Java Programming for Beginners

By Krishna Rungta

Copyright 2018 - All Rights Reserved – Krishna Rungta

ALL RIGHTS RESERVED. No part of this publication may be reproduced or transmitted in any form whatsoever, electronic, or mechanical, including photocopying, recording, or by any informational storage or retrieval system without express written, dated and signed permission from the author.

Table Of Content

Chapter 1: Introduction to Java Platform

What is Java?

Java is a programming language and a computing platform for application development. It was first released by Sun Microsystem in 1995 and later acquired by Oracle Corporation. It is one of the most used programming languages.

What is Java Platform?

Java platform is a collection of programs that help to develop and run programs written in the Java programming language. Java platform includes an execution engine, a compiler, and a set of libraries. JAVA is platform-independent language. It is not specific to any processor or operating system.

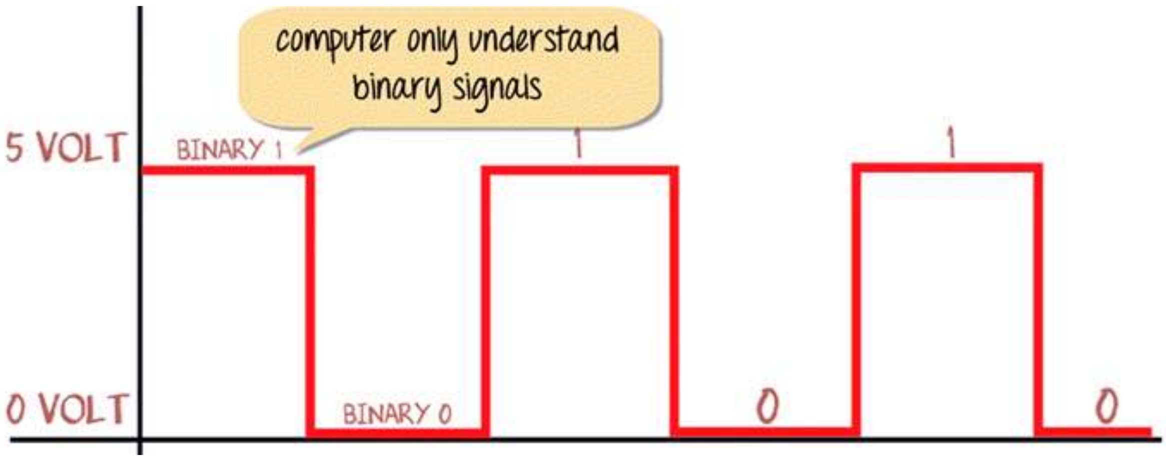
To understand JAVA programming language, we need to understand some basic concept of how a computer program can run a command and execute the action.

What is PC?

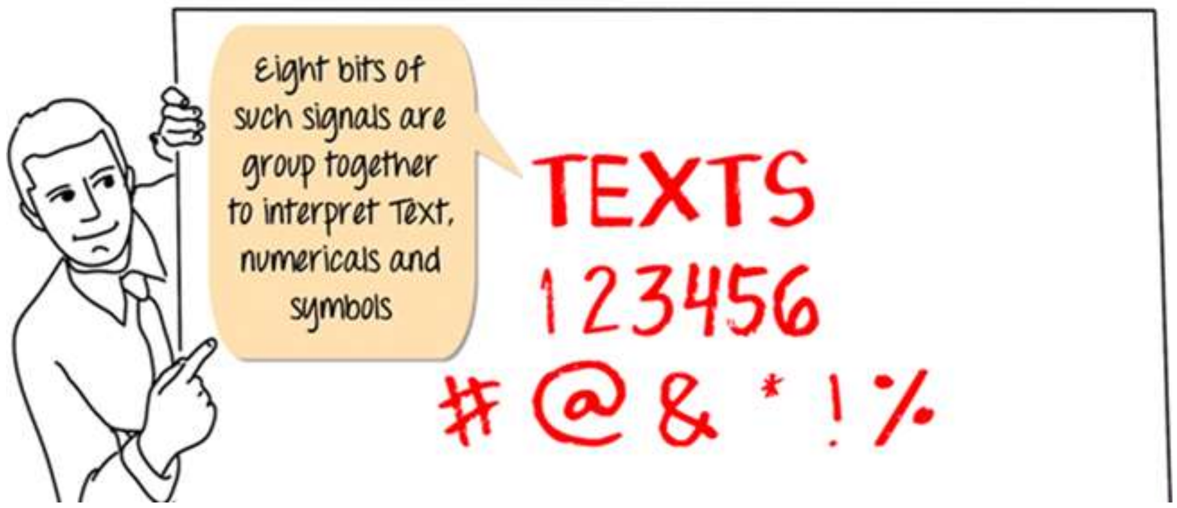
A computer is an electronic device capable of performing computations, and we all know that it is composed of a monitor, keyboard, mouse, and memory to store information. But the most important component of the computer is a PROCESSOR. Which does all thinking of computer, but the question is how the computer does this thinking? How does it understand text, images, videos, etc.?

What is Assembly Language?

