Week 9 - S9 - Advanced OOP - Object Class Methods, Inner Classes - Assignment Problem (HW)

Name: Ramesh Harisabapathi Chettiar

Date of Submission:15/10/25

Problem Statement 1:

Create a class Employee with fields id, name, and salary. Override the toString()

method to print employee details in a readable format. In the main method, create multiple

Employee objects and print their class name using getClass().getName().

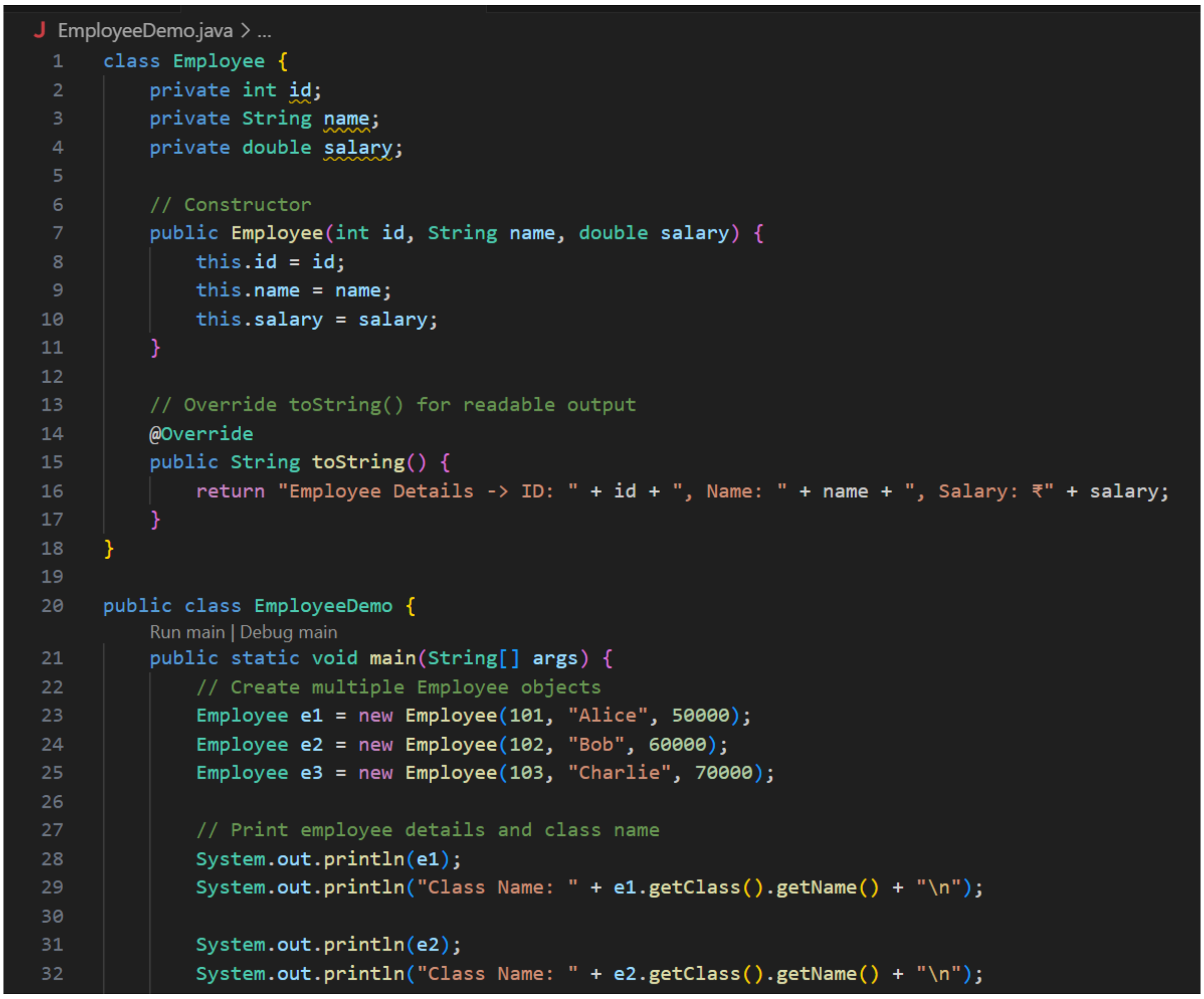
Hints:

● Override toString() to provide a meaningful string representation.

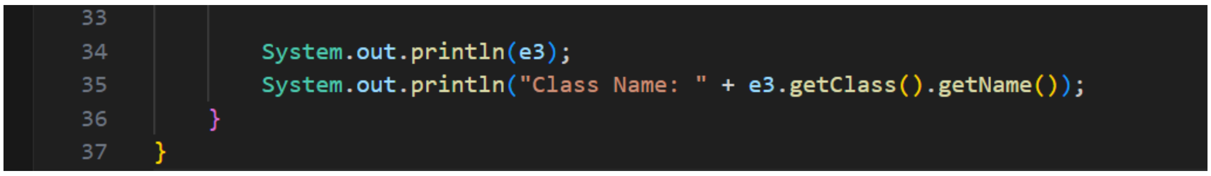
● Use getClass() to obtain runtime class information.

● Display both the object details and its class name.

EmployeeDemo.java

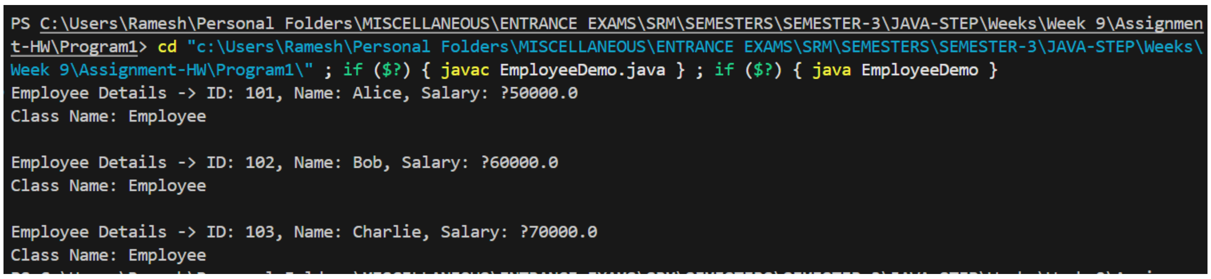


```java
class Employee {
    private int id;
    private String name;
    private double salary;

    // Constructor
    public Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }

    // Override toString() for readable output
    @Override
    public String toString() {
        return "Employee Details -> ID: " + id + ", Name: " + name + ", Salary: ₹" + salary;
    }
}

public class EmployeeDemo {
    public static void main(String[] args) {
        // Create multiple Employee objects
        Employee e1 = new Employee(101, "Alice", 50000);
        Employee e2 = new Employee(102, "Bob", 60000);
        Employee e3 = new Employee(103, "Charlie", 70000);

        // Print employee details and class name
        System.out.println(e1);
        System.out.println("Class Name: " + e1.getClass().getName() + "\n");

        System.out.println(e2);
        System.out.println("Class Name: " + e2.getClass().getName() + "\n");
```



```java

        System.out.println(e3);
        System.out.println("Class Name: " + e3.getClass().getName());
    }
}
```

OUTPUT→



```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 9\Assignmen
t-HW\Program1> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\
Week 9\Assignment-HW\Program1\" ; if ($?) { javac EmployeeDemo.java } ; if ($?) { java EmployeeDemo }
Employee Details -> ID: 101, Name: Alice, Salary: ?50000.0
Class Name: Employee

Employee Details -> ID: 102, Name: Bob, Salary: ?60000.0
Class Name: Employee

Employee Details -> ID: 103, Name: Charlie, Salary: ?70000.0
Class Name: Employee
```

