Object Oriented Programming using Java

Comprehensive overview of Java

sandeepkulange@gmail.com | 9527325202

Session Overview

- Naming Convention
- Overview of java.lang.Object
- toString() Method
- Static Field
- Static Method
- Singleton Class

Naming Convention

- **Camel Case Naming Convention**
  - In this case, the first letter of the identifier is lowercase, and each subsequent concatenated word starts with an uppercase letter.
  - Example:
    - main
    - parseInt, firstName, lastName etc.
    - showInputDialog, numberOfStudents, calculateTotalPrice etc.
  - In Java, this case is typically used for the following:
    - Method local variable
    - Method parameter
    - Field
    - Method

Naming Convention

- **Pascal Case Naming Convention**
  - In this case, the first letter of each concatenated word starts with an uppercase letter.
  - Example:
    - System, String, Scanner, Program etc.
    - StringBuffer, StringBuilder, UserProfile etc.
    - NumberFormatException, NullPointerException, ArrayIndexOutOfBoundsException etc.
  - In Java, this case is typically used for the following:
    - Type Name(Interface, Class, Enum)
    - File Name

Naming Convention

- **Snake Case Naming Convention**
  - In this case, words are separated by underscores (_) and all letters are typically in lowercase.
  - Example:
    - this_is_snake_case
    - example_variable_name
    - constant_value
  - In Java, the snake case convention is **not commonly used** for naming variables, methods, or classes.
  - Java uses **UPPER_SNAKE_CASE** for final (constant) values
  - Example
    - public static final int MIN_VALUE
    - public static final int MAX_VALUE
    - public static final int MIN_PRIORITY
    - public static final int MAX_PRIORITY

Overview of java.lang.Object class

- Object is a class declared in java.lang package.
    - ➢ It is a **non final class.** It means that we can create child class i.e sub class of it.
    - ➢ It is a **concrete class.** It means that we can instantiate it.

- It is **ultimate base class** / Super cosmic base class / root of java class hierarchy.
    - ➢ java.lang.Object class do not have parent/super class.
    - ➢ java.lang.Object class do not implement any interface.

- Every Java class is directly or indirectly extended from java.lang.Object class.

- **Note:** Super type of interface can be interface only. Hence interfaces do not extend java.lang.Object class.

- **Points to remember:**
    - ➢ java.lang.Object class do not contain nested type.
    - ➢ java.lang.Object class do not contain field
    - ➢ java.lang.Object class contain only parameterless constructor(actually default ctor.)
    - ➢ java.lang.Object class contain 11 methods(5 non final & 6 final methods).

Overview of java.lang.Object class

- Since java.lang.Object class contains only parameterless constructor, we can not instantiate it by passing argument.

- Consider below examples:

```
1. Object o1 = new Object("Sandeep");        //Not OK
2. Object o2 = new Integer( 123 );           //Not OK
3. Object o3 = new Object();                 //OK
```

Overview of java.lang.Object class

- java.lang.Object class contains 11 methods:

    1. public String **toString**()
    2. public boolean **equals**(Object obj)
    3. public **native** int **hashCode**()
    4. protected **native** Object **clone**() throws CloneNotSupportedException
    5. protected void **finalize**() throws Throwable

    6. public final **native** Class<?> **getClass**()
    7. public final void **wait**() throws InterruptedException
    8. public final **native** void **wait**(long timeout) throws InterruptedException
    9. public final void **wait**(long timeout, int nanos) throws InterruptedException
    10. public final **native** void **notify**()
    11. public final **native** void **notifyAll**()

