Getting Started With Java

Sisoft Technologies Pvt Ltd
SRC E7, Shipra Riviera Bazar, Gyan Khand-3, Indirapuram, Ghaziabad
Website: www.sisoft.in Email:info@sisoft.in
Phone: +91-9999-283-283

Java - Overview

- James Gosling initiated the Java language project in June 1991 for use in one of his many set-top box projects. The language, initially called Oak after an oak tree that stood outside Gosling's office, also went by the name Green and ended up later renamed as Java, from a list of random words.

- Sun released the first public implementation as Java 1.0 in 1995. It promised **Write Once, Run Anywhere** (WORA), providing no-cost run-times on popular platforms.

- On 13 November 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL).

- On 8 May 2007 Sun finished the process, making all of Java's core code free and open-source, aside from a small portion of code to which Sun did not hold the copyright.

Java Comparison

	C	C++	Java
Style/Pradigm	Procedural	Object Oriented	Object Oriented
Run Environment	Runs as native executable machine code	Runs as native executable machine code	Runs on a virtual machine
Platform Dependence	Platform Dependent	Platform Dependent	Platform Independent

Java Features

Java Features

- **Object Oriented :** In java everything is an Object. Java can be easily extended since it is based on the Object model.

- **Platform independent:** Unlike many other programming languages including C and C++ when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by virtual Machine (JVM) on whichever platform it is being run.

Java Features

- **Architectural- neutral :**Java compiler generates an architecture-neutral object file format which makes the compiled code to be executable on many processors, with the presence Java runtime system.

- **Portable :**being architectural neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler and Java is written in ANSI C with a clean portability boundary which is a POSIX subset.

- **Multi-threaded :** With Java's multi-threaded feature it is possible to write programs that can do many tasks simultaneously. This design feature allows developers to construct smoothly running interactive applications.

Object Oriented - Properties

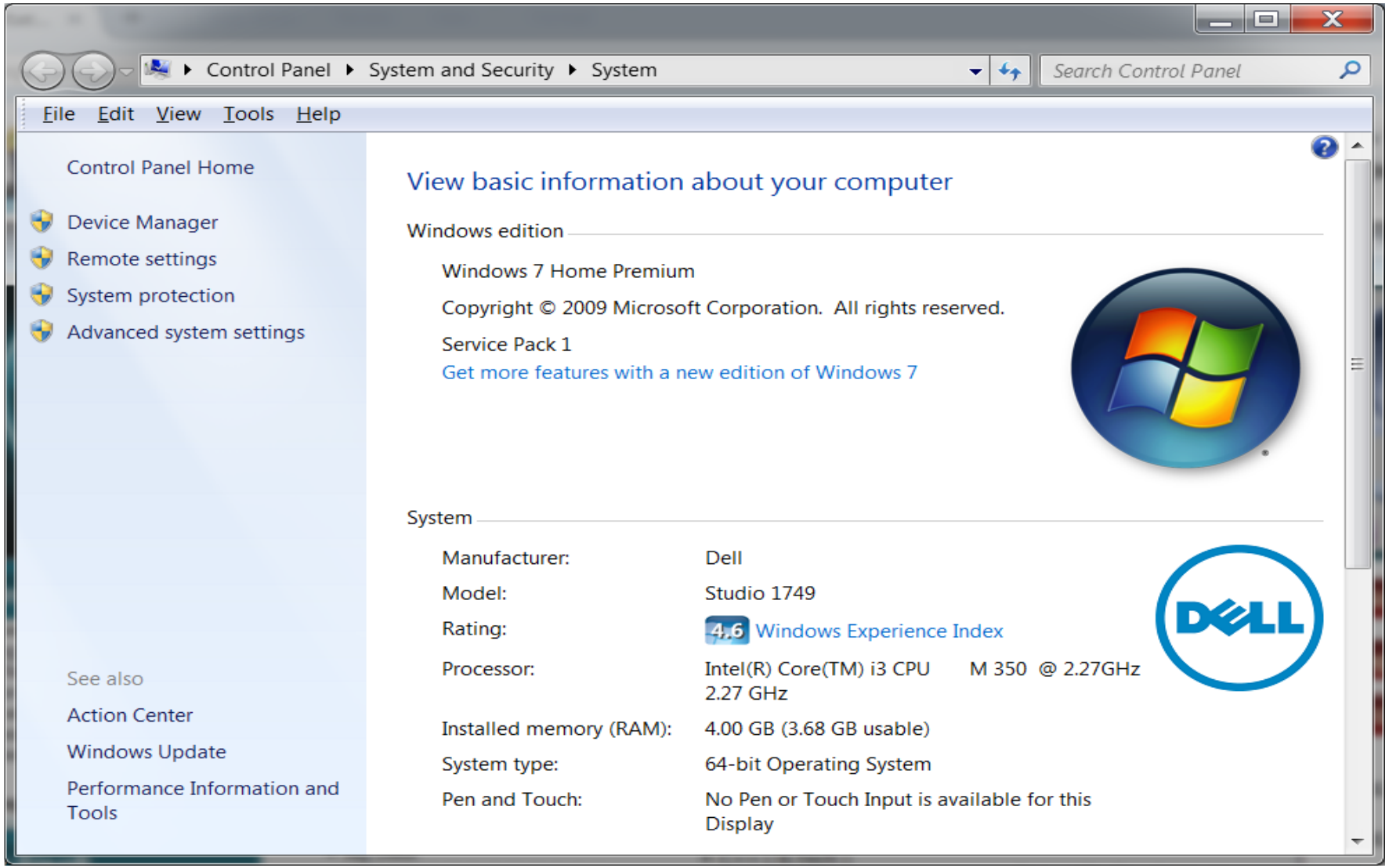
- **Encapsulation:**
  - Construct to define attributes and actions together
- **Data Abstraction:**
  - Data hiding and access restriction
- **Inheritance:**
  - Improve reusability and productivity
- **Polymorphism:**
  - Many forms
  - Overloading & Overriding

