Head First

Chapter-1:
Breaking the Surface - JAVA working process

Upcode Software
Engineer Team

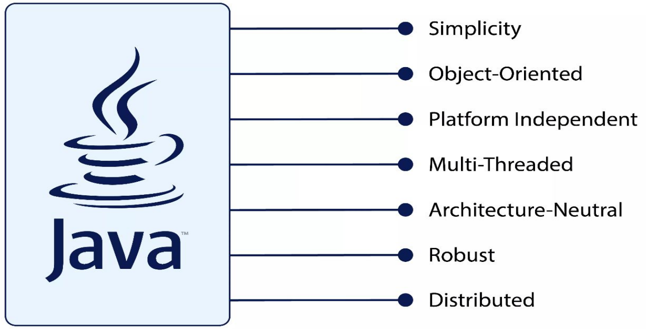
-2023-

CONTENT

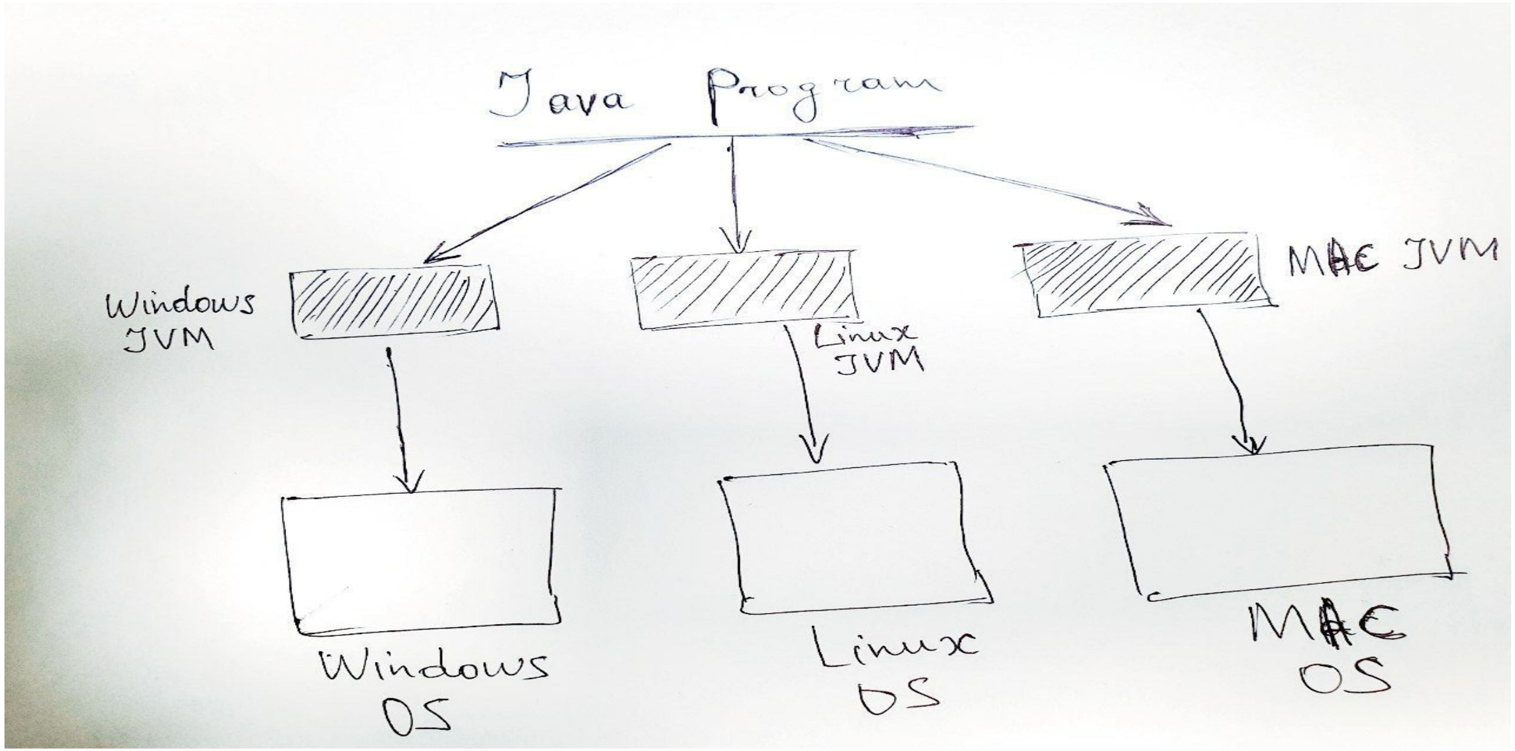
1. What is Java ?
2. Why Java is independent from any OS ?
3. How JAVA works and what happens behind the scenes ?
4. What is a JDK, JRE and JVM ?
5. Used materials and references

1.What is Java?