toString() method

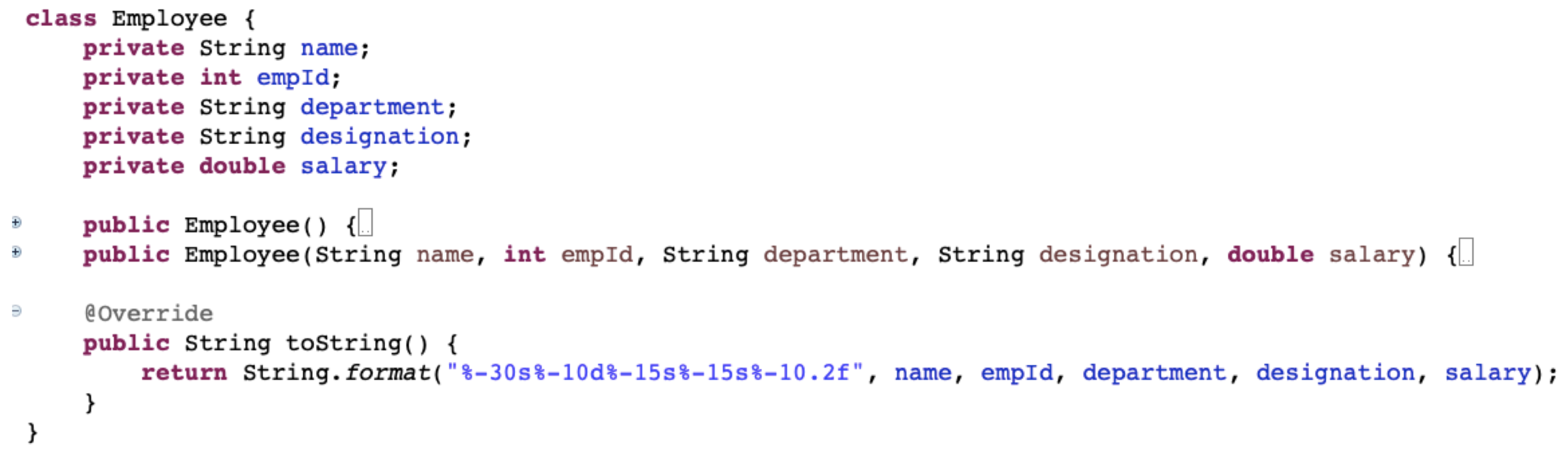
- toString() is a non final method of java.lang.Object class.
- Signature:
  - ➢ public String toString( );
- To represent state of the instance in String format, we should use toString().
- Default implementation:

```
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

- In general, according to business logic if implementation of existing class toString method is logically incomplete or partially complete then we should redefine i.e. override toString method in sub class.
- **Note:** toString() method should return short and clear description of the instance that should be easy for a person to understand.

toString() method

- Example 1:

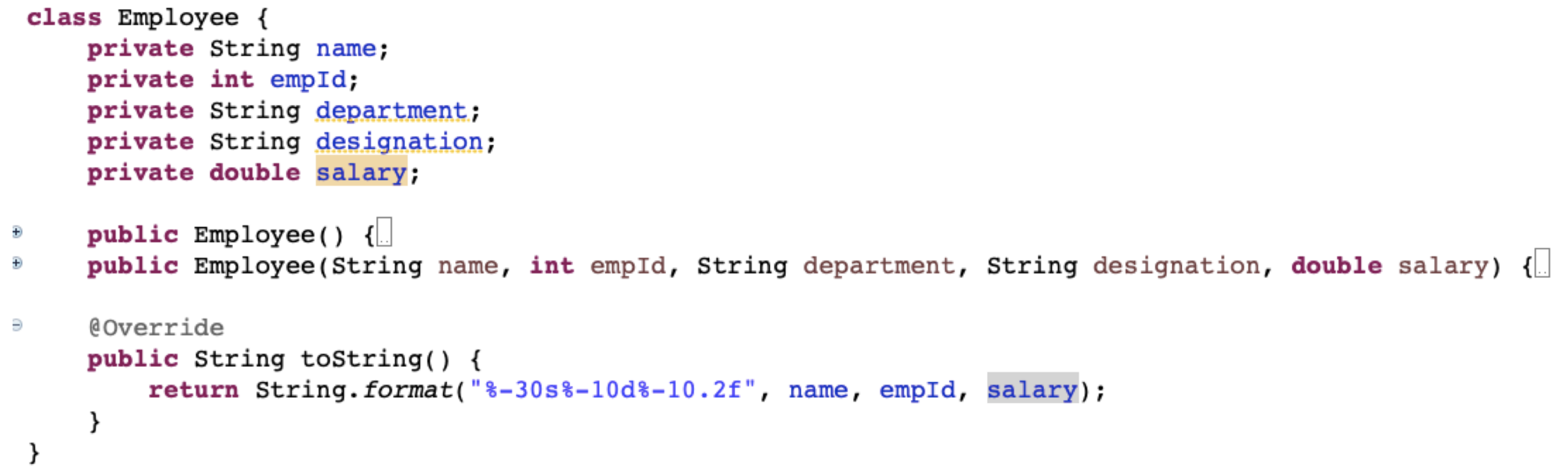


