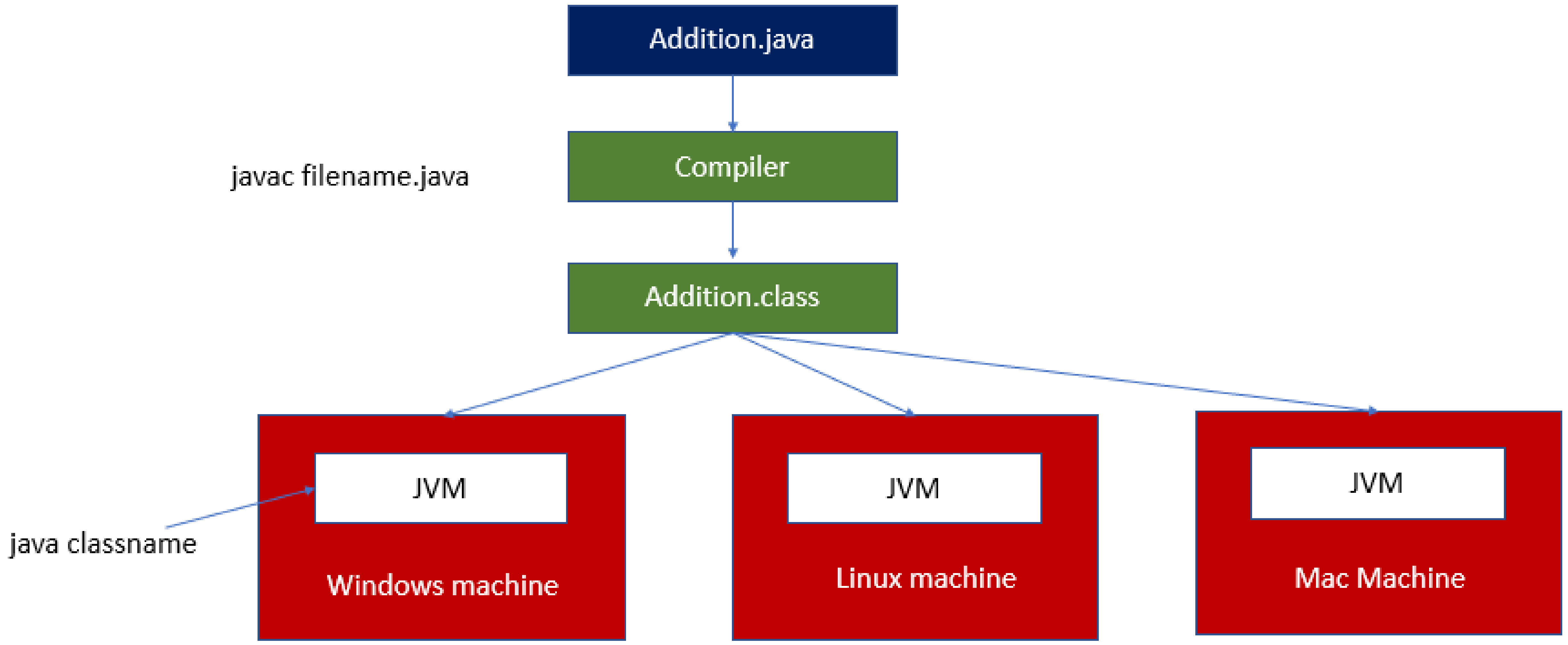
Chapter 7

What is JVM
(Java Virtual Machine)

Other courses in our channel

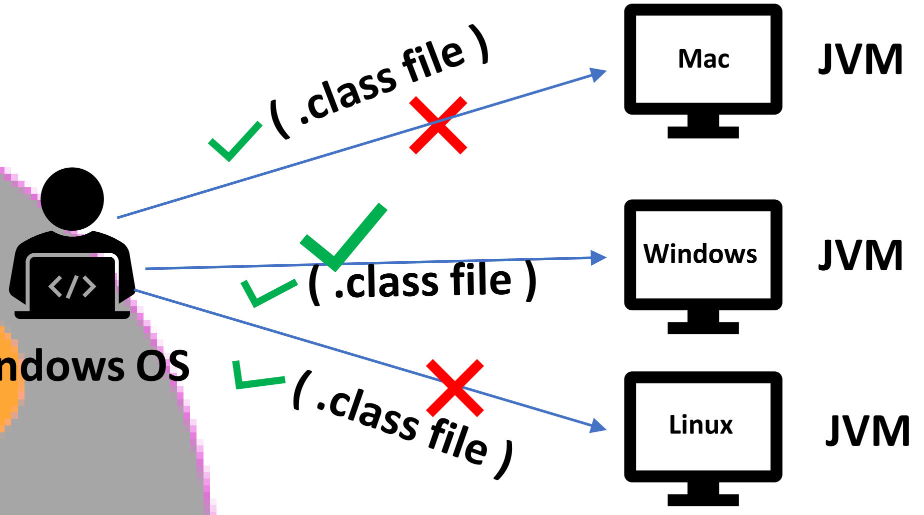
- C 18 Hours Full course: https://youtu.be/3JF7ndGauZk
- Python 11 hours complete course: https://youtu.be/hXN0JBWIya8
- 20 Programs for interview: https://youtu.be/16MFbFib7v8
- What is programming: https://youtu.be/UGfuscUWi-E
- Java in 10 minutes: https://youtu.be/cM82qnE_TPc
- Git Telugu course: https://youtu.be/LIhE7L__E6M
- Git English course: https://youtu.be/aysYDoEH-54
- HTML Full course Telugu: https://youtu.be/6P6yillxZY4

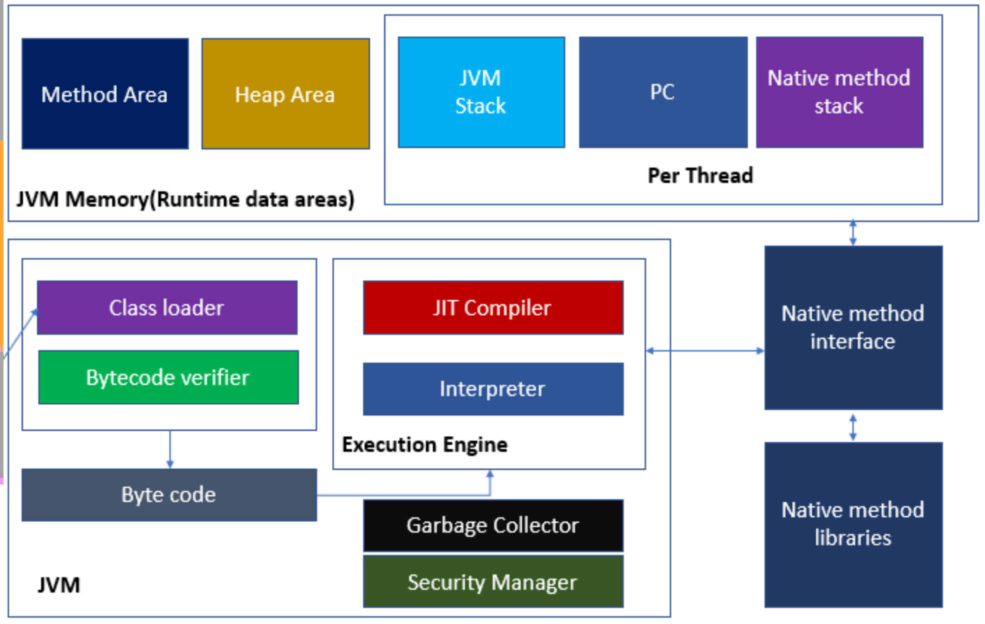
Compilation and execution of Java Program



JVM is a sensation

- Java achieves **platform independence** by **using byte code which get's executed by a JVM**
- Very big topic, need a **detailed discussion for 3 hours**
- It is **not required for beginners**, so we will look at the **overview of the JVM and important interview questions on JVM**

