will run in any platforms. But how does Java make this possible?

As we discussed early, first the Java code is compiled by the Java compiler and generates the bytecode. This bytecode will be stored in class files. Java Virtual Machine (JVM) is unique for each platform. Though JVM is unique for each platform, all interpret the same bytecode and

convert it into machine code required for its own platform and this machine code will be directly executed by the machine in which java program runs. This makes Java platform independent and portable.

Let's make it more clear with the help of the following diagram. Here the same compiled Java bytecode is interpreted by two different JVMS to make it run in Windows and Linux platforms.



4. Java Runtime Environment (JRE) and Java Architecture in Detail

Java Runtime Environment contains JVM, class libraries and other supporting components. As you know the Java source code is compiled into bytecode by Java compiler. This bytecode will be stored in class files. During runtime, this bytecode will be loaded, verified and JVM interprets the bytecode into machine code which will be executed in the machine in which the Java program runs.

A Java Runtime Environment performs the following main tasks respectively.

1. Loads the class : This is done by the class loader
2. Verifies the bytecode : This is done by bytecode verifier.
3. Interprets the bytecode : This is done by the JVM

These tasks are described in detail in the subsequent sessions. A detailed Java architecture can be drawn as given below.

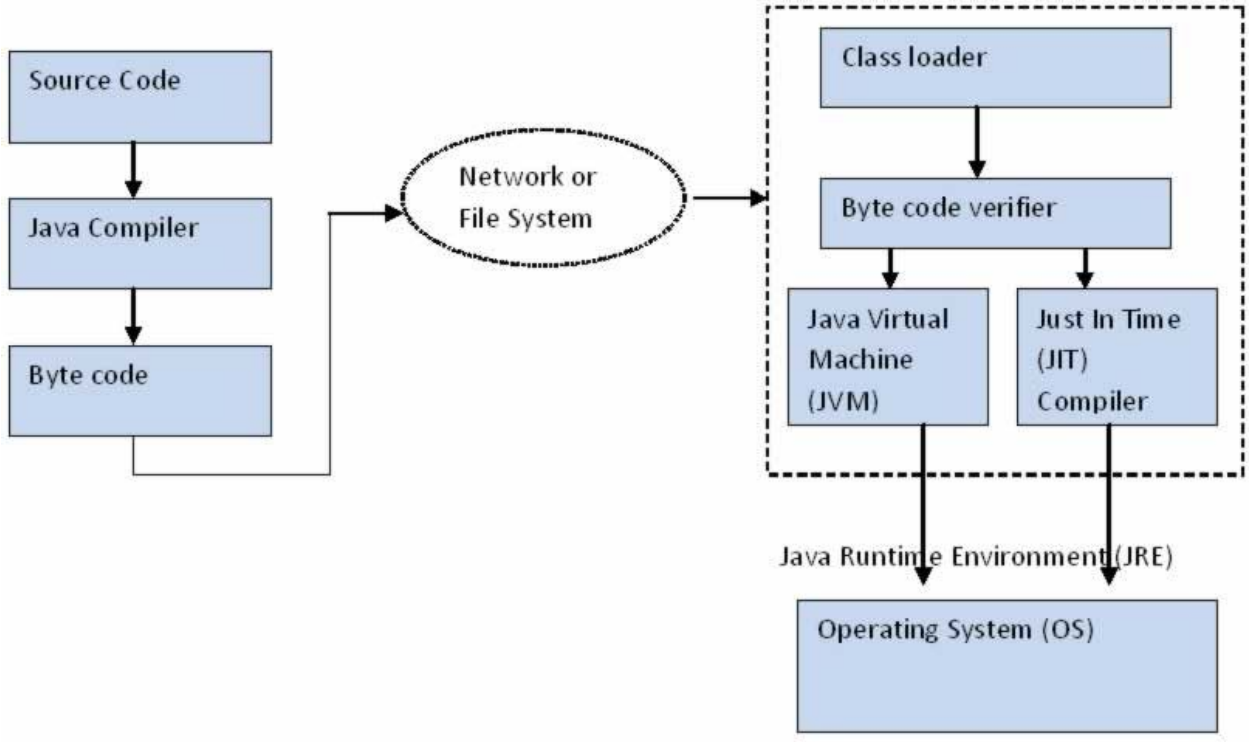


Figure 3: Java Architecture in Detail

4.1. Class loader

Class loader loads all the class files required to execute the program. Class loader makes the program secure by separating the namespace for the classes obtained through the network from the classes available locally. Once the bytecode is loaded successfully, then next step is bytecode verification by bytecode verifier.