- Java is a widely used object-oriented programming language and software platform that runs on billions of devices, including notebook computers, mobile devices, gaming consoles, medical devices and many others.
- The rules and syntax of Java are based on the C and C++ languages.



2. Why Java is independent from any OS



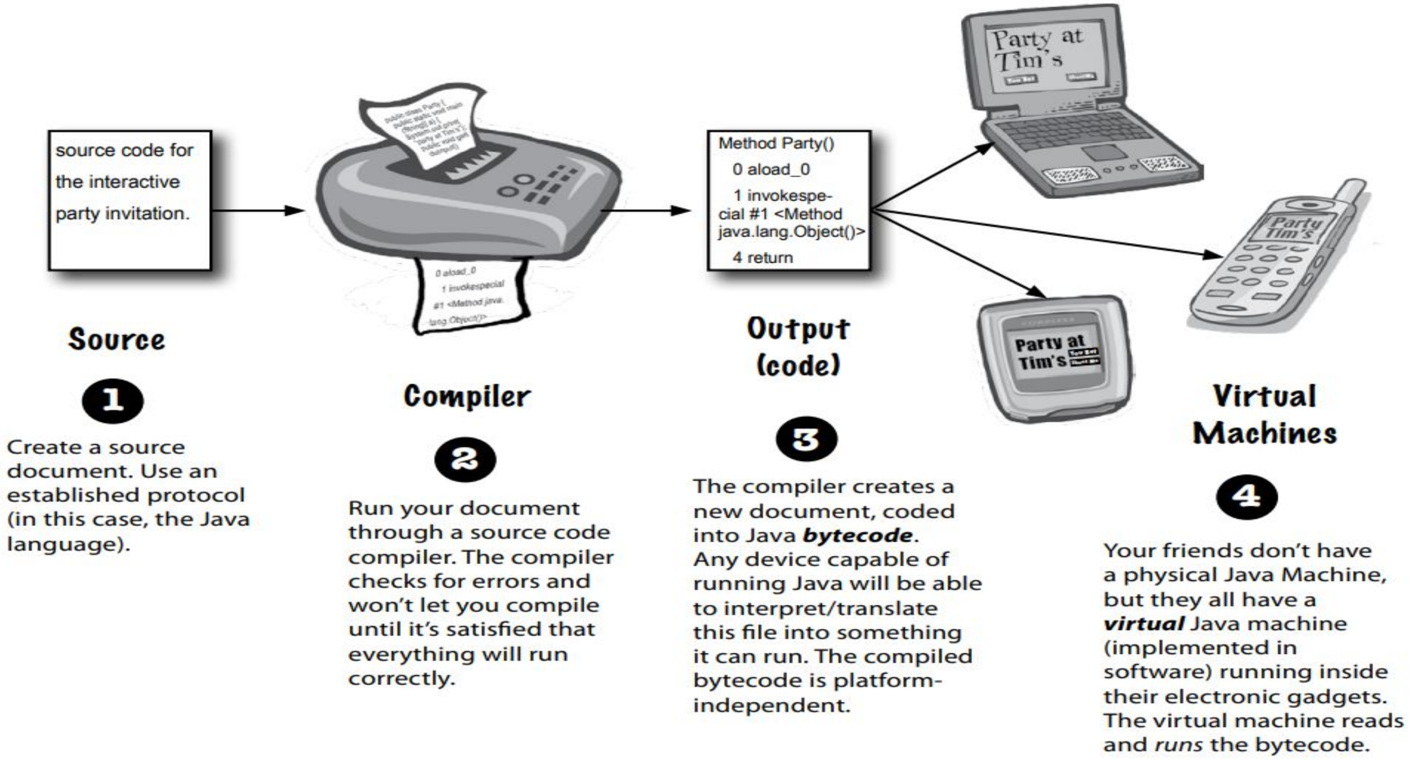
3. Why Java is independent from any OS

1. **Java is platform-independent because it uses virtual machine.**
2. **Platform-independent languages like Java offer several advantages. Some of them are:**
   - **Portability**