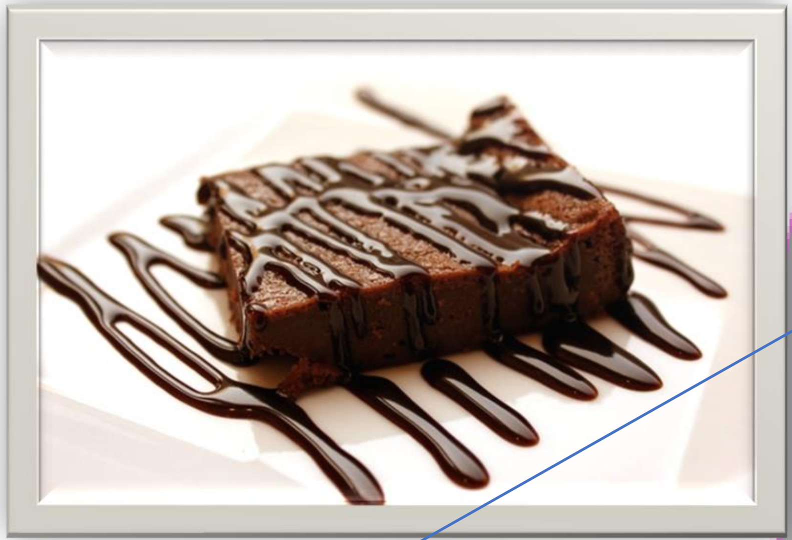




JVM Architecture

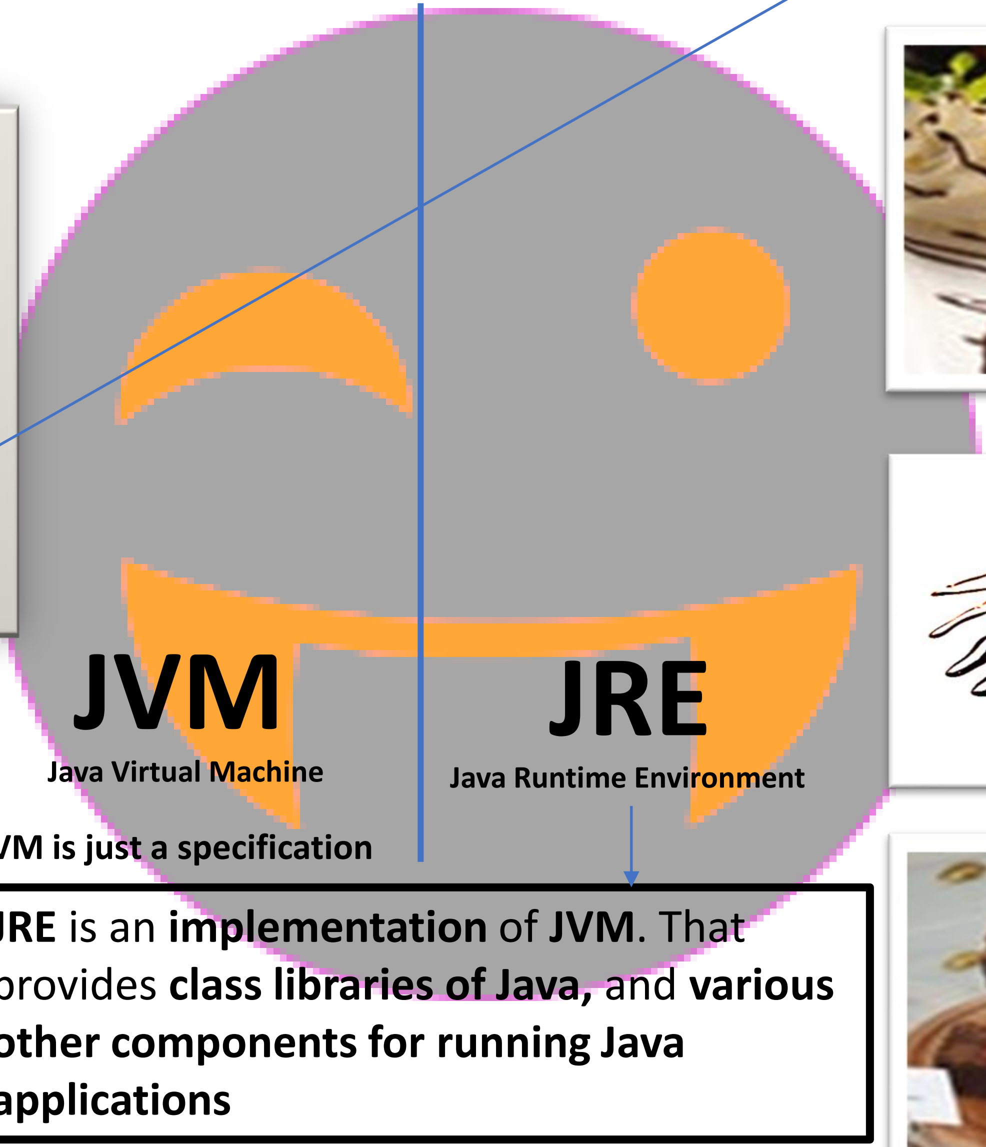
Kirkiri-mango sweet

Implementations



Specifications

1. There should be **bread**
2. There should be **milk**
3. Two slices of **mango**
4. There should be **chocolate**
5. Should be **tasty**
6. Add little **sugar**

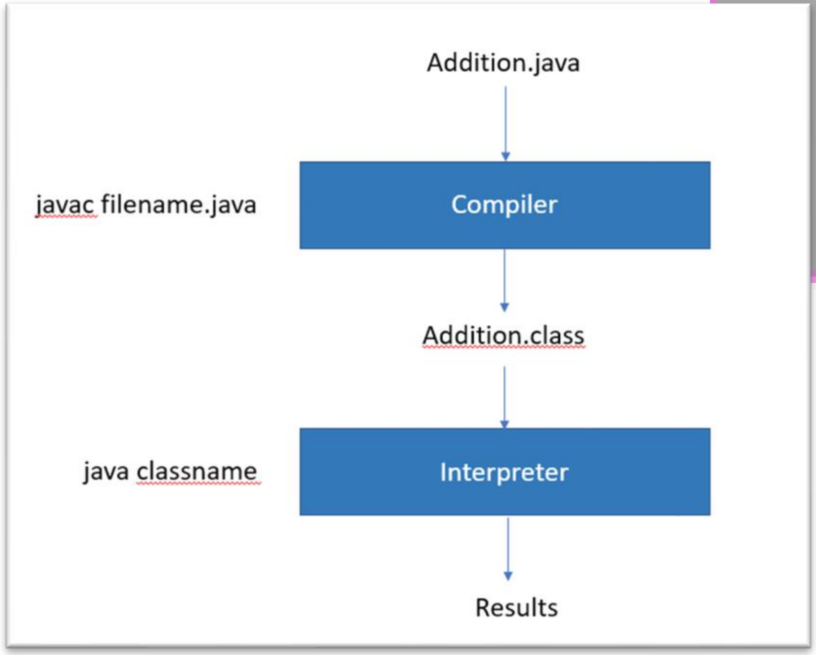


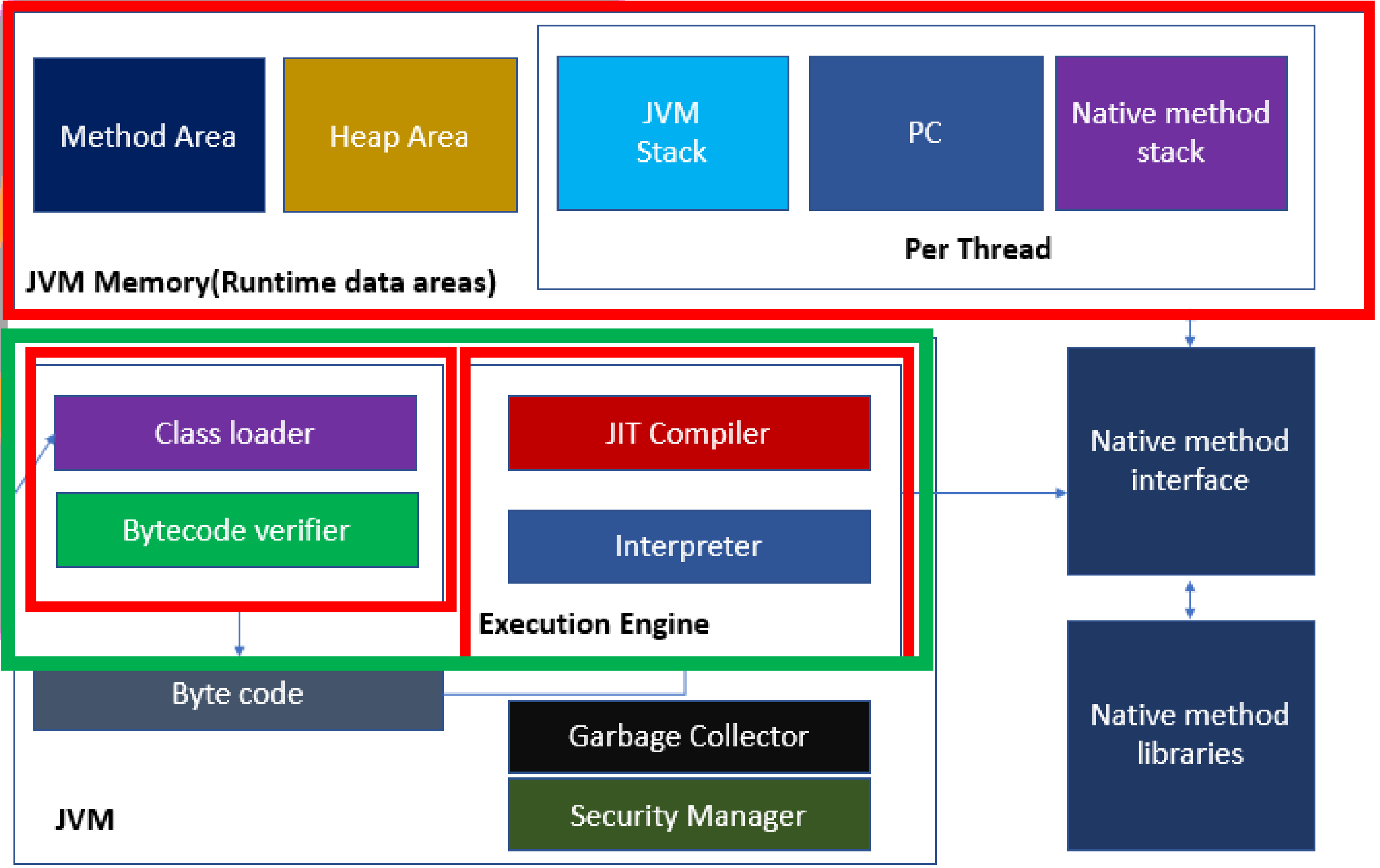
JVM is just a specification

JRE is an **implementation** of **JVM**. That provides **class libraries of Java,** and **various other components for running Java applications**

Architecture of JVM (Very important)

1. **Class loader subsystem**
2. **Runtime data areas**
3. **Execution Engine**

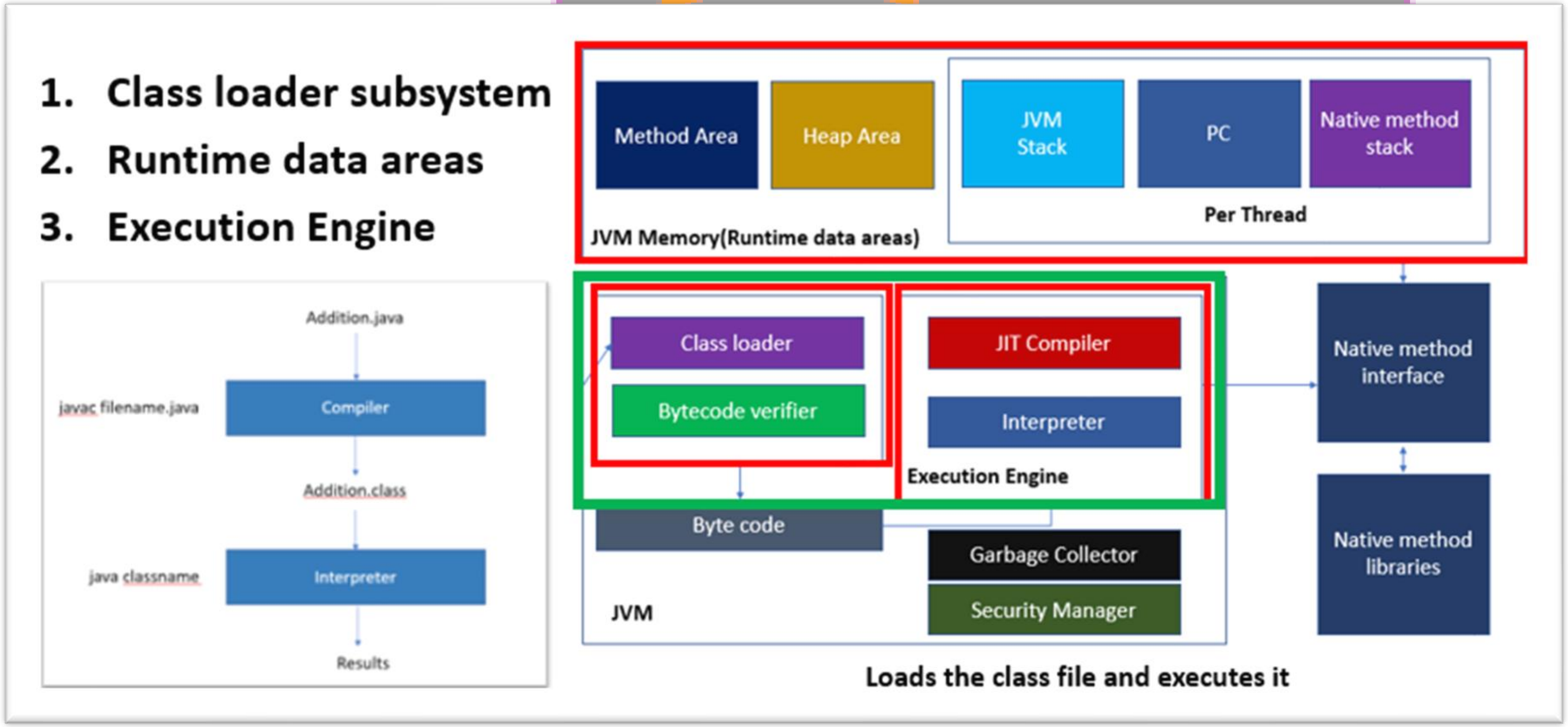




Loads the class file and executes it

Note

- Note that **a JVM instance can run only one java application**
- If you want to run another application at the same time, then you create another instance of the JVM (**java Multiplication**)

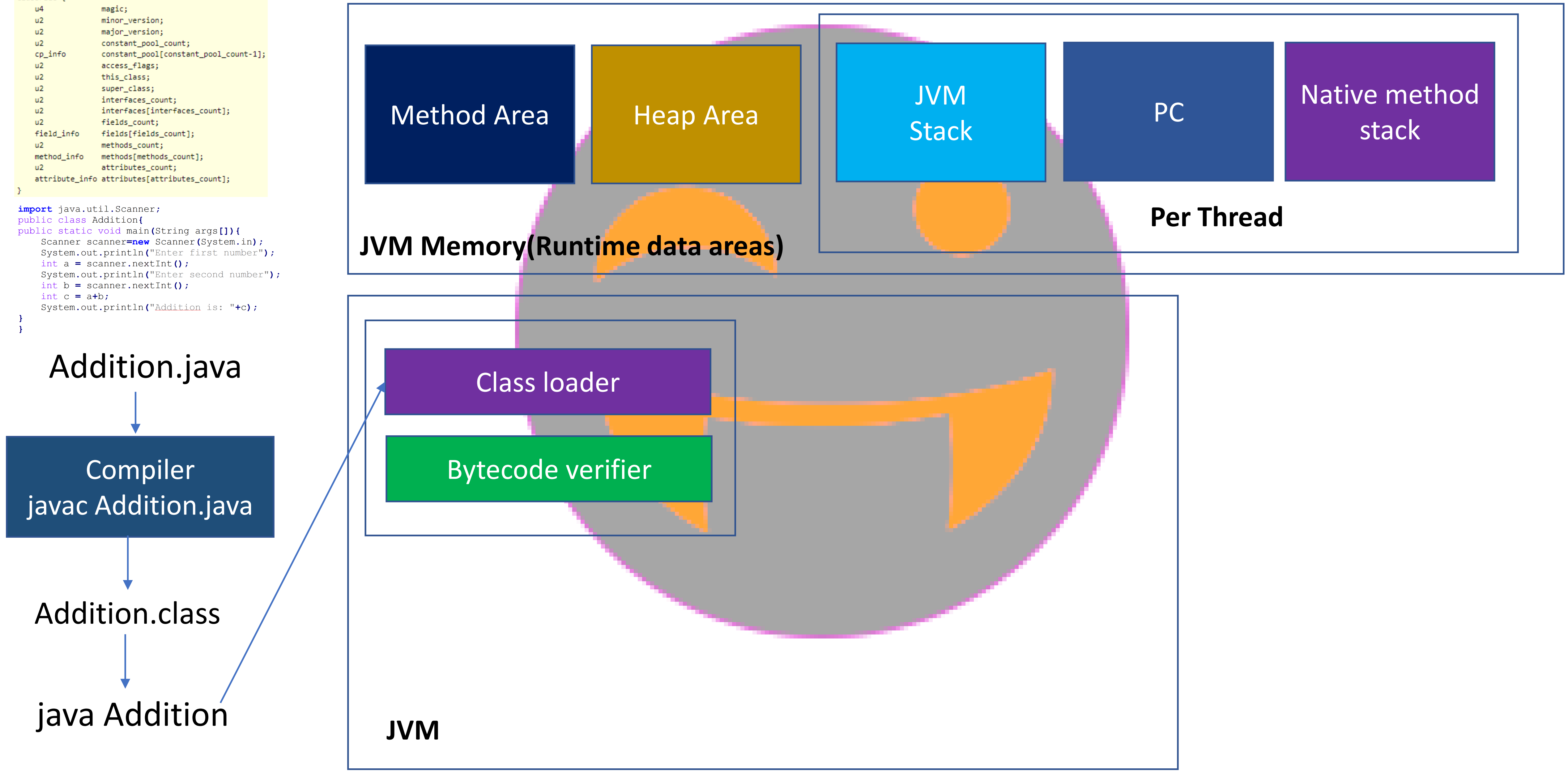


1. **Class loader subsystem**
2. **Runtime data areas**
3. **Execution Engine**

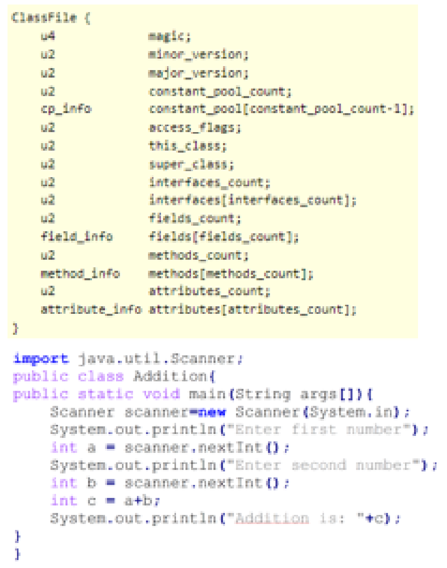
JVM architecture overview

```
ClassFile {
    u4             magic;
    u2             minor_version;
    u2             major_version;
    u2             constant_pool_count;
    cp_info        constant_pool[constant_pool_count-1];
    u2             access_flags;
    u2             this_class;
    u2             super_class;
    u2             interfaces_count;
    u2             interfaces[interfaces_count];
    u2             fields_count;
    field_info     fields[fields_count];
    u2             methods_count;
    method_info    methods[methods_count];
    u2             attributes_count;
    attribute_info attributes[attributes_count];
}
```

```java
import java.util.Scanner;
public class Addition{
public static void main(String args[]){
    Scanner scanner=new Scanner(System.in);
    System.out.println("Enter first number");
    int a = scanner.nextInt();
    System.out.println("Enter second number");
    int b = scanner.nextInt();
    int c = a+b;
    System.out.println("Addition is: "+c);
}
}
```

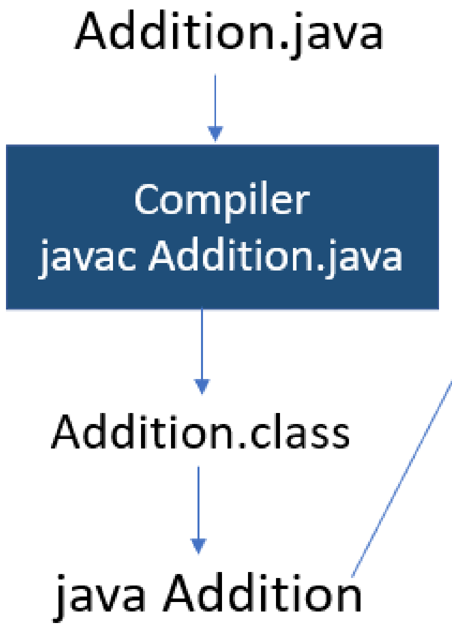


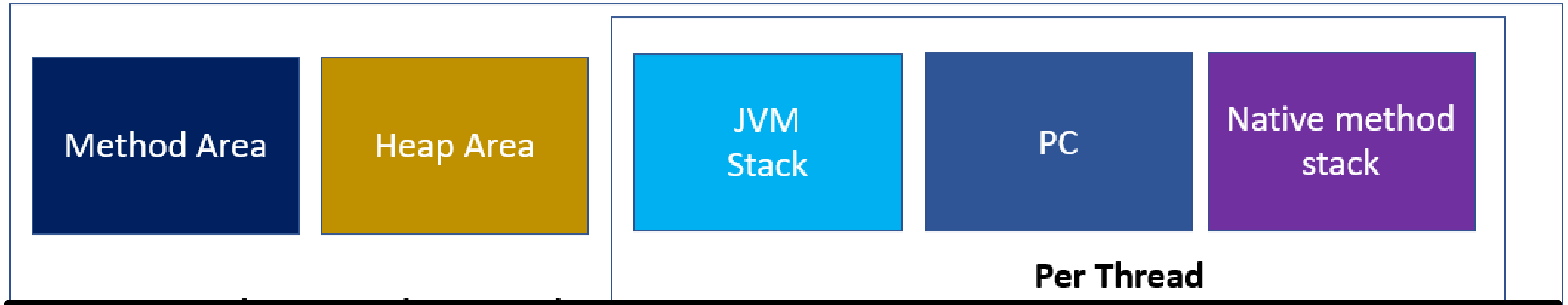
JVM architecture overview



```
ClassFile {
    u4             magic;
    u2             minor_version;
    u2             major_version;
    u2             constant_pool_count;
    cp_info        constant_pool[constant_pool_count-1];
    u2             access_flags;
    u2             this_class;
    u2             super_class;
    u2             interfaces_count;
    u2             interfaces[interfaces_count];
    u2             fields_count;
    field_info     fields[fields_count];
    u2             methods_count;
    method_info    methods[methods_count];
    u2             attributes_count;
    attribute_info attributes[attributes_count];
}
```

```
import java.util.Scanner;
public class Addition{
public static void main(String args[]){
    Scanner scanner=new Scanner(System.in);
    System.out.println("Enter first number");
    int a = scanner.nextInt();
    System.out.println("Enter second number");
    int b = scanner.nextInt();
    int c = a+b;
    System.out.println("Addition is: "+c);
}
}
```



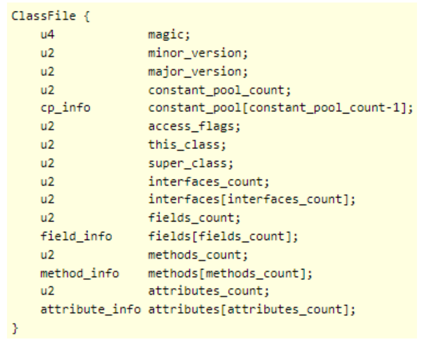


- **Method Area:** Is also called class area. Contains class level information like class name, methods, variables etc. are stored. Along with **that static variables** are stored.

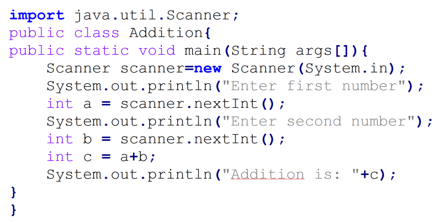
- **Heap Area:** Heap is place where **all objects are stored. Instance variables** are created under object name.

- **Per Thread:**
  - **Stack Area:**
    - For every thread, JVM creates **one run-time stack** which is stored here.
    - Every block of this stack is called a stack frame or activation record, which stores methods calls.
    - All **local variables** of that method are stored in their corresponding **stack frame**.
  - **Program counter registers (PC Registers):**
    - Stores **address of currently executed instruction** and increments by 1 to point to the next instruction to be executed
  - **Native method stack:** Stores native method information

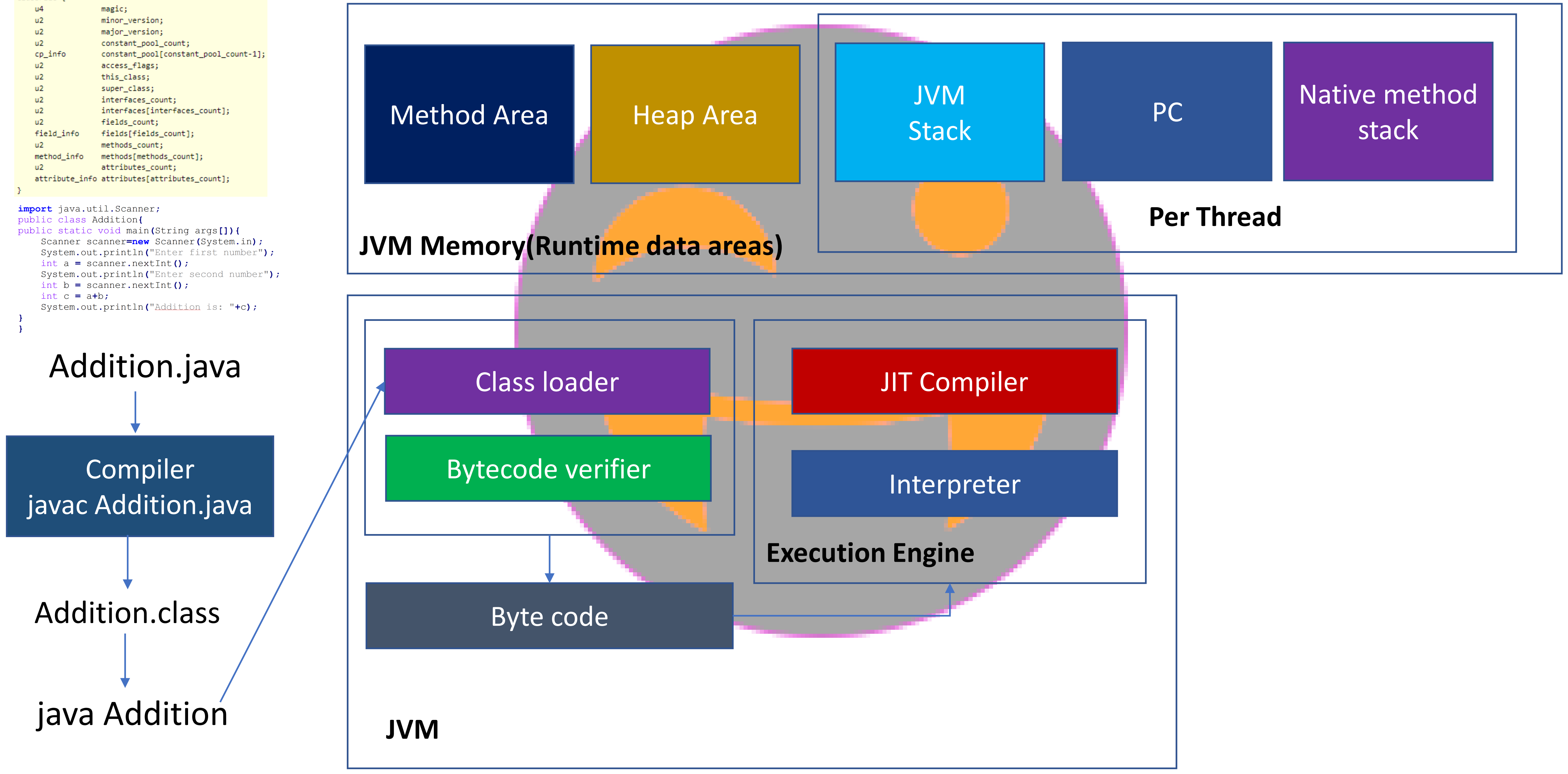
JVM architecture overview



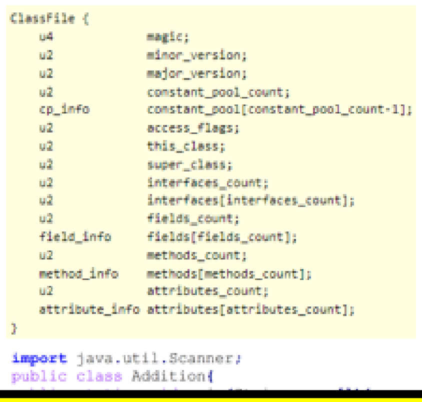
```
ClassFile {
    u4             magic;
    u2             minor_version;
    u2             major_version;
    u2             constant_pool_count;
    cp_info        constant_pool[constant_pool_count-1];
    u2             access_flags;
    u2             this_class;
    u2             super_class;
    u2             interfaces_count;
    u2             interfaces[interfaces_count];
    u2             fields_count;
    field_info     fields[fields_count];
    u2             methods_count;
    method_info    methods[methods_count];
    u2             attributes_count;
    attribute_info attributes[attributes_count];
}
```



```
import java.util.Scanner;
public class Addition{
public static void main(String args[]){
    Scanner scanner=new Scanner(System.in);
    System.out.println("Enter first number");
    int a = scanner.nextInt();
    System.out.println("Enter second number");
    int b = scanner.nextInt();
    int c = a+b;
    System.out.println("Addition is: "+c);
}
}
```

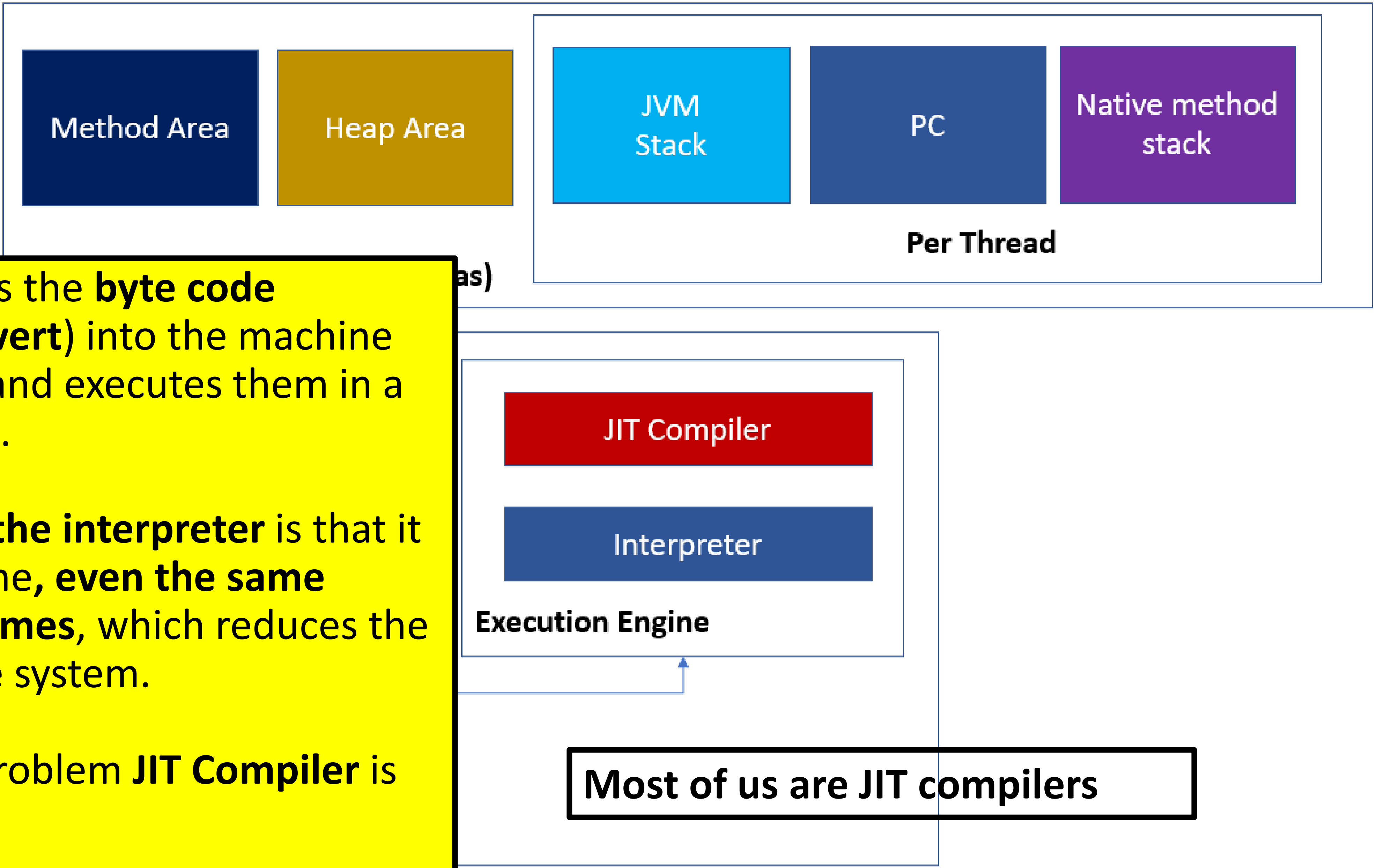


JVM architecture overview



```
ClassFile {
    u4             magic;
    u2             minor_version;
    u2             major_version;
    u2             constant_pool_count;
    cp_info        constant_pool[constant_pool_count-1];
    u2             access_flags;
    u2             this_class;
    u2             super_class;
    u2             interfaces_count;
    u2             interfaces[interfaces_count];
    u2             fields_count;
    field_info     fields[fields_count];
    u2             methods_count;
    method_info    methods[methods_count];
    u2             attributes_count;
    attribute_info attributes[attributes_count];
}
```

```
import java.util.Scanner;
public class Addition{
```



- **Interpreter:** It reads the **byte code** and interprets(**convert**) into the machine code(native code) and executes them in a **sequential manner**.

- The **problem with the interpreter** is that it interprets every time**, even the same method multiple times**, which reduces the performance of the system.