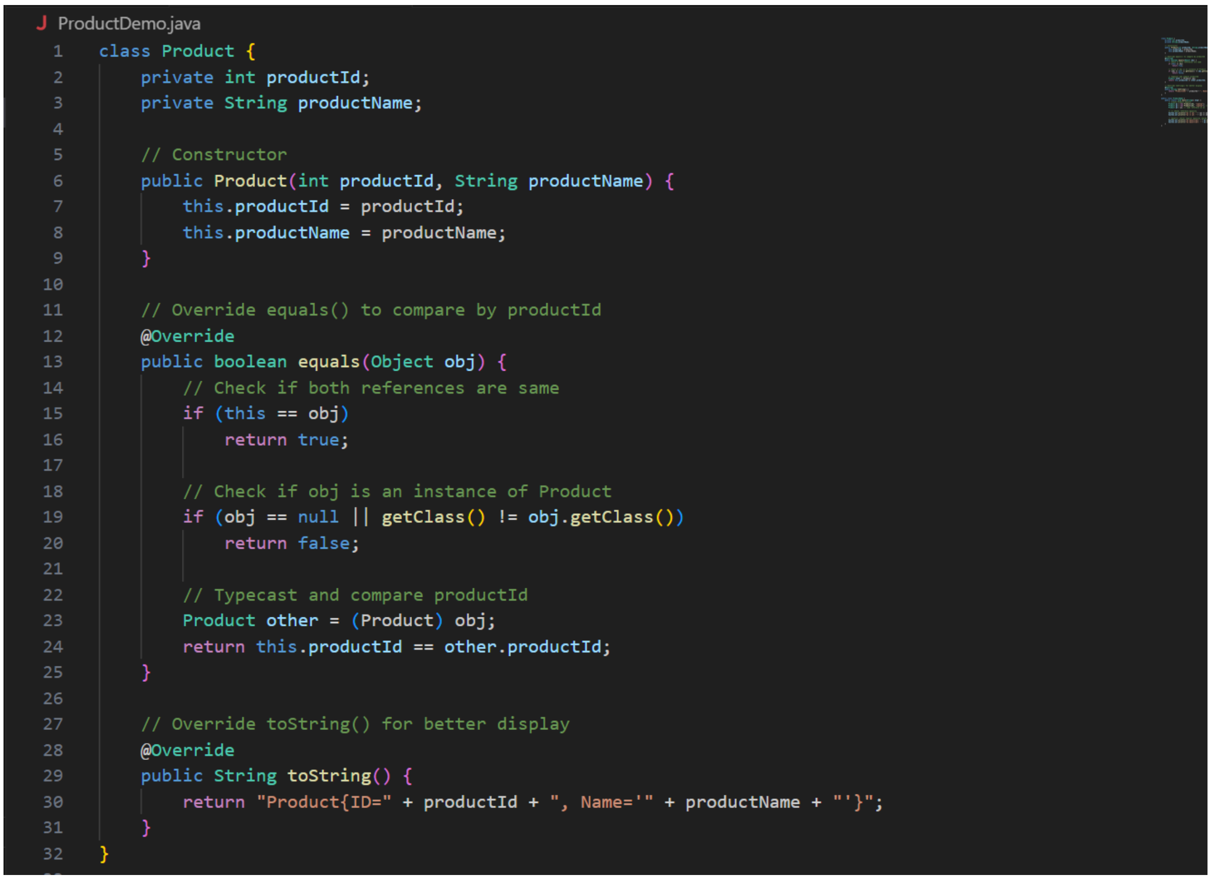
Problem Statement 2:

Create a class Product with productId and productName fields. Compare two Product objects using both == and .equals() to demonstrate the difference between reference and content comparison. Override the equals() method to compare objects by productId.

Hints:

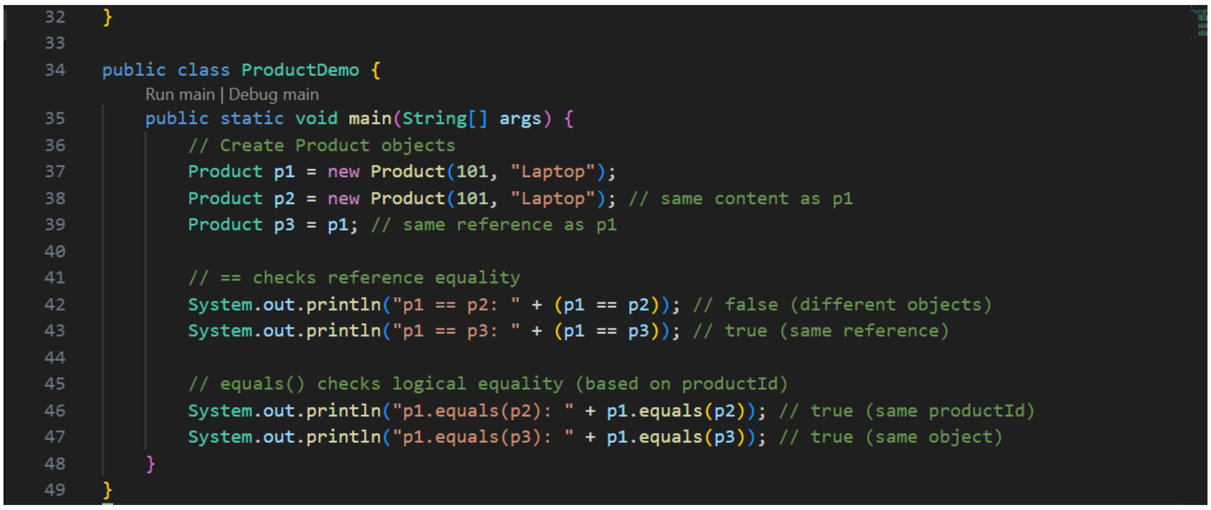
- == checks reference equality, .equals() checks logical equality.
- Override equals() properly using the @Override annotation.
- Print results of both comparisons for clarity.

ProductDemo.java



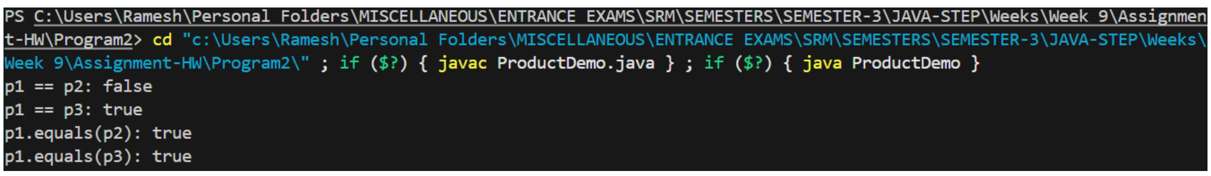
ProductDemo.java

```java
class Product {
    private int productId;
    private String productName;

    // Constructor
    public Product(int productId, String productName) {
        this.productId = productId;
        this.productName = productName;
    }

    // Override equals() to compare by productId
    @Override
    public boolean equals(Object obj) {
        // Check if both references are same
        if (this == obj)
            return true;

        // Check if obj is an instance of Product
        if (obj == null || getClass() != obj.getClass())
            return false;

        // Typecast and compare productId
        Product other = (Product) obj;
        return this.productId == other.productId;
    }

    // Override toString() for better display
    @Override
    public String toString() {
        return "Product{ID=" + productId + ", Name='" + productName + "'}";
    }
}
```



```java
}

public class ProductDemo {
    Run main | Debug main
    public static void main(String[] args) {
        // Create Product objects
        Product p1 = new Product(101, "Laptop");
        Product p2 = new Product(101, "Laptop"); // same content as p1
        Product p3 = p1; // same reference as p1

        // == checks reference equality
        System.out.println("p1 == p2: " + (p1 == p2)); // false (different objects)
        System.out.println("p1 == p3: " + (p1 == p3)); // true (same reference)

        // equals() checks logical equality (based on productId)
        System.out.println("p1.equals(p2): " + p1.equals(p2)); // true (same productId)
        System.out.println("p1.equals(p3): " + p1.equals(p3)); // true (same object)
    }
}
```

