Week 9 - S9 - Advanced OOP - Object Class Methods, Inner Classes - Lab Problem

Name: Ramesh Harisabapathi Chettiar

Date of Submission:15/10/2025

QNO1→

Create a Book class with title and author fields. Override the equals() method to

compare two books based on their title and author. Demonstrate the difference between == and

.equals() using two Book objects.

Hints:

- Use .equals() for content comparison and == for reference comparison
- Override equals() properly to avoid reference equality
- Use @Override annotation when redefining equals()

BookDemo.java

```java
class Book {
    private String title;
    private String author;

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

    // Override equals() for content comparison
    @Override
    public boolean equals(Object obj) {
        // Check if both references point to same object
        if (this == obj)
            return true;

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

        // Typecast obj and compare fields
        Book other = (Book) obj;
        return this.title.equals(other.title) && this.author.equals(other.author);
    }
}

public class BookDemo {
    public static void main(String[] args) {
        Book book1 = new Book("1984", "George Orwell");
        Book book2 = new Book("1984", "George Orwell");
        Book book3 = book1; // Same reference as book1

        // == compares references
        System.out.println("book1 == book2: " + (book1 == book2));  // false
        System.out.println("book1 == book3: " + (book1 == book3));  // true

        // equals() compares content
        System.out.println("book1.equals(book2): " + book1.equals(book2)); // true
        System.out.println("book1.equals(book3): " + book1.equals(book3)); // true
    }
}
```

OUTPUT→

```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 9\Lab Probl
ems\Program1> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\W
eek 9\Lab Problems\Program1\" ; if ($?) { javac BookDemo.java } ; if ($?) { java BookDemo }
book1 == book2: false
book1 == book3: true
book1.equals(book2): true
book1.equals(book3): true
```

QNO2→

Create a Car class with brand, model, and price fields. Override the toString() method to display object details. In the main method, print the class name of an object using getClass().getName().

Hints:

- Use toString() for readable object representation
- Use getClass() to get runtime class information
- Print both the object (to invoke toString()) and its class name

CarDemo.java

```java
class Car {
    private String brand;
    private String model;
    private double price;

    // Constructor
    public Car(String brand, String model, double price) {
        this.brand = brand;
        this.model = model;
        this.price = price;
    }

    // Override toString() to display car details in readable form
    @Override
    public String toString() {
        return "Car Details -> Brand: " + brand + ", Model: " + model + ", Price: ₹" + price;
    }
}

public class CarDemo {
    public static void main(String[] args) {
        // Create a Car object
        Car car1 = new Car("Tesla", "Model S", 8999999.99);

        // Print object (calls toString() automatically)
        System.out.println(car1);

        // Print class name using getClass().getName()
        System.out.println("Class Name: " + car1.getClass().getName());
    }
}
```

OUTPUT→

```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 9\Lab Prob
ems\Program2> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\W
eek 9\Lab Problems\Program2\" ; if ($?) { javac CarDemo.java } ; if ($?) { java CarDemo }
Car Details -> Brand: Tesla, Model: Model S, Price: ?8999999.99
Class Name: Car
```

QNO3→

Create a Student class with id and name fields. Override both equals() and hashCode() methods to ensure two students with the same id are treated as equal. Demonstrate storing Student objects in a HashSet and show how duplicates are handled.

Hints:

- Use Objects.hash(id) for hashCode()
- Ensure consistent results for equals() and hashCode()
- Print HashSet elements to verify duplicates are avoided

StudentDemo.java

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

class Student {
    private int id;
    private String name;

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

    // Override equals() to compare based on 'id'
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;

        if (obj == null || getClass() != obj.getClass())
            return false;

        Student other = (Student) obj;
        return this.id == other.id; // equality based on id only
    }

    // Override hashCode() to be consistent with equals()
    @Override
    public int hashCode() {
        return Objects.hash(id);
    }
    // Override toString() for readable output
    @Override
    public String toString() {
        return "Student{id=" + id + ", name='" + name + "'}";
    }
}

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

        // Create Student objects
        Student s1 = new Student(101, "Alice");
        Student s2 = new Student(102, "Bob");
        Student s3 = new Student(101, "Charlie"); // same id as s1 → treated as duplicate

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

        // Print all students (duplicates avoided automatically)
        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\Lab Prob
ems\Program3> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\W
eek 9\Lab Problems\Program3\" ; if ($?) { javac StudentDemo.java } ; if ($?) { java StudentDemo }
Students in HashSet:
Student{id=101, name='Alice'}
Student{id=102, name='Bob'}
```

QNO4→

Create a Person class with name and Address object as fields. Implement Cloneable and demonstrate both shallow and deep cloning.

Hints:

- Use implements Cloneable and super.clone()
- For deep copy, create a new Address object manually in clone()
- Print original and cloned objects to compare changes

CloneDemo.java

```java
class Address {
    String city;
    String state;

    // Constructor
    public Address(String city, String state) {
        this.city = city;
        this.state = state;
    }

    @Override
    public String toString() {
        return city + ", " + state;
    }
}

class Person implements Cloneable {
    String name;
    Address address;

    // Constructor
    public Person(String name, Address address) {
        this.name = name;
        this.address = address;
    }

    // Shallow clone: copies only references
    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone(); // default shallow copy
    }
```

```java
    // Deep clone: manually copy inner object
    protected Person deepClone() throws CloneNotSupportedException {
        Person cloned = (Person) super.clone();
        cloned.address = new Address(this.address.city, this.address.state); // new Address object
        return cloned;
    }

    @Override
    public String toString() {
        return "Person{name='" + name + "', address=" + address + "}";
    }
}

public class CloneDemo {
    Run main | Debug main
    public static void main(String[] args) throws CloneNotSupportedException {
        Address addr = new Address("Chennai", "Tamil Nadu");
        Person original = new Person("Ramesh", addr);

        // Shallow copy
        Person shallowCopy = (Person) original.clone();

        // Deep copy
        Person deepCopy = original.deepClone();

        System.out.println("=== Before modification ===");
        System.out.println("Original: " + original);
        System.out.println("Shallow Copy: " + shallowCopy);
        System.out.println("Deep Copy: " + deepCopy);

        // Modify original address
        original.address.city = "Bangalore";

        System.out.println("\n=== After modifying original address ===");
        System.out.println("Original: " + original);
        System.out.println("Shallow Copy: " + shallowCopy);
        System.out.println("Deep Copy: " + deepCopy);
    }
}
```

OUTPUT→

```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 9\Lab Probl
ems\Program4> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\W
eek 9\Lab Problems\Program4\" ; if ($?) { javac CloneDemo.java } ; if ