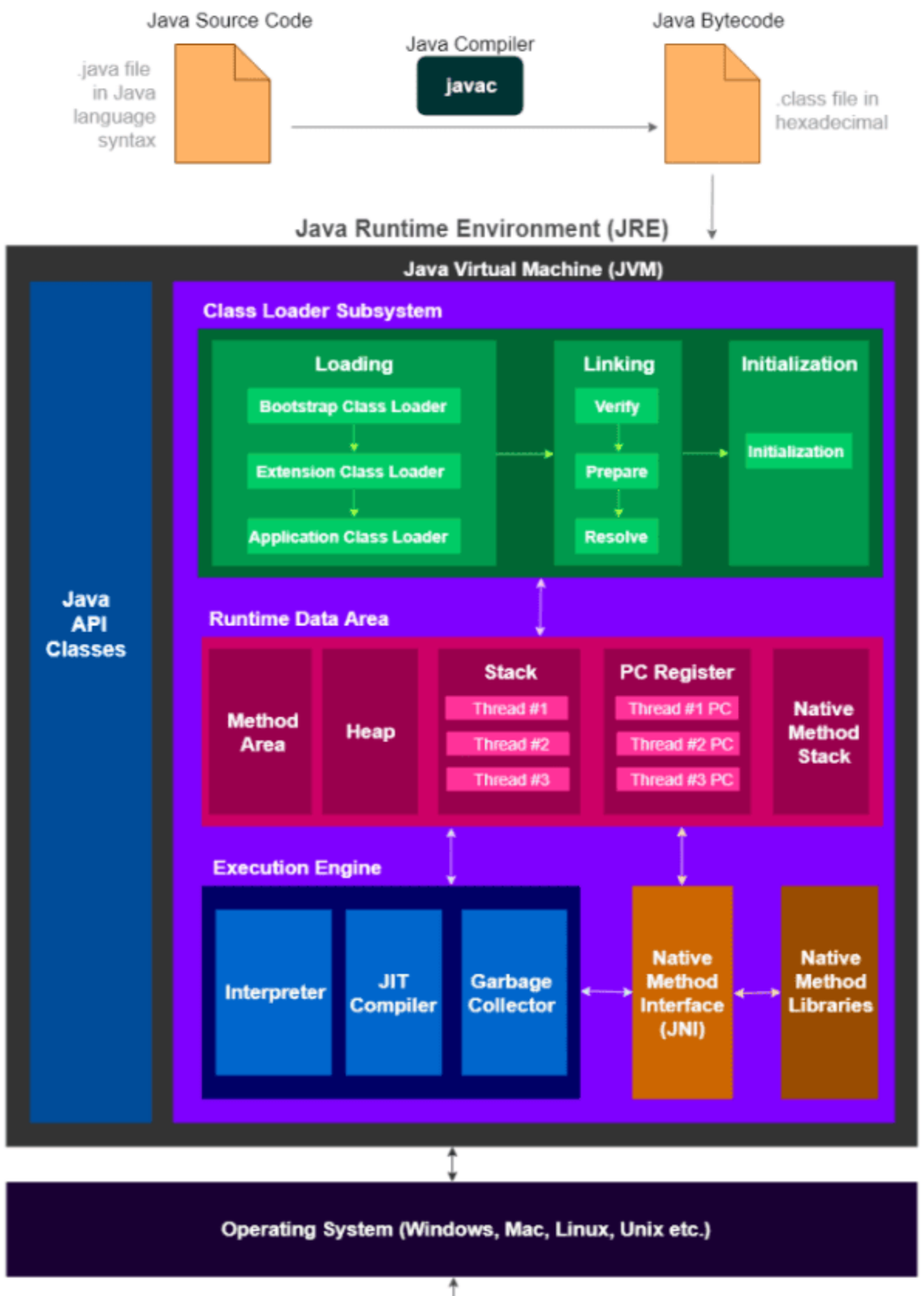
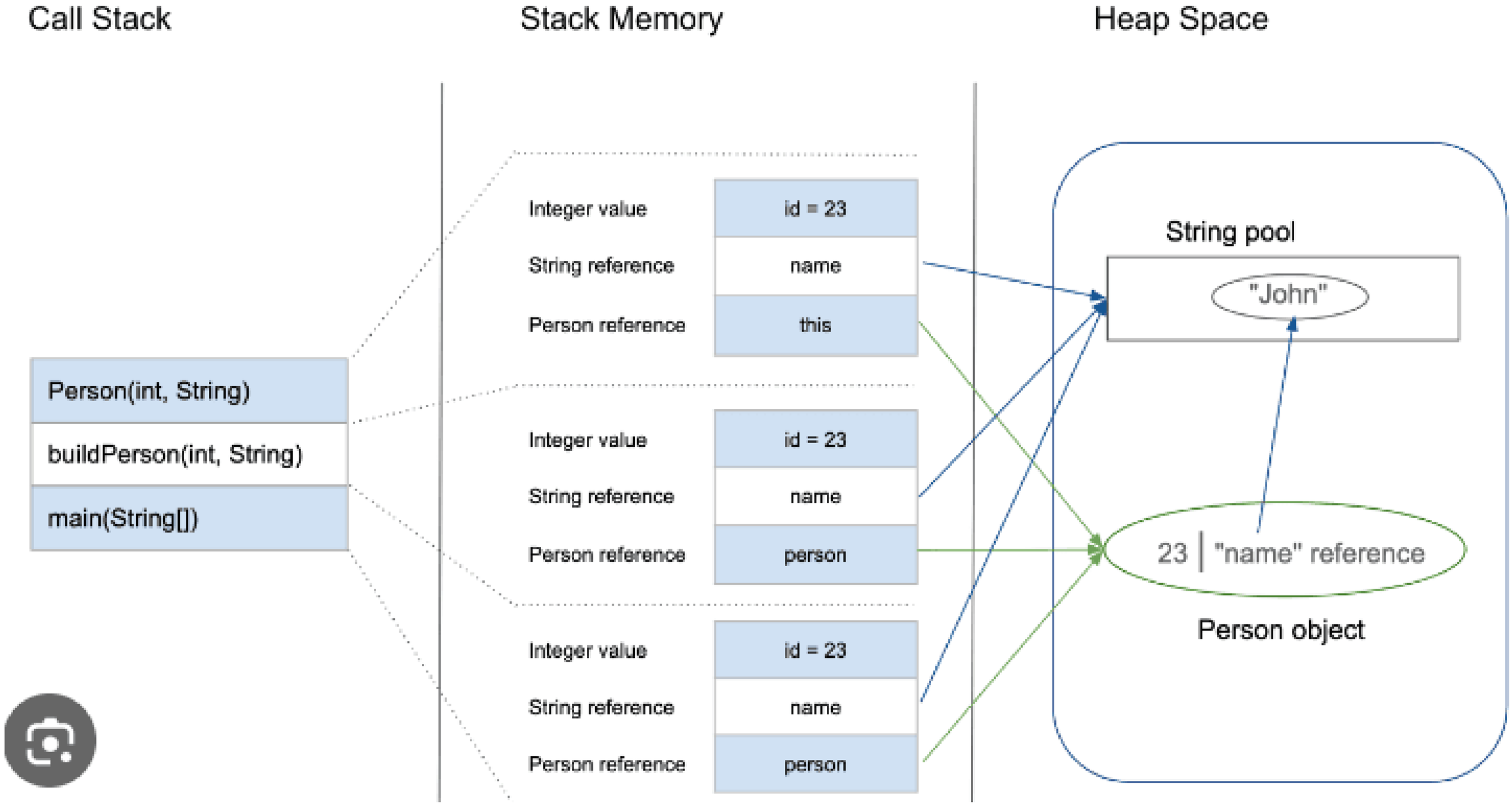