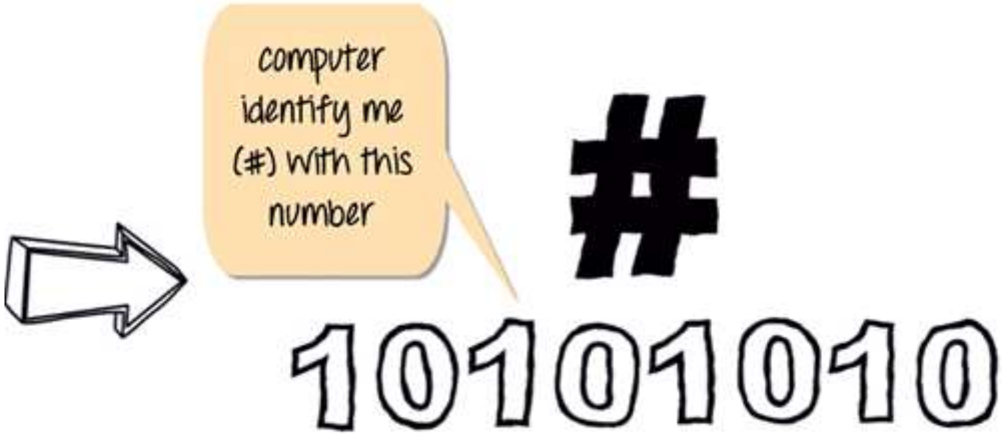
The computer is an electronic device, and it can only understand electronic signals or binary signals. For example, the 5-volt electronic signal may represent binary number 1 while 0 volts may represent binary number 0. So your PC is continuously bombarded with these signals.



Eight bits of such signals are group together to interpret Text, numerical and symbols.


Eight bits of such signals are group together to interpret Text, numericals and symbols
TEXTS
123456
# @ & * ! %

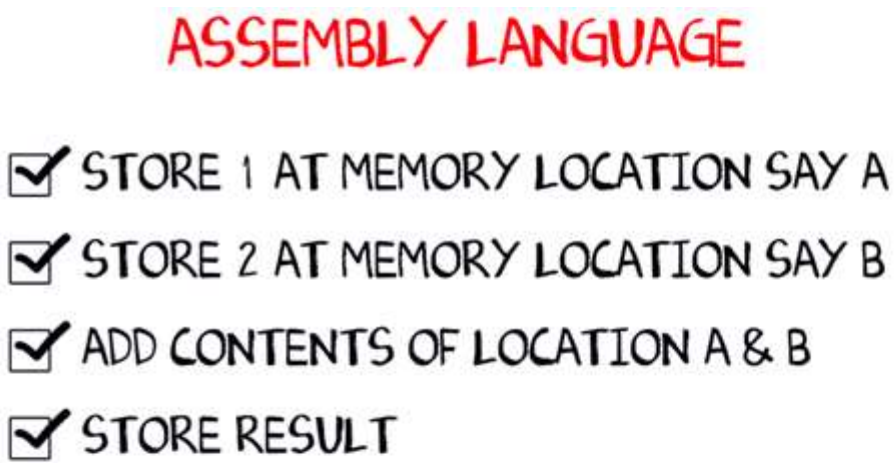
For example, the # symbol is identified by computer as 10101010. Similarly, the pattern for adding a function is represented by 10000011.



This is known as 8-bit computing. Current day processor is capable of decoding 64-bit time. But what is the relation of this concept with the programming language JAVA? Let understand these as an example.

Suppose if you want to tell the computer to add two number (1+2) which is represented by some binary numbers (10000011), how are you going to tell the computer? Yes, we going to use assembly language to get our code executed.

We are going to give the command to a computer in this format as shown below. Your code to add two numbers in this language would be in this order.

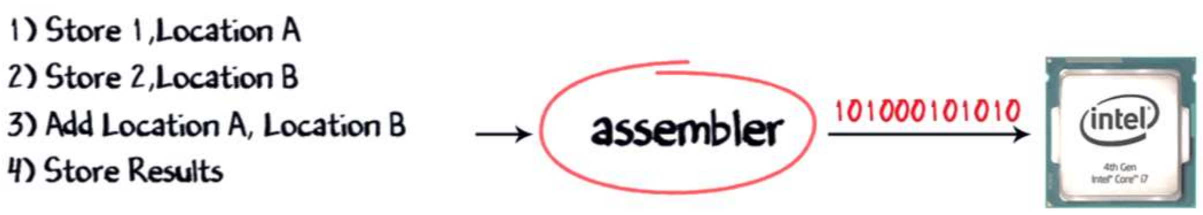

ASSEMBLY LANGUAGE
STORE 1 AT MEMORY LOCATION SAY A
STORE 2 AT MEMORY LOCATION SAY B
ADD CONTENTS OF LOCATION A & B
STORE RESULT

- Store number 1 at memory location say A
- Store number 2 at memory location say B
- Add contents of Location A & B
- Store results

But how are we going to do this? Back in 1950's when computers were huge and consumed a great deal of power, you would convert your assembly code into corresponding machine code to 1 and 0's using mapping sheets. Later this code will be punched into the machine cards and feed to the computer. The computer will read these code and execute the program. These would be a long process then until ASSEMBLER came to help.

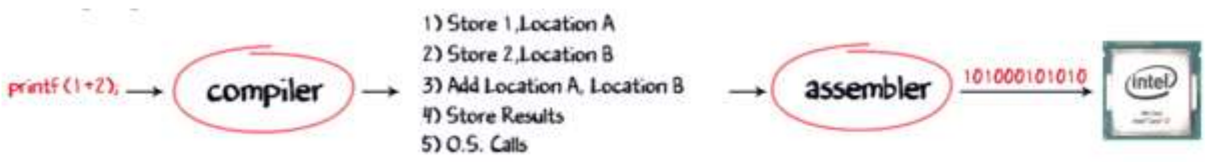
What are Assembler and Compiler?

With the advancement in technology i/o devices were invented, you could directly type your program into the PC using a program called ASSEMBLER. It converts it into corresponding machine code (110001..) and feeds to your processor. So coming back to our example addition of (1+2), the assembler will convert this code into machine code and give the output.



That apart, you will also have to make calls to create Operating System provided functions to display the output of the code.

