**Working of Java Virtual Machine(JVM) & its Architecture**

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.

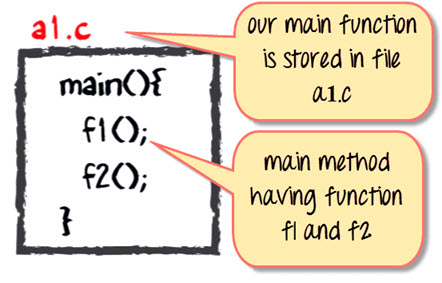
**4) Loader**– To load the files from your secondary storage device like Hard Disk, Flash Drive , CD into RAM for execution. The loading is automatically done when your execute your code.

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

## C code Compilation and Execution process

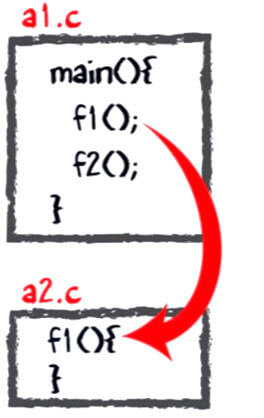
To understand the Java compiling process in Java. Let's first take a quick look to compiling and linking process in C.

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

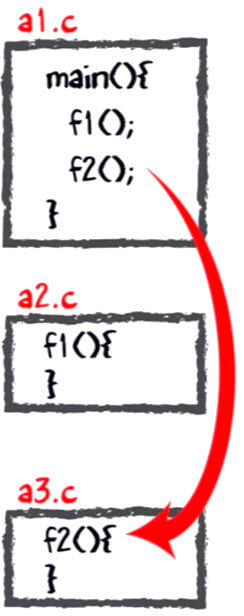


Function f1 is stored in a file a1.c

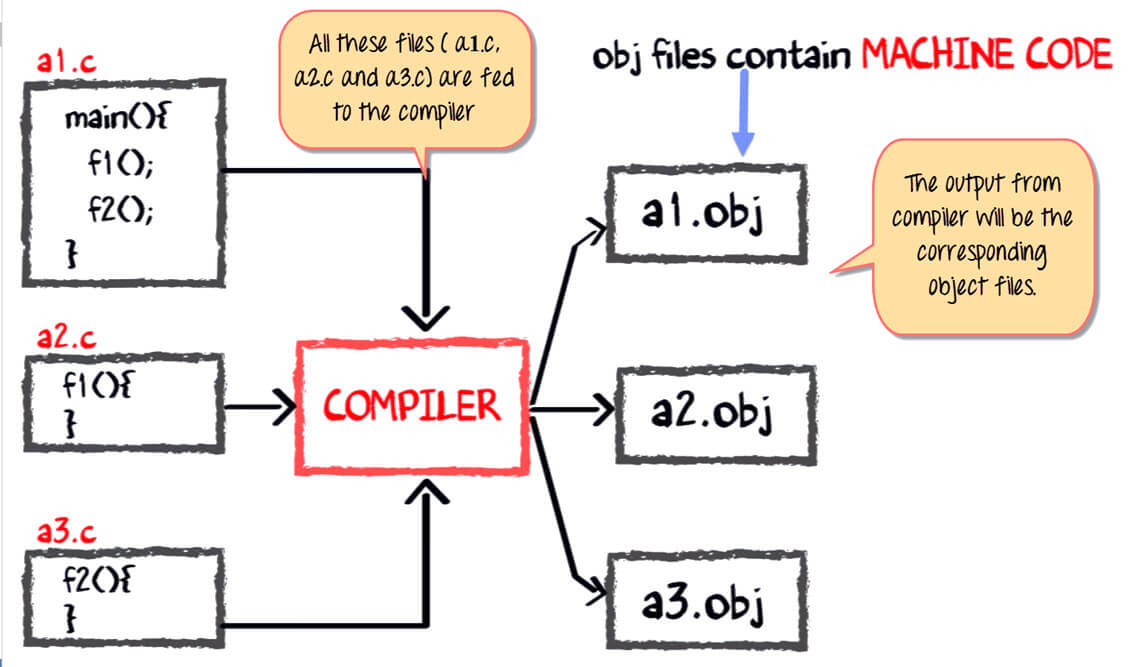
Function f1 is stored in a file a2.c



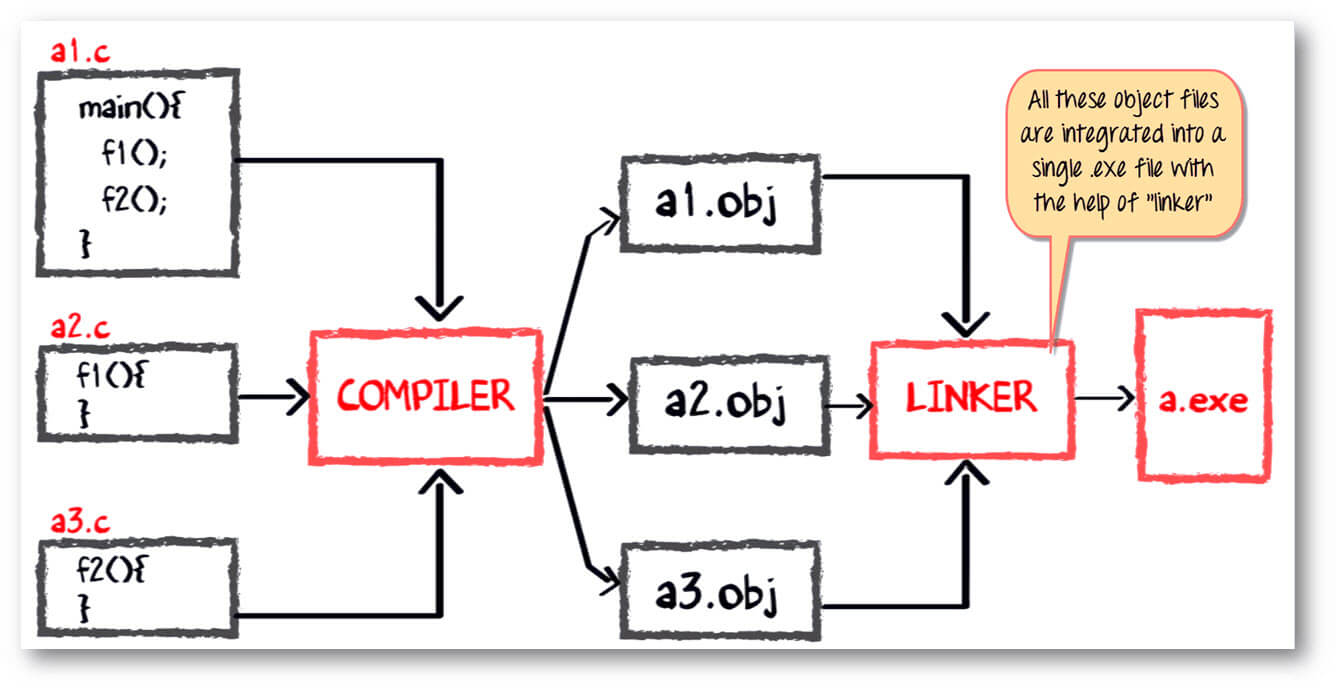
Function f2 is stored in a file a3.c



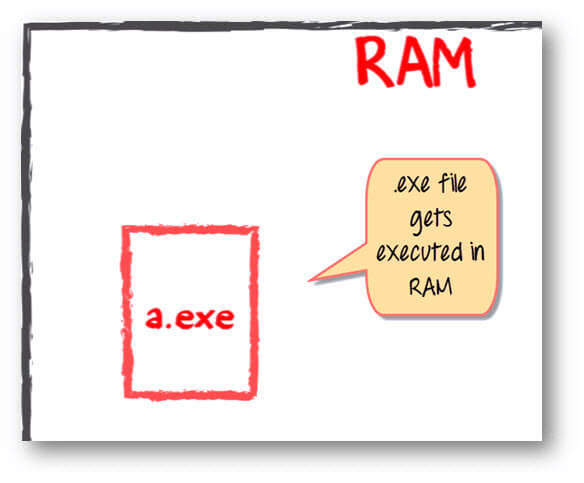
All these files, i.e., a1.c, a2.c, and a3.c, is fed to the compiler. Whose output is the corresponding object files which is the machine code.



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.