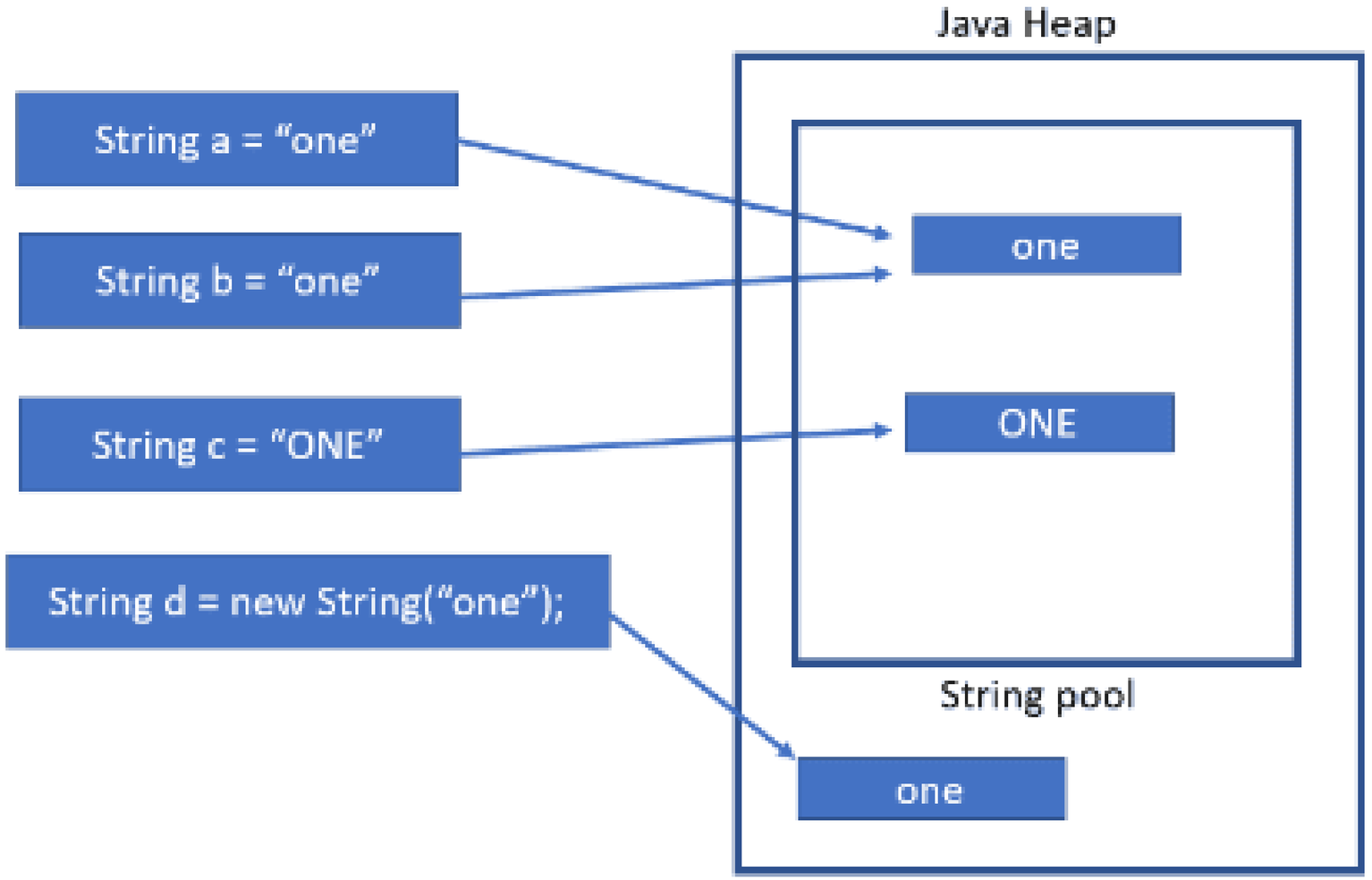
UNRAVELING THE MYSTERIES OF JAVA MEMORY: A JOURNEY THROUGH ITS INTRIGUING PARTS