Object Oriented - Terms

- **Class:**
  - blueprint or prototype from which objects are created
- **Object:**
  - An instance of class
- **Package:**
  - a namespace that organizes a set of related classes and interfaces
  - similar to different folders on your computer

Java Installation and Environment Setup

Find the computer system type(32-bit or 64-bit)



Download JDK

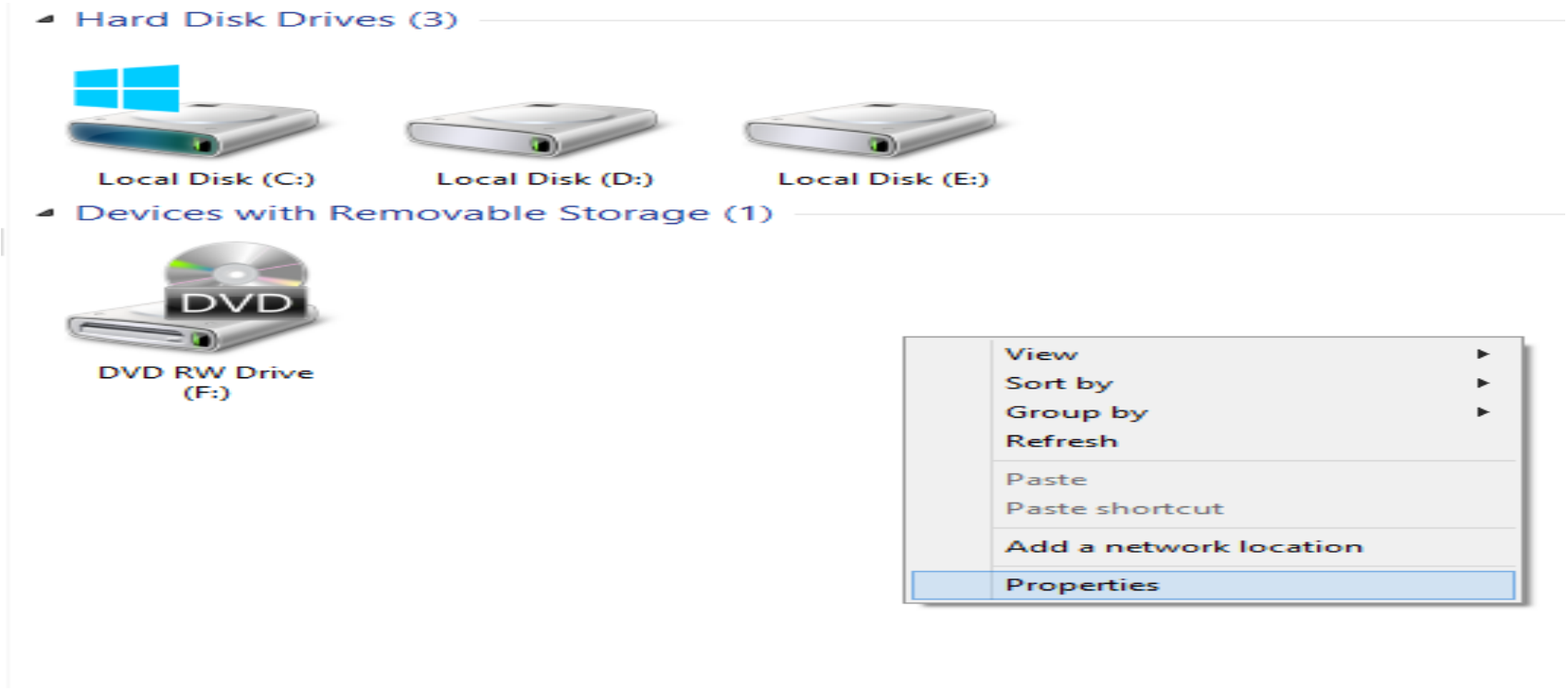
- JDK Download link:
- http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
- X86 is for 32 bit and x64 for 64 bit
- After downloading, run the installer and install JDK in c:\program files\java\

Java: Environment Setup

Setting up the path for windows 2000/XP/win7/win8:

Assuming you have installed Java in *c:\Program Files\java\jdk* directory:

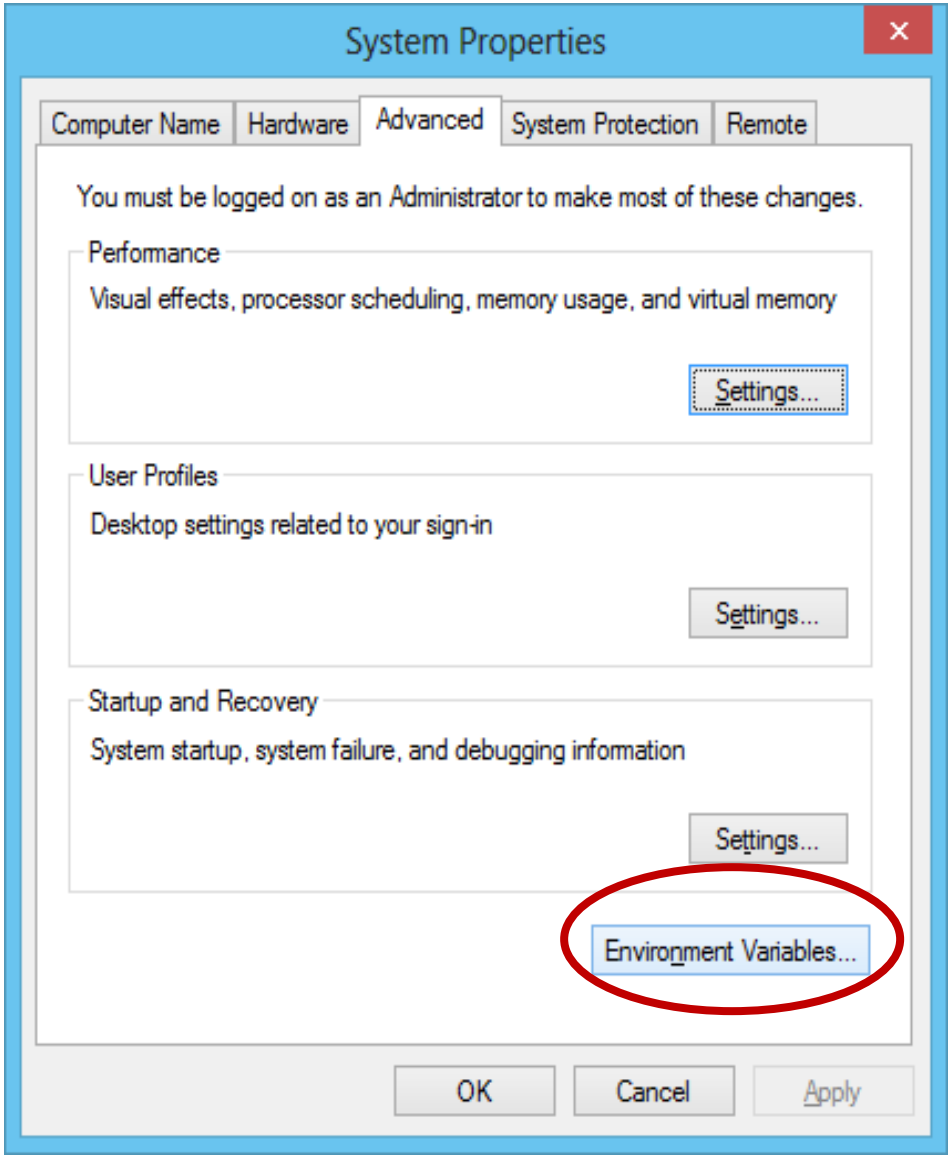
- Right-click on 'My Computer' and select 'Properties'.

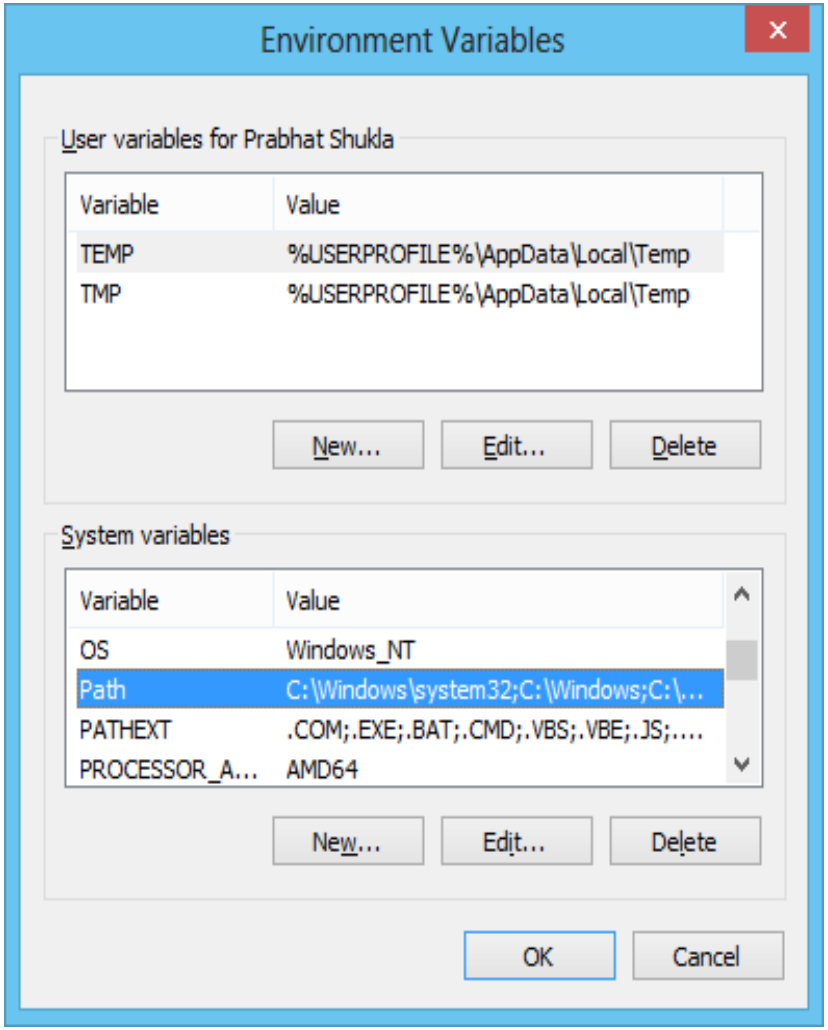


Java: Environment Setup

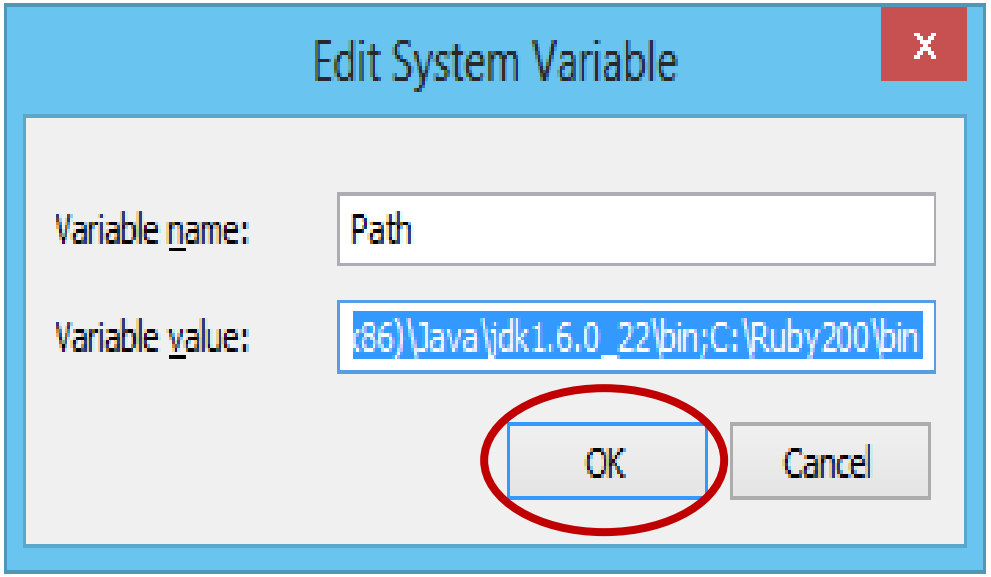
- Click on the 'Environment variables' button under the 'Advanced' tab.
- Now alter the 'Path' variable so that it also contains the path to the Java executable. Example, if the path is currently set to 'C:\WINDOWS\SYSTEM32', then change your path to read 'C:\WINDOWS\SYSTEM32;c:\Program Files\java**jdk\bin**'