But alone assembler is not involved in this whole process; it also requires the compiler to compile the long code into a small chunk of codes. With advancement in software development languages, this entire assembly code could shrink into just one line **print f 1+2 A** with the help of software called COMPILER. It is used to convert your c language code into assembly code, and the assembler converts it into corresponding machine code, and this machine code will be transmitted to the processor. The most common processor used in PC or Computers are Intel processor.



Though present-day compilers come bundled with assembler can directly convert your higher language code into machine code.

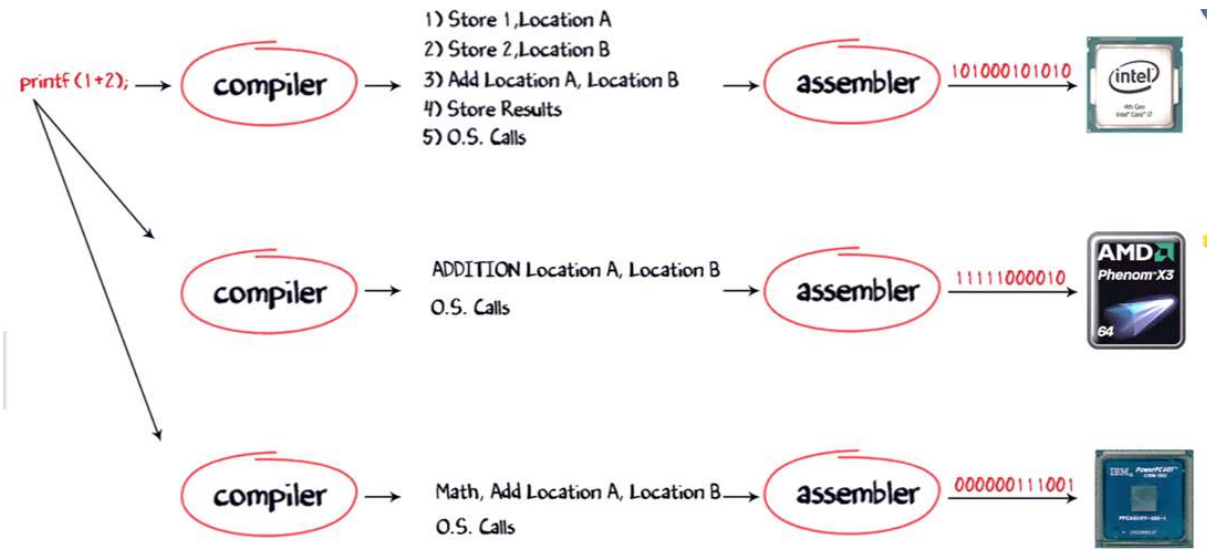
Now, suppose Windows operating system is running on this Intel processor, a combination of Operating System plus the processor is called the PLATFORM. The most common platform in the world is the Windows, and Intel called the Wintel Platform. The other popular platforms are AMD and Linux, Power PC, and Mac OS X.

Now, with a change in processor, the assembly instructions will also change. For example the

- Add instruction in Intel may be called ADDITION for AMD
- OR Math ADD for Power PC

And obviously, with a change in Operating System, the level and nature of O.S level calls will also change.

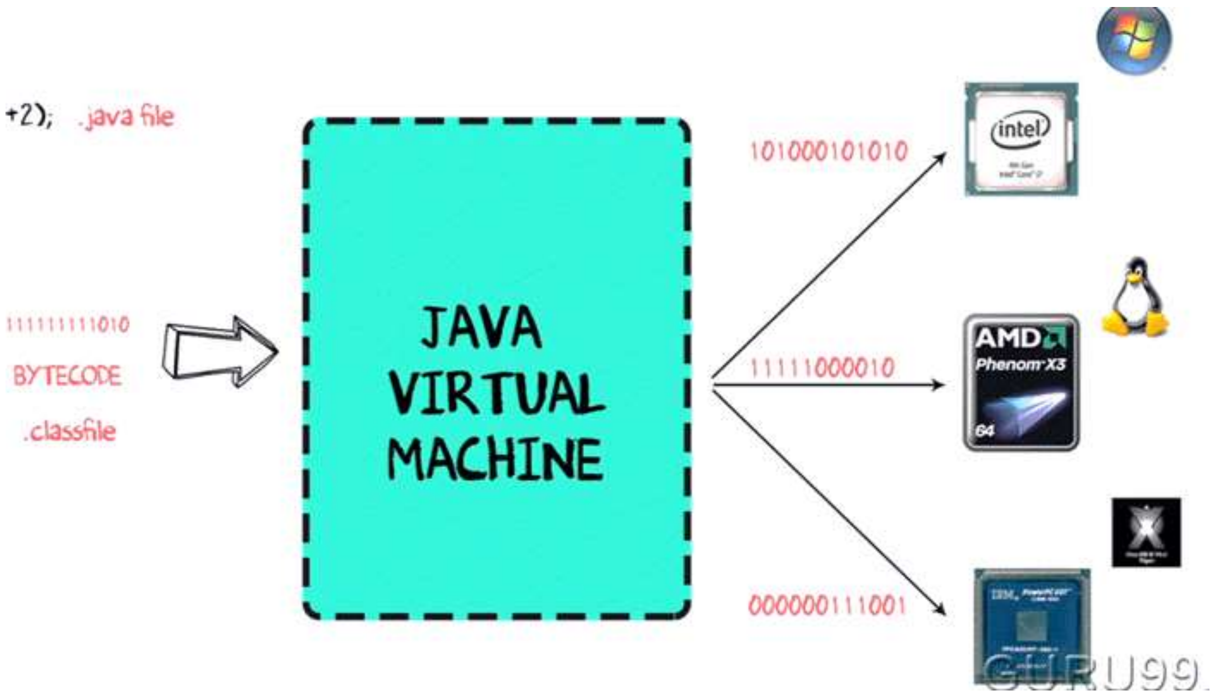
As a developer, I want my software program to work on all platforms available, to maximize my revenues. So I would have to buy separate compilers which convert my print f command into the native machine code.