By Deepak


Java Source Code
.java file in Java language syntax
Java Compiler
javac
Java Bytecode
.class file in hexadecimal
Java Runtime Environment (JRE)
Java Virtual Machine (JVM)
Java API Classes
Class Loader Subsystem
Loading
Bootstrap Class Loader
Extension Class Loader
Application Class Loader
Linking
Verify
Prepare
Resolve
Initialization
Initialization
Runtime Data Area
Method Area
Heap
Stack
Thread #1
Thread #2
Thread #3
PC Register
Thread #1 PC
Thread #2 PC
Thread #3 PC
Native Method Stack
Execution Engine
Interpreter
JIT Compiler
Garbage Collector
Native Method Interface (JNI)
Native Method Libraries
Operating System (Windows, Mac, Linux, Unix etc.)

Overview

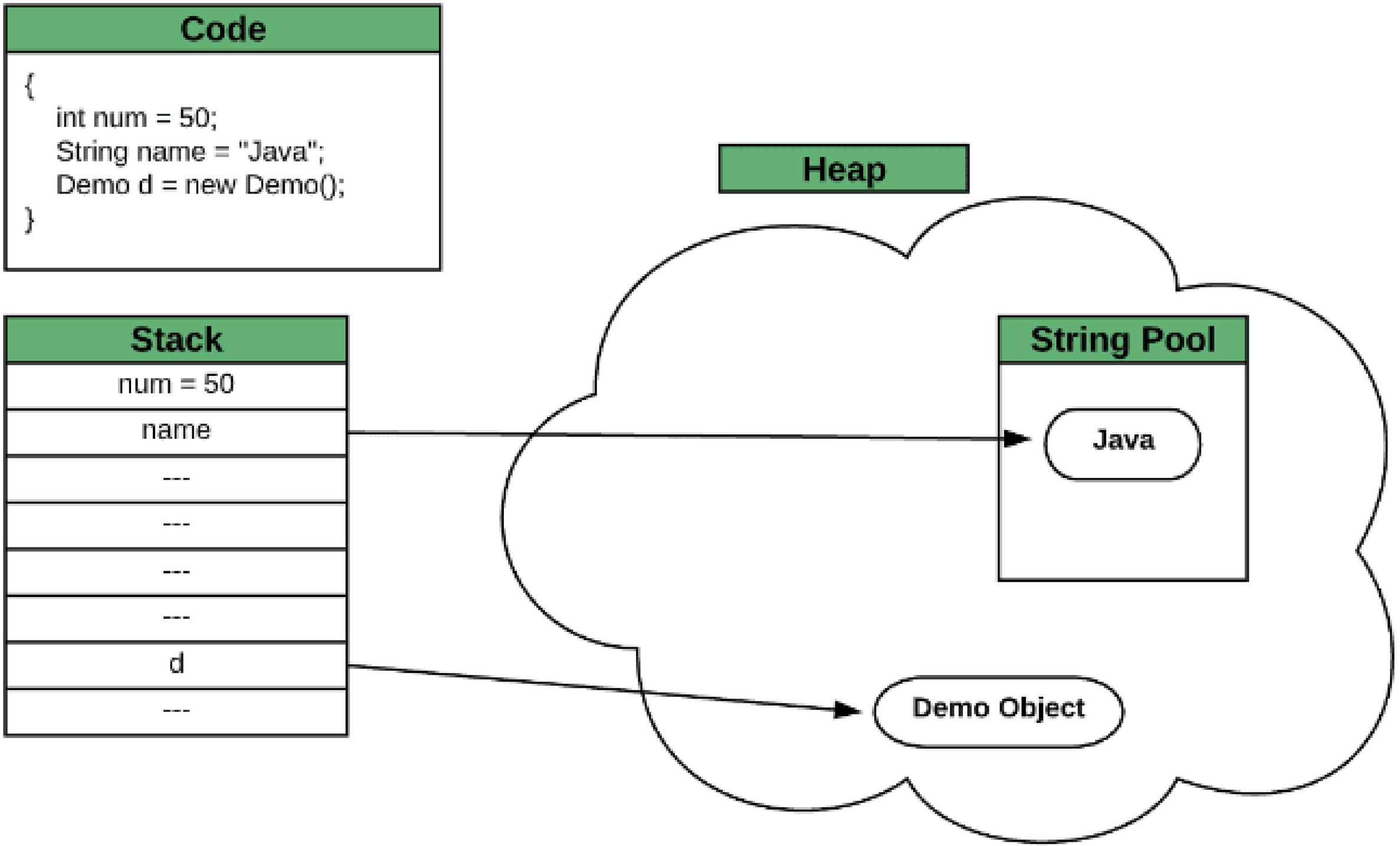

Call Stack
Stack Memory
Heap Space
Person(int, String)
buildPerson(int, String)
main(String[])
Integer value
id = 23
String reference
name
Person reference
this
Integer value
id = 23
String reference
name
Person reference
person
Integer value
id = 23
String reference
name
Person reference
person
String pool
"John"
23 | "name" reference
Person object




```
a == b;  // true
a == c;  // false
a.equals(b);  // true
a.equalsIgnoreCase(c);  // true
a == d;  // false
a.equals(d);  // true
```

HEAP MEMORY

The **Heap Memory** is where objects are stored in Java. We will dive into its structure and learn about the different generations of objects in Java's Heap Memory.


