Lecture 02: OOPs Principles, JVM Architecture

Mahesh Kumar
Assistant Professor (Adhoc)

Department of Computer Science
Acharya Narendra Dev College
University of Delhi

Course webpage
[http://www.mkbhandari.com/mkwiki]

Outline

1. OOPs Principles [class notes]
2. First Sample Program
3. Java Virtual Machine(JVM) Architecture

The First Sample Program


```
1
// This is a simple Java Program
2  3
class HelloWorld{

        public static void main(String args[] {

                System.out.println("Hello World!");

        }

}
```

1. **Comment**
   - Describes how the program works or what a specific feature does.
   - The contents of a comment are ignored by the compiler.
2. **class**
   - The keyword class is used to define/create a new class.
3. **HelloWorld**
   - HelloWorld is an identifier that is the name of the class.
   - The entire class definition, including all of its members, will be between the opening curly brace ({) and the closing curly brace (}).
   - in Java, all program activity occurs within class.

The First Sample Program

```
1
// This is a simple Java Program
2  3
class HelloWorld{
        4  5  6
        public static void main(String args[] {

                System.out.println("Hello World!");

        }

}
```

4 public

- The keyword public is an access modifier (visibility specifier).
- Regulate access to classes, fields and methods in Java.

5 static

- The keyword static allows main( ) to be called without creating object of the class.
- This is necessary since main( ) is called by the Java Virtual Machine before any objects are made.

6 void

- The keyword void simply tells the compiler that main( ) does not return a value.

The First Sample Program

```
1
// This is a simple Java Program
2 3
class HelloWorld{
    4 5 6 7 8
    public static void main(String args[] ) {
        9
        System.out.println("Hello World!");

    }

}
```

7 main( )

- The main( ) method is the starting point for JVM to start execution of a Java program.
- Without the main() method, JVM will not execute the program.

8 String args[ ]

- The parameter args is used to receive any command-line arguments when the program is executed.

9 System.out

- System is a predefined class that provides access to the system
- out is the output stream that is connected to the console(terminal window).

The First Sample Program

```
// This is a simple Java Program
class HelloWorld{
        public static void main(String args[] ) {
                System.out.println("Hello World!");
        }
}
```

10 println( )

- The bulit-in println( ) method displays the string which is passed to it.

11 Other important information

- A complex program will have dozens of classes, only one of which will need to have a main( ) method to get things started.
- In some cases, you won't need main( ) at all. For example, when creating applets.
- All statements in Java end with a semicolon.

Lexical Issues (Tokens)

- Java programs are a collection of whitespaces, identifiers, literals, comments, operators, separators, and keywords.

1. **Whitespace**
   - whitespace is a space, tab, or newline. Java is a free-form language.
2. **Identifiers**
   - Identifiers are used to name things, such as classes, variables, and methods. Java is case-sensitive, so Name is a different identifier than name.
3. **Literals**
   - A constant value in Java is created by using a literal representation of it.
4. **Comments**
   - Three types of comments defined by Java (Single-line, Multiline, Documentation).
5. **Separators**
   - There are 7 types of separatrors(**( )**, **{ }**, **[ ]**, **;** , **,** , **.** , **::**)
6. **Keywords**
   - There are 51 keywords currently defined in the Java language(49 are in use)

Separators

Symbol	Name	Purpose
( )	Parentheses	Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates statements.
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a **for** statement.
.	Period	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.
::	Colons	Used to create a method or constructor reference. (Added by JDK 8.)