   Programs can be written once and executed on multiple platforms, reducing development effort and maintenance.
   - **Wider Audience**

   Platform-independent programs can reach a broader audience as they are not limited to a specific operating system or hardware architecture.

3. The How Java works and what happens behind the



```
Method Party()
  0 aload_0
  1 invokespe-
cial #1 <Method
java.lang.Object()>
  4 return
```

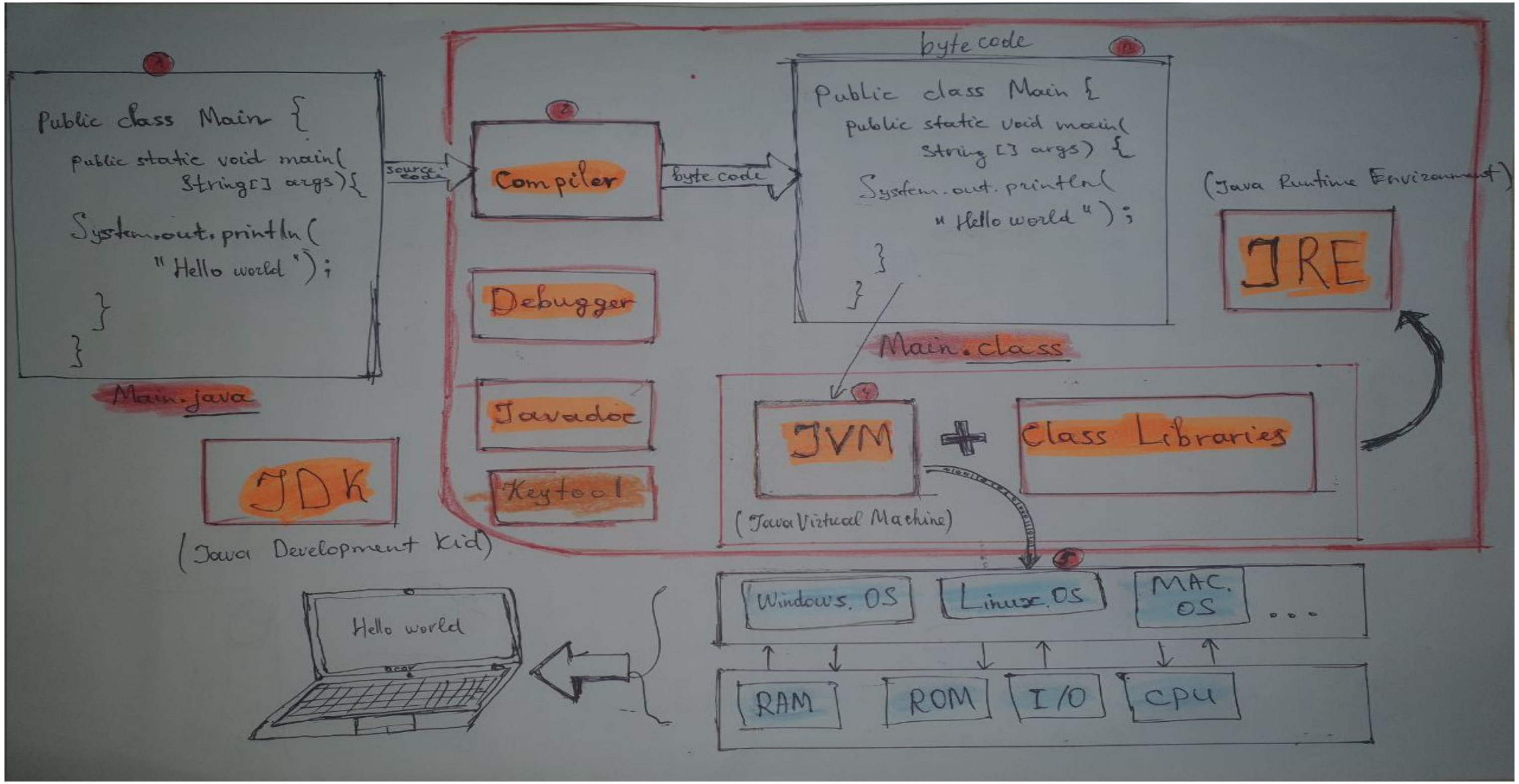
Create a source document. Use an established protocol (in this case, the Java language).