Code
{
int num = 50;
String name = "Java";
Demo d = new Demo();
}
Heap
Stack
num = 50
name

---
d

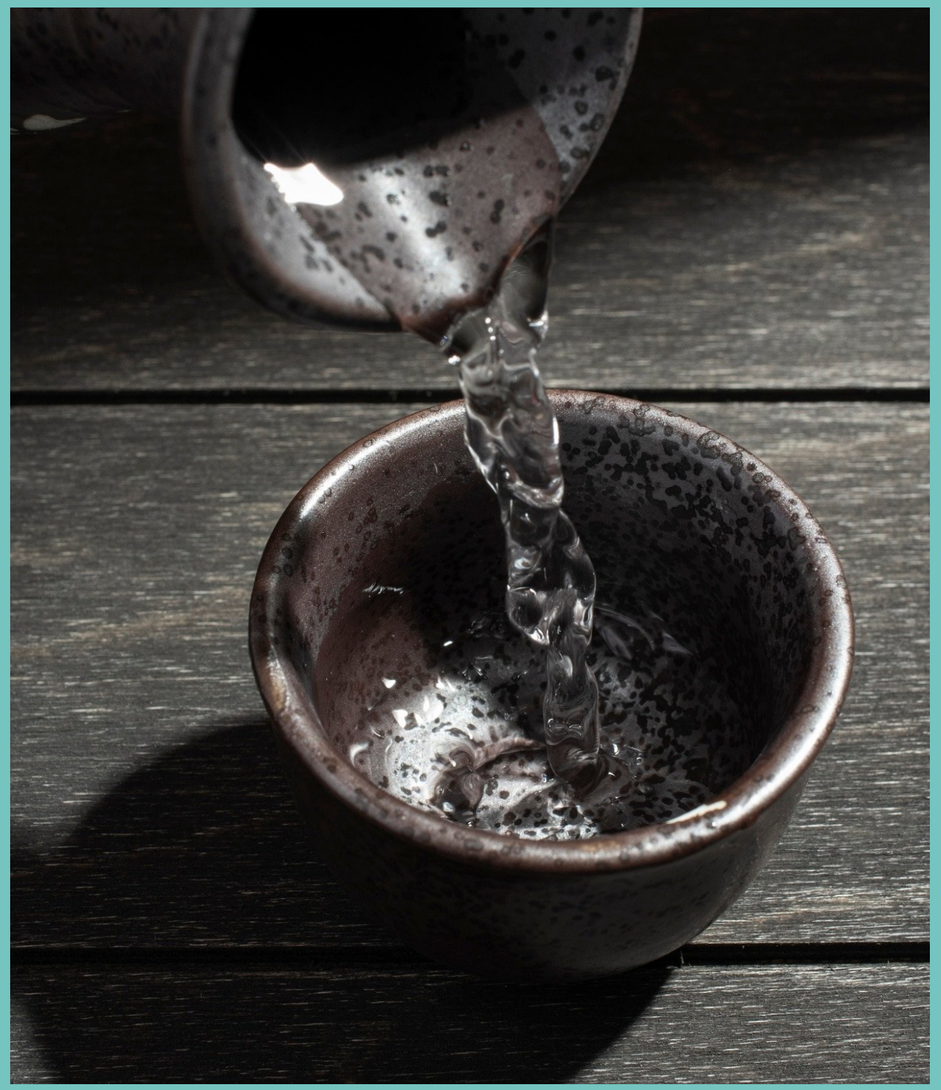
String Pool
Java
Demo Object

GARBAGE COLLECTION

The **Garbage Collector** is responsible for managing the Heap Memory by freeing unused objects. We will explore the different algorithms used by the Garbage Collector and how they impact performance.

STACK MEMORY

The **Stack Memory** is where method invocations and local variables are stored in Java. We will see how the Stack Memory works and how it is used to manage method calls and nested invocations.



MEMORY LEAKS

A **Memory Leak** occurs when objects are not released properly by the Garbage Collector, causing memory consumption to grow over time. We will learn how to identify and prevent Memory Leaks in Java applications.

CONCLUSION

We hope you have enjoyed our journey through the mysteries of Java Memory. We have explored its most intriguing parts and learned how to manage it effectively. Remember to use this knowledge wisely and create amazing applications!

THANKS