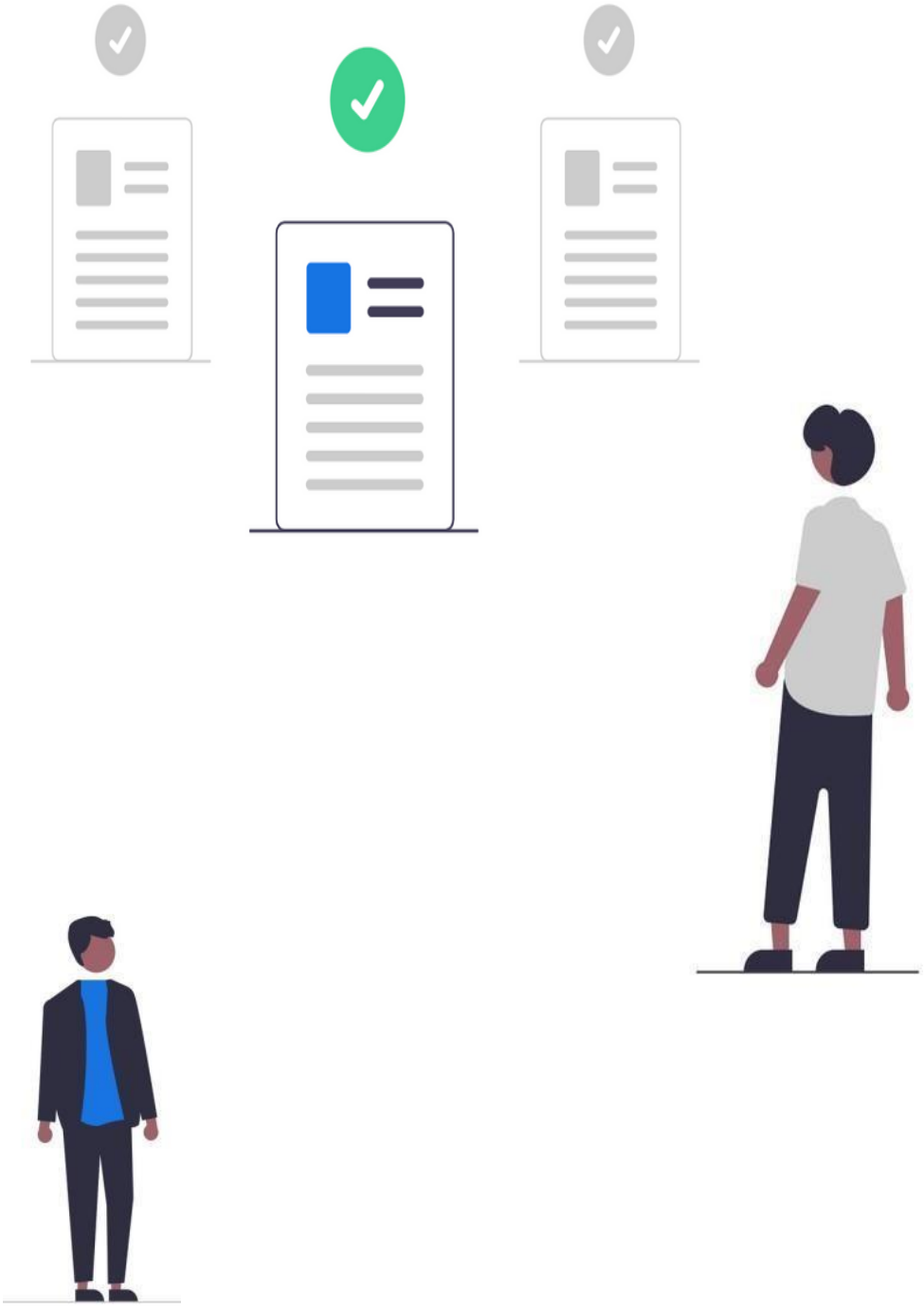
www.qastack.net
+92 300 1404 303

Difference Between Instance Variable and Static Variable in Java



Instance Variable:

- **Definition:** An instance variable is a variable that belongs to an instance of a class. Each object created from the class has its own copy of instance variables.
- **Memory Allocation:** Instance variables are allocated memory in the heap as part of the object. Each object gets its own copy.
- **Scope:** The scope of an instance variable is within the object; different objects have different values for these variables.
- **Access:** Accessed via the object reference.
- **Lifecycle:** The lifecycle of an instance variable is tied to the object lifecycle.