Java: Environment Setup





Java: Environment Setup



JDK Files

Assuming the JDK software is installed at /jdk1.8.0, here are some of the most important directories:

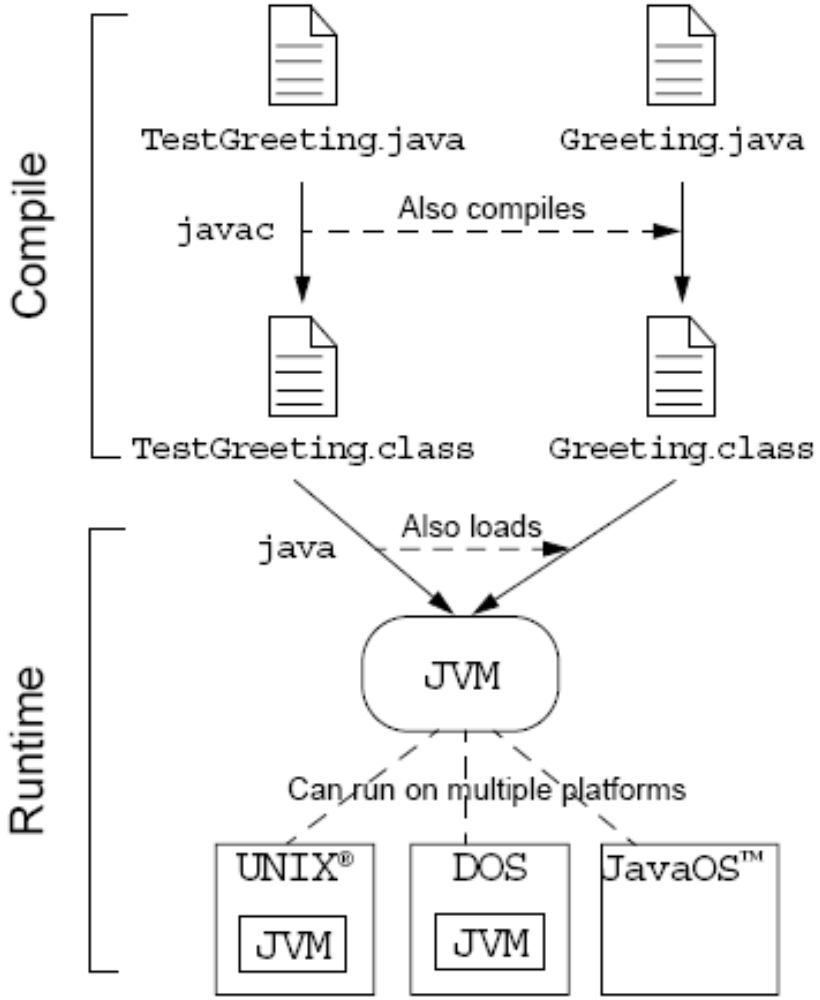
- **/jdk1.8.0/bin:** Executables for all the development tools contained in the JDK. The PATH environment variable should contain an entry for this directory.
- **/jdk1.8.0/lib:** Files used by the development tools. Includes tools.jar, which contains non-core classes for support of the tools and utilities in the JDK. Also includes dt.jar, the DesignTime archive of BeanInfo files that tell interactive development environments (IDEs) how to display the Java components and how to let the developer customize them for an application.
- **/jdk1.8.0/jre:** Root directory of the Java Runtime Environment (JRE) used by the JDK development tools. The runtime environment is an implementation of the Java platform. This is the directory referred to by the java.home system property.
- **/jdk1.8.0/db:** Contains Java DB. See Java DB Technical Documentation at http://docs.oracle.com/javadb/
- **/jdk1.8.0/include:** C-language header files that support native-code programming with the Java Native Interface and the Java Virtual Machine (JVM) Debugger Interface. See Java Native Interface at

http://docs.oracle.com/javase/8/docs/technotes/guides/jni/index.html

First Java Program
“Hello sisoft”