- To overcome this problem **JIT Compiler** is introduced

Most of us are JIT compilers

Repeatedly used items will be fixed in our mind

Ramu – 5 times
Somu – 7 times
Jelly – 4 times
JIT – 13 times

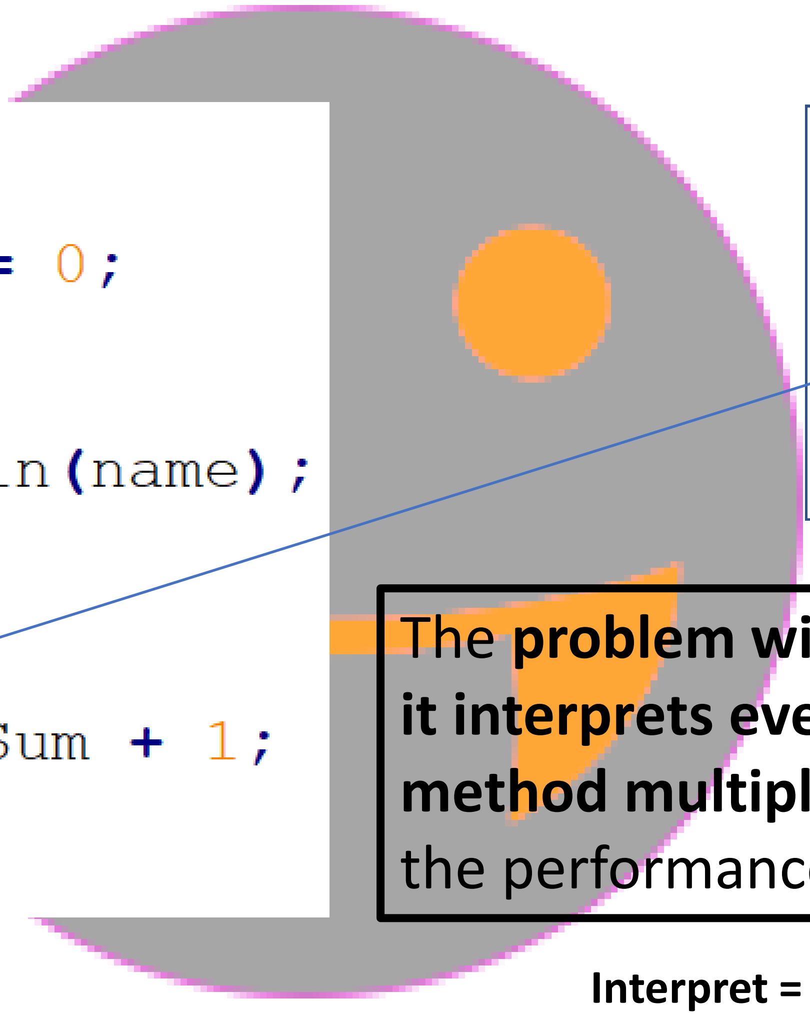
Threshold value

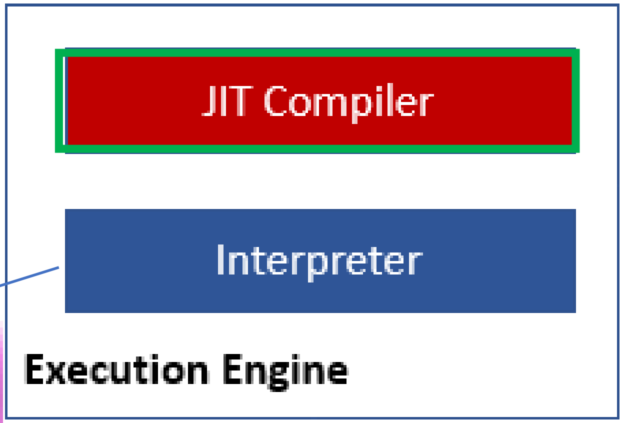
JIT Compiler

Method call count > 1000 = save the machine code for future calls

Threshold value = 1000

```
class JITTest{
        String name="JIT";
        static int totalSum = 0;

        void display(){
                System.out.println(name);
        }

        void calculate(){
                totalSum = totalSum + 1;
        }
}
```

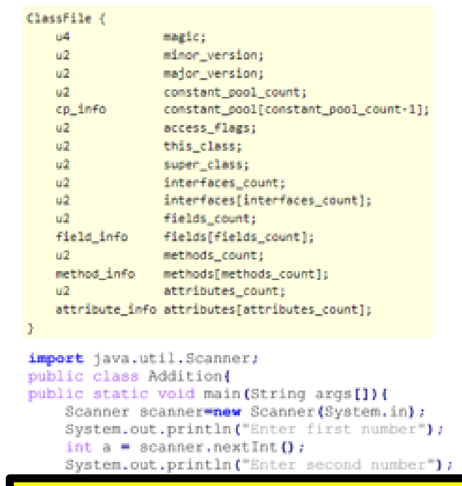




The **problem with the interpreter** is that **it interprets every time, even the same method multiple times**, which reduces the performance of the system.

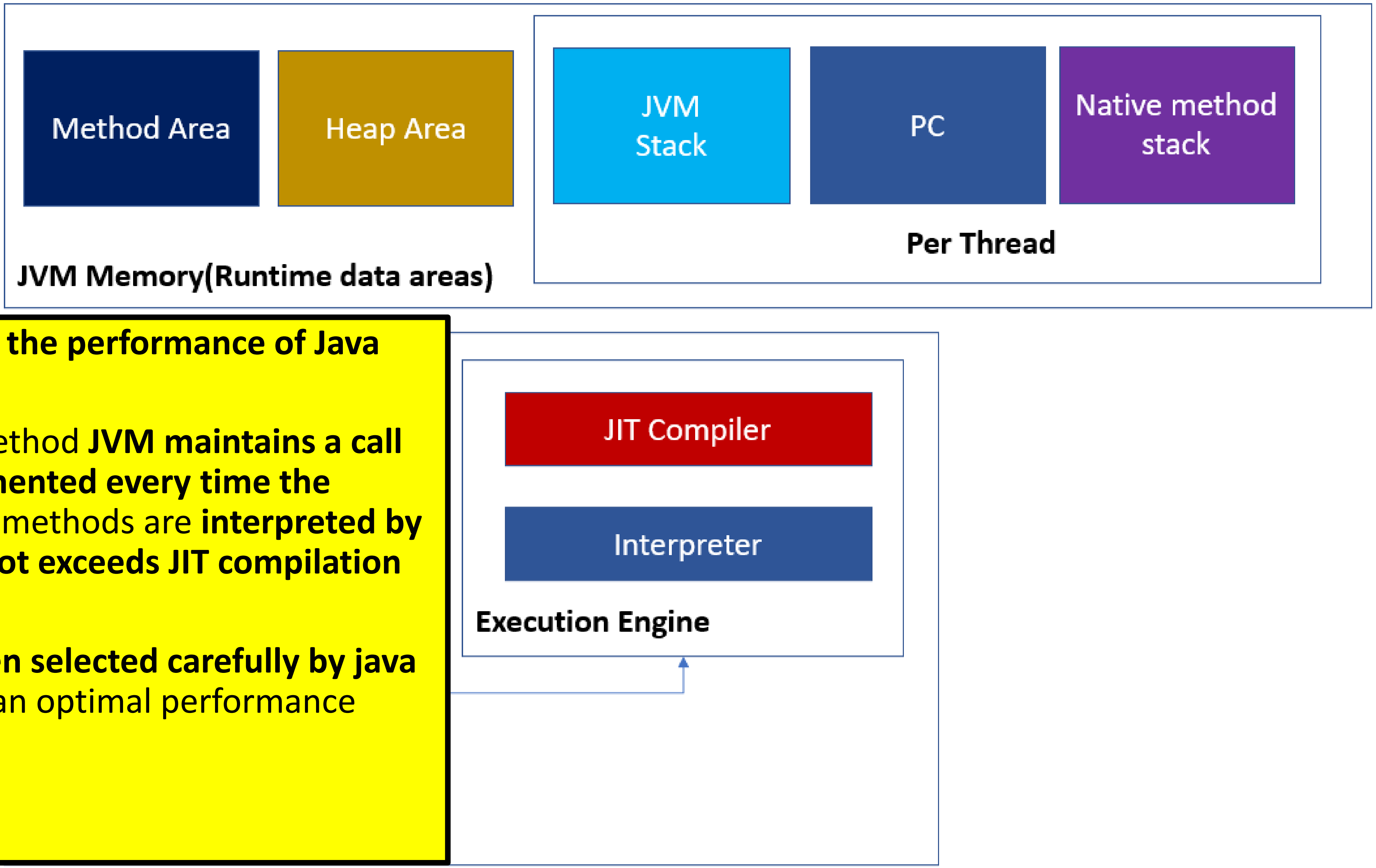
Interpret = converting byte code to machine code

JVM architecture overview



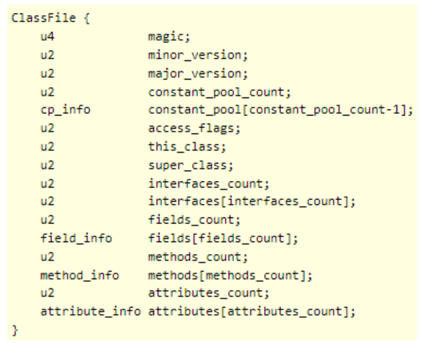
```
ClassFile {
    u4             magic;
    u2             minor_version;
    u2             major_version;
    u2             constant_pool_count;
    cp_info        constant_pool[constant_pool_count-1];
    u2             access_flags;
    u2             this_class;
    u2             super_class;
    u2             interfaces_count;
    u2             interfaces[interfaces_count];
    u2             fields_count;
    field_info     fields[fields_count];
    u2             methods_count;
    method_info    methods[methods_count];
    u2             attributes_count;
    attribute_info attributes[attributes_count];
}
```

```
import java.util.Scanner;
public class Addition{
public static void main(String args[]){
    Scanner scanner=new Scanner(System.in);
    System.out.println("Enter first number");
    int a = scanner.nextInt();
    System.out.println("Enter second number");
```