Static Variable:

- **Definition:** A static variable belongs to the class itself, not to any particular object. It is shared among all instances of the class.
- **Memory Allocation:** Static variables are stored in the method area of the memory and there's only one copy, shared across all objects.
- **Scope:** The scope of a static variable is within the class; all instances share the same value.
- **Access:** Accessed via the class name or object reference, though using the class name is preferred.
- **Lifecycle:** The lifecycle of a static variable starts when the class is loaded into memory and ends when the class is unloaded.

Example:

```
class Example {
    int instanceVar = 10; // Instance
    variable static int staticVar = 20; //
    Static variable
    public static void main(String[] args) {
        Example obj1 = new Example();
        Example obj2 = new Example();
        // Accessing instance variable
        obj1.instanceVar = 30;
        System.out.println("obj1 instanceVar: " + obj1.instanceVar); //
```

```
        30 System.out.println("obj2 instanceVar: " + obj2.instanceVar);
        // 10
        // Accessing static variable
        obj1.staticVar = 40;
        System.out.println("obj1 staticVar: " + obj1.staticVar); // 40
        System.out.println("obj2 staticVar: " + obj2.staticVar); // 40
        System.out.println("Example staticVar: " + Example.staticVar);
        // 40
    }
}
```

Internal Working:

- **Instance Variable:** Each time a new object is created, a new memory space is allocated for the instance variables.
- **Static Variable:** Only one memory space is allocated for the static variable when the class is loaded, shared among all instances.

What is String Pool?

The **String Pool** (also known as String Intern Pool) is a special memory region in the Java heap where Java stores string literals. When a string literal is created, the JVM checks the string pool first:

- If the string already exists in the pool, a reference to the pooled instance is returned.
- If the string does not exist, the JVM adds it to the pool and returns a reference to the new string.

Why Use String Pool?

- It helps save memory by avoiding the creation of multiple copies of the same string.

Example:

```
public class StringPoolExample {
    public static void main(String[] args) {

        String str1 = "Hello";
        String str2 = "Hello";
```

```
        // Both str1 and str2 point to the same object in the string pool
        System.out.println(str1 == str2); // true
        String str3 = new String("Hello");

        // str3 is a new object in the heap, not in the string pool
        System.out.println(str1 == str3); // false
    }
}
```

Internal Working:

- When String str1 = "Hello"; is executed, the JVM checks if "Hello" exists in the string pool. If it does, str1 points to it. If not, it creates "Hello" in the pool.
- The statement new String("Hello"); creates a new object in the heap, independent of the string pool.

In Java, the concept of a "global variable" as it exists in some other programming languages does not directly apply. However, you can achieve similar functionality using static variables, which are class-level variables. Here's a comparison between what could be considered a "global" variable in Java (a static variable) and an instance variable.

1. Instance Variable

Definition:

- An instance variable is a variable that is declared within a class but outside any method or constructor. It belongs to an instance of the class, meaning each object created from the class has its own copy of the instance variable.

Scope:

- The scope of an instance variable is within the object. Each object can have different values for the instance variables.

Memory Allocation:

- Instance variables are stored in the heap memory as part of the object.