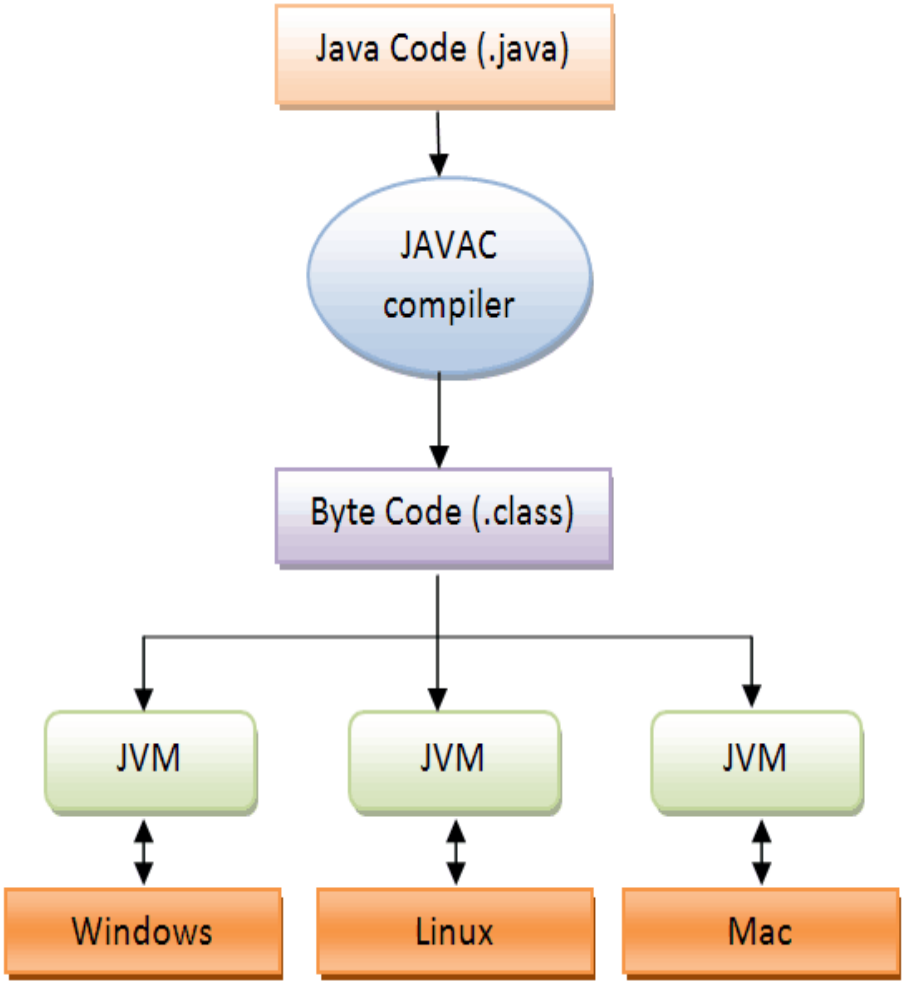
Java Run Environment

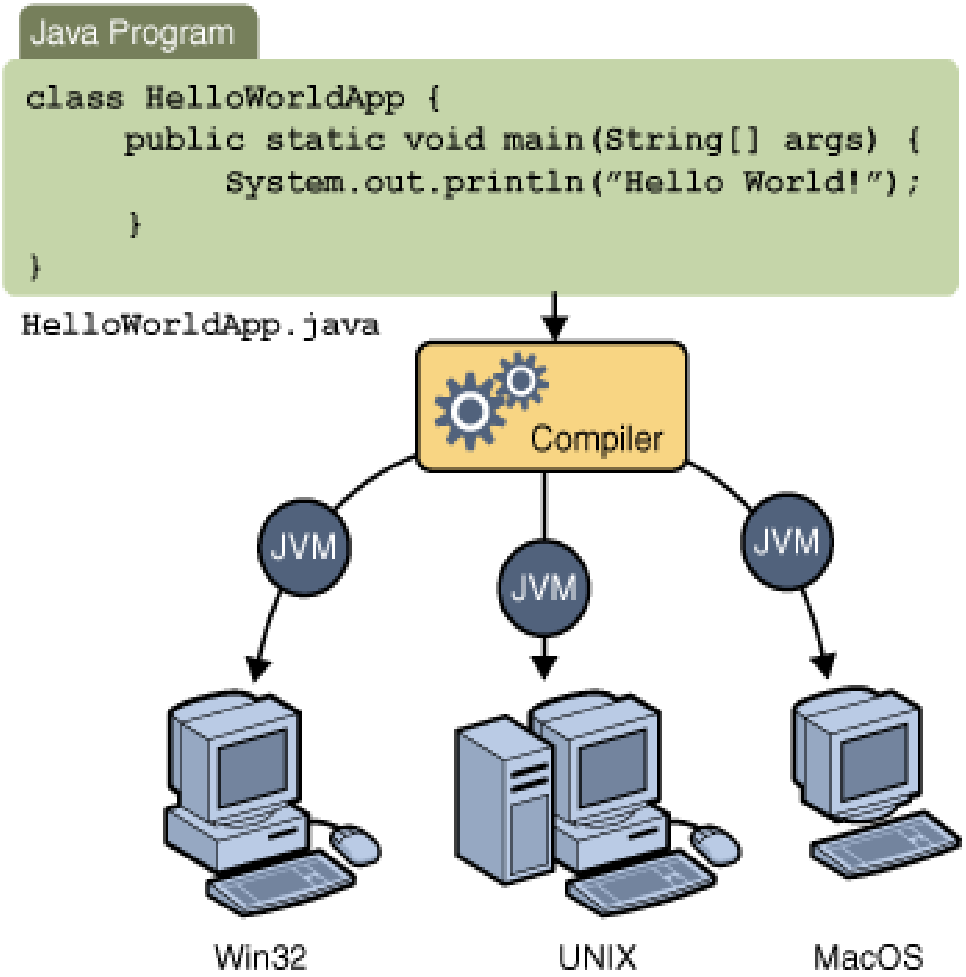
Java Technology Runtime Environment



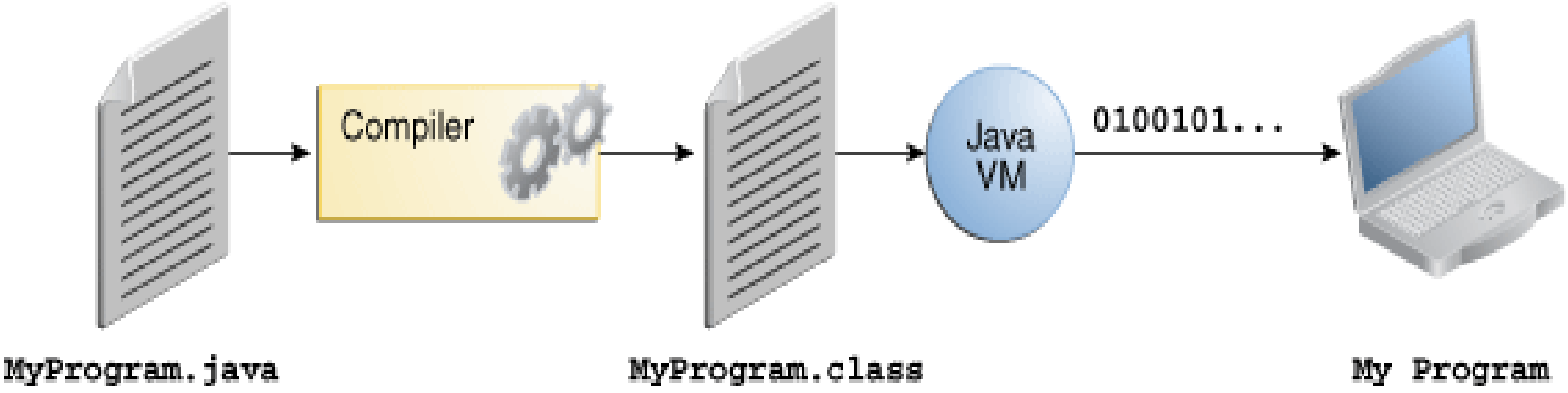
Platform independent

- You can run one byte code on different-2 o/s





Execution of Java program



Structure of Java programs

```
public class <name> {
    public static void main(String[] args) {
        <statement>;
        <statement>;
        ...
        <statement>;
    }
}
```

- Every executable Java program consists of a **class**
  - that contains a **method** named `main`
    - that contains the **statements** (commands) to be executed
- Name of file should match name of class

First Java Program:

```
public class Hellojava
{
/* This is my first java program. * This will print 'Hello World' as the output */
public static void main(String []args)
{
   System.out.println("Hello Sisoft"); // prints Hello World
 } }
```

Open a command prompt window and go to the directory where you saved the class. Assume its C:\

```
C : > javac  Hellojava.java
C : > java  Hellojava
Hello Sisoft
```

Understanding Main Method

- The public static void main function is an important function in Java programming language. Now we will learn about main function.
- The main method is the first method, which the Java Virtual Machine(JVM) executes. When you run a class with the java interpreter, the Runtime System begin by calling the class's `main()` method.
- The `main()` method then calls all other methods required to execute our application.
- It may be said that the main method is the starting point in the Java program, and java program can't run without this method.
- The signature of `main()` method seems like:
- `public static void main(String args[])`
- The method signature for the main() method contains the 3 modifiers:
  - Here public notify that the main() method can be called by any object.
  - Here static notify that the main() method is a class method.
  - Here void notify that the main() method has never return the value.

System.out.println

- `System.out.println` : A statement to instruct the computer to print a line of output on the console.
  - pronounced "*print-linn*"
  - sometimes called a "*println statement"* for short
- Two ways to use `System.out.println`:

  `System.out.println("`***<Message>***"`);`
  - Prints the given message as a line of text on the console.

  `System.out.println();`
  - Prints a blank line on the console.

System.out.println

- **System – is a final class** in java.lang package. Among the facilities provided by the System class are standard input, standard output, and error output streams; access to externally defined properties and environment variables; a means of loading files and libraries; and a utility method for quickly copying a portion of an array.
- **out – is a static member field of System class and is of type PrintStream**. Its access specifiers are public final. This gets instantiated during startup and gets mapped with standard output console of the host. This stream is open by itself immediately after its instantiation and ready to accept data.