```
class Employee {
    private String name;
    private int empId;
    private String department;
    private String designation;
    private double salary;

    public Employee() {
    public Employee(String name, int empId, String department, String designation, double salary) {

    @Override
    public String toString() {
        return String.format("%-30s%-10d%-15s%-15s%-10.2f", name, empId, department, designation, salary);
    }
}
```

- Example 2:



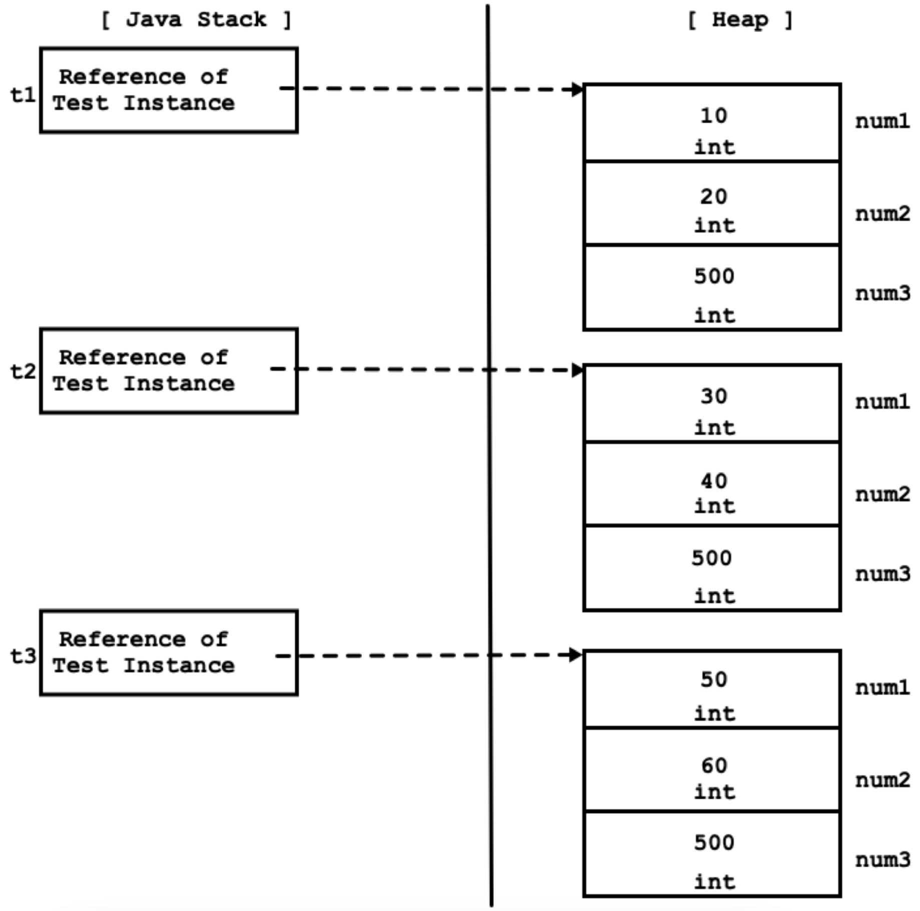
```
class Employee {
    private String name;
    private int empId;
    private String department;
    private String designation;
    private double salary;

    public Employee() {
    public Employee(String name, int empId, String department, String designation, double salary) {