Example:

```
class Example {
    int instanceVar = 10; // Instance variable
```

```
    void display() {
        System.out.println("Instance Variable: " + instanceVar);
    }

    public static void main(String[] args) {
        Example obj1 = new Example();
        Example obj2 = new Example();
        obj1.instanceVar = 20; // Change instance variable for obj1
        obj2.instanceVar = 30; // Change instance variable for obj2
        obj1.display(); // Instance Variable:
        20 obj2.display(); // Instance
        Variable: 30
    }
}
```

2. "Global" Variable (Static Variable) Definition:

- A static variable in Java is akin to a "global" variable in other languages. It is a class-level variable that is shared among all instances of the class. There is only one copy of a static variable, regardless of how many objects are created from the class.

Scope:

- The scope of a static variable is the entire class. It can be accessed from any method in the class, whether static or instance method, and even without creating an object of the class.

Memory Allocation:

- Static variables are stored in the method area (a specific part of the heap) and are shared among all instances of the class.

Example:

```
class Example {
    static int globalVar = 100; // Static variable, akin to a global variable
    void display() {
```

```
        System.out.println("Global Variable: " + globalVar);
    }

    public static void main(String[] args) {
        Example obj1 = new Example();
        Example obj2 = new Example();
        obj1.display(); // Global Variable: 100

        // Modify static variable using obj1
        obj1.globalVar = 200;
        obj2.display(); // Global Variable: 200 (since it's shared across all instances)

        // Modify static variable using class name
        Example.globalVar = 300;
        obj1.display(); // Global Variable: 300
        obj2.display(); // Global Variable: 300
    }
}
```

Key Differences:

1. **Ownership:**
    - **Instance Variable:** Each object has its own copy of an instance variable.
    - **Static Variable:** Only one copy of a static variable exists, shared among all instances of the class.
2. **Access:**
    - **Instance Variable:** Accessed through object references.
    - **Static Variable:** Can be accessed via the class name or object reference (though using the class name is preferred).
3. **Memory Allocation:**
    - **Instance Variable:** Memory is allocated in the heap as part of the object.
    - **Static Variable:** Memory is allocated in the method area, independent of any object.

4. **Lifecycle:**
    - **Instance Variable:** The lifecycle is tied to the object's lifecycle.
    - **Static Variable:** The lifecycle starts when the class is loaded into memory and ends when the class is unloaded.

In summary, while Java does not have true "global" variables, static variables fulfill a similar role within the scope of a class. They are accessible across all instances of the class and maintain a single state across the application, making them similar to global variables in other languages.

Let's explore the concepts of instance variables, static variables, and what is commonly referred to as "global" variables (which are essentially static variables in Java) through five different real- time programming examples.

Real time Examples ?

Example 1: Bank Account Management

Scenario: Managing multiple bank accounts, each having its own balance (instance variable) while sharing a common interest rate (static variable).

```
class BankAccount {
   int balance; // Instance variable
   static double interestRate = 5.0; // Static variable (common for all accounts)
   BankAccount(int initialBalance) {
      this.balance = initialBalance;
   }
   void displayBalance() {
      System.out.println("Balance: " + balance + ", Interest Rate: " + interestRate + "%");
   }
   public static void main(String[] args) {
      BankAccount account1 = new
      BankAccount(1000); BankAccount account2 =
      new BankAccount(2000);
```

```
        account1.displayBalance(); // Balance: 1000, Interest Rate: 5.0%
        account2.displayBalance(); // Balance: 2000, Interest Rate: 5.0%
        // Change interest rate (static
        variable) BankAccount.interestRate
        = 6.0;
        account1.displayBalance(); // Balance: 1000, Interest Rate: 6.0%
        account2.displayBalance(); // Balance: 2000, Interest Rate: 6.0%
    }
}
```