- **println – is a method of PrintStream class**. println prints the argument passed to the standard console and a newline. There are multiple println methods with different arguments (overloading). Every println makes a call to print method and adds a newline. print calls write() and the story goes on like that.

Another Java program

```
public class Hello2 {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
        System.out.println();
        System.out.println("This program produces");
        System.out.println("four lines of output");
    }
}
```

- The code in this program instructs the computer to print four messages on the screen.
  - Its output:

```
Hello, world!

This program produces
four lines of output
```

Scanner Class

- Scanner is used to parse primitive types and String using regular expression
- This is also used to read input
- Import java.util.Scanner
- Code to read input from System.in

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
```

Comments

- **comment**: A note written in the source code by the programmer to make the code easier to understand.
  - Comments are not executed when your program runs.
  - Most Java editors show your comments with a special color.

- Java language provides three styles (*Single Line, Multi-line, and Javadoc*) of comments.
  - Java single line comments or slash-slash comments or end of the line comments (//...)
    `//` ***<comment text, on one line>***
  - Java multi-line or traditional comments (/*...*/)
    `/*` ***<comment text; may span multiple lines>*** `*/`
  - Javadoc or Java documentation comments (/**...*/)

- Examples:

```
/* A comment goes here. */
/* It can even span
   multiple lines. */
// This is a one-line comment.
```

Using comments

- Where to place comments:
  - at the top of each file (also called a "comment header"), naming the author and explaining what the program does
  - at the start of every method, describing its behavior
  - inside methods, to explain complex pieces of code (more useful later)
- Comments provide important documentation.
  - Later programs will span hundreds of lines with many methods.
  - Comments provide a simple description of what each class, method, etc. is doing.
  - When multiple programmers work together, comments help one programmer understand the other's code.

Comments example

```
/* Suzy Student
   CS 101, Fall 2019
   This program prints lyrics from my favorite song! */
public class MyFavoriteSong {
    /* Runs the overall program to print the song
       on the console. */
    public static void main(String[] args) {
        sing();

        // Separate the two verses with a blank line
        System.out.println();

        sing();
    }

    // Displays the first verse of the theme song.
    public static void sing() {
        System.out.println("Now this is the story all about how");
        System.out.println("My life got flipped turned upside-down");
    }
}
```

How to comment: methods

- Do not describe the syntax/statements in detail.
- Instead, provide a short English description of the observed behavior when the method is run.
  - Example:

```
// This method prints the lyrics to the first verse
// of my favorite TV theme song.
// Blank lines separate the parts of the verse.
public static void verse1() {
    System.out.println("Now this is the story all about how");
    System.out.println("My life got flipped turned upside-down");
    System.out.println();
    System.out.println("And I'd like to take a minute,");
    System.out.println("just sit right there");
    System.out.println("I'll tell you how I became the prince");
    System.out.println("of a town called Bel-Air");
}
```

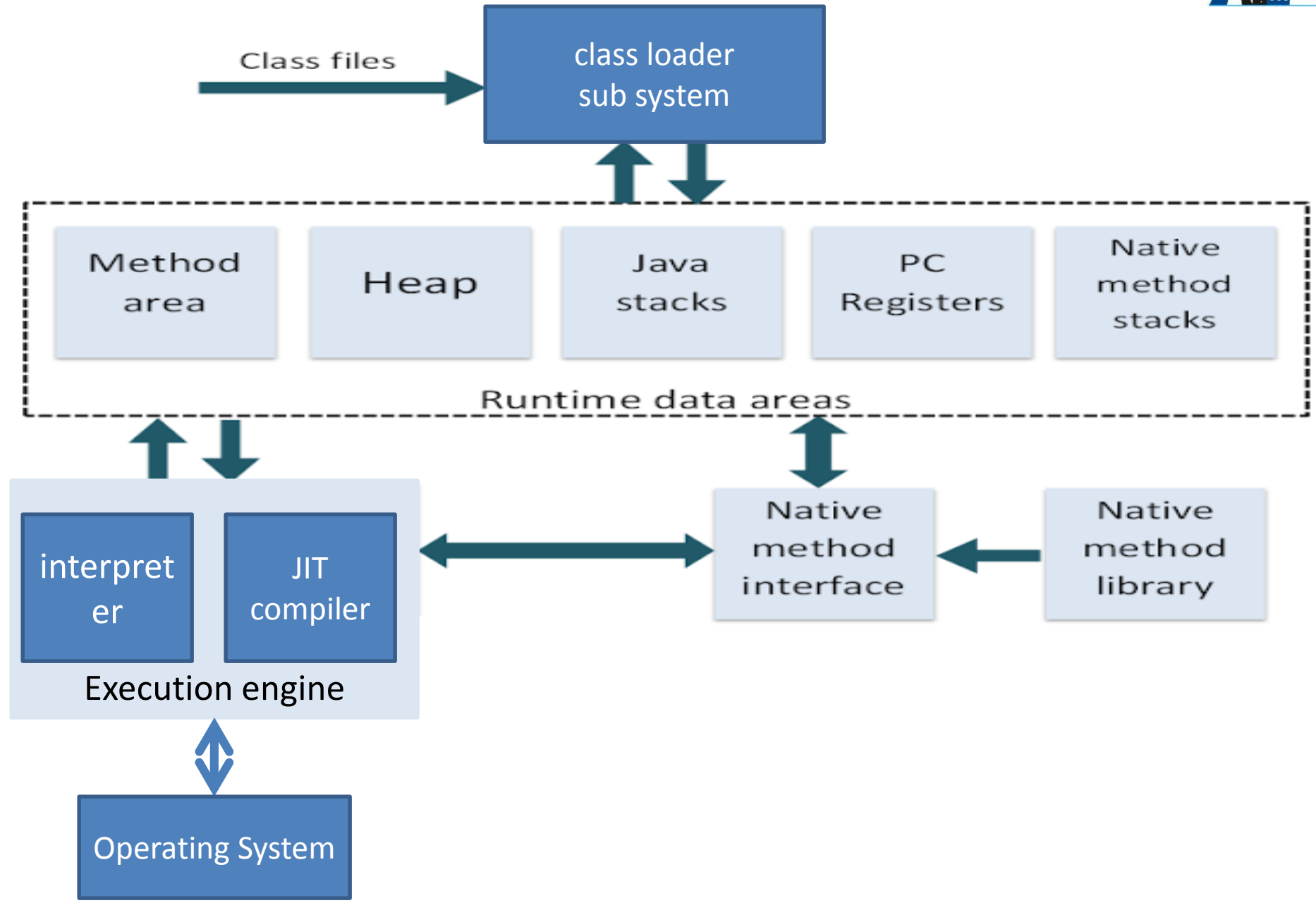
JVM
(Java Virtual Machine)
Architecture

JVM tasks

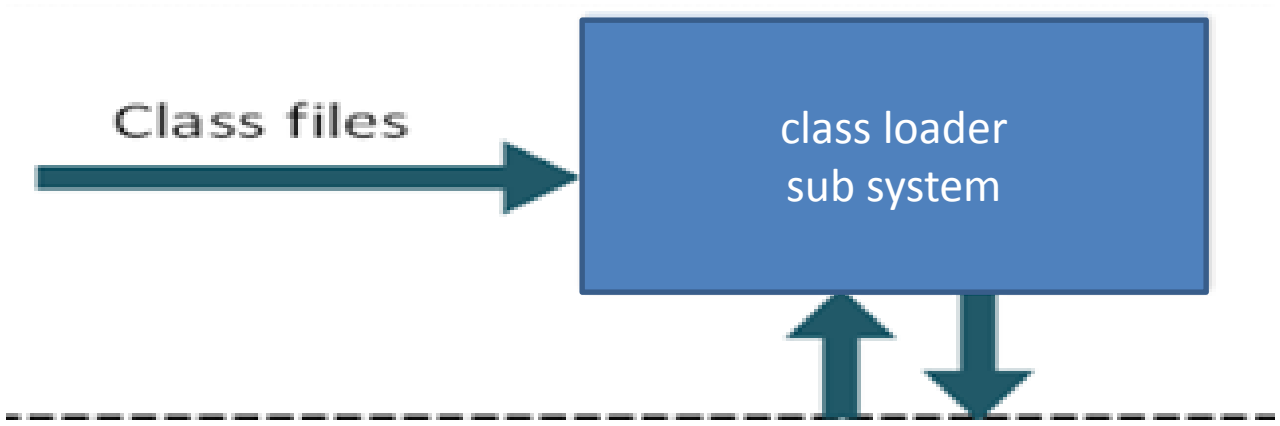
The JVM performs three main tasks:

- Loads code
- Verifies code
- Executes code

JVM Architecture



JVM Architecture



1. **Class loader sub system:** JVM's class loader sub system performs 3 tasks
   a. It loads .class file into memory.
   b. It verifies byte code instructions.
   c. It allots memory required for the program.

The Class Loader

- Loads all classes necessary for the execution of a program
- Maintains classes of the local file system in separate *namespaces*
- Prevents spoofing

JVM Architecture

2. Run time data area: This is the memory resource used by JVM and it is divided into 5 parts