But compilers come expensive, and there is a chance of compatibility issues. So buying and installing a separate compiler for different O.S and processor is not feasible. So, what can be an alternative solution? Enter Java language.

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.

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.

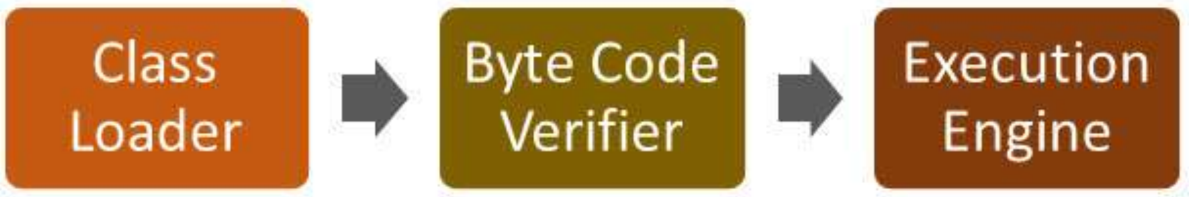
Chapter 2: Introduction to Java Virtual Machine (JVM)

What is JVM?

JVM is a engine that provides runtime environment to drive the Java Code or applications. It converts Java bytecode into machines language. JVM is a part of JRE(Java Run Environment). It stands for Java Virtual Machine

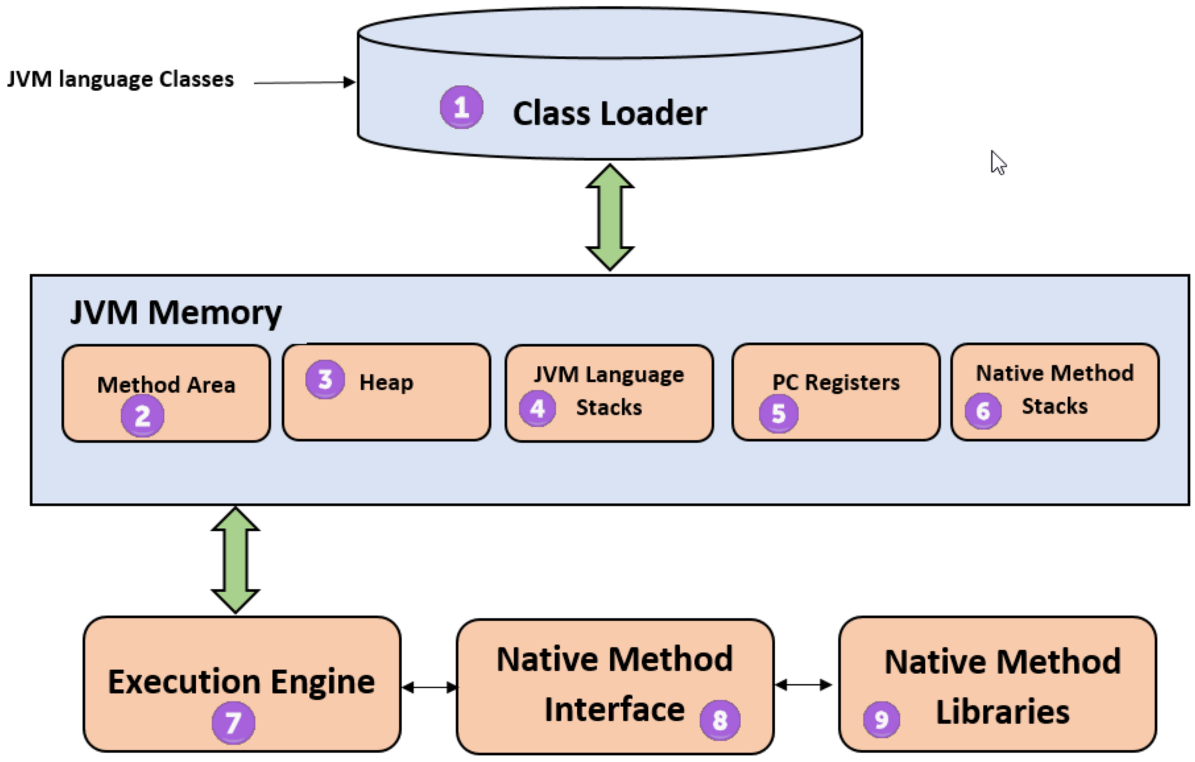
- In other programming languages, the compiler produces machine code for a particular system. However, Java compiler produces code for a Virtual Machine known as Java Virtual Machine.
- First, Java code is complied into bytecode. This bytecode gets interpreted on different machines
- Between host system and Java source, Bytecode is an intermediary language.

- JVM is responsible for allocating memory space.



JVM Architecture

Let's understand the Architecture of JVM. It contains classloader, memory area, execution engine etc.



1) ClassLoader

The class loader is a subsystem used for loading class files. It performs three major functions viz. Loading, Linking, and Initialization.

2) Method Area

JVM Method Area stores class structures like metadata, the constant runtime pool, and the code for methods.

3) Heap

All the Objects, their related instance variables, and arrays are stored in the heap. This memory is common and shared across multiple threads.

4) JVM language Stacks

Java language Stacks store local variables, and it's partial results. Each thread has its own JVM stack, created simultaneously as the thread is created. A new frame is created whenever a method is invoked, and it is deleted when method invocation process is complete.

5) PC Registers

PC register store the address of the Java virtual machine instruction which is currently executing. In Java, each thread has its separate PC register.