4.2. Byte-code verifier

The bytecode verifier verifies the byte code to see if any security problems are there in the code. It checks the byte code and ensures the followings.

1. The code follows JVM specifications.
2. There is no unauthorized access to memory.
3. The code does not cause any stack overflows.
4. There are no illegal data conversions in the code such as float to object references.

Once this code is verified and proven that there is no security issues with the code, JVM will convert the byte-code into machine code which will be directly executed by the machine in which the Java program runs.

4.3. Just in Time Compiler

You might have noticed the component “Just in Time” (JIT) compiler in Figure 3. This is a component which helps the program execution to happen faster. How? Let’s see in detail.

As we discussed earlier when the Java program is executed, the byte code is interpreted by JVM. But this interpretation is a slower process. To overcome this difficulty, JRE include the component JIT compiler. JIT makes the execution faster.

If the JIT Compiler library exists, when a particular bytecode is executed first time, JIT compiler compiles it into native machine code which can be directly executed by the machine in which the Java program runs. Once the byte code is recompiled by JIT compiler, the execution time needed will be much lesser. This compilation happens when the byte code is about to be executed and hence the name “Just in Time”.

Once the bytecode is compiled into that particular machine code, it is cached by the JIT compiler and will be reused for the future needs. Hence the main performance improvement by using JIT compiler can be seen when the same code is executed again and again because JIT make use of the machine code which is cached and stored.

5. Why Java is Secure?

As you have noticed in the prior session “Java Runtime Environment (JRE) and Java Architecture in Detail”, the byte code is inspected carefully before execution by Java Runtime Environment (JRE). This is mainly done by the “Class loader” and “Byte code verifier”. Hence a high level of security is achieved.

Advantages of Java

- Java is easy to learn.
- Java was designed to be easy to use and is therefore easy to write, compile, debug, and learn than other programming languages.
- Java is object-oriented.
- This allows you to create modular programs and reusable code.
- Java is platform-independent.
- One of the most significant advantages of Java is its ability to move easily from one computer system to another. The ability to run the same program on many different systems is crucial to World Wide Web software, and Java succeeds at this by being platform-independent at both the source and binary levels.

- **Platform Independence** - Java compilers do not produce native object code for a particular platform but rather ‘byte code’ instructions for the Java Virtual Machine (JVM). Making Java code work on a particular platform is then simply a matter of writing a byte code interpreter to simulate a JVM. What this all means is that the same compiled byte code will run unmodified on any platform that supports Java.
- **Object Orientation** - Java is a pure object-oriented language. This means that everything in a Java program is an object and everything is descended from a root object class.

- **Rich Standard Library** - One of Java's most attractive features is its standard library. The Java environment includes hundreds of classes and methods in six major functional areas.
    - *Language Support* classes for advanced language features such as strings, arrays, threads, and exception handling.
    - *Utility* classes like a random number generator, date and time functions, and container classes.
    - *Input/output* classes to read and write data of many types to and from a variety of sources.
    - *Networking* classes to allow inter-computer communications over a local network or the Internet.
    - *Abstract Window Toolkit* for creating platform-independent GUI applications.
    - *Applet* is a class that lets you create Java programs that can be downloaded and run on a client browser.
- **Applet Interface** - In addition to being able to create stand-alone applications, Java developers can create programs that can downloaded from a web page and run on a client browser.
- **Familiar C++-like Syntax** - One of the factors enabling the rapid adoption of Java is the similarity of the Java syntax to that of the popular C++ programming language.
- **Garbage Collection** - Java does not require programmers to explicitly free dynamically allocated memory. This makes Java programs easier to write and less prone to memory errors.