OUTPUT→



```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 9\Assignment-HW\Program2> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 9\Assignment-HW\Program2\" ; if ($?) { javac ProductDemo.java } ; if ($?) { java ProductDemo }
p1 == p2: false
p1 == p3: true
p1.equals(p2): true
p1.equals(p3): true
```

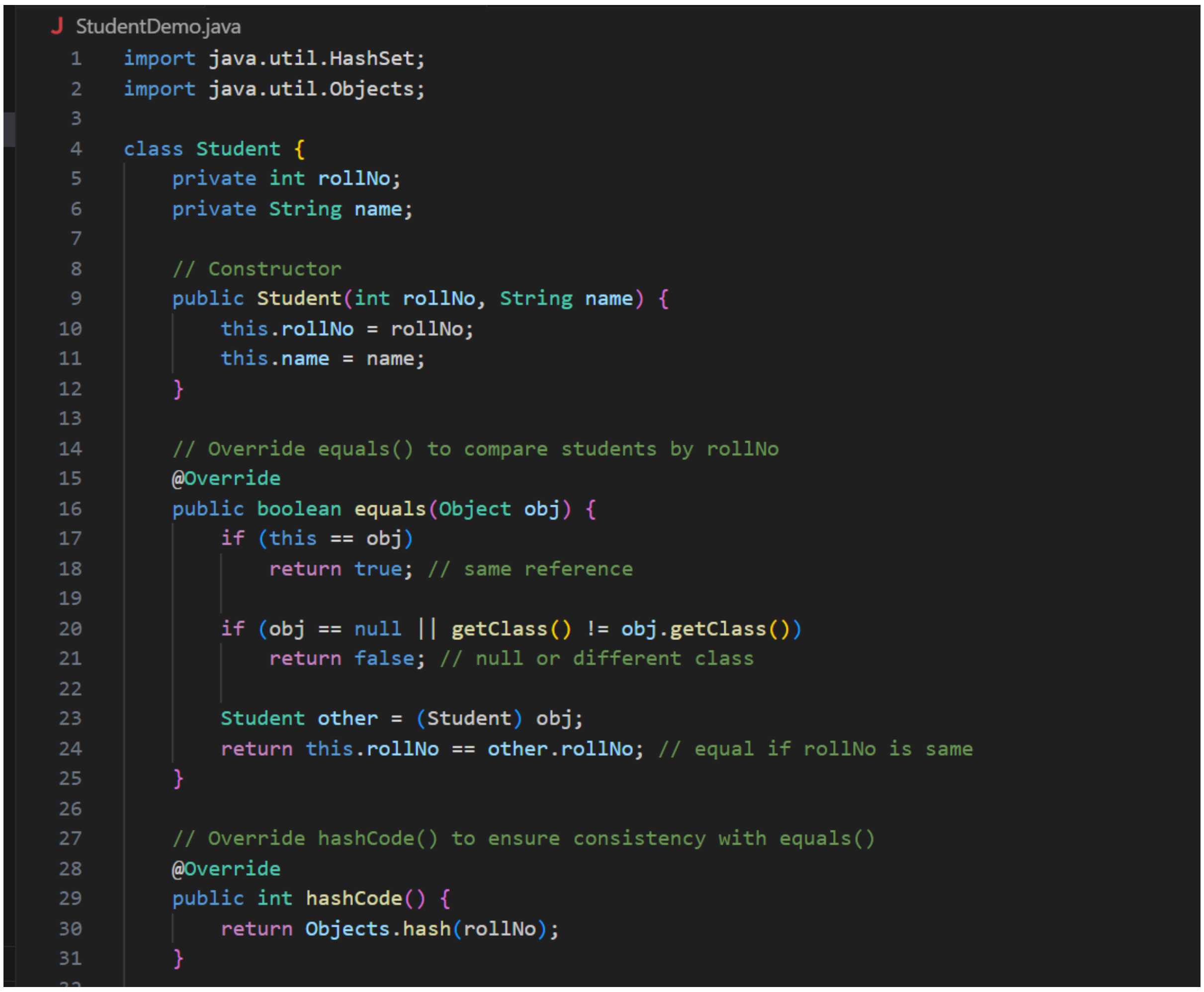
Problem Statement 3:

Create a Student class with rollNo and name fields. Override both equals() and hashCode() so that two students with the same roll number are considered equal. Demonstrate how these methods affect object storage in a HashSet.

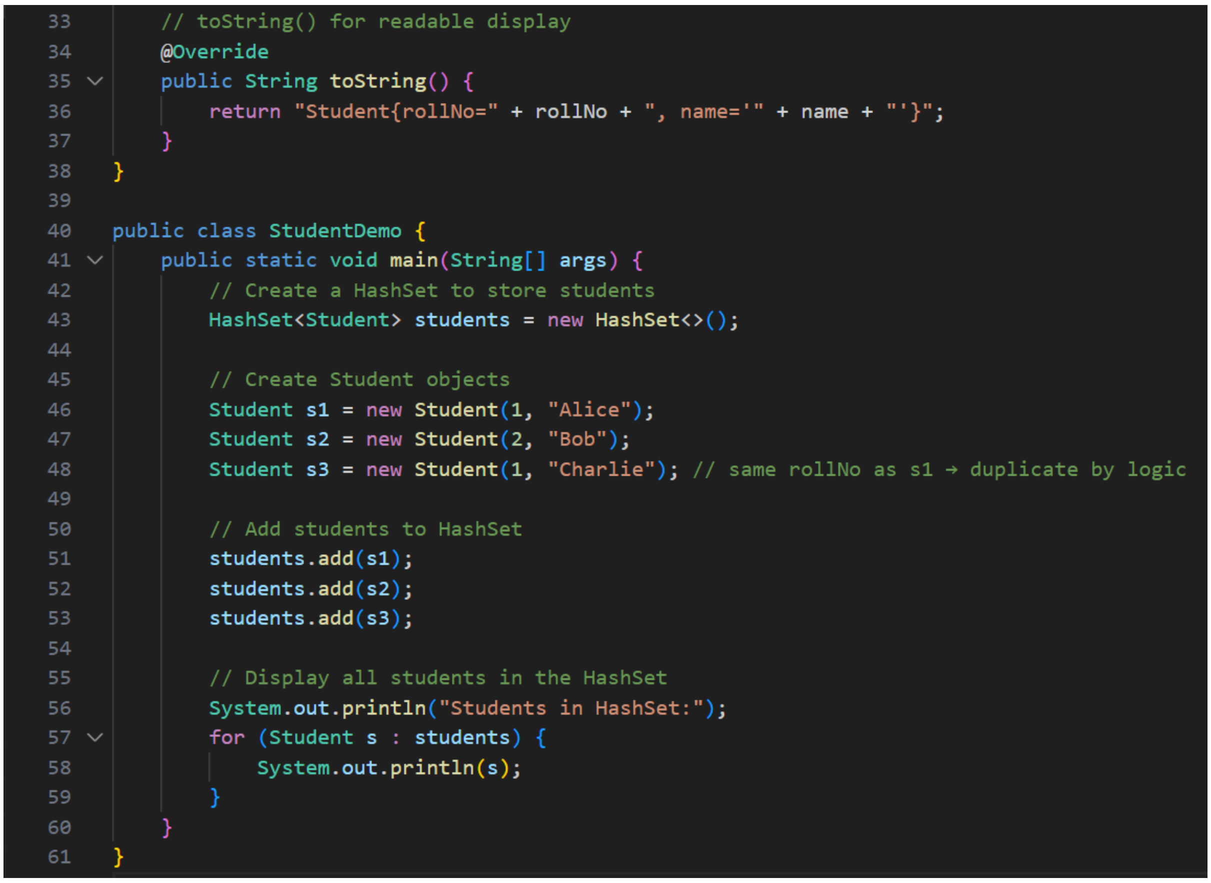
Hints:

- Use Objects.hash() to generate hash codes.
- Ensure equals() and hashCode() produce consistent results.
- Add duplicate objects to a HashSet and observe the output.