Run your document through a source code compiler. The compiler checks for errors and won't let you compile until it's satisfied that everything will run correctly.

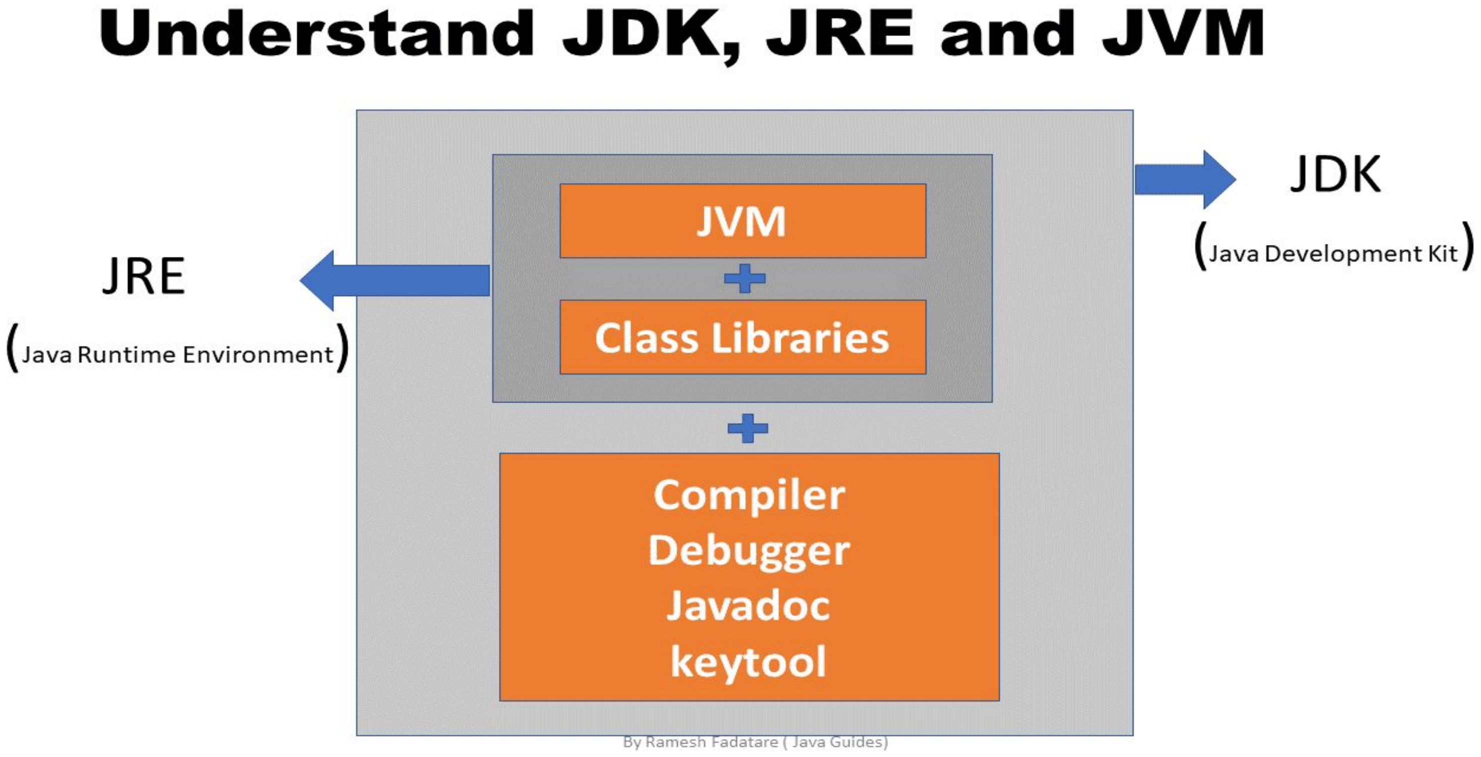
The compiler creates a new document, coded into Java ***bytecode***. Any device capable of running Java will be able to interpret/translate this file into something it can run. The compiled bytecode is platform-independent.

