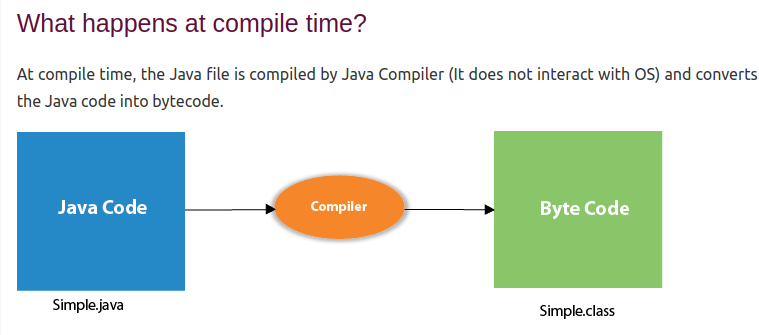
**JAVA**

**What happens at compile and run time?**

****

### **Compile Time**

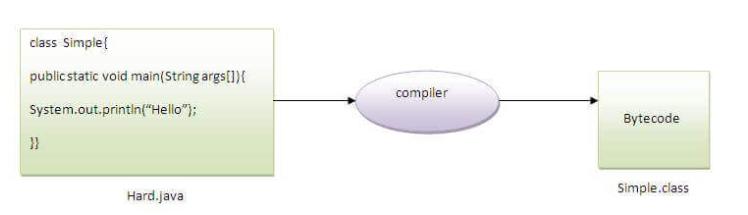
1. **Source Code to Bytecode:**
   * The Java compiler (javac) takes the source code (.java files) and compiles it into bytecode (.class files). Bytecode is a platform-independent set of instructions that can be executed by the Java Virtual Machine (JVM).
2. **Syntax and Semantic Checking:**
   * The compiler checks the source code for syntax errors (e.g., missing semicolons, unbalanced parentheses) and semantic errors (e.g., type mismatches, undeclared variables).
3. **Generation of Class Files:**
   * If the source code is error-free, the compiler generates .class files, each containing the bytecode for a corresponding .java file.

### **Run Time**

1. **Class Loading:**
   * The JVM loads the necessary class files into memory. The class loader subsystem is responsible for finding and loading the classes.
2. **Bytecode Verification:**
   * The JVM verifies the bytecode to ensure it adheres to Java's security constraints and doesn't perform any illegal operations. This step prevents malicious code from compromising the JVM.
3. **Just-In-Time (JIT) Compilation:**
   * The JVM's JIT compiler translates bytecode into native machine code for the host CPU. This process improves performance by enabling the JVM to execute native code instead of interpreting bytecode.
4. **Execution:**
   * The JVM executes the native code generated by the JIT compiler. The JVM manages memory, performs garbage collection, and handles other runtime responsibilities.
5. **Runtime Environment:**
   * The JVM provides a runtime environment that includes the Java Runtime Environment (JRE), libraries, and other resources needed by the Java application.

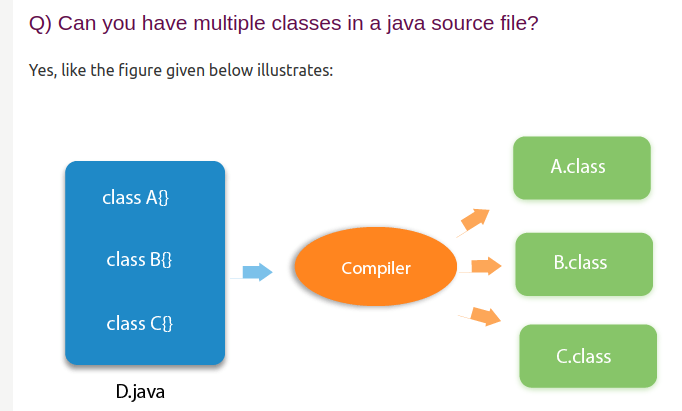


### Q) Can you save a Java source file by another name than the class name?



| **To compile:** | javac Hard.java |
| --- | --- |
| **To execute:** | java Simple |

Observe that, we have compiled the code with file name but running the program with class name. Therefore, we can save a Java program other than class name.



**PATH SETTING**

# 

Here's a brief overview of the process:

1. **Compilation**: During compilation, the Java source code (.java files) is converted into bytecode (.class files) by the Java compiler (javac). This bytecode is platform-independent and can be executed on any system with a Java Virtual Machine (JVM).
2. **Class Loading**: At runtime, the JVM loads the necessary classes from the compiled bytecode. The class loader is responsible for finding and loading class files into the JVM. This process includes:
   * **Loading**: The class loader reads the bytecode of a class and brings it into the JVM.
   * **Linking**: The bytecode is verified, prepared (static fields are allocated memory and initialized to default values), and optionally resolved (symbolic references are replaced with direct references).
   * **Initialization**: The class's static initializers and static fields are executed and initialized.