Example 2: Library System

Scenario: Each book in a library has a unique identifier (instance variable), while all books share a common library name (static variable). We can think of the library name as a global variable.

```
class Book {
    String title; // Instance variable
    static String libraryName = "City Library"; // Static variable (global across all books)

    Book(String title) {
        this.title = title;
    }

    void displayBookInfo() {
        System.out.println("Title: " + title + ", Library: " + libraryName);
    }

    public static void main(String[] args) {
        Book book1 = new Book("Java Programming");
        Book book2 = new Book("Python
        Programming");
        book1.displayBookInfo(); // Title: Java Programming, Library: City Library
        book2.displayBookInfo(); // Title: Python Programming, Library: City Library
        // Change library name (static
```

```
        variable) Book.libraryName =
        "National Library";
        book1.displayBookInfo(); // Title: Java Programming, Library: National Library
        book2.displayBookInfo(); // Title: Python Programming, Library: National Library
    }
}
```

Example 3: Online Shopping Cart

Scenario: Each product has a price (instance variable), while all products share a common discount rate (static variable). The discount rate could be considered a global variable within the shopping cart context.

```
class Product {
    double price; // Instance variable
    static double discountRate = 10.0; // Static variable (global across all products)

    Product(double price)
        { this.price = price;
    }

    double getDiscountedPrice() {
        return price - (price * discountRate / 100);
    }

    public static void main(String[] args) {
        Product product1 = new
        Product(100.0); Product product2 =
        new Product(200.0);
        System.out.println("Product 1 Price after Discount: " + product1.getDiscountedPrice()); //
90.0
        System.out.println("Product 2 Price after Discount: " + product2.getDiscountedPrice()); //
180.0

        // Change discount rate (static
```

```
    variable) Product.discountRate =
    20.0;
    System.out.println("Product 1 Price after Discount: " + product1.getDiscountedPrice()); //
80.0
    System.out.println("Product 2 Price after Discount: " + product2.getDiscountedPrice()); //
16
0.0
 }
}
```

Example 4: University Management System

Scenario: Each student has a name (instance variable), while all students belong to the same university (static variable). The university name acts like a global variable.

```
class Student {
  String name; // Instance variable
  static String universityName = "Global University"; // Static variable (global across all
students)
  Student(String name) {
    this.name = name;
  }
  void displayStudentInfo() {
    System.out.println("Student: " + name + ", University: " + universityName);
  }
  public static void main(String[] args) {
    Student student1 = new
    Student("Alice"); Student student2 =
    new Student("Bob");
    student1.displayStudentInfo(); // Student: Alice, University: Global University
    student2.displayStudentInfo(); // Student: Bob, University: Global University
    // Change university name (static variable)
```

```
    Student.universityName = "International University";

    student1.displayStudentInfo(); // Student: Alice, University: International University

    student2.displayStudentInfo(); // Student: Bob, University: International University

  }

}
```

Example 5: Employee Management System

Scenario: Each employee has a unique ID (instance variable), while all employees share a common company name (static variable), which could be thought of as a global variable.

```
class Employee {

  int employeeId; // Instance variable

  static String companyName = "Tech Corp"; // Static variable (global across all employees)

  Employee(int id) {

    this.employeeId = id;

  }
  void displayEmployeeInfo() {

    System.out.println("Employee ID: " + employeeId + ", Company: " + companyName);

  }

  public static void main(String[] args) {

    Employee emp1 = new

    Employee(101); Employee emp2 =

    new Employee(102);

    emp1.displayEmployeeInfo(); // Employee ID: 101, Company: Tech Corp

    emp2.displayEmployeeInfo(); // Employee ID: 102, Company: Tech Corp

    // Change company name (static

    variable) Employee.companyName =

    "Global Tech";

    emp1.displayEmployeeInfo(); // Employee ID: 101, Company: Global Tech

    emp2.displayEmployeeInfo(); // Employee ID: 102, Company: Global Tech
```

```
    }
}
```

Summary:

1. **Instance Variables** are specific to each object and represent the state of that object.
2. **Static Variables** are shared across all objects of a class and represent the state of the class.
3. **Global Variables** (in the context of Java, these are static variables) are shared across all instances and accessible across the entire class.