6) Native Method Stacks

Native method stacks hold the instruction of native code depends on the native library. It is written in another language instead of Java.

7) Execution Engine

It is a type of software used to test hardware, software, or complete systems. The test execution engine never carries any information about the tested product.

8) Native Method interface

The Native Method Interface is a programming framework. It allows Java code which is running in a JVM to call by libraries and native applications.

9) Native Method Libraries

Native Libraries is a collection of the Native Libraries(C, C++) which are needed by the Execution Engine.

Software Code Compilation & Execution process

In order to write and execute a software program, you need the following

1) Editor – To type your program into, a notepad could be used for this

2) Compiler – To convert your high language program into native machine code

3) Linker – To combine different program files reference in your main program together.

4) Loader – To load the files from your secondary storage device like Hard Disk, Flash Drive, CD into RAM for execution. The loading is automatically done when you execute your code.

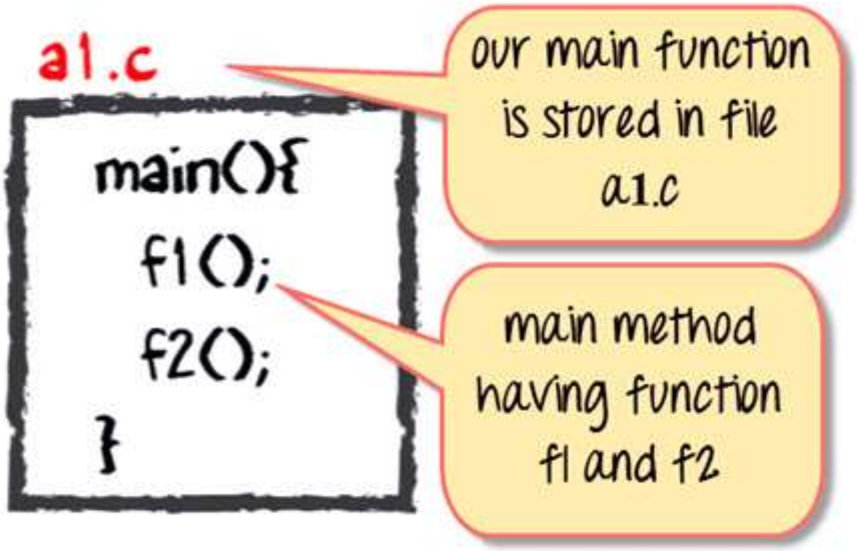
5) Execution – Actual execution of the code which is handled by your OS & processor.

With this background, refer the following video & learn the working and architecture of the Java Virtual Machine.

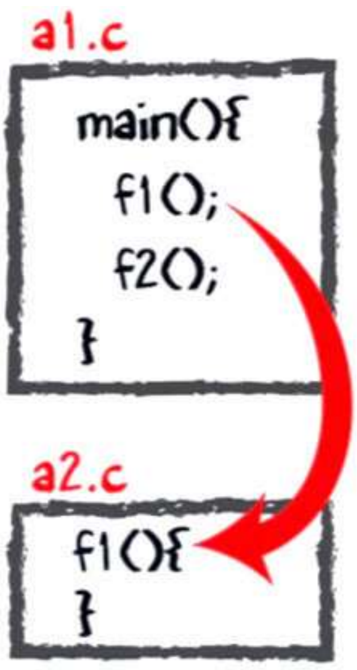
C code Compilation and Execution process

To understand the Java compiling process in Java. Let's first take a quick look to compiling and linking process in C.

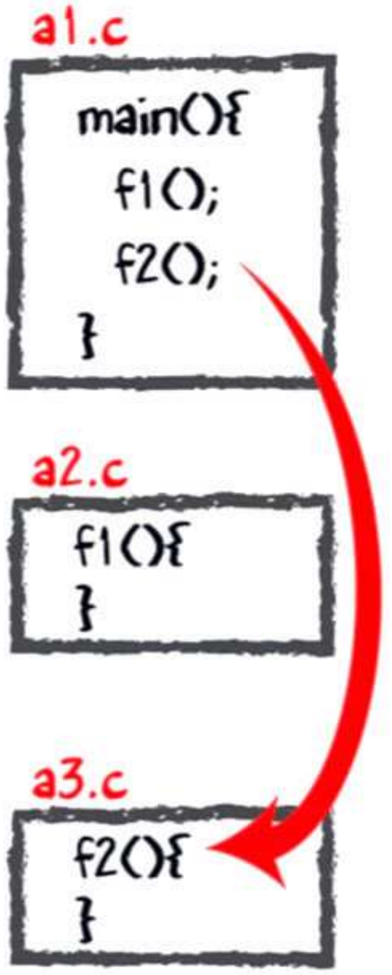
Suppose in the main, you have called two function f1 and f2. The main function is stored in file a1.c.


a1.c
main(){
f1();
f2();
}
our main function is stored in file a1.c
main method having function f1 and f2