Keywords

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

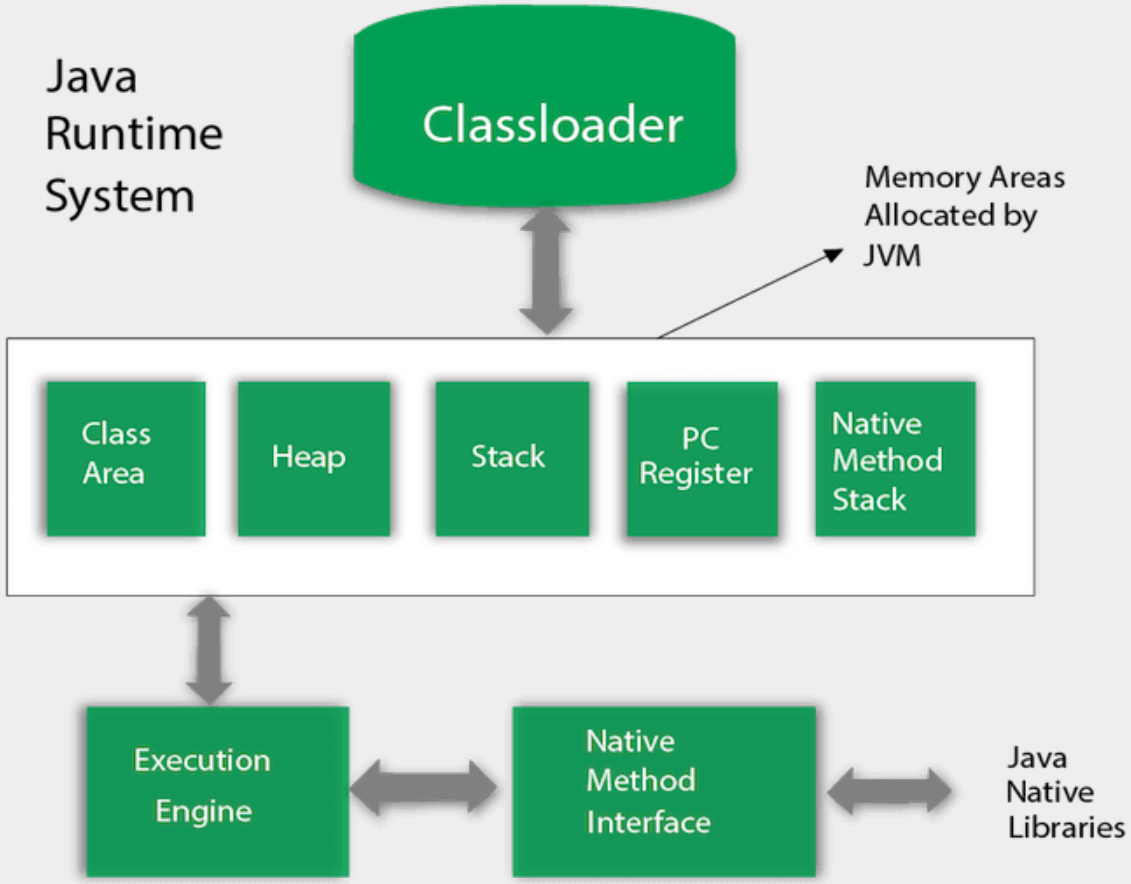
JVM Architecture

- JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.
- JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).
- What is JVM?
  1. **A specification:** where working of Java Virtual Machine is specified. Its implementation has been provided by Oracle and other companies.
  2. **An implementation:** Its implementation is known as JRE (Java Runtime Environment).
  3. **Runtime Instance:** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

JVM Architecture

- The JVM performs following operation:
  - 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.

JVM Architecture



1 Classloader

- Classloader is a subsystem of JVM which is used to load class files.
- Whenever we run the java program, it is loaded first by the classloader.

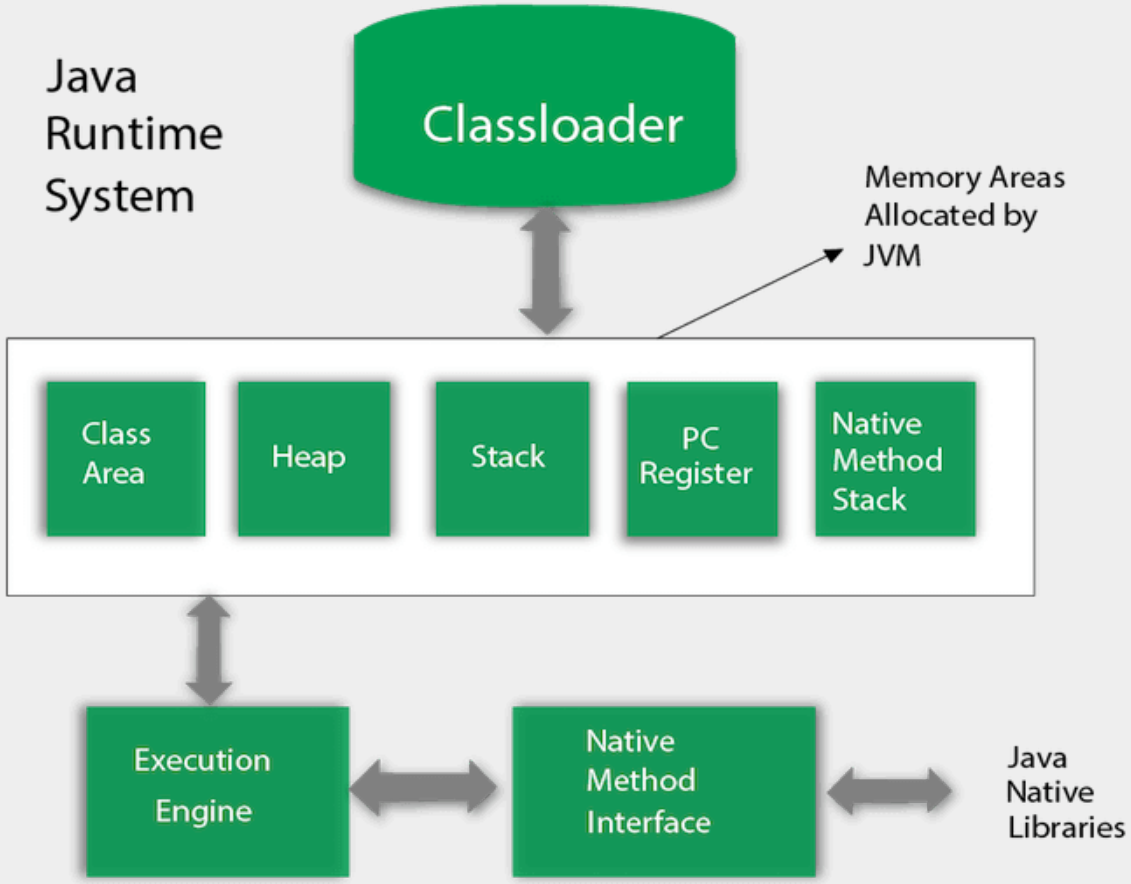
2 Class(Method) Area

- Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods

3 Heap

- It is the runtime data area in which objects are allocated.