    @Override
    public String toString() {
        return String.format("%-30s%-10d%-10.2f", name, empId, salary);
    }
}
```

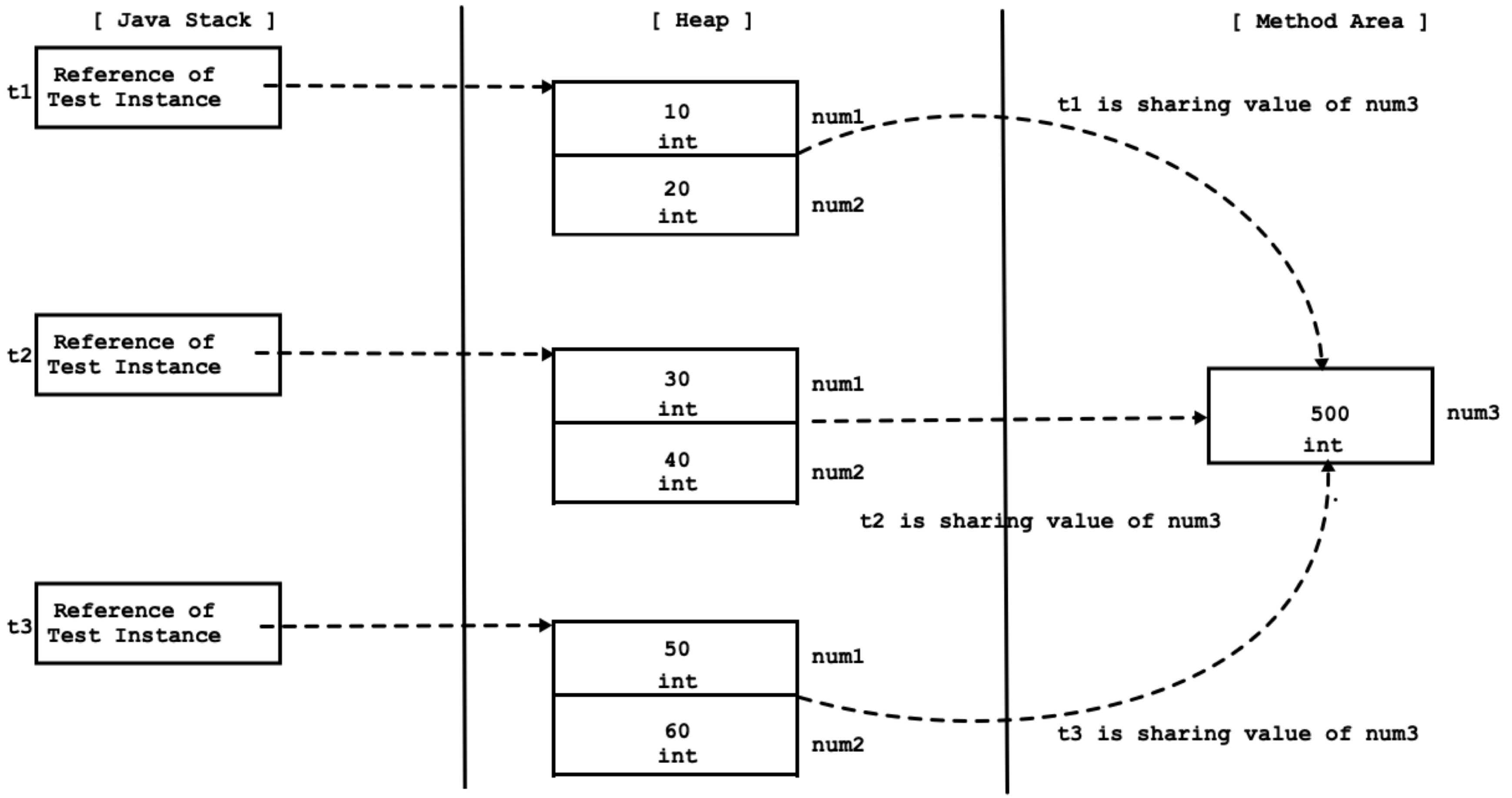
Non Static Field

- Non static field is also called as instance variable.
- Instance variable get space once per instance according to order of its declaration.



Static Field

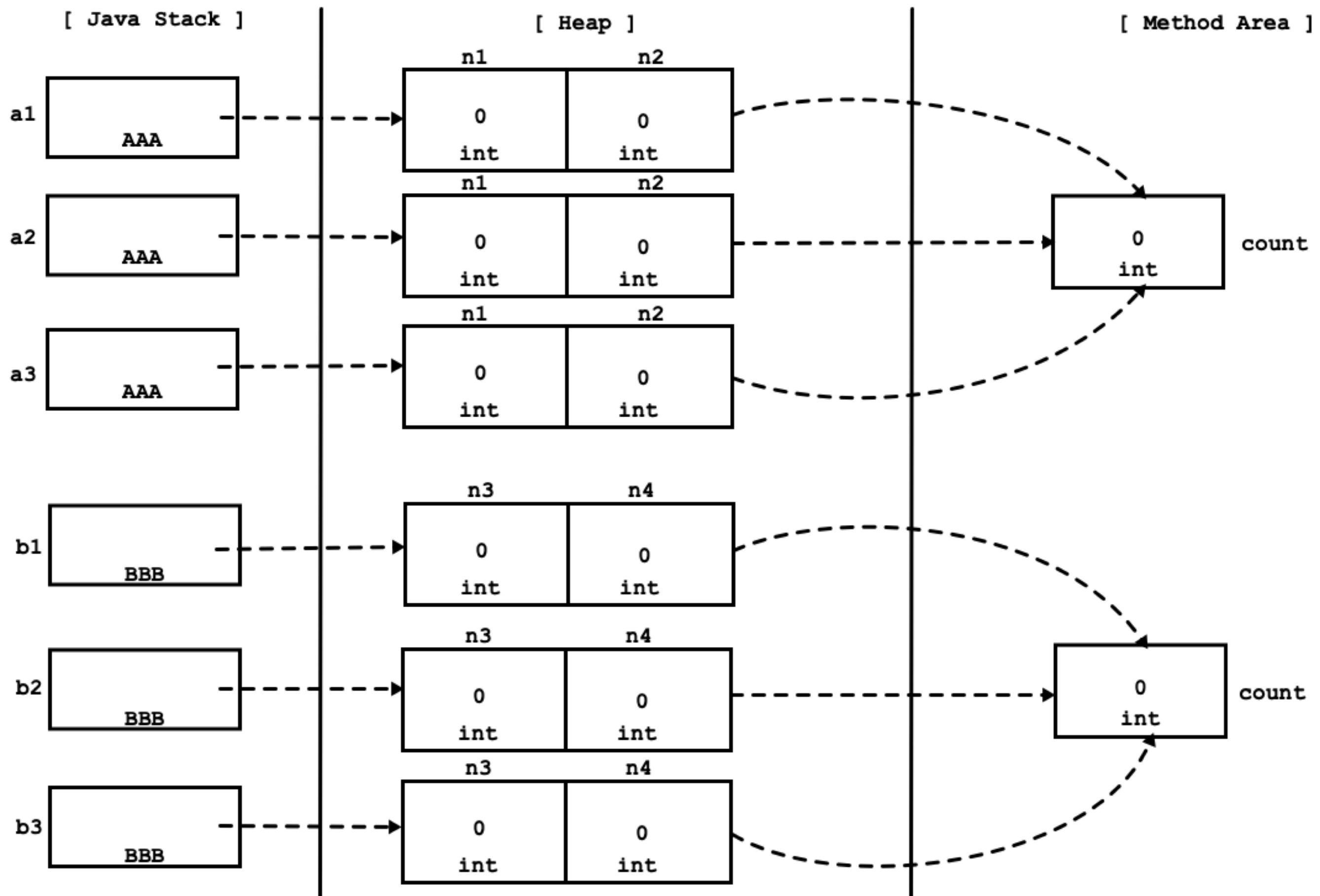
- If we want to share value of a field in all the instances of same class then field should be static.



[Java Stack]
[Heap]
[Method Area]
t1
Reference of Test Instance
10 int
num1