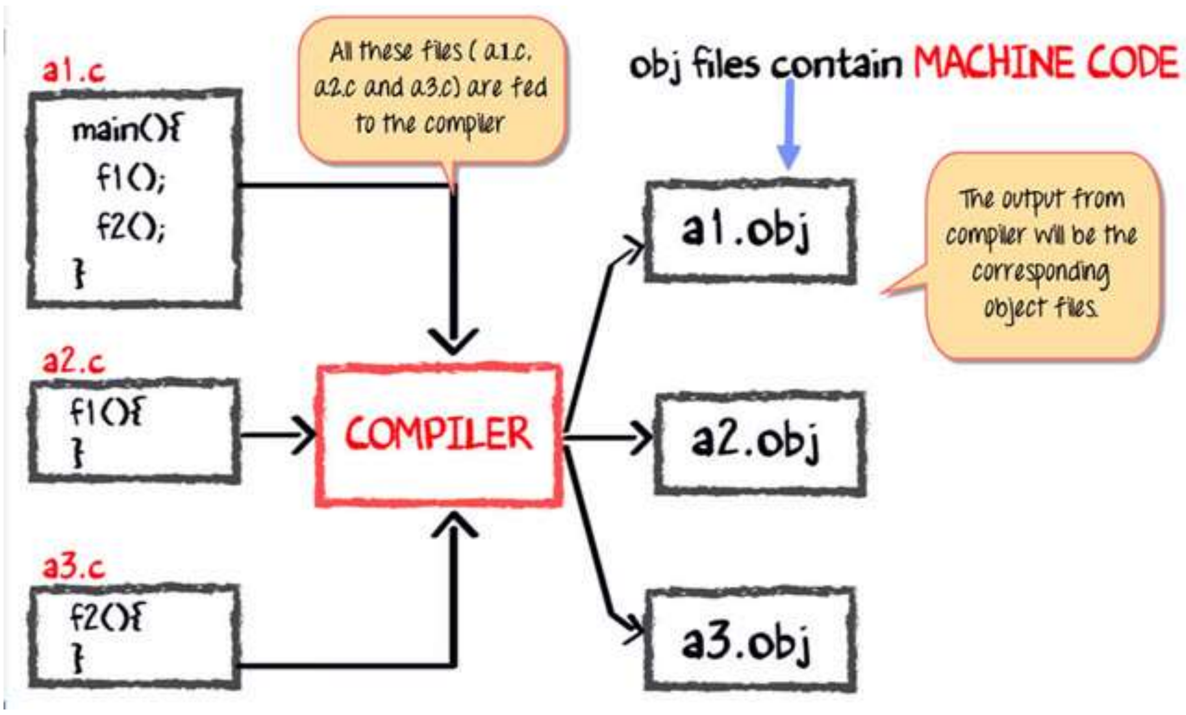
Function f1 is stored in a file a2.c



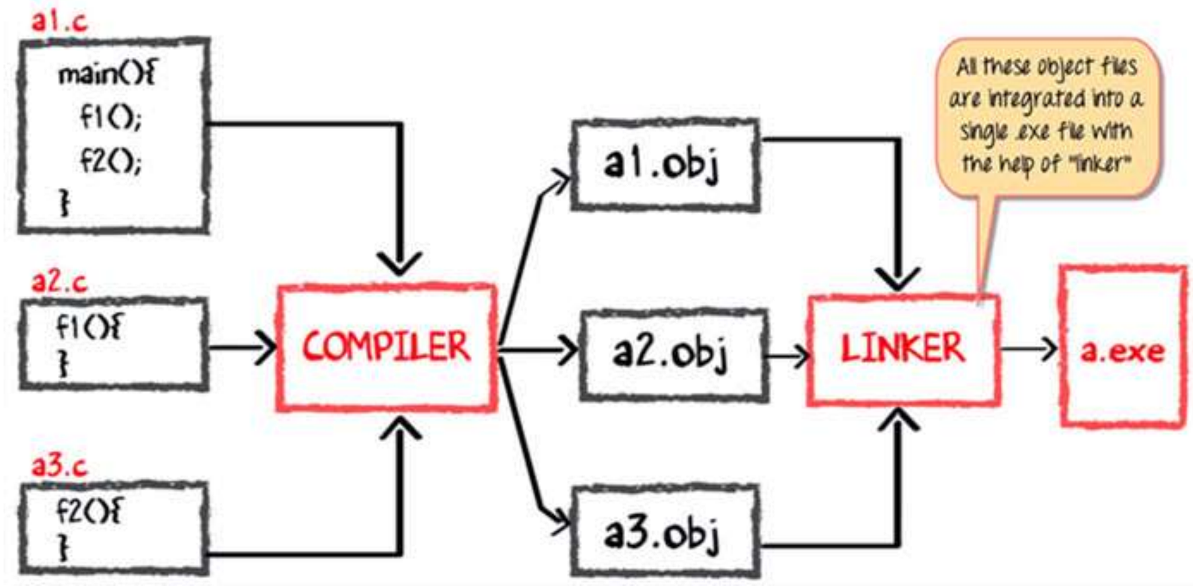
Function f2 is stored in a file a3.c



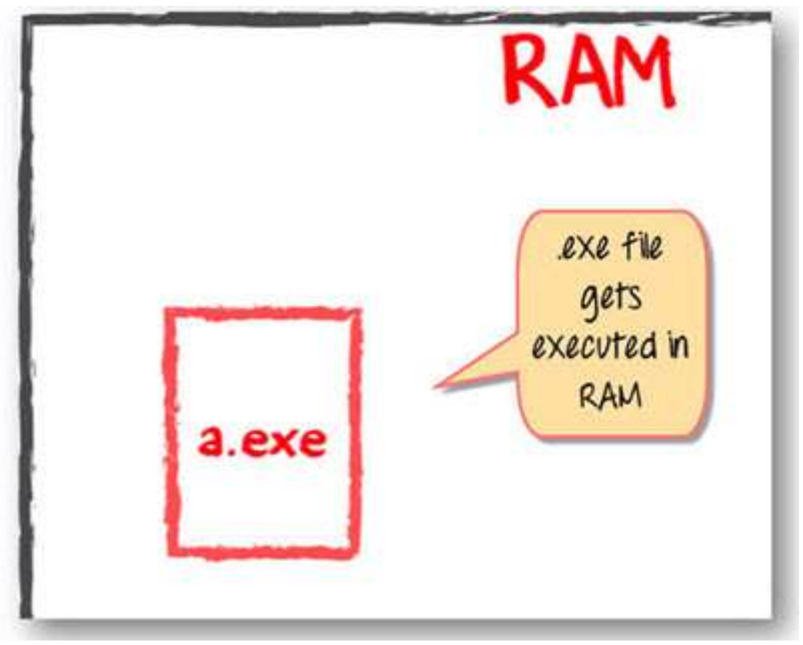
All these files, i.e., a1.c, a2.c, and a3.c, is fed to the compiler. Whose output is the corresponding object files which are the machine code.