Your friends don't have a physical Java Machine, but they all have a ***virtual*** Java machine (implemented in software) running inside their electronic gadgets. The virtual machine reads and *runs* the bytecode.

How to work JDK



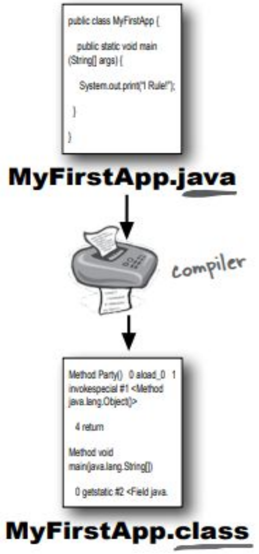
4. What is a JDK, JRE and JVM?


Understand JDK, JRE and JVM
JVM
Class Libraries
Compiler
Debugger
Javadoc
keytool
JRE
(Java Runtime Environment)
JDK
(Java Development Kit)
By Ramesh Fadatare (Java Guides)

Why use JDK?

Important reasons to use JDK:

- The **JDK** has a private **JVM** and a few other resources necessary for the development of a Java Application.
- The **JDK** contains the necessary tools for writing Java programs and the JRE for running them.
- It includes a **Compiler**
- Java application launcher
- It uses **JRE** to run applications developed in Java and **JRE** loads the necessary classes, and invokes its main methods.



```
public class MyFirstApp {

  public static void main
(String[] args) {

    System.out.print("I Rule!");

  }

}
```

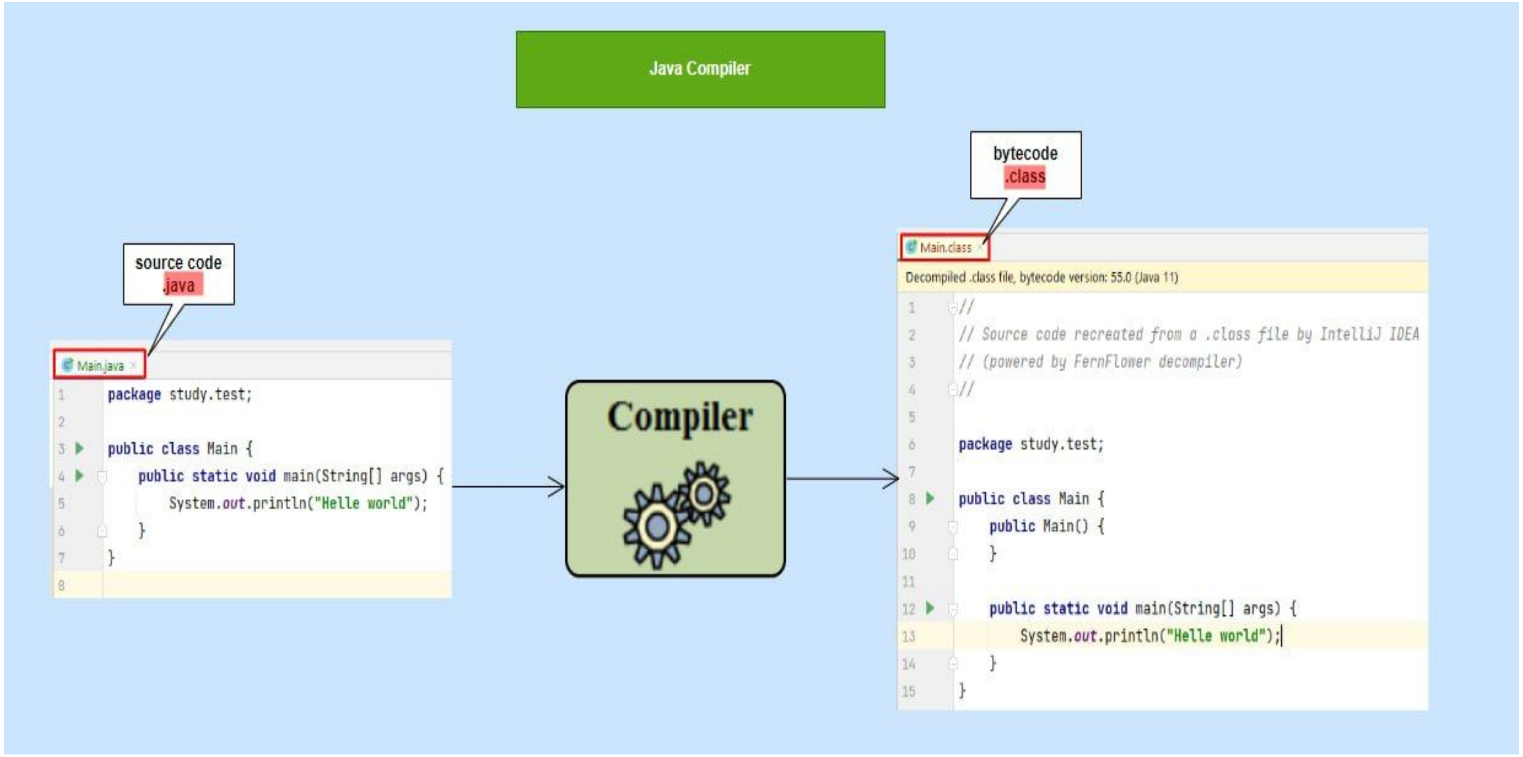
```
Method Party()  0 aload_0  1
invokespecial #1 <Method
java.lang.Object()>

  4 return

Method void
main(java.lang.String[])

  0 getstatic #2 <Field java.
```