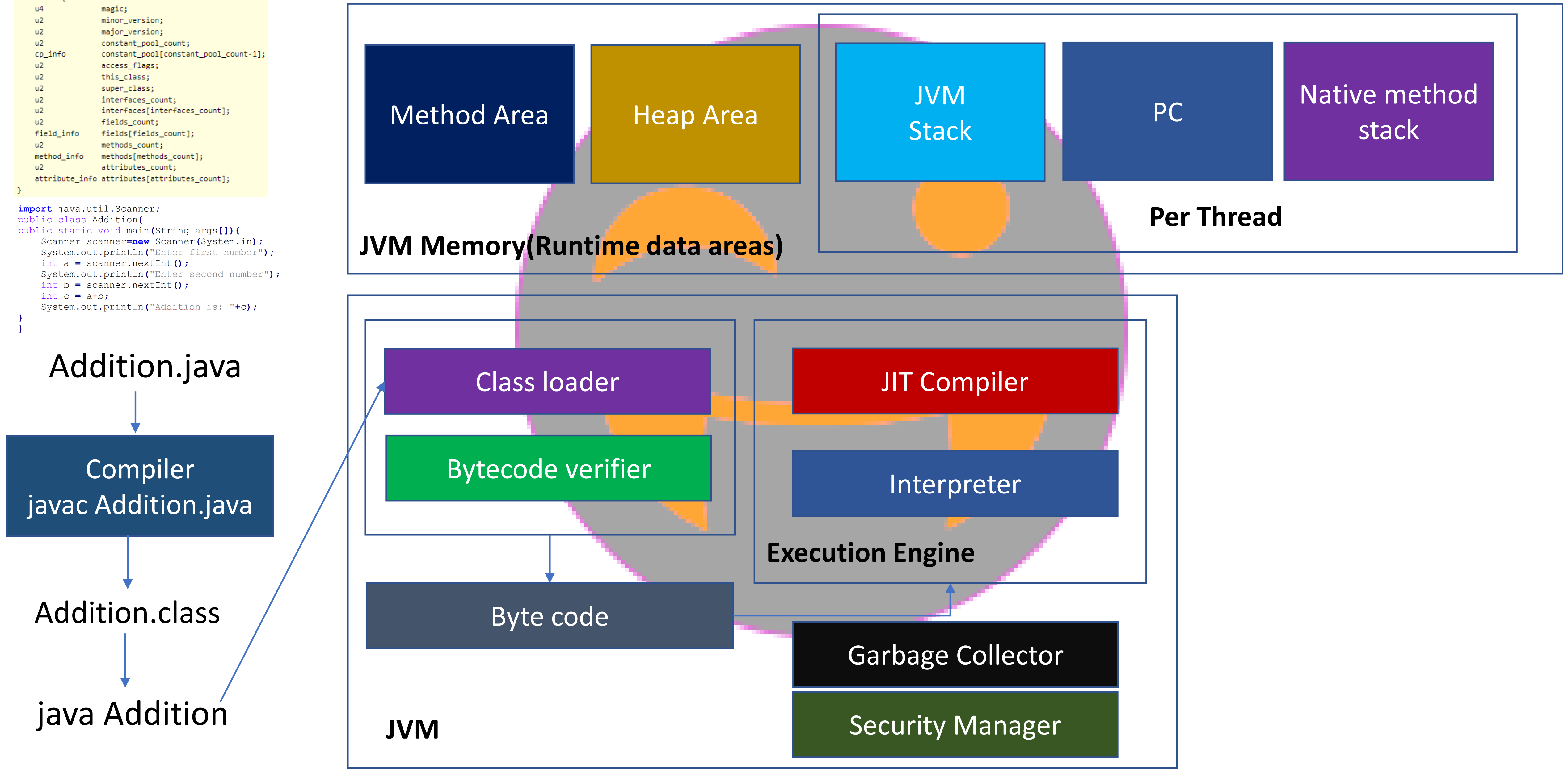
- JIT compiler **improves the performance of Java applications**
- For each and every method **JVM maintains a call count**, which is **incremented every time the method is called**. The methods are **interpreted by JVM until call count not exceeds JIT compilation threshold**
- The threshold has **been selected carefully by java developers** to obtain an optimal performance

JVM architecture overview



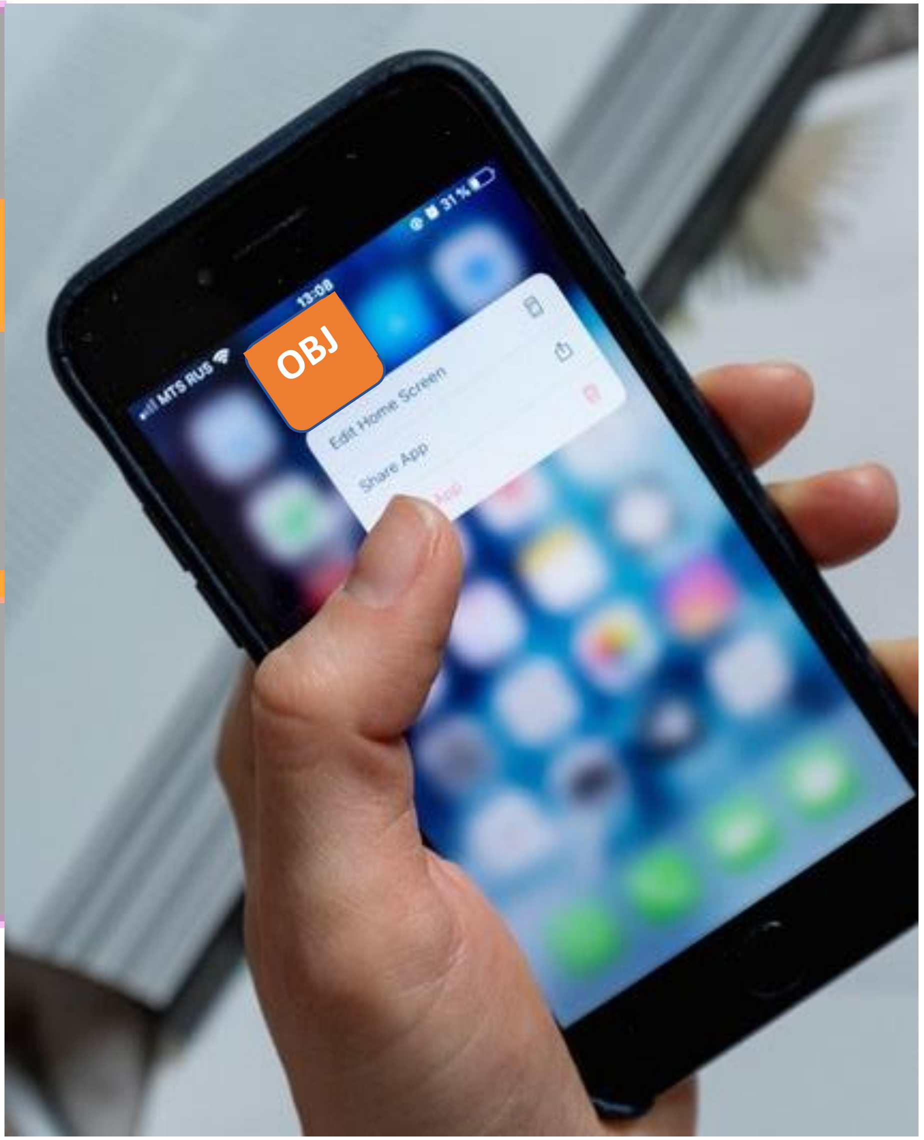
```
ClassFile {
    u4             magic;
    u2             minor_version;
    u2             major_version;
    u2             constant_pool_count;
    cp_info        constant_pool[constant_pool_count-1];
    u2             access_flags;
    u2             this_class;
    u2             super_class;
    u2             interfaces_count;
    u2             interfaces[interfaces_count];
    u2             fields_count;
    field_info     fields[fields_count];
    u2             methods_count;
    method_info    methods[methods_count];
    u2             attributes_count;
    attribute_info attributes[attributes_count];
}
```

```java
import java.util.Scanner;
public class Addition{
public static void main(String args[]){
    Scanner scanner=new Scanner(System.in);
    System.out.println("Enter first number");
    int a = scanner.nextInt();
    System.out.println("Enter second number");
    int b = scanner.nextInt();
    int c = a+b;
    System.out.println("Addition is: "+c);
}
}
```



Garbage collector

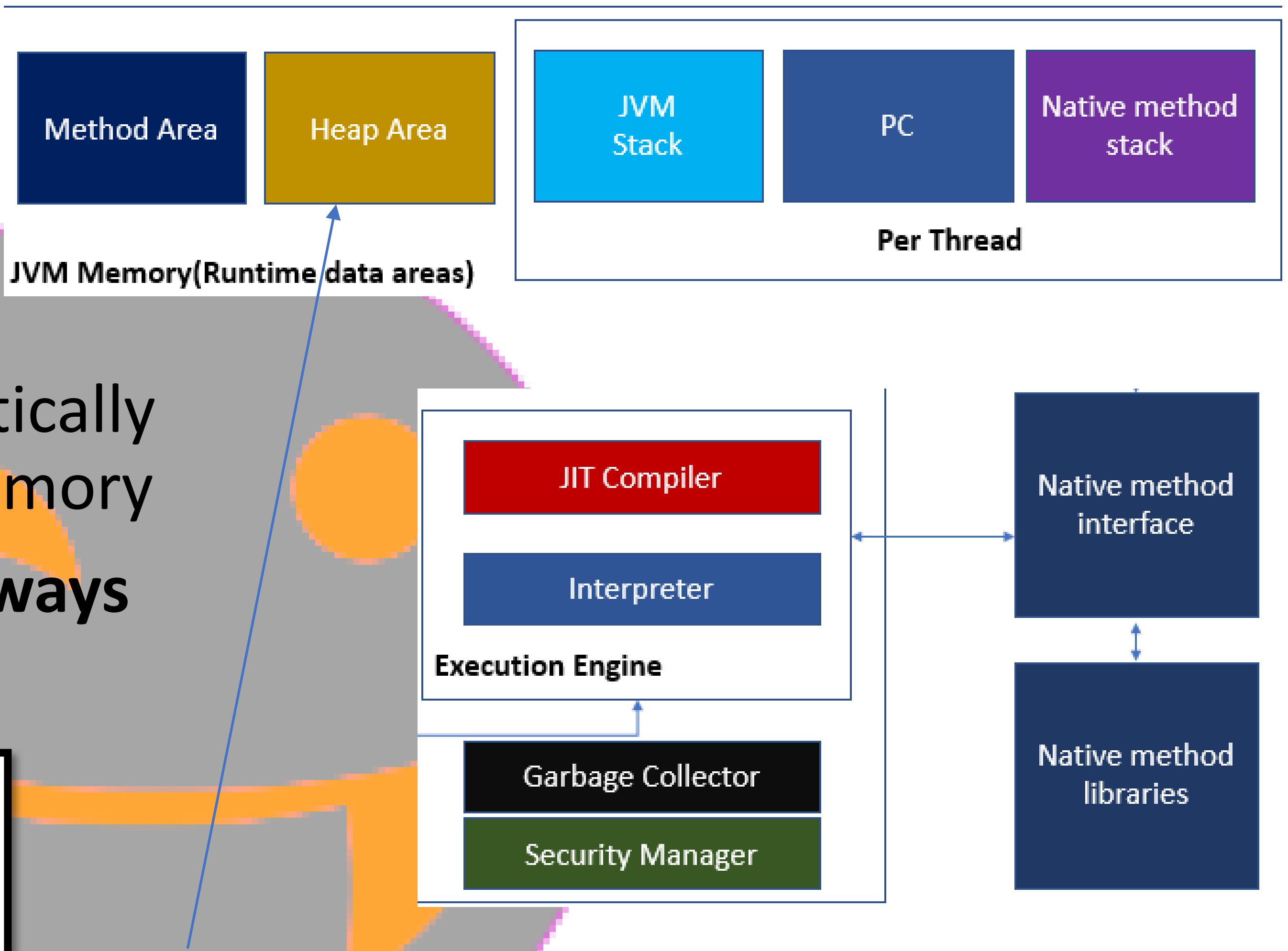
RAM = 32 GB/ 64 GB



Garbage collector

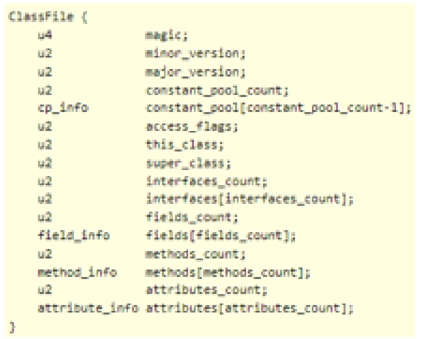
- **Manages the memory** automatically by cleaning unwanted heap memory
- It is a **daemon** thread **which always runs in the background**

```
class JITTest{
    String name="JIT";
    static int totalSum = 0;

    void display(){
        System.out.println(name);
    }

    void calculate(){
        totalSum = totalSum + 1;
    }
}
```