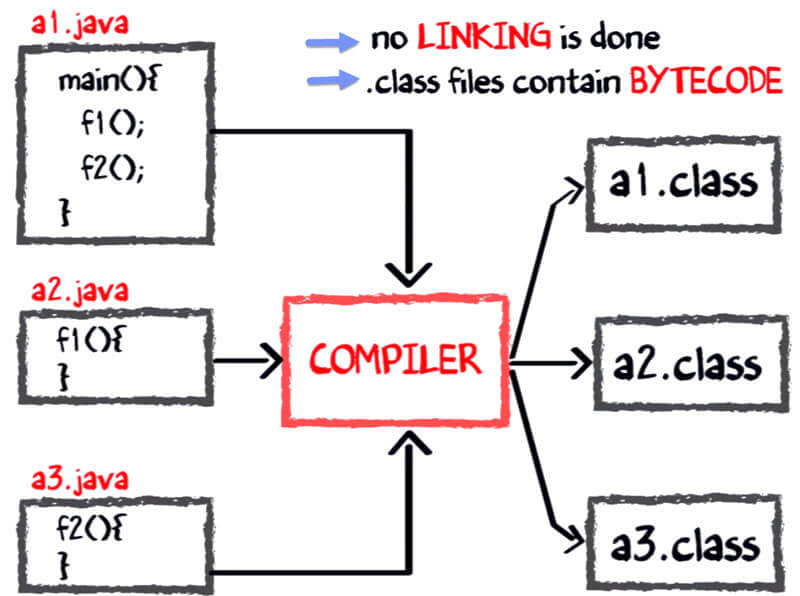
During program run a loader program will load a.exe into the RAM for the execution.



## Java code compilation and execution in Java VM

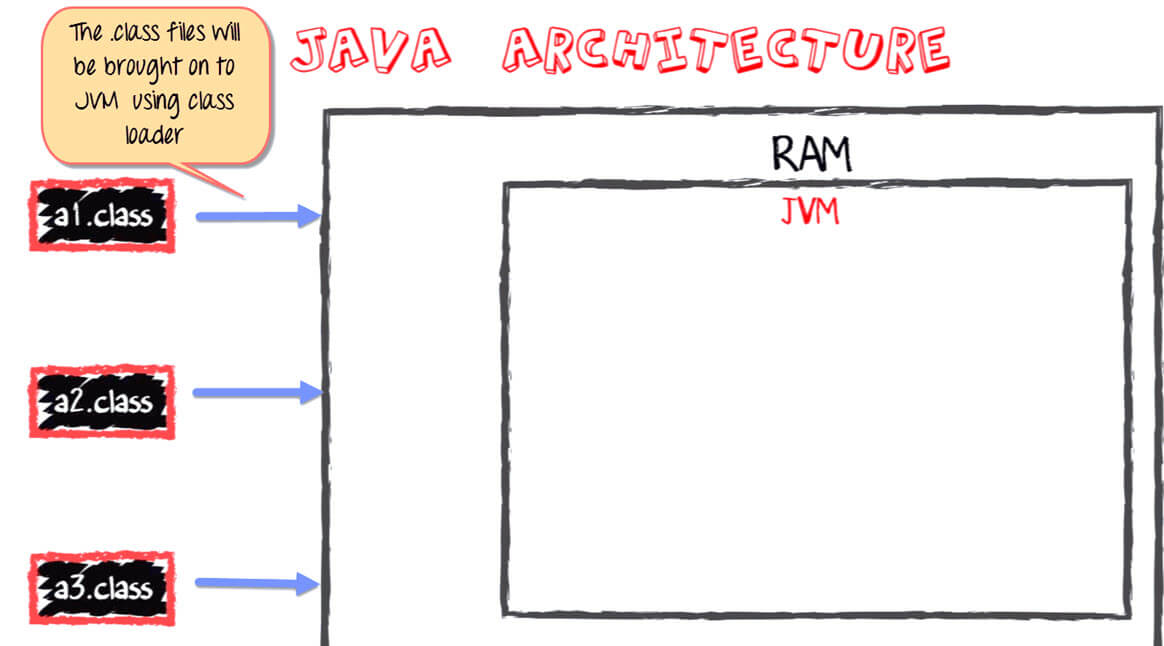
Let's look at the process for JAVA. In your main you have two methods f1 and f2.