JVM Architecture



4 Stack

- It holds local variables and partial results, and plays a part in method invocation and return.
- A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

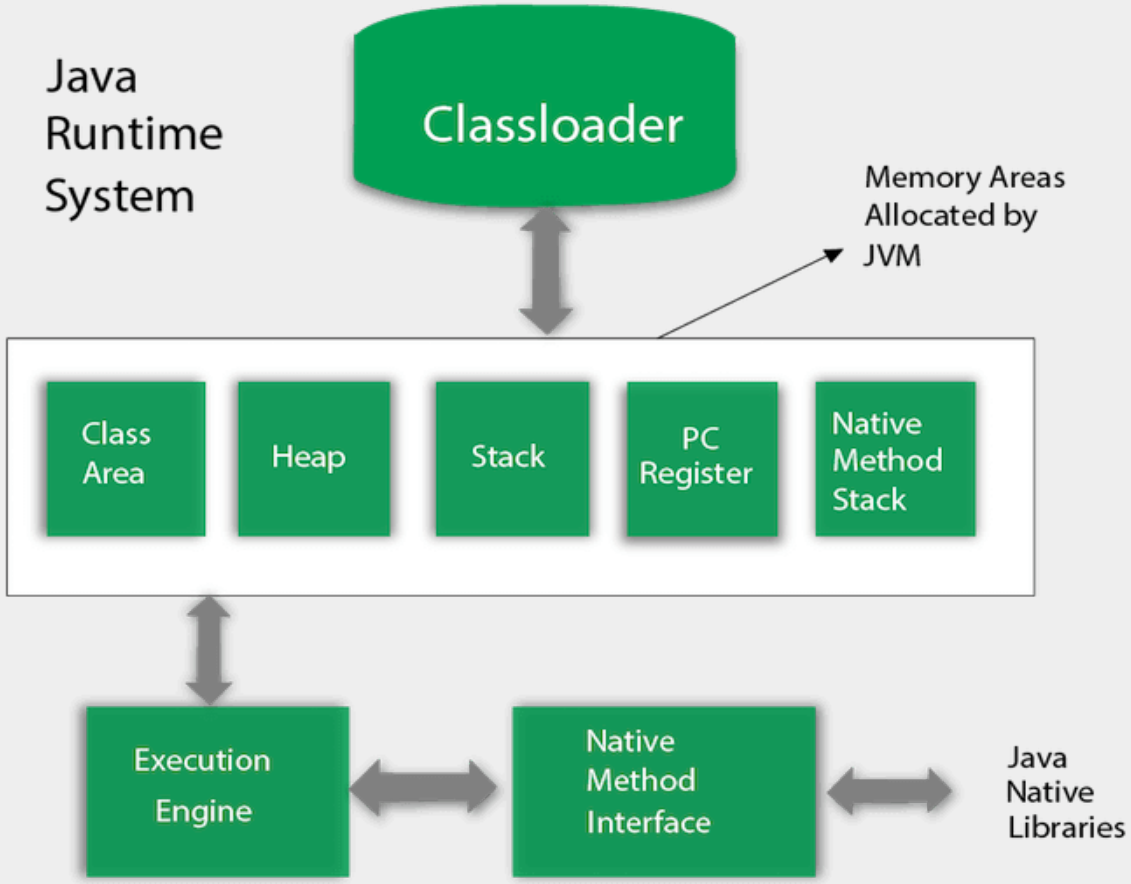
5 PC Register

- PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.

6 Native Method Stack

- It contains all the native methods used in the application.

JVM Architecture



7. **Execution Engine** contains:
   - A virtual processor
   - Interpreter: Read bytecode stream then execute the instructions.
   - Just-In-Time(**JIT**) compiler: helps improve the performance of Java programs by compiling bytecodes into native machine code at run time.

8. **Java Native Interface**
   - Java Native Interface (JNI) is a framework which provides an interface to communicate with application written in C, C++, Assembly etc. Java uses JNI framework to send output to the Console or interact with OS libraries.

References

Reference for this topic

- [Book: Java: The Complete Reference, Ninth Edition: Herbert Schildt]
  https://www.amazon.in/Java-Complete-Reference-Herbert-Schildt/dp/0071808558

- [Web: Java T Point tutorial]
  https://www.javatpoint.com/java-tutorial

- [Web: GeeksforGeeks]
  https://www.geeksforgeeks.org/java/