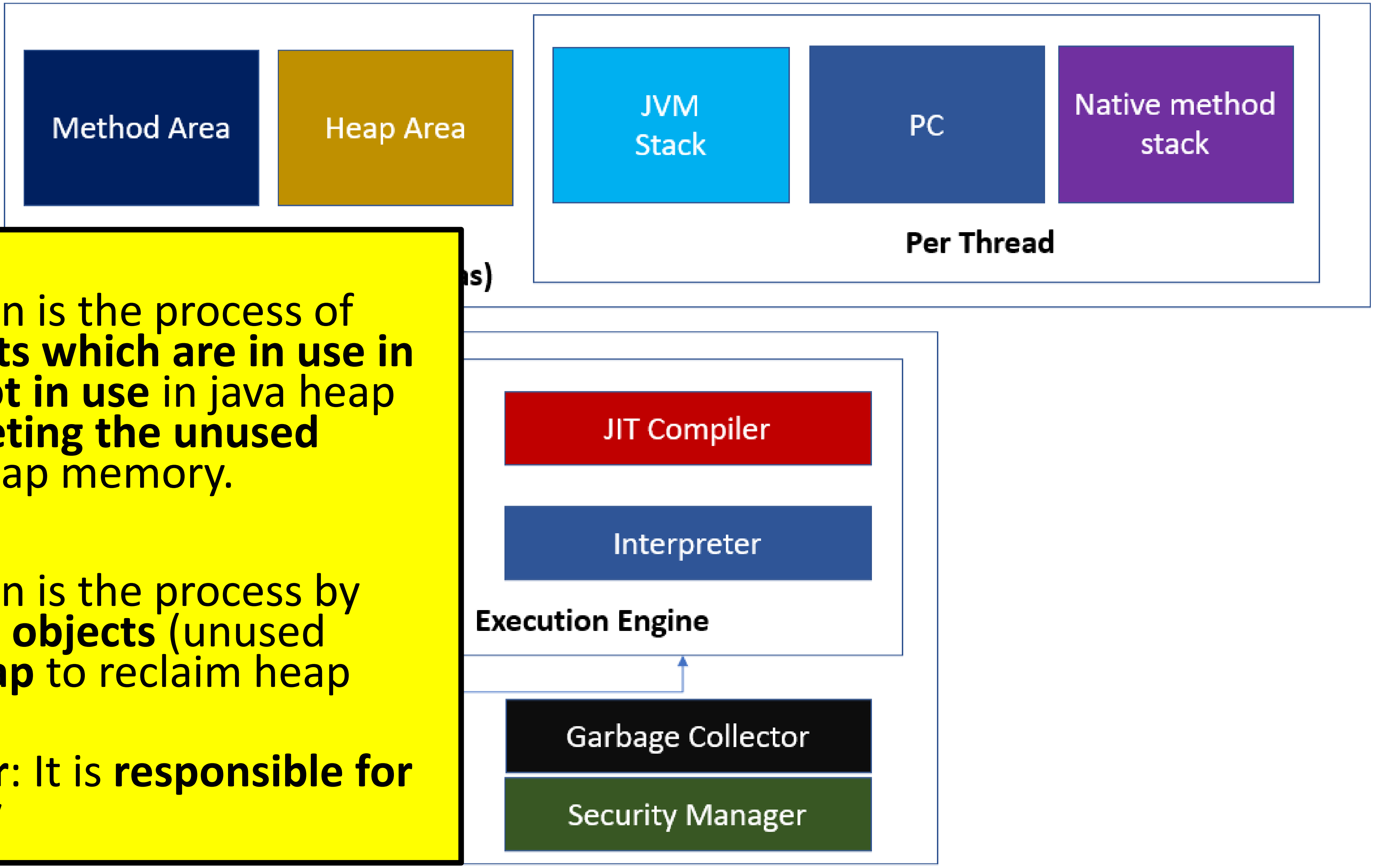


```
JITTest t1 = new JITTest();
JITTest t2 = new JITTest();
JITTest t3 = new JITTest();
JITTest t4 = new JITTest();
JITTest t5 = new JITTest();
JITTest t6 = new JITTest();
```

JVM architecture overview



```
ClassFile {
    u4             magic;
    u2             minor_version;
    u2             major_version;
    u2             constant_pool_count;
    cp_info        constant_pool[constant_pool_count-1];
    u2             access_flags;
    u2             this_class;
    u2             super_class;
    u2             interfaces_count;
    u2             interfaces[interfaces_count];
    u2             fields_count;
    field_info     fields[fields_count];
    u2             methods_count;
    method_info    methods[methods_count];
    u2             attributes_count;
    attribute_info attributes[attributes_count];
}
```

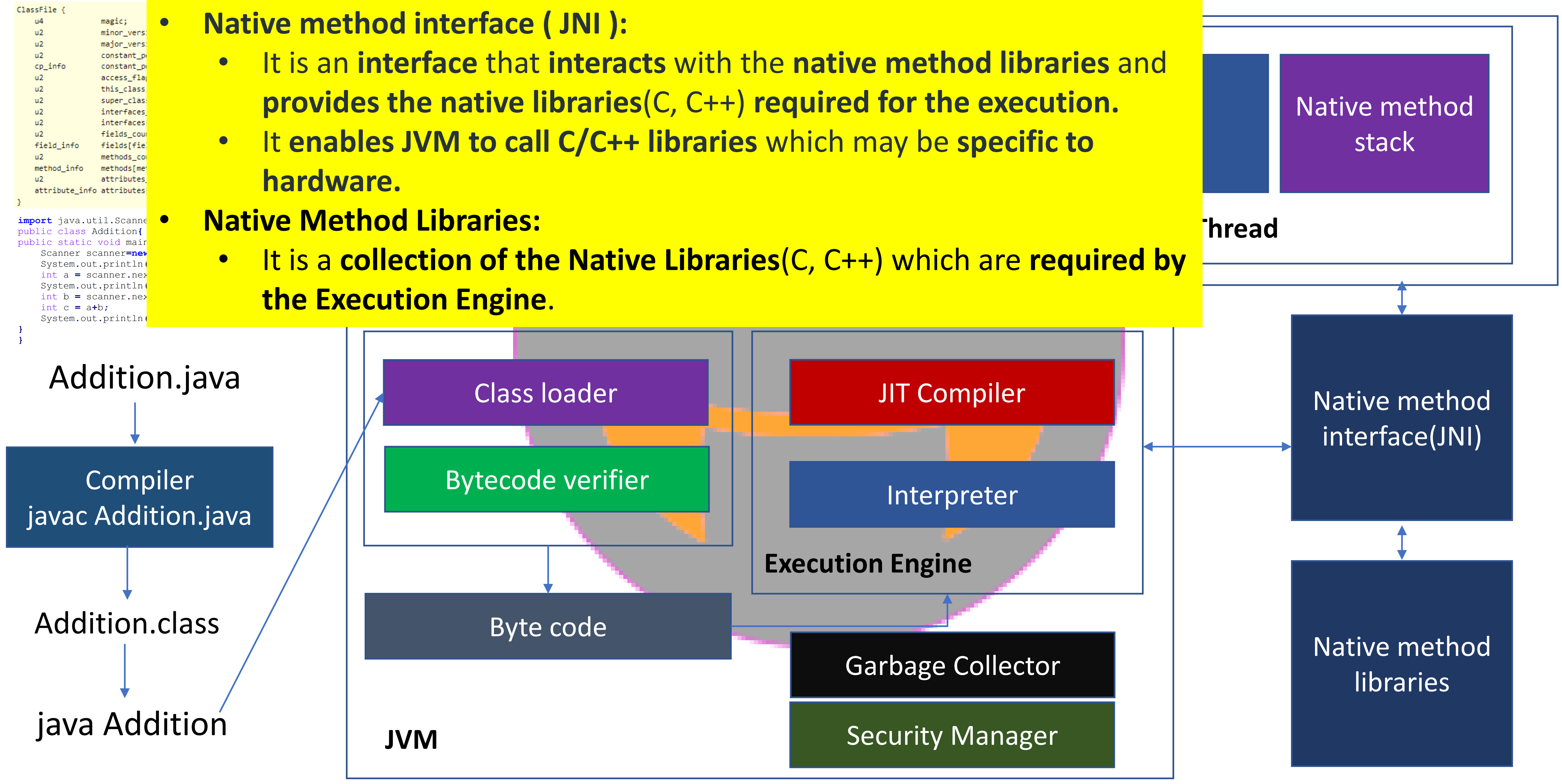


- Garbage collection is the process of **identifying objects which are in use in and which are not in use** in java heap memory and **deleting the unused objects** in java heap memory.

- Garbage collection is the process by which **JVM clears objects** (unused objects) **from heap** to reclaim heap space
- **Security manager**: It is **responsible for ensuring security**

JVM architecture overview

- **Native method interface (JNI):**
  - It is an **interface** that **interacts** with the **native method libraries** and **provides the native libraries**(C, C++) **required for the execution.**
  - It **enables JVM to call C/C++ libraries** which may be **specific to hardware.**
- **Native Method Libraries:**
  - It is a **collection of the Native Libraries**(C, C++) which are **required by the Execution Engine**.



