JVM, JRE and JDK

Before jumping into the internals of Java, let's understand how a Java source file is executed.

1. We write the Java source code in Simple.Java file using an editor or IDE (**integrated development environment**) e.g. Sublime, *Eclipse* or *IntelliJ Idea* etc.
2. Program has to be compiled into bytecode. Java compiler (javac) compiles the sourcecode to Simple.class file.
3. This class file can be executed in any platform/OS by JVM (**Java virtual machine**).
4. JVM translates bytecode into native machine code which machines can execute.

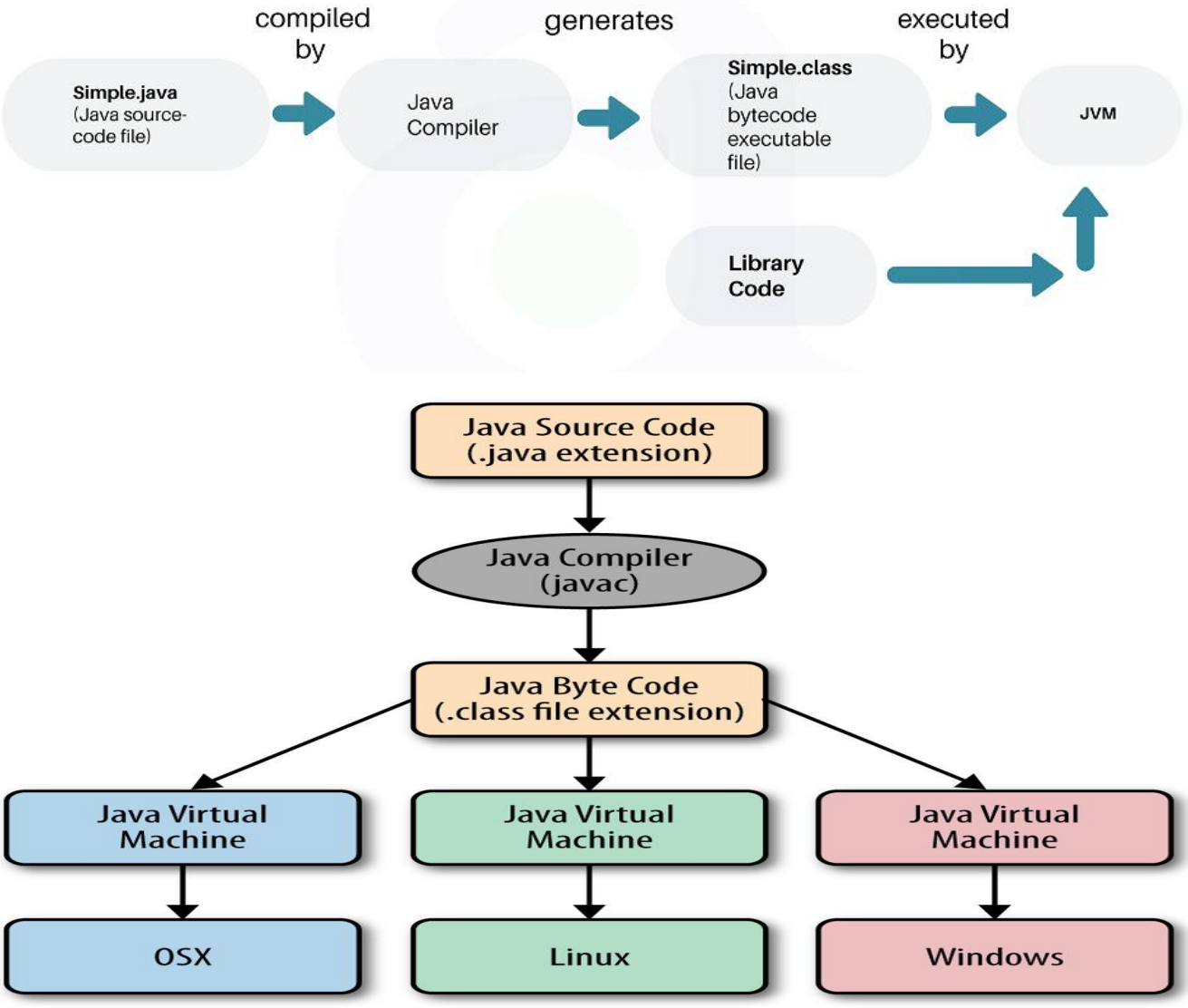


Fig: JAVA Program Execution Process

2. What is JVM?

Java Virtual machine (JVM) is the virtual machine that runs the Java bytecodes. You get this bytecode by compiling the .java files into .class files. .class files contain the bytecodes understood by the JVM.