StudentDemo.java

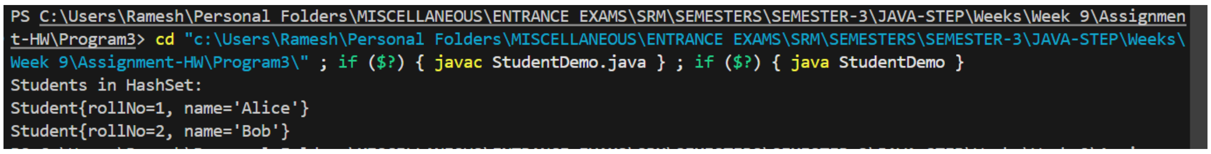


```java
import java.util.HashSet;
import java.util.Objects;

class Student {
    private int rollNo;
    private String name;

    // Constructor
    public Student(int rollNo, String name) {
        this.rollNo = rollNo;
        this.name = name;
    }

    // Override equals() to compare students by rollNo
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true; // same reference

        if (obj == null || getClass() != obj.getClass())
            return false; // null or different class

        Student other = (Student) obj;
        return this.rollNo == other.rollNo; // equal if rollNo is same
    }

    // Override hashCode() to ensure consistency with equals()
    @Override
    public int hashCode() {
        return Objects.hash(rollNo);
    }
```



```java
    // toString() for readable display
    @Override
    public String toString() {
        return "Student{rollNo=" + rollNo + ", name='" + name + "'}";
    }
}

public class StudentDemo {
    public static void main(String[] args) {
        // Create a HashSet to store students
        HashSet<Student> students = new HashSet<>();

        // Create Student objects
        Student s1 = new Student(1, "Alice");
        Student s2 = new Student(2, "Bob");
        Student s3 = new Student(1, "Charlie"); // same rollNo as s1 → duplicate by logic

        // Add students to HashSet
        students.add(s1);
        students.add(s2);
        students.add(s3);

        // Display all students in the HashSet
        System.out.println("Students in HashSet:");
        for (Student s : students) {
            System.out.println(s);
        }
    }
}
```

OUTPUT→



```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 9\Assignmen
t-HW\Program3> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\
Week 9\Assignment-HW\Program3\" ; if ($?) { javac StudentDemo.java } ; if ($?) { java StudentDemo }
Students in HashSet:
Student{rollNo=1, name='Alice'}
Student{rollNo=2, name='Bob'}
```

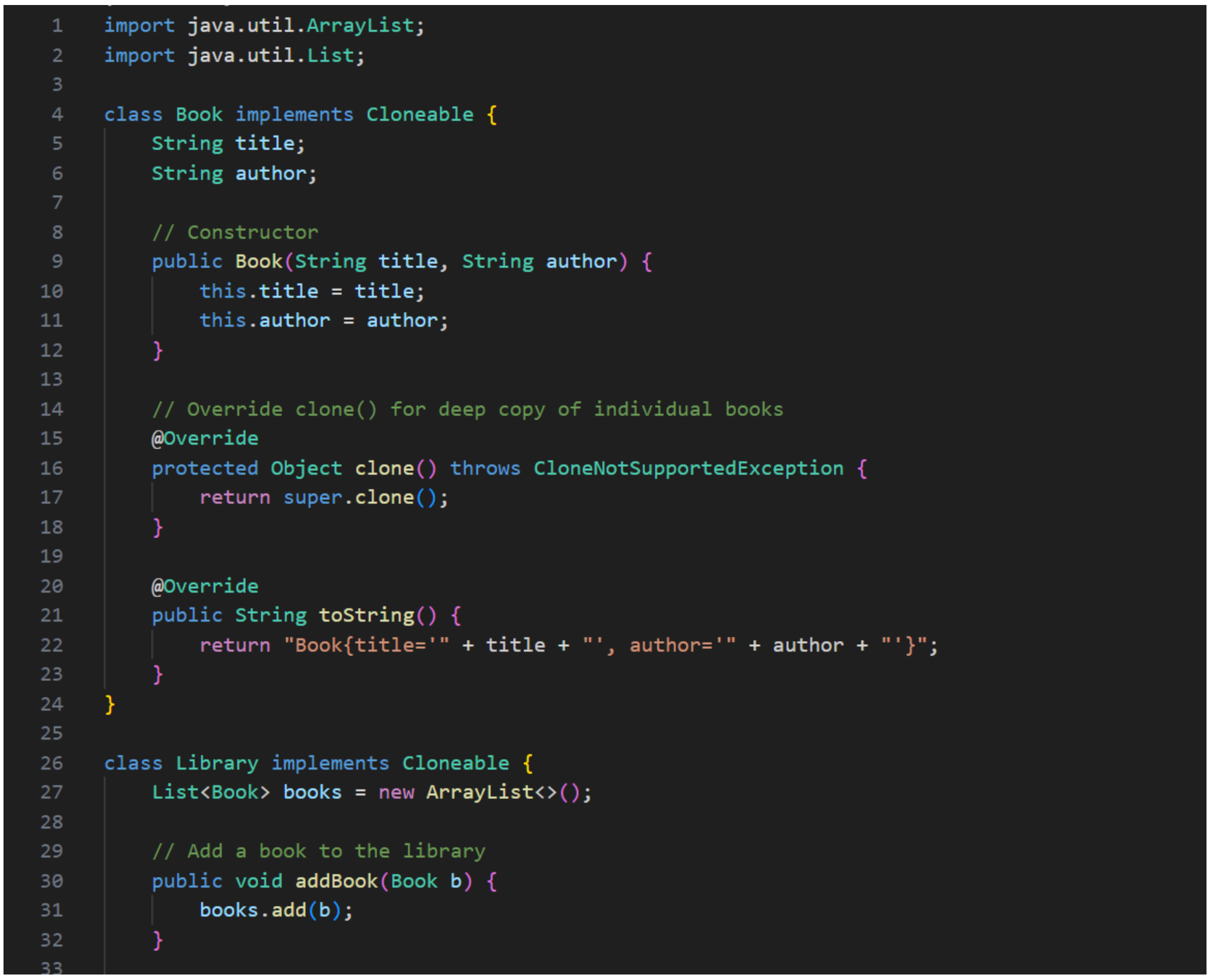
Problem Statement 4:

Create a class Library containing a list of Book objects. Implement cloning such that shallow cloning only copies object references while deep cloning copies the entire list with individual book data. Modify one book in the cloned object and observe its effect on the original.