a. Method area: Method area stores class code and method code.

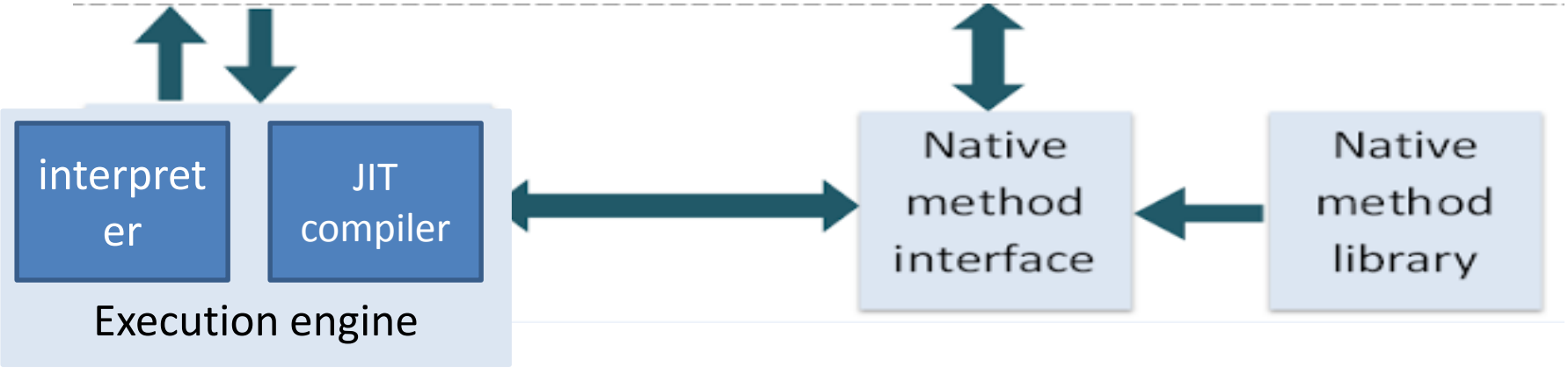
b. Heap: This is the area where objects are created .

c. Java stacks: Java stacks are the places where the Java methods are executed. A Java stack contains frames. On each frame, a separate method is executed.

d. Program counter (PC)registers: The program counter registers store memory address of the instruction to be executed by the micro processor.

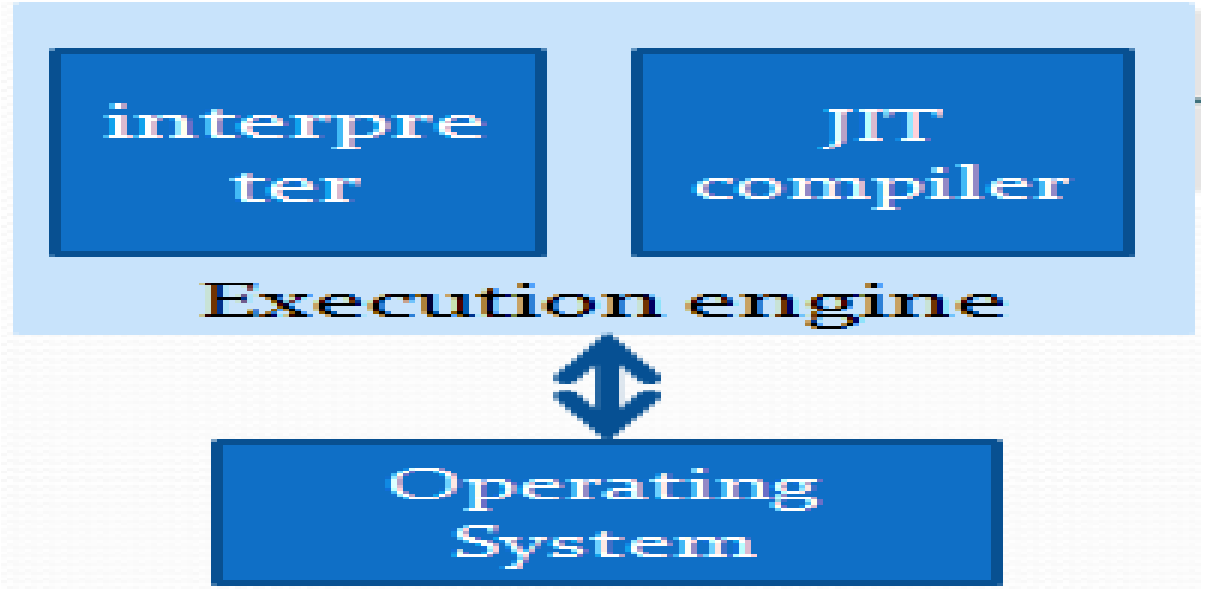
e. Native method stacks: The native method stacks are places where native methods (for example, C language programs) are executed. Native method is a function, which is written in another language other than Java.

JVM



- **3. Native method interface:** Native method interface is a program that connects native methods libraries (C header files) with JVM for executing native methods.

  4. Native method library: holds the native libraries information.



- **5. Execution engine:** Execution engine contains interpreter and JIT compiler, which covert byte code into machine code. JVM uses optimization technique to decide which part to be interpreted and which part to be used with JIT compiler. The HotSpot represent the block of code executed by JIT compiler

Questions ?