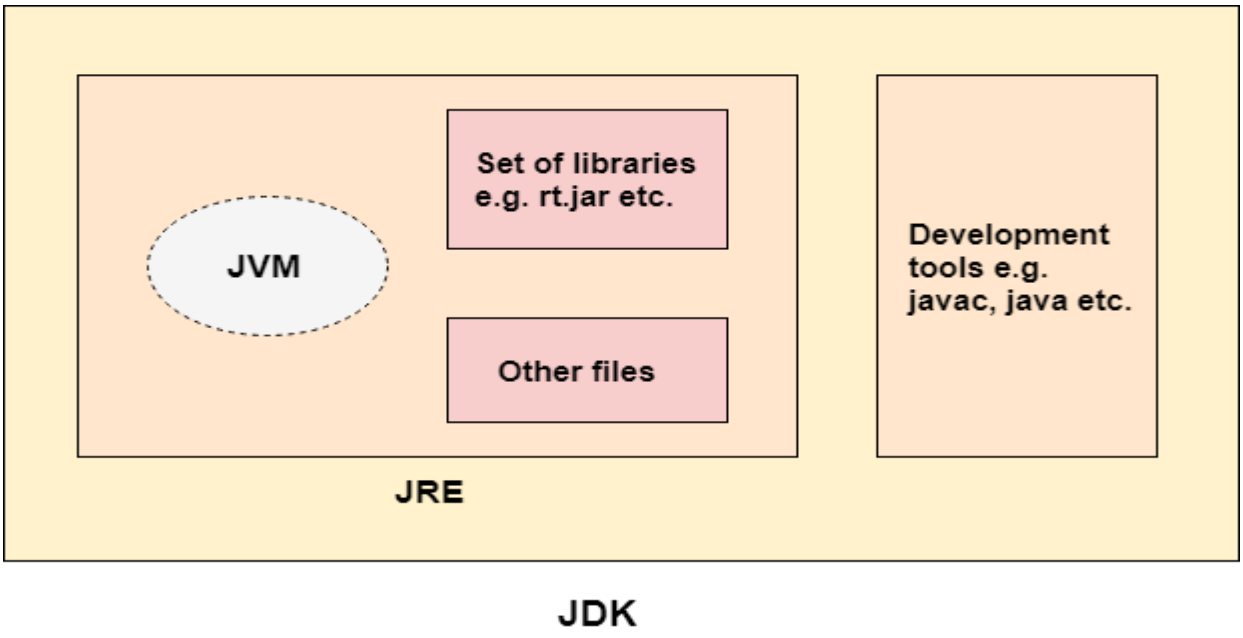
In the real world, JVM is a specification that provides a runtime environment in which Java bytecode can be executed. Different vendors provide different implementations of this specification. For example, this wiki page lists down different JVM implementations.

JVM delivers the optimal performance for Java applications using many advanced techniques, incorporating a state-of-the-art memory model, **garbage collector**, and **adaptive optimizer**.

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.

The JVM performs the following tasks:

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



JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

The following actions occur at runtime.

- **Class Loader**

The Class Loader loads all necessary classes needed for the execution of a program. It provides security by separating the namespaces of the local file system from that imported through the network. These files are loaded either from a hard disk, a network or from other sources.

- **Byte Code Verifier**

The JVM puts the code through the Byte Code Verifier that checks the format and checks for an illegal code. Illegal code, for example, is code that violates access rights on objects or violates the implementation of pointers.

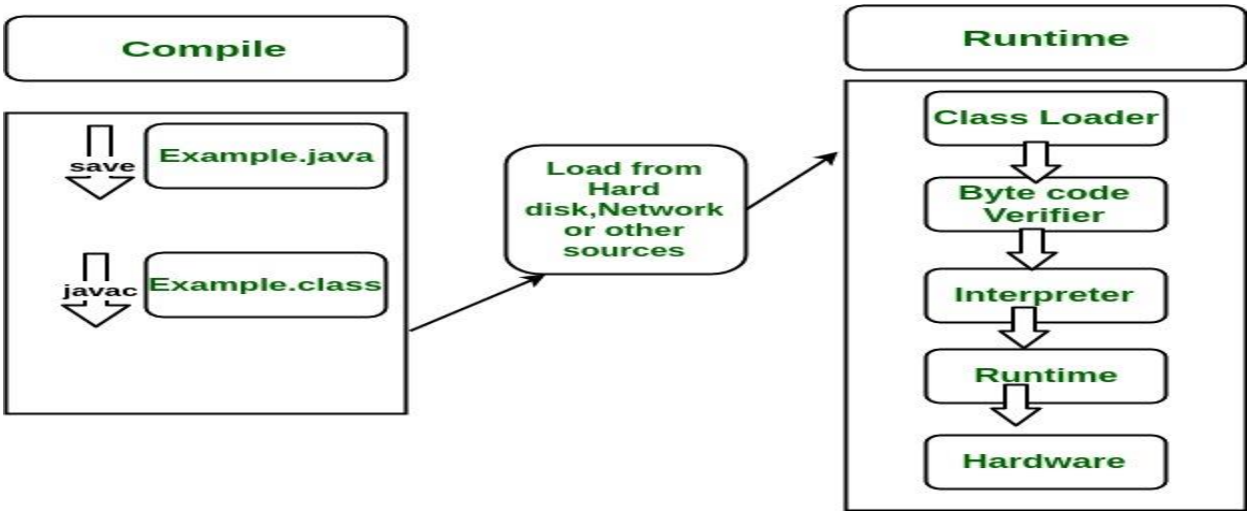
The Byte Code verifier ensures that the code adheres to the JVM specification and does not violate system integrity.