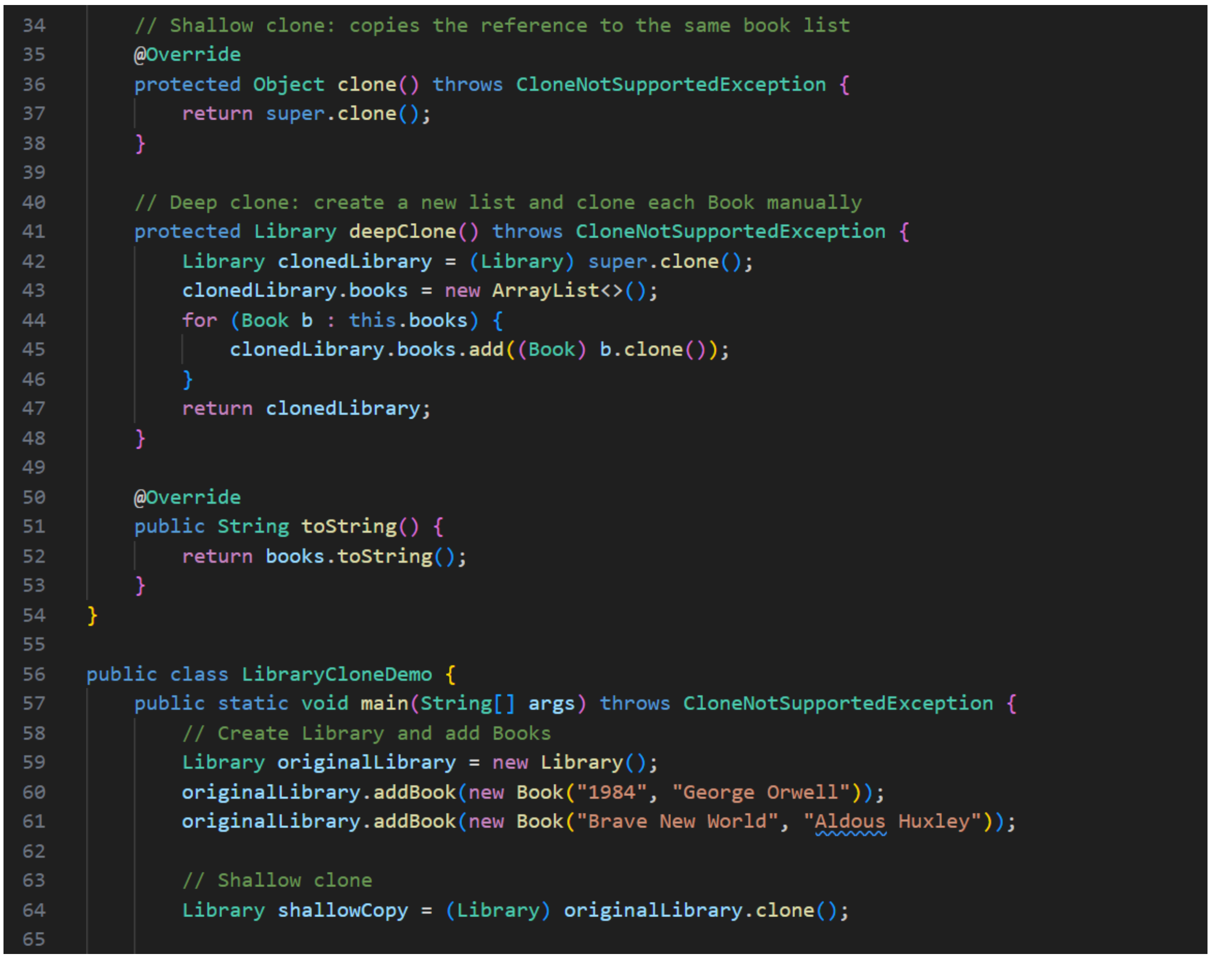
Hints:

- Use Cloneable interface and override clone().
- For deep cloning, clone each Book object inside the list manually.
- Use loops or streams to copy nested objects.

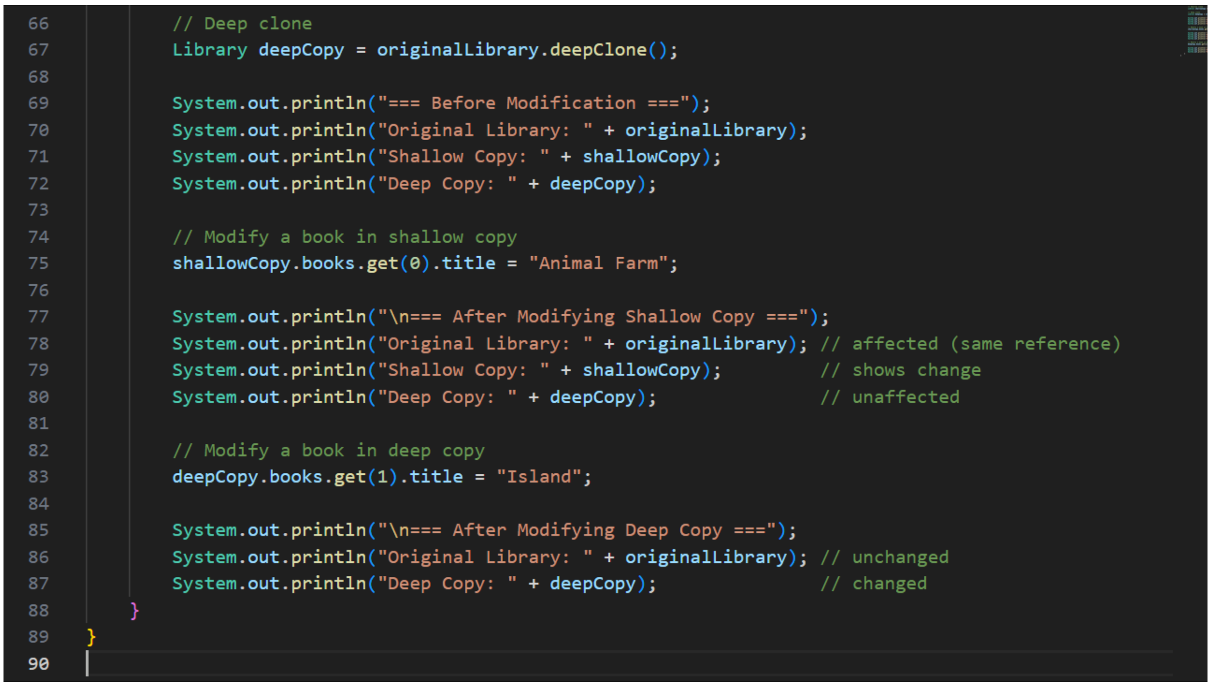
LibraryCloneDemo.java



```java
import java.util.ArrayList;
import java.util.List;

class Book implements Cloneable {
    String title;
    String author;

    // Constructor
    public Book(String title, String author) {
        this.title = title;
        this.author = author;
    }

    // Override clone() for deep copy of individual books
    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();
    }

    @Override
    public String toString() {
        return "Book{title='" + title + "', author='" + author + "'}";
    }
}

class Library implements Cloneable {
    List<Book> books = new ArrayList<>();

    // Add a book to the library
    public void addBook(Book b) {
        books.add(b);
    }