a1.c
main(){
f1();
f2();
}
a2.c
f1(){
}
a3.c
f2(){
}
All these files (a1.c, a2.c and a3.c) are fed to the compiler
COMPILER
obj files contain MACHINE CODE
a1.obj
a2.obj
a3.obj
The output from compiler will be the corresponding object files.

The next step is integrating all these object files into a single .exe file with the help of linker. The linker will club all these files together and produces the .exe file.


a1.c
main(){
f1();
f2();
}
a2.c
f1(){
}
a3.c
f2(){
}
COMPILER
a1.obj
a2.obj
a3.obj
LINKER
a.exe
All these object files are integrated into a single exe file with the help of "linker"

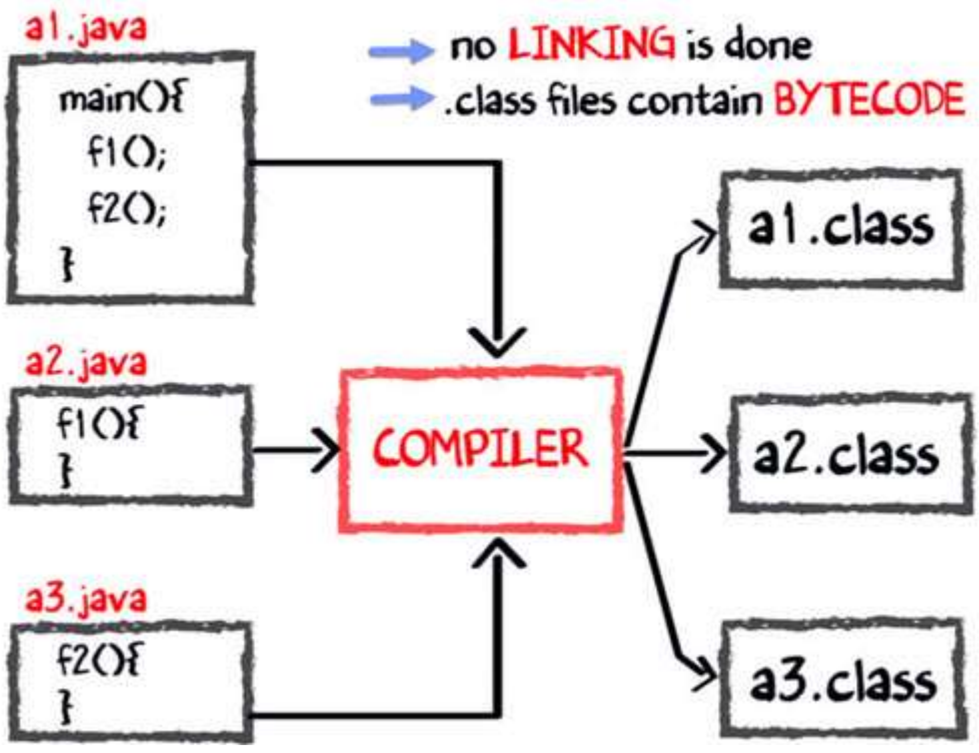
During program run, a loader program will load a.exe into the RAM for the execution.



Java code Compilation and Execution in Java VM

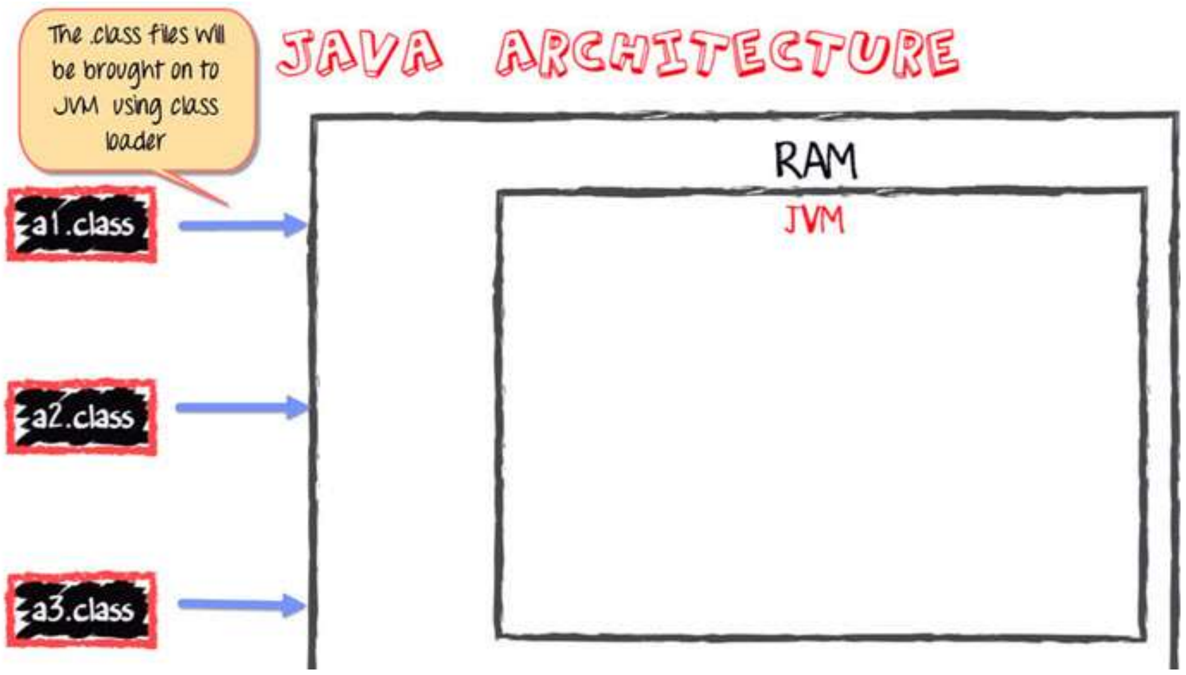
Let's look at the process for JAVA. In your main, you have two methods f1 and f2.