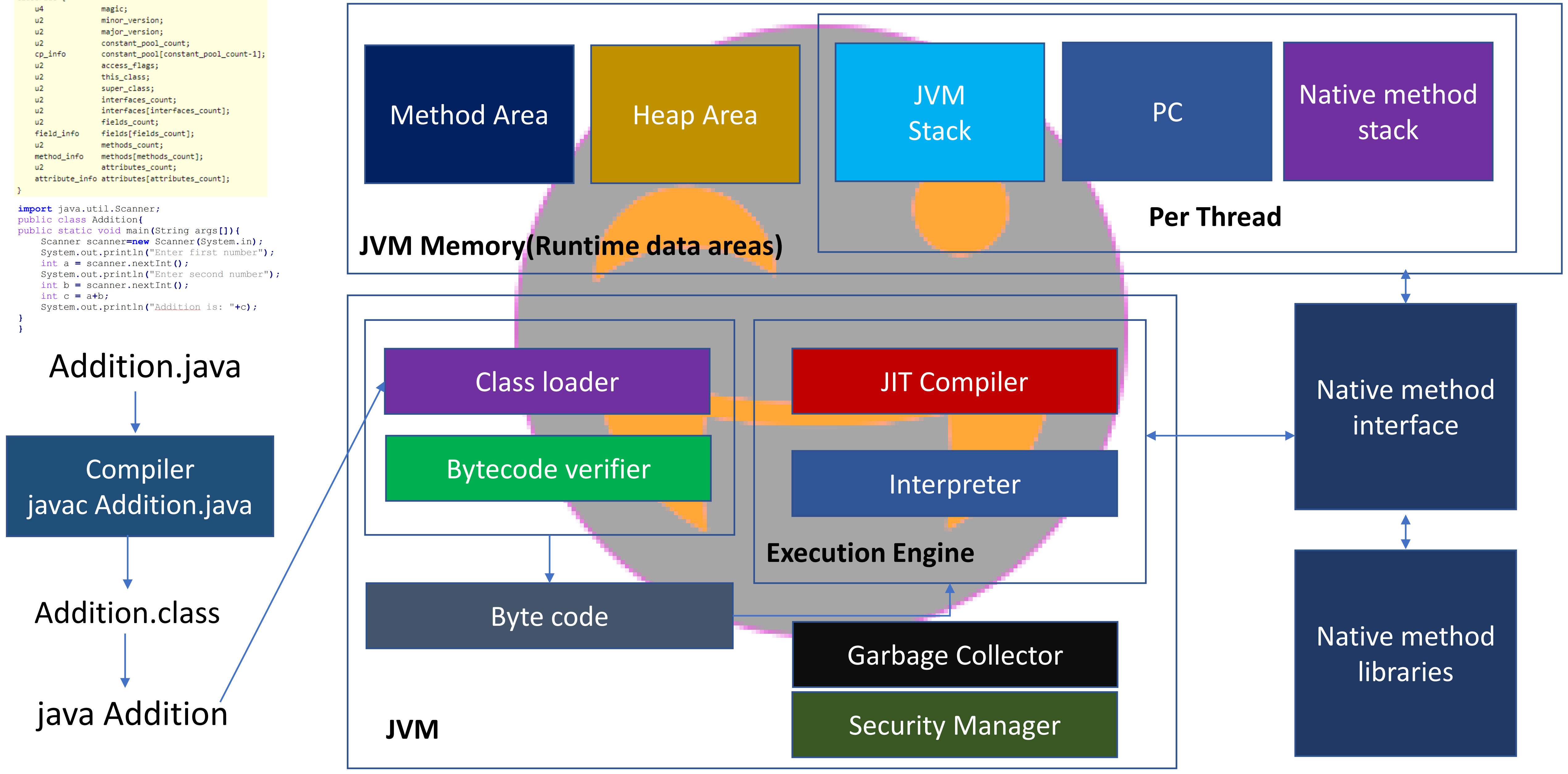
```
ClassFile {
    u4             magic;
    u2             minor_vers
    u2             major_vers
    u2             constant_p
    cp_info        constant_p
    u2             access_fla
    u2             this_class
    u2             super_clas
    u2             interfaces
    u2             interfaces
    u2             fields_cou
    field_info     fields[fie
    u2             methods_co
    method_info    methods[me
    u2             attributes
    attribute_info attributes
}
```

```
import java.util.Scanne
public class Addition{
public static void main
    Scanner scanner=new
    System.out.println(
    int a = scanner.nex
    System.out.println(
    int b = scanner.nex
    int c = a+b;
    System.out.println(
}
}
```

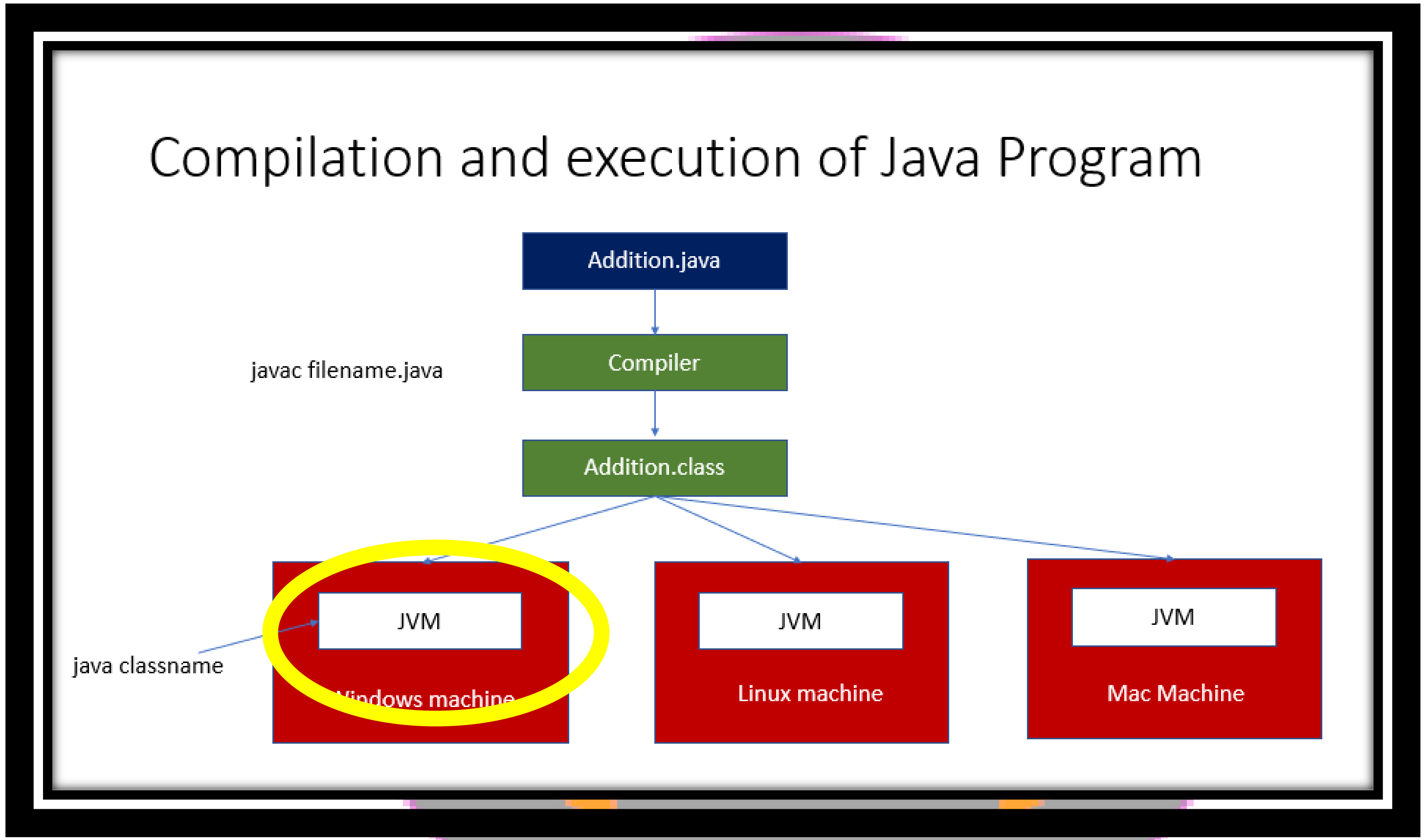
JVM architecture overview

```
ClassFile {
    u4             magic;
    u2             minor_version;
    u2             major_version;
    u2             constant_pool_count;
    cp_info        constant_pool[constant_pool_count-1];
    u2             access_flags;
    u2             this_class;
    u2             super_class;
    u2             interfaces_count;
    u2             interfaces[interfaces_count];
    u2             fields_count;
    field_info     fields[fields_count];
    u2             methods_count;
    method_info    methods[methods_count];
    u2             attributes_count;
    attribute_info attributes[attributes_count];
}
```

```java
import java.util.Scanner;
public class Addition{
public static void main(String args[]){
    Scanner scanner=new Scanner(System.in);
    System.out.println("Enter first number");
    int a = scanner.nextInt();
    System.out.println("Enter second number");
    int b = scanner.nextInt();
    int c = a+b;
    System.out.println("Addition is: "+c);
}
}
```



NOTE



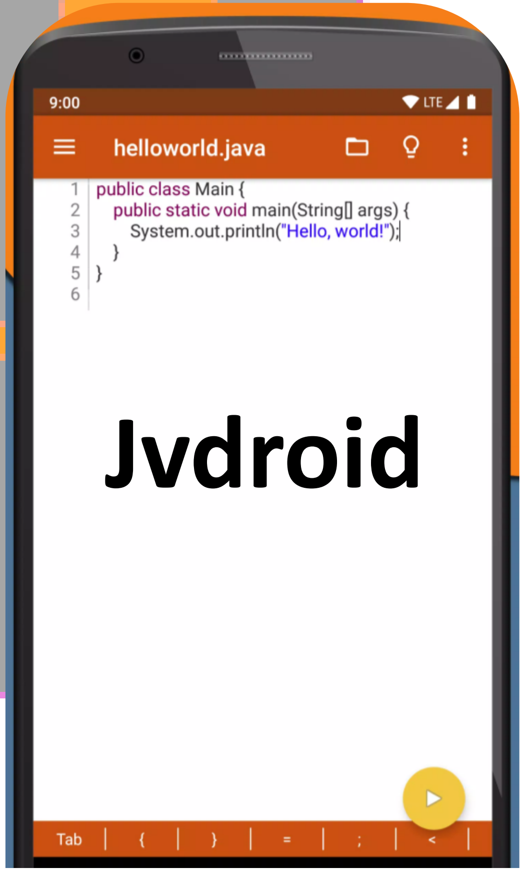
- JVM is an **important topic**
- Will discuss **interview** questions after **few chapters**

```
import java.util.Scanner;
public class Addition{
public static void main(String args[]){
    Scanner scanner=new Scanner(System.in);
    System.out.println("Enter first number");
    int a = scanner.nextInt();
    System.out.println("Enter second number");
    int b = scanner.nextInt();
    int c = a+b;
    System.out.println("Addition is: "+c);
}
}
```

JDK

(Java Development Kit)

Mobile/Tablet



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

Jvdroid

What next?

Install JAVA (JDK)

చిన్న బ్రేక్ చిటికలో వచ్చేస్తా

Other courses in our channel

☞ C 18 Hours Full course: https://youtu.be/3JF7ndGauZk

☞ Python 11 hours complete course: https://youtu.be/hXN0JBWIya8

☞ 20 Programs for interview: https://youtu.be/16MFbFib7v8

☞ What is programming: https://youtu.be/UGfuscUWi-E

☞ Java in 10 minutes: https://youtu.be/cM82qnE_TPc

☞ Git Telugu course: https://youtu.be/LIhE7L__E6M

☞ Git English course: https://youtu.be/aysYDoEH-54

☞ HTML Full course Telugu: https://youtu.be/6P6yillxZY4