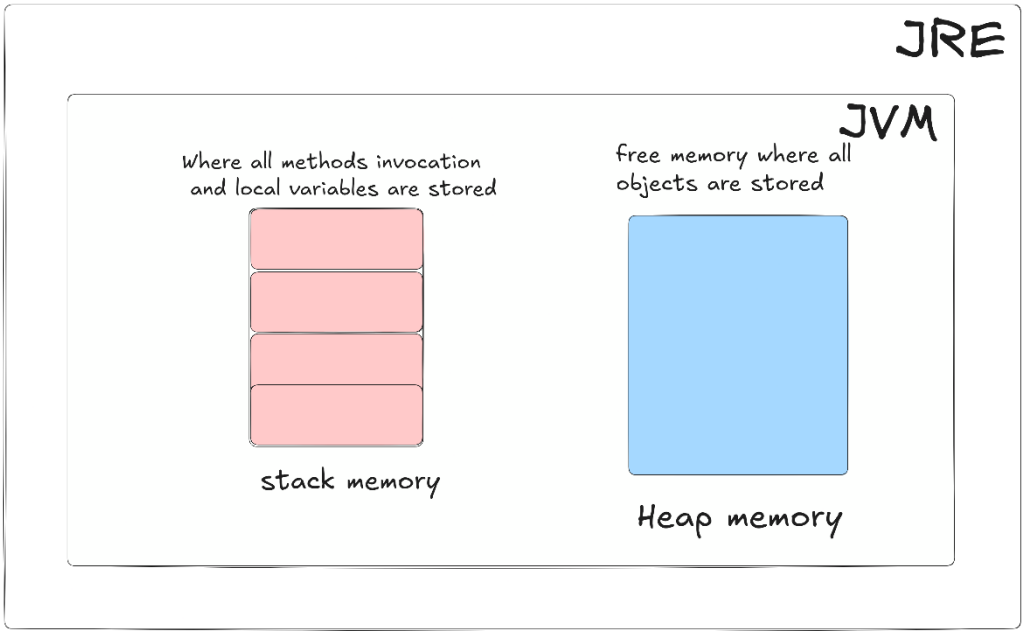
07-Stack and Heap

Understanding Stack and Heap Memory in Java

In Java, memory management is crucial for efficient application performance. The Java Virtual Machine (JVM) divides memory into two primary areas: **Stack Memory** and **Heap Memory**. Understanding how these memory areas work is key to writing optimized Java programs.



What's Inside the JVM?

The JVM manages the execution of Java code, including memory allocation and deallocation. Here's an overview of how memory is structured within the JVM:

1. Stack Memory

- **What is Stack Memory?**
    - Stack memory is allocated to each thread at runtime and is used to store:
        - Local variables
        - Method call details (like the order of execution)
        - References to objects and partial results
- **How Does Stack Memory Work?**

- It follows the **Last-In-First-Out (LIFO)** principle, meaning the last item added to the stack is the first one to be removed. Each method in Java has its own stack, and the memory is automatically freed up when the method completes.

- **Code Example**:

```
public class Calculator {
    public static void main(String[] args) {
        int data = 10; // Local variable stored in stack memory

        Calculator calc = new Calculator(); // Reference stored in stack, object in heap
        int result = calc.add(3, 4); // Method call, adds to stack

        System.out.println("Result: " + result);
    }

    public int add(int num1, int num2) {
        int sum = num1 + num2; // Local variables num1, num2, and sum are in stack memory
        return sum;
    }
}
```

Explanation:

- In the main method, the variable data is stored in the stack.
- When the add method is called, a new stack frame is created for this method, and local variables num1, num2, and sum are stored in this stack frame.
- The reference to the Calculator object (calc) is stored in the stack, but the actual Calculator object is stored in the heap.

Output:

Result: 7

Explanation of Output:

- The add method is called with the arguments 3 and 4. These values are stored in the stack, added together, and returned as 7. This result is then printed to the console.

2. Heap Memory

- **What is Heap Memory?**
    - Heap memory is used for dynamic memory allocation. It stores:
        - Objects
        - Classes loaded by the JVM
- **How Does Heap Memory Work?**
    - Heap memory is not bound by a specific order like the stack. Memory allocation and deallocation are dynamic, meaning objects can grow and shrink as needed.
- **Code Example**:

```
public class Calculator {
   int num=5; // Instance variable stored in heap memory

   public static void main(String[] args) {
      Calculator calc = new Calculator(); // Reference in stack, object in heap
      int result = calc.add(3, 4); // Method call
      System.out.println("Result: " + result);
   }

   public int add(int num1, int num2) {
      return num1 + num2 + num; // Accessing instance variable from heap
   }
}
```

Explanation:

- The instance variable num is stored in the heap memory because it belongs to an instance of the Calculator class.
- The reference calc is stored in the stack, pointing to the object in the heap.
- The add method uses the instance variable num, demonstrating how the stack and heap interact.

Output:

Result: 12

Explanation:

- The instance variable num is initialized to 5 in the heap memory. When add is called, num1 and num2 (with values 3 and 4) are added along with num (which is 5), giving a total of 12, which is printed.

3. Interaction Between Stack and Heap

- The stack and heap work together to manage memory efficiently. When a method uses an object, the reference is stored in the stack, and the actual object is in the heap. The JVM uses this reference to access and manipulate the object in the heap.