What does the compiler do?



```
package study.test;

public class Main {
    public static void main(String[] args) {
        System.out.println("Helle world");
    }
}
```

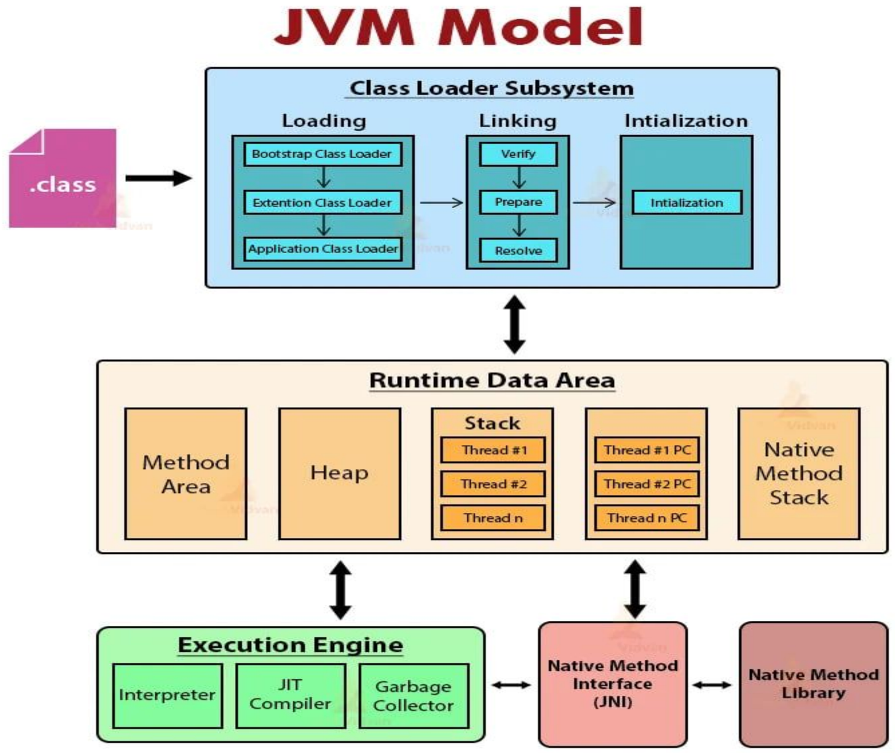
```
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by FernFlower decompiler)
//

package study.test;

public class Main {
    public Main() {
    }

    public static void main(String[] args) {
        System.out.println("Helle world");
    }
}
```