```

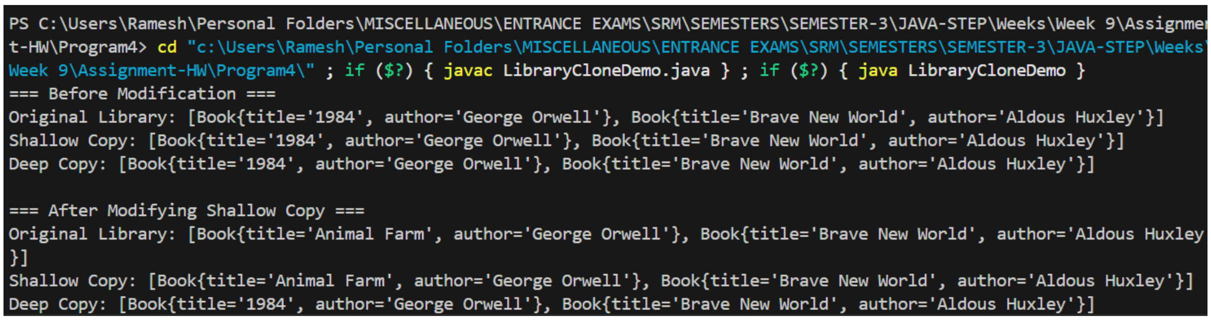


```java
    // Shallow clone: copies the reference to the same book list
    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();
    }

    // Deep clone: create a new list and clone each Book manually
    protected Library deepClone() throws CloneNotSupportedException {
        Library clonedLibrary = (Library) super.clone();
        clonedLibrary.books = new ArrayList<>();
        for (Book b : this.books) {
            clonedLibrary.books.add((Book) b.clone());
        }
        return clonedLibrary;
    }

    @Override
    public String toString() {
        return books.toString();
    }
}

public class LibraryCloneDemo {
    public static void main(String[] args) throws CloneNotSupportedException {
        // Create Library and add Books
        Library originalLibrary = new Library();
        originalLibrary.addBook(new Book("1984", "George Orwell"));
        originalLibrary.addBook(new Book("Brave New World", "Aldous Huxley"));

        // Shallow clone
        Library shallowCopy = (Library) originalLibrary.clone();

```

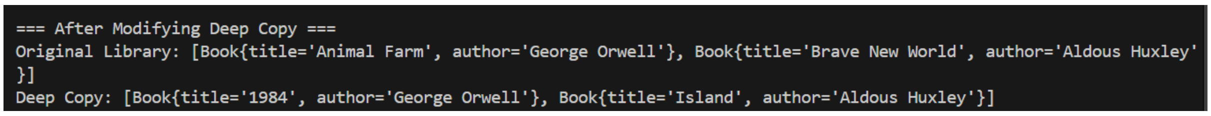


```java
        // Deep clone
        Library deepCopy = originalLibrary.deepClone();

        System.out.println("=== Before Modification ===");
        System.out.println("Original Library: " + originalLibrary);
        System.out.println("Shallow Copy: " + shallowCopy);
        System.out.println("Deep Copy: " + deepCopy);

        // Modify a book in shallow copy
        shallowCopy.books.get(0).title = "Animal Farm";

        System.out.println("\n=== After Modifying Shallow Copy ===");
        System.out.println("Original Library: " + originalLibrary); // affected (same reference)
        System.out.println("Shallow Copy: " + shallowCopy);          // shows change
        System.out.println("Deep Copy: " + deepCopy);                // unaffected

        // Modify a book in deep copy
        deepCopy.books.get(1).title = "Island";

        System.out.println("\n=== After Modifying Deep Copy ===");
        System.out.println("Original Library: " + originalLibrary); // unchanged
        System.out.println("Deep Copy: " + deepCopy);                // changed
    }
}

```

OUTPUT→



```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 9\Assignme
t-HW\Program4> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks
Week 9\Assignment-HW\Program4\" ; if ($?) { javac LibraryCloneDemo.java } ; if ($?) { java LibraryCloneDemo }
=== Before Modification ===
Original Library: [Book{title='1984', author='George Orwell'}, Book{title='Brave New World', author='Aldous Huxley'}]
Shallow Copy: [Book{title='1984', author='George Orwell'}, Book{title='Brave New World', author='Aldous Huxley'}]
Deep Copy: [Book{title='1984', author='George Orwell'}, Book{title='Brave New World', author='Aldous Huxley'}]

=== After Modifying Shallow Copy ===
Original Library: [Book{title='Animal Farm', author='George Orwell'}, Book{title='Brave New World', author='Aldous Huxley
}]
Shallow Copy: [Book{title='Animal Farm', author='George Orwell'}, Book{title='Brave New World', author='Aldous Huxley'}]
Deep Copy: [Book{title='1984', author='George Orwell'}, Book{title='Brave New World', author='Aldous Huxley'}]
```



```
=== After Modifying Deep Copy ===
Original Library: [Book{title='Animal Farm', author='George Orwell'}, Book{title='Brave New World', author='Aldous Huxley'
}]
Deep Copy: [Book{title='1984', author='George Orwell'}, Book{title='Island', author='Aldous Huxley'}]
```

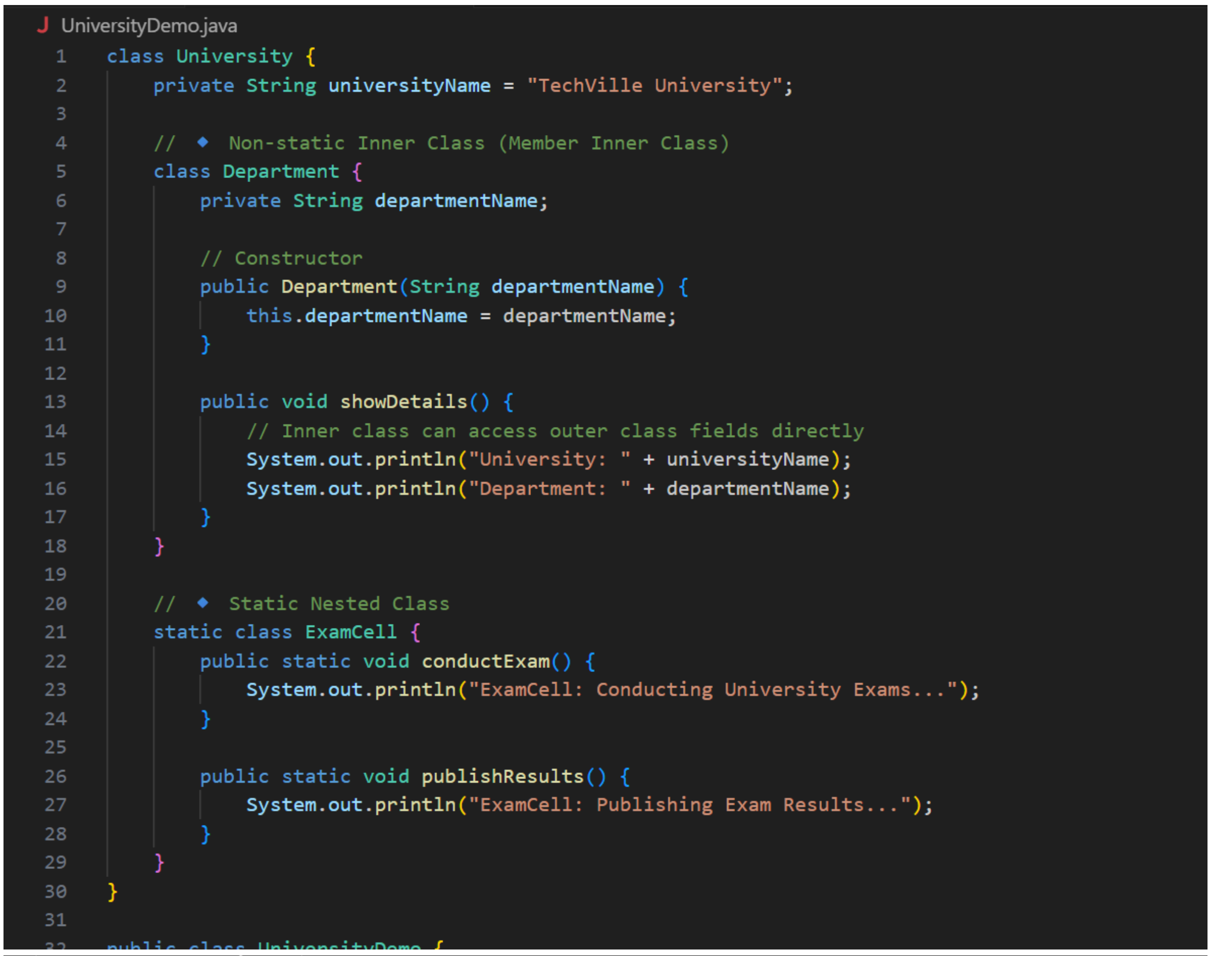
Problem Statement 5:

Create an University class with a non-static inner class Department and a static nested class ExamCell. The Department class should access outer class data, while the ExamCell performs general exam operations. Demonstrate access of both inner types from the main method.