20 int
num2
t1 is sharing value of num3
t2
Reference of Test Instance
30 int
num1
40 int
num2
500 int
num3
t2 is sharing value of num3
t3
Reference of Test Instance
50 int
num1
60 int
num2
t3 is sharing value of num3

Static Field

- Static variable is also called as class level variable.
- Class level variable get space once per class during class loading on method area.

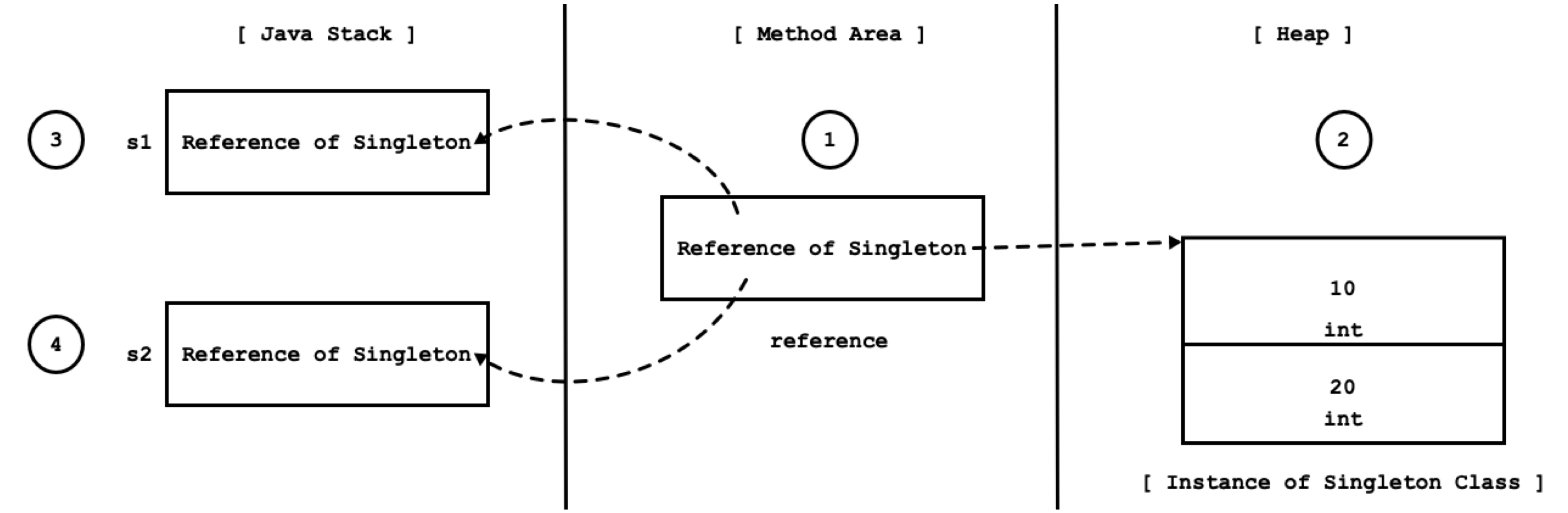


Static Method

- To access state of instance variable we should define non static method inside class.
- To access state of class level variable we should define static method inside class.
- Non static methods are designed to call on instance whereas static methods are designed to call on class name.
    - Since static methods are designed to call on class name, it doesn’t get this reference.
- Inside non static method, we can access static as well as non static members but inside static method we can access only static members of the class.
- Using instance, we can access non static members inside static method.

Singleton Class

- A class from which we can create only one instance is called as singleton class.



Q
A