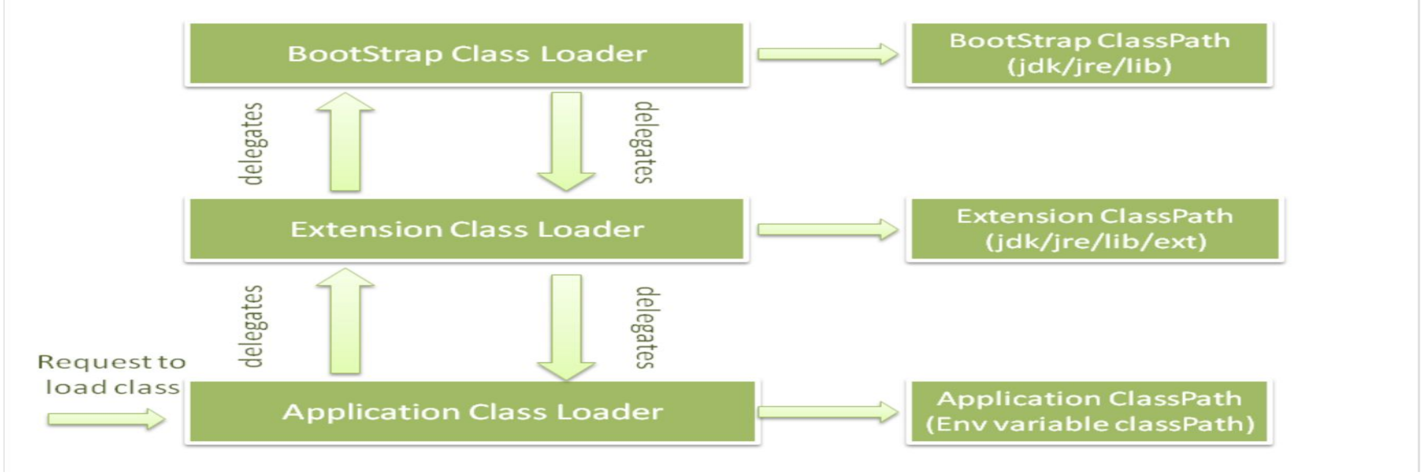
JVM Architecture



JVM Architecture

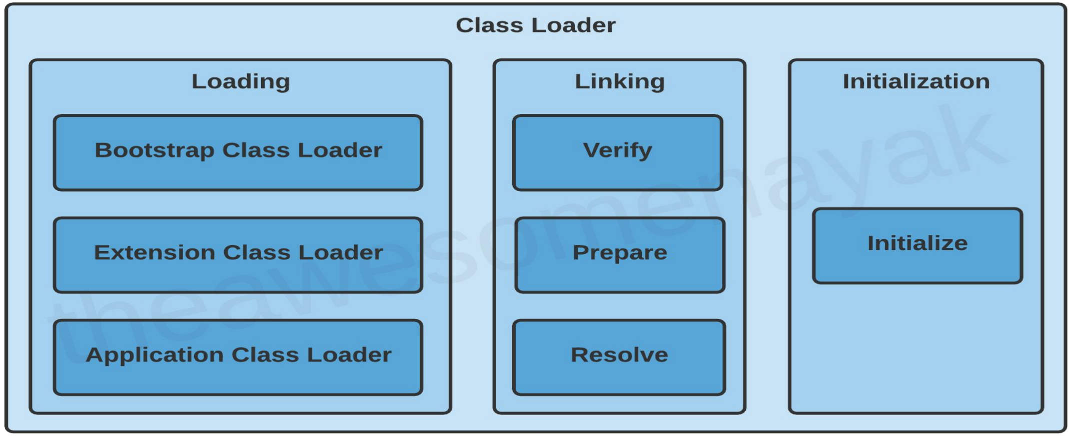
The JVM consists of three distinct components

1. Class Loader - Loaders of classes into memory
2. Runtime Data Area - the part of memory that stores data in the implementation process
3. Execution Engine - the part that performs actions



