Java Memory Model, JVM, Stack & Heap – Corrected and Detailed Notes

1. Java as a Strongly Typed, Object-Oriented Language

Java is a **strongly typed programming language**, which means:

- Every variable must have a **declared data type**
- Type checking is done at **compile time**
- Type mismatches are caught early, reducing runtime errors

Java is also a **pure object-oriented language** (except for primitive types).

Because Java follows Object-Oriented Programming (OOP):

- Real-world entities can be modeled easily
- Programs are designed using **classes and objects**
- Concepts like **encapsulation, inheritance, polymorphism, and abstraction** make applications modular and maintainable

---

2. Classes, Objects, and Instantiation

- A **class** is a blueprint or template
- An **object** is a real instance of a class

Creating an object using the `new` keyword is called **instantiation**.

Example:

```
Student s = new Student();
```

Here:

- `Student` → class
- `s` → reference variable (stored in stack)
- `new Student()` → object (stored in heap)

---

3. JVM, RAM, and Execution Environment

When a Java program runs:

- JVM is loaded into **RAM**

- JVM creates a **runtime environment** for Java execution

Inside JVM, memory is divided into **Runtime Data Areas**.

---

4. JVM Runtime Data Areas

The JVM runtime data areas are:

1. Method Area
2. Heap Area
3. Java Stack Area
4. Native Method Stack
5. Program Counter (PC) Register

Among these, **Stack Area and Heap Area are the most important** for understanding memory and interviews.

---

5. Method Area (Class Area)

The Method Area stores:

- Class metadata
- Method bytecode
- Static variables
- Constant pool

This area is shared by all threads.

---

6. Stack Area (Java Stack)

The **stack area** stores:

- Method calls
- Local variables
- Method parameters
- Reference variables

Key points:

- Each thread has its own stack
- Memory is allocated in the form of **stack frames**
- Stack follows **LIFO (Last In First Out)** principle

When a method is called:

- A stack frame is created

When a method completes execution:

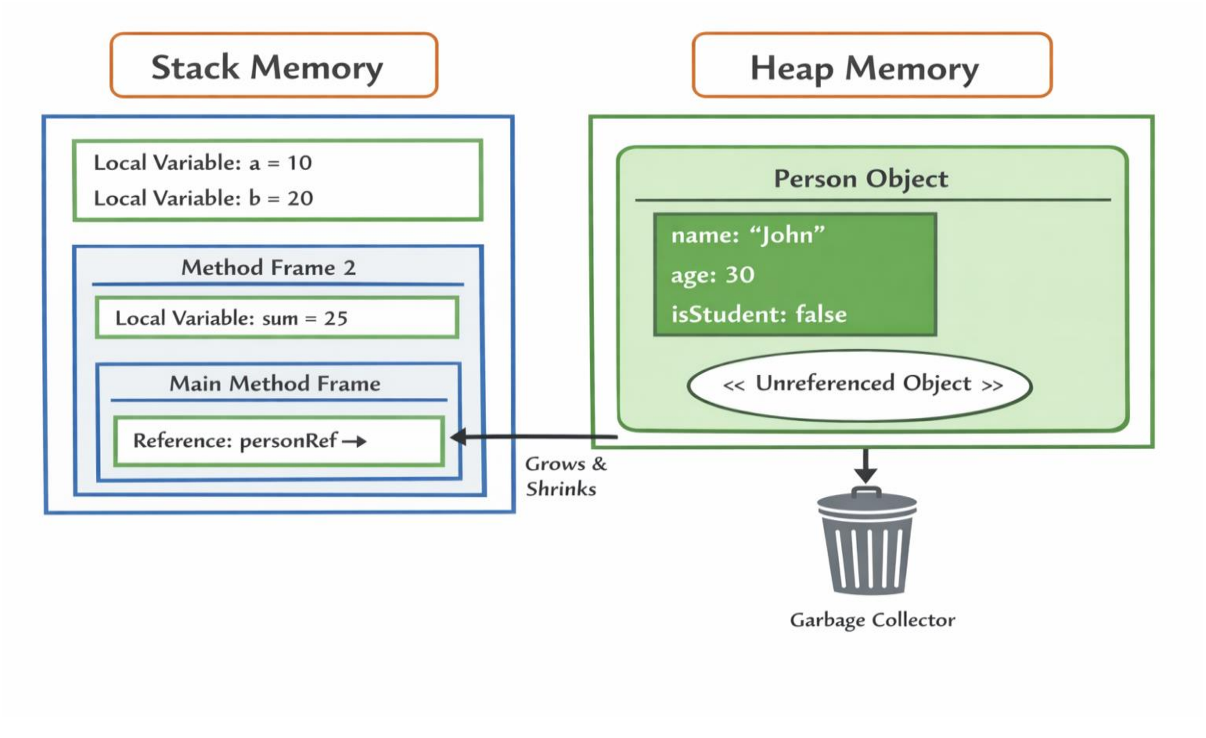
- Its stack frame is destroyed automatically

The `main()` method:

- Is also stored in the stack area
- Is the first method pushed onto the stack

Local variables:

- Are declared inside methods, blocks, or loops
- Stored in the stack
- Have **no default values**
- Scope is limited to the block in which they are declared



7. Heap Area

The **heap area** stores:

- Objects
- Instance variables (object variables)

Key points:

- Heap is shared among all threads
- Objects are created using `new`
- Memory allocation happens dynamically

Instance variables:

- Declared directly inside a class but outside methods
- Memory is allocated **inside the object** in heap
- Have default values assigned by Java
- Scope is throughout the class (via object)

---

8. Static Variables Clarification

Static variables:

- Are declared using `static` keyword
- Belong to the **class**, not to objects
- Stored in the **Method Area**, not in heap

(This corrects a common misconception.)

---

9. Variable Types Based on Memory Location

Local Variables

- Declared inside methods or blocks
- Stored in stack
- No default values
- Destroyed when stack frame is removed

Instance Variables

- Declared in class, outside methods
- Stored in heap inside objects
- Default values provided
- Destroyed when object is garbage collected

---

10. Garbage Collection

Garbage Collector works on **heap memory only**.

An object becomes eligible for garbage collection when:

- It has no active references from stack or method area

Garbage Collector:

- Automatically deallocates heap memory
- Does NOT work on stack memory

Stack memory is cleared automatically when methods complete execution.

---

11. Program Termination Flow

1. `main()` starts execution (stack)
2. Objects are created (heap)
3. Methods execute and return
4. Stack frames are destroyed
5. After last line of `main()`, control returns to JVM
6. `main()` stack frame is removed
7. Heap objects without references are cleaned by GC

---

12. Interview One-Line Summary

- Stack stores method execution and local variables
- Heap stores objects and instance variables
- References live in stack, objects live in heap
- Garbage Collector works only on heap