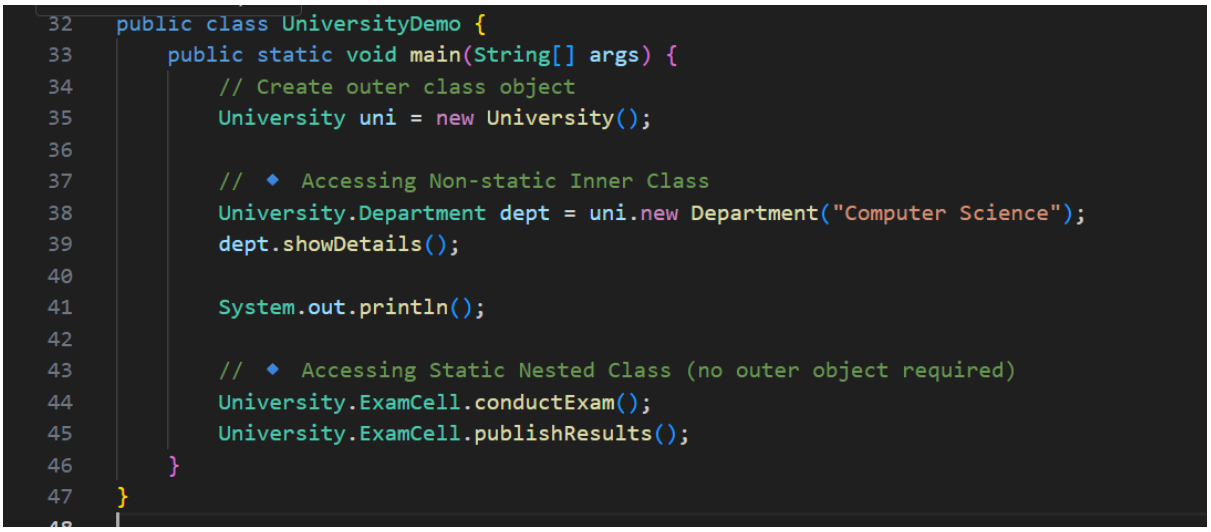
Hints:

- Use Outer.Inner syntax to create a member inner class object.
- Access outer class fields directly from member inner class.
- Use class name to access static nested class methods.

UniversityDemo.java

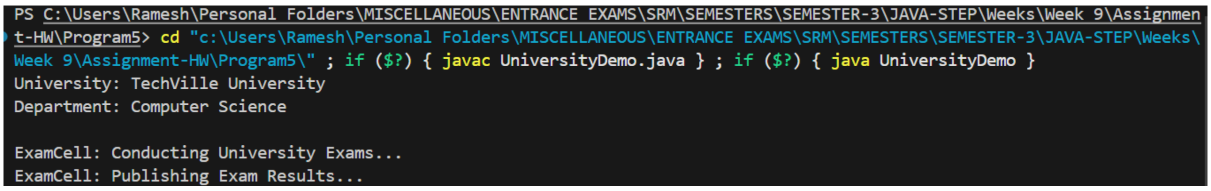


```java
class University {
    private String universityName = "TechVille University";

    // ◆ Non-static Inner Class (Member Inner Class)
    class Department {
        private String departmentName;

        // Constructor
        public Department(String departmentName) {
            this.departmentName = departmentName;
        }

        public void showDetails() {
            // Inner class can access outer class fields directly
            System.out.println("University: " + universityName);
            System.out.println("Department: " + departmentName);
        }
    }

    // ◆ Static Nested Class
    static class ExamCell {
        public static void conductExam() {
            System.out.println("ExamCell: Conducting University Exams...");
        }

        public static void publishResults() {
            System.out.println("ExamCell: Publishing Exam Results...");
        }
    }
}

```



```java
public class UniversityDemo {
    public static void main(String[] args) {
        // Create outer class object
        University uni = new University();

        // ◆ Accessing Non-static Inner Class
        University.Department dept = uni.new Department("Computer Science");
        dept.showDetails();

        System.out.println();

        // ◆ Accessing Static Nested Class (no outer object required)
        University.ExamCell.conductExam();
        University.ExamCell.publishResults();
    }
}
```

OUTPUT→



```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 9\Assignmen
t-HW\Program5> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\
Week 9\Assignment-HW\Program5\" ; if ($?) { javac UniversityDemo.java } ; if ($?) { java UniversityDemo }
University: TechVille University
Department: Computer Science

ExamCell: Conducting University Exams...
ExamCell: Publishing Exam Results...
```

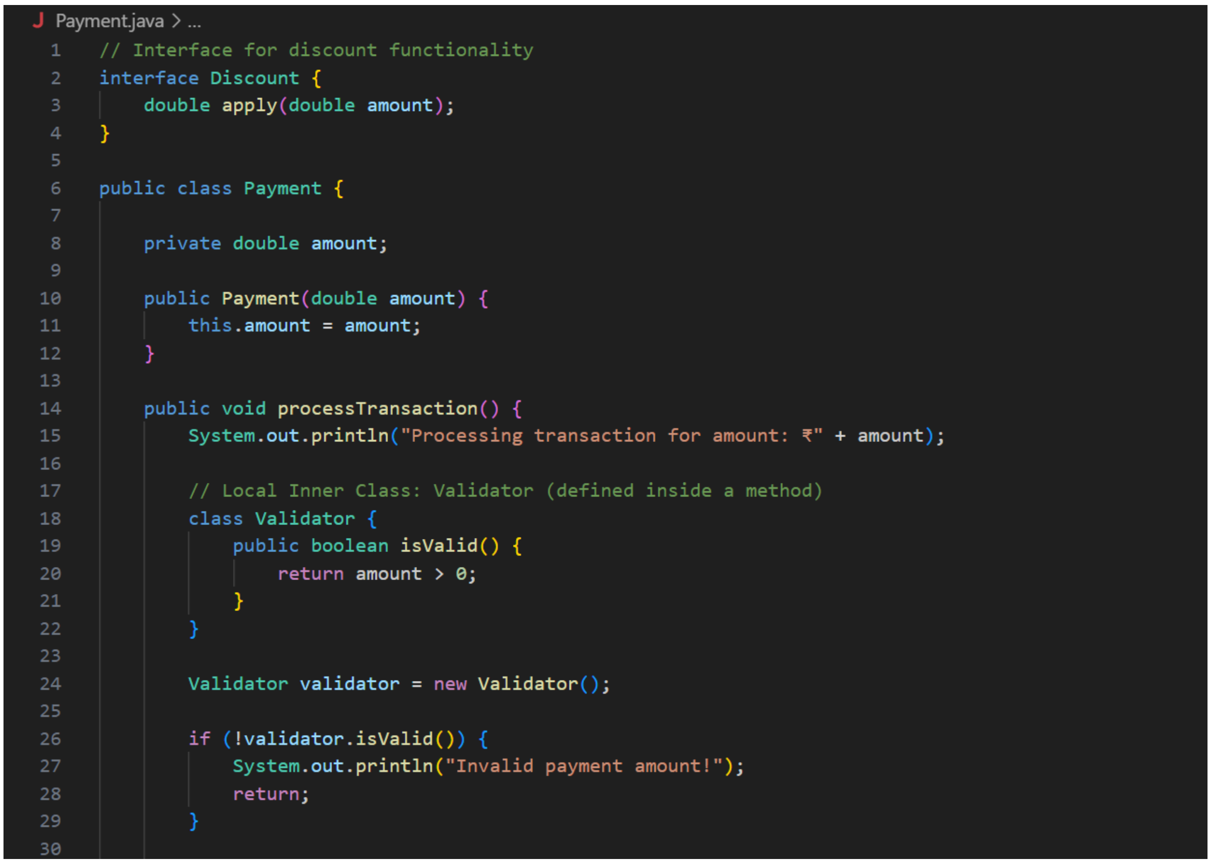
Problem Statement 6:

Create a Payment class with a method processTransaction(). Inside it, define a local inner class Validator that checks if payment amount is valid. Also, create an anonymous inner class implementing an interface Discount to apply discount dynamically.

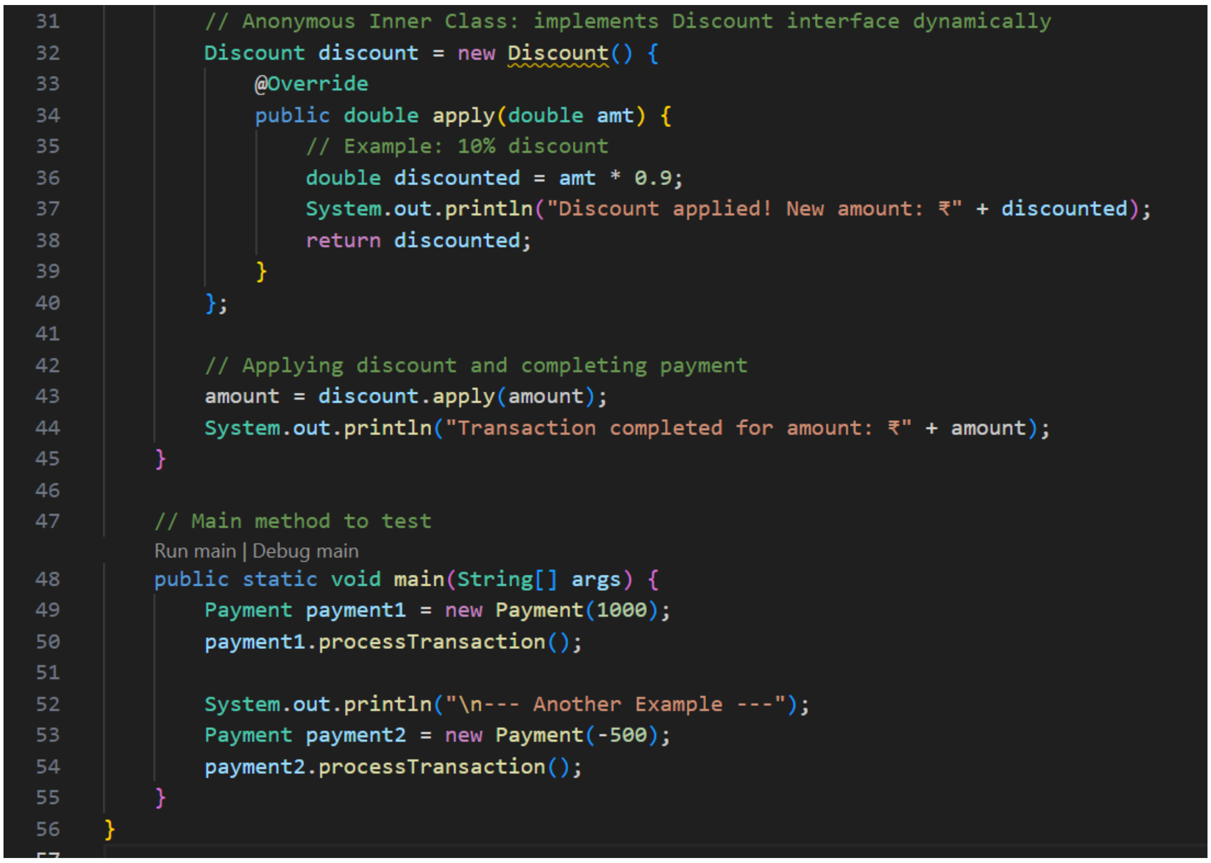
Hints:

- Define local inner class inside a method body.
- Use anonymous inner class for one-time interface implementation.
- Call methods of both classes inside processTransaction().

Payment.java

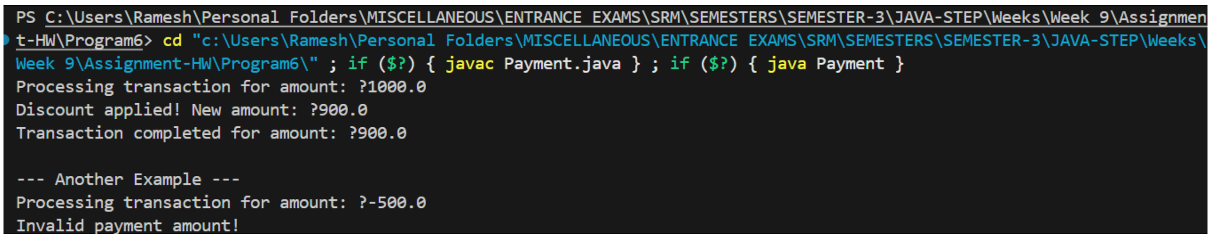


```java
// Interface for discount functionality
interface Discount {
    double apply(double amount);
}

public class Payment {

    private double amount;

    public Payment(double amount) {
        this.amount = amount;
    }

    public void processTransaction() {
        System.out.println("Processing transaction for amount: ₹" + amount);

        // Local Inner Class: Validator (defined inside a method)
        class Validator {
            public boolean isValid() {
                return amount > 0;
            }
        }

        Validator validator = new Validator();

        if (!validator.isValid()) {
            System.out.println("Invalid payment amount!");
            return;
        }

```



```java
        // Anonymous Inner Class: implements Discount interface dynamically
        Discount discount = new Discount() {
            @Override
            public double apply(double amt) {
                // Example: 10% discount
                double discounted = amt * 0.9;
                System.out.println("Discount applied! New amount: ₹" + discounted);
                return discounted;
            }
        };

        // Applying discount and completing payment
        amount = discount.apply(amount);
        System.out.println("Transaction completed for amount: ₹" + amount);
    }

    // Main method to test
    public static void main(String[] args) {
        Payment payment1 = new Payment(1000);
        payment1.processTransaction();

        System.out.println("\n--- Another Example ---");
        Payment payment2 = new Payment(-500);
        payment2.processTransaction();
    }
}
```

OUTPUT→



```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 9\Assignmen
t-HW\Program6> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\
Week 9\Assignment-HW\Program6\" ; if ($?) { javac Payment.java } ; if ($?) { java Payment }
Processing transaction for amount: ?1000.0
Discount applied! New amount: ?900.0
Transaction completed for amount: ?900.0

--- Another Example ---
Processing transaction for amount: ?-500.0
Invalid payment amount!
```