* main method is stored in file a1.java
* f1 is stored in file as a2.java
* f2 is stored in file as a3.java

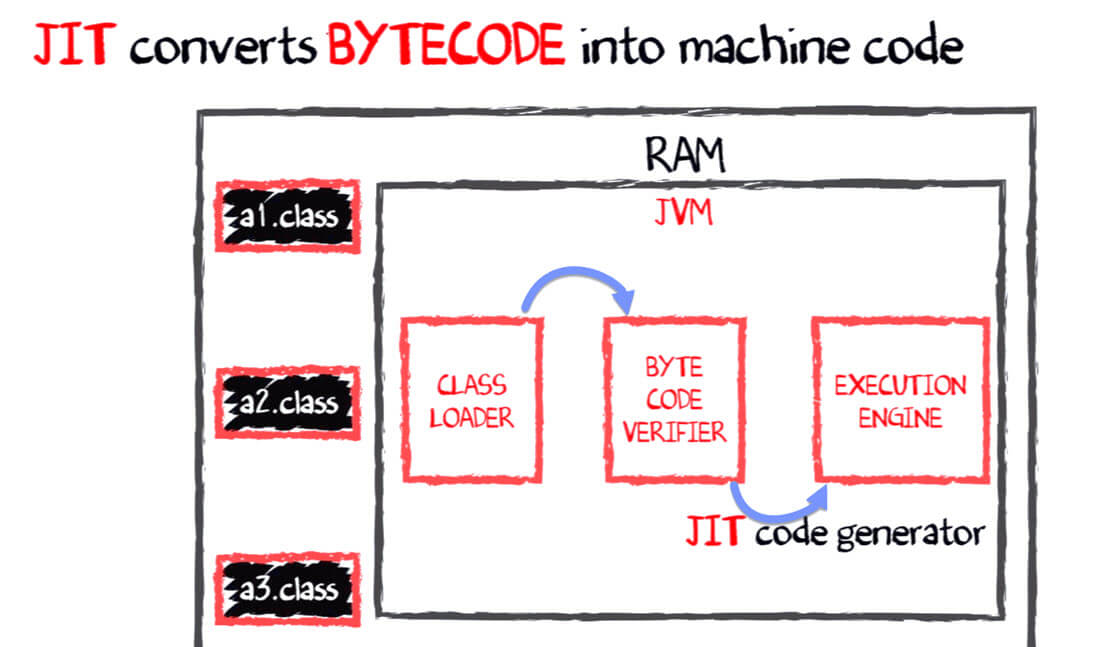


The compiler will compile the three files and produces a corresponding .class file which consists of BYTE code. **Unlike C, no linking is done**.

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.



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.

## What is JVM?

JVM stands for Java Virtual Machine. It is the engine that drives the Java Code. It converts Java bytecode into machines language.

* In other programming language, the compiler produces code for a particular system. But Java compiler produces code for a Virtual Machine.
* In JVM, Java code is compiled 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 a memory space.



## Why is Java both interpreted and compiled language?

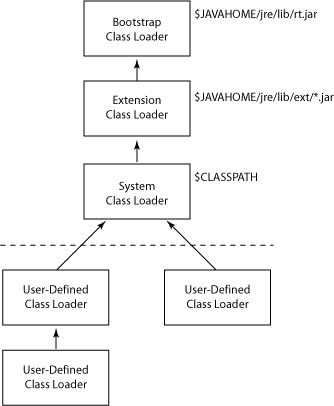
* Programming languages are classifies as
* Higher Level Language Ex. C++ , Java
* Middle Level Languages Ex. C
* Low Level Language Ex Assembly
* finally the lowest level as the Machine Language.

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 is a convert's high level java code into bytecode (which is also a type of machine code).

A **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.



|  |  |
| --- | --- |
| **Bootstrap** | The Bootstrap Classloader looks for runtime classes in rt.jar and internationalization classes in i18n.jar. |
| **Extension** | The Installed Extensions Classloader looks for classes in JAR files in the lib/ext directory of the JRE. |
| **System** | The System Classpath Classloader looks for classes in JAR files on paths specified by the system property java.class.path. To have a class loaded by the System Classloader, you must include the relevant directory in the class path. This means either in KJS (on Unix), the environment (on Unix or NT), or in the SoftwareiPlanetApplication Server6.0JavaClasspath entry (on NT.) |
| **Module** | The iPlanet Application Server Module Classloader looks for classes in all directories under <iPlanet Application Server install>/ias/APPS/modules/\*. |
| **Application** | Each registered J2EE application is loaded by its own class loader, which looks for classes under <iPlanet Application Server install>/ias/APPS/<app\_name> including all subdirectories. |

**The Java Platform architecture**