Class loading happens dynamically and only when needed, allowing Java to be flexible and modular. This also enables features like dynamic class loading and unloading, which are used in various advanced Java frameworks and applications.

# How to set path in Java

* [How to set the path of JDK in Windows OS](https://www.javatpoint.com/how-to-set-path-in-java#)
  + [Setting Temporary Path of JDK](https://www.javatpoint.com/how-to-set-path-in-java#pathtemporary)
  + [Setting Permanent Path of JDK](https://www.javatpoint.com/how-to-set-path-in-java#pathpermanent)
* [How to set the path of JDK in Linux OS](https://www.javatpoint.com/how-to-set-path-in-java#pathlinux)

The path is required to be set for using tools such as javac, java, etc.

If you are saving the Java source file inside the JDK/bin directory, the path is not required to be set because all the tools will be available in the current directory.

However, if you have your Java file outside the JDK/bin folder, it is necessary to set the path of JDK.

**JVM**

### JVM

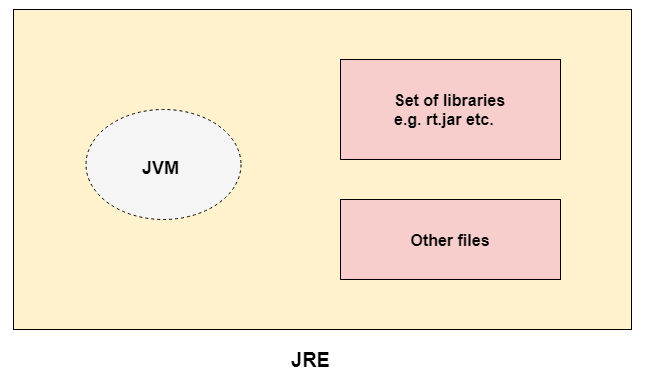
JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed.

The JVM performs the following main tasks:

* Loads code
* Verifies code
* Executes code
* Provides runtime environment

### JRE

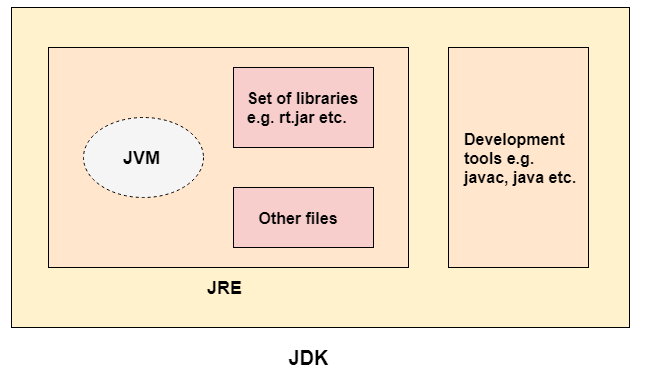
JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.



### JDK

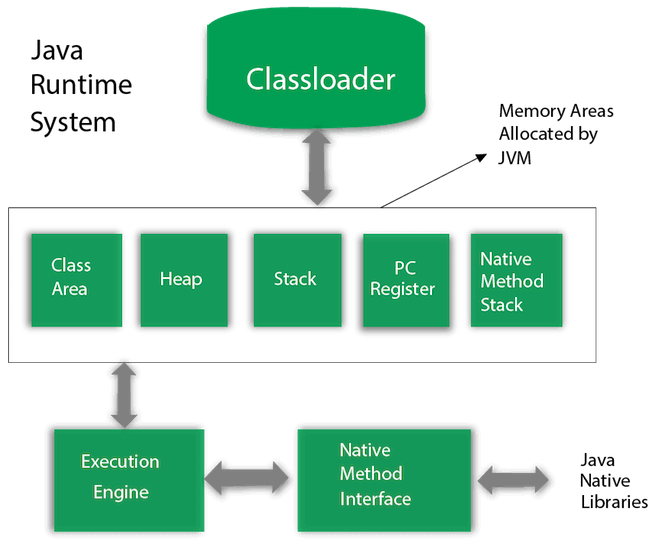
JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and [applets](https://www.javatpoint.com/java-applet). It physically exists. It contains JRE + development tools.

he JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.



## JVM Architecture

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



**Basic concepts of OOPs are:**

1. [**Object**](https://www.javatpoint.com/object-and-class-in-java)
2. [**Class**](https://www.javatpoint.com/object-and-class-in-java#class)
3. [**Inheritance**](https://www.javatpoint.com/inheritance-in-java)
4. [**Polymorphism**](https://www.javatpoint.com/runtime-polymorphism-in-java)
5. [**Abstraction**](https://www.javatpoint.com/abstract-class-in-java)
6. [**Encapsulation**](https://www.javatpoint.com/encapsulation)

**STATIC KEYWORD**

* **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.