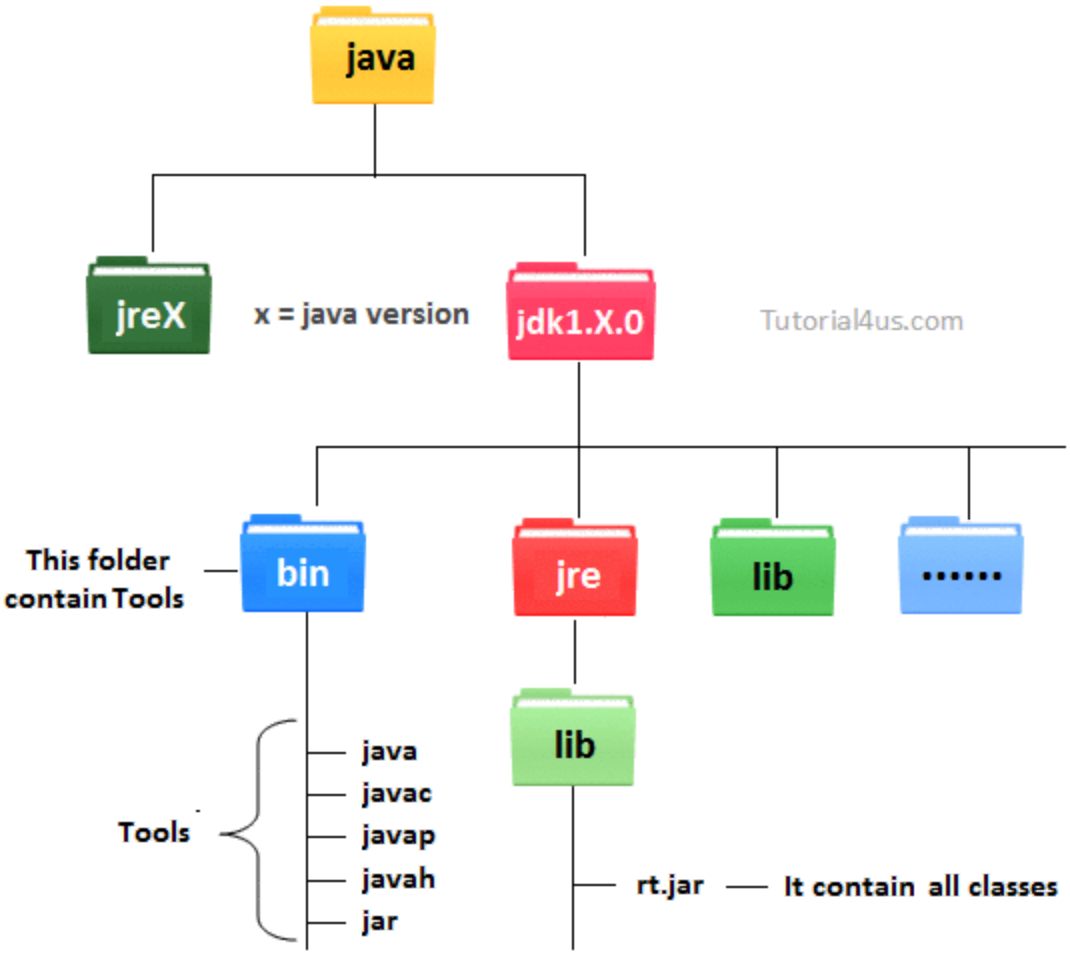
PATH and CLASSPATH variables

PATH

Path variable is set for providing path for all Java tools like java, javac, javap, javah, jar, appletviewer. In Java to run any program we use ***java*** tool and for compile Java code use ***javac*** tool. All these tools are available in bin folder so we set path upto bin folder.

CLASSPATH

classpath variable is set for providing path of all Java classes which is used in the applications. All classes are available in lib/rt.jar so we set classpath upto lib/rt.jar.



PATH	CLASSPATH
An environment variable which is used by the operating system to find the executables	An environment variable which is used by the Java compiler to find the path, of classes.ie in J2EE we give the path of jar files
PATH is nothing but setting up an environment for operating system. Operating System will look in this PATH for executables	Classpath is nothing but setting up the environment for Java. Java will use to find compiled classes
Refers to the system	Refers to the Developing Envornment

Compiling and Running Java Programs

Writing first Java class (program)

- Open a text editor and write the following code.

- Save file as ***Swastik.java***. The file name has to be same as its Class name, i.e. ***Swastik.java***.
  Note: File names are case sensitive so filename has to be exactly same as class name.

```
class Swastik{
   public static void main(String args[]){
    System.out.println("Hello Java Students !");
   }
}
```

Understanding first java program

Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().

- ***class*** keyword is used to declare a class in java.
- ***public*** keyword is an access modifier which represents visibility, it means it is visible to all.
- ***static*** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.
- ***void*** is the return type of the method, it means it doesn't return any value.
- ***main*** represents startup of the program.
- ***String[] args*** is used for command line argument. We will learn it later.
- ***System.out.println()*** is used print statement. We will learn about the internal working of System.out.println statement later.

Compiling

javac command is used to compile the java code. Go to command prompt in Windows or open terminal in Linux, then type the following command to compile:

$ javac Swastik.java

If we want to save the compiled class files in different path than the current, we can give the path to ***-d*** option of javac command like below:

$ javac Swastik.java -d classes

This will (create if not exists ***classes***) store the compiled class files in classes directory.