Class Loader Subsysteam

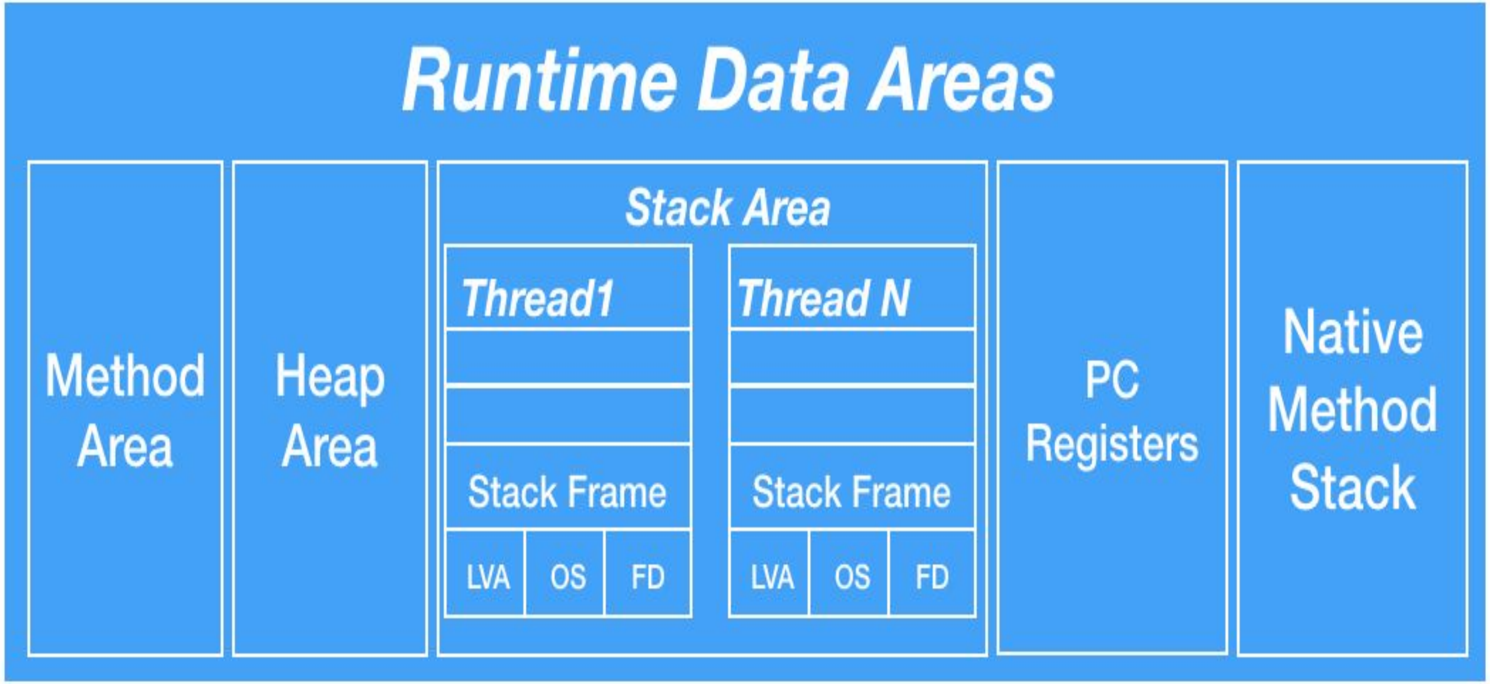
1. Classloader is a subsystem of JVM which is used to load class files.
2. Whenever we run the java program, it is loaded first by the classloader.
3. Classloaders have three main functions:
    a. Loading
    b. Linking
    c. Initialization



Class Loader

1. **Loading:** Loading involves taking the binary representation (bytecode) of a class or interface with a particular name, and generating the original class or interface from that.
2. **Linking:** After a class is loaded into memory, it undergoes the linking process. Linking a class or interface involves combining the different elements and dependencies of the program together.
3. **Initialization:** Initialisation involves executing the initialization method of the class or interface. This can include calling the class's constructor, executing the static block, and assigning values to all the static variables. This is the final stage of class loading.

Runtime Data Area



LVA: Local Variable Array. *OS: Operand Stack.* *FD: Frame Data*

Thank you!

Presented by

Temurmalik Nomozov

(temirmaliknomozov@gmail.com)