- **Interpreter**

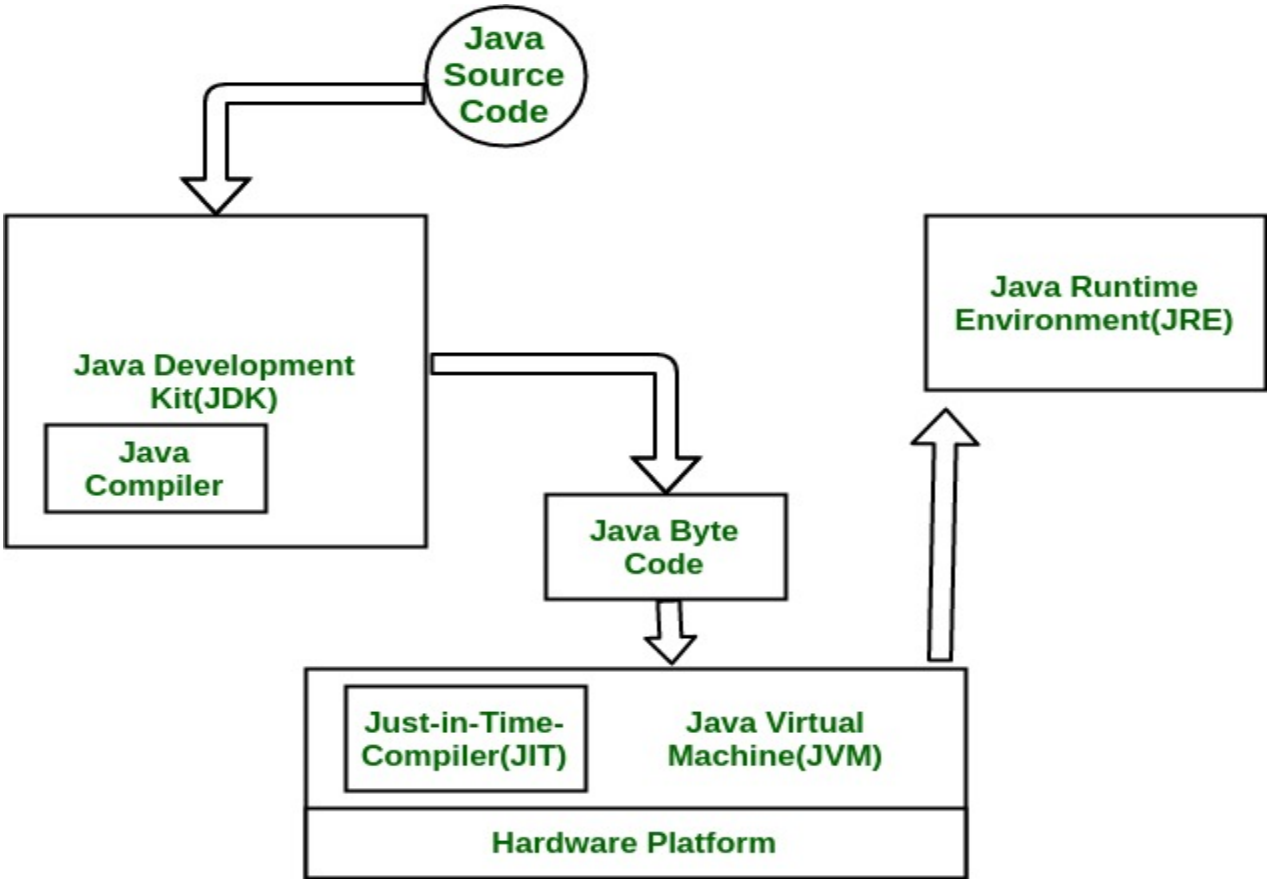
At runtime the Byte Code is loaded, checked and run by the interpreter. The interpreter has the following two functions:

- Execute the Byte Code
- Make appropriate calls to the underlying hardware

Both operations can be shown as:



To understand the interactions between JDK and JRE consider the following diagram.



JVM Architecture