Running Java

java command is used to execute/run the Java class. After compiling the class the .java file, there should be one class file created in the same path or given path.

$ java Swastik

Do not need to provide the file extension .class, just the class name i.e. Swastik instead of Swastik.class but need to give the path to class file if ***-d*** was used while compiling.

$ java classes/Swastik

Class and Object

Creating Classes

Interfaces

Creating Objects

Access Modifiers

Arrays

Packages

Inheritance

Exception Handling and Threading

Try

Catch

Finally

Throws

Creating Multithreaded Programs

Thread Life Cycle

File IO

Byte Stream Classes (FileInputStream and FileOutputStream)

Character Stream Classes (FileReader and File Writer)

Random Access File Class

Unit 2: User Interface Components with Swing

10 Hrs.

Swing and MVC Design Patterns

Design Pattern

MVC Pattern

MVC Analysis of Swing Buttons

Layout Management:
Border Layout
Grid layout
Gridbag Layout
Group Layout
Using NO layout managers
Custom layout Managers
Text Input
Text Fields
Password Fields
Text Areas
Scroll Pane
Label and Labeling Components
Choice Components
Check Boxes
Radio Buttons
Borders
Combo boxes
Sliders
Menus
Menu Building
Icons in Menu Items
Check box and Radio Buttons in Menu Items
op-up Menus
Keyboard Mnemonics and Accelerators
Enabling and Disabling menu Items
Toolbars
Tooltips
Dialog Boxes
Option Dialogs
Creating Dialogs
Data Exchange
File Choosers
Color Choosers
Components Organizers
split Panes
Tabbed Panes
Desktop Panes and Internal Frames
Cascading and Tiling

Advance Swing Components
List
Trees
Tables
Progress Bars
Unit 3: Event Handling
4 Hrs.
Introduction:
Standard Event Handling
Using Delegated Class
Using Action Commands
Listener Interfaces
Adapter Classes
Handling Events
Action Events
Key Events
Focus Events
Window Event
Mouse Event
Item Events
Unit 4: Database Connectivity
4 Hrs.
Design of JDBC
Driver Types
Typical Uses of JDBC
JDBC Configuration
Database URLs
Driver JAR Files
Starting Database
Registering Driver Class
Connecting to the database
Executing SZL Statements
Managing Connections
Statements
Result Set
SQL Exceptions
Populating Database
Query Execution
Prepared Statements

Reading and Writing LOBs
SQL Escapes
Multiple Results
Scrollable Result Sets Updatable Result Sets
Row Sets and Cached Row Sets, Transactions

Unit 5: Network Programming

5 Hrs.

Networking Basics

Transmission control Protocol (TCP)
User Datagram Protocol (UDP)
Ports
IP Address Network Classes in JDK

Working with URLs

Connecting to URLs
Reading Directly from URLs
InetAddress Class

Sockets

TCP Sockets
UDP Sockets
Serving Multiple Clients
Half Close
Interruptible Sockets
Sending Email

Unit 6: Java Beans

3 Hrs.

Introduction

Creating, Updating and Reading From JAR Files
Java Beans
Advantages of Java Beans
Class Vs. Beans
BDK and Bean Box

Java Bean

Creating a Java Bean
Creating a Bean Mainfest File
Creating a Bean JAR File
Using a New Bean
Adding Controls to Beans
Giving a Bean Properties
Creating Bound Properties

Giving a Bean Methods
Giving a Bean an Icon

Unit 7: Servlets and Java Server pages

8Hrs.

Servlets

Introduction to Servlets
Life Cycle of Servlets
Java Servlets Development Kit
Creating
Compiling and running servlets
The servlets API (javax.servlet package)
Reading the servlet Parameters
Reading Initialization Parameter
The javax.servlet.http.Package
Handling HTTP Request and Response (GET/POST Request)
Using Cookies, Session Tracking

Java Server Pages:

Advantages of JSP technology (Comparison with ASP/Servlet)
JSP Architecture
JSP Access Model
JSP Implicit Object
Object Scope
Synchronization Issue
Exception Handling
Session management
Creating and Processing Forms

Unit 8: RMI and CORBA

3 Hrs.

Remote Method Invocation

Introduction of RMI
Architecture of RMI
Remote Objects
Creating and Executing RMI Applications

CORBA

Introduction to CORBA
Architecture of CORBA
Functioning of CORBA Applications
CORBA Services