- The main method is stored in file a1.java
- f1 is stored in a file as a2.java
- f2 is stored in a file as a3.java

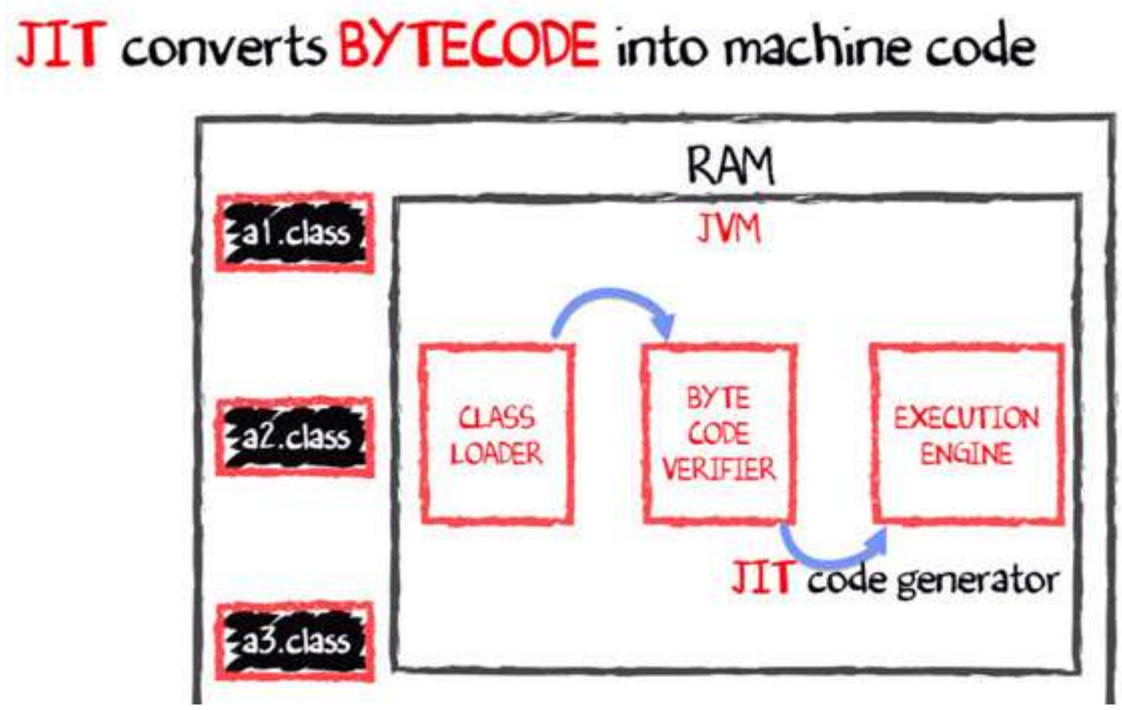


The compiler will compile the three files and produces 3 corresponding .class file which consists of BYTE code. **Unlike C, no linking is done**.

The Java VM or Java Virtual Machine resides on the RAM. During execution, using the class loader the class files are brought on the RAM. The BYTE code is verified for any security breaches.


The .class files will be brought on to JVM using class loader
JAVA ARCHITECTURE
RAM
JVM
a1.class
a2.class
a3.class

Next, the execution engine will convert the Bytecode into Native machine code. This is just in time compiling. It is one of the main reason why Java is comparatively slow.



NOTE: JIT or Just-in-time compiler is the part of the Java Virtual Machine (JVM). It interprets part of the Byte Code that has similar functionality at the same time.

Why is Java both Interpreted and Compiled Language?

Programming languages are classified as

- Higher Level Language Ex. C++, Java
- Middle-Level Languages Ex. C
- Low-Level Language Ex Assembly
- finally the lowest level as the Machine Language.

A **compiler** is a program which converts a program from one level of language to another. Example conversion of C++ program into machine code.

The java compiler converts high-level java code into bytecode (which is also a type of machine code).

An **interpreter** is a program which converts a program at one level to another programming language at the **same level.** Example conversion of Java program into C++

In Java, the Just In Time Code generator converts the bytecode into the native machine code which are at the same programming levels.

Hence, Java is both compiled as well as interpreted language.

Why is Java slow?

The two main reasons behind the slowness of Java are

1. **Dynamic Linking:** Unlike C, linking is done at run-time, every time the program is run in Java.
2. **Run-time Interpreter:** The conversion of byte code into native machine code is done at run-time in Java which furthers slows down the speed

However, the latest version of Java has addressed the performance bottlenecks to a great extent.

Summary:

- JVM or Java Virtual Machine is the engine that drives the Java Code. It converts Java bytecode into machines language.
- In JVM, Java code is compiled to bytecode. This bytecode gets interpreted on different machines
- JIT or Just-in-time compiler is the part of the Java Virtual Machine (JVM). It is used to speed up the execution time
- In comparison to other compiler machines, Java may be slow in execution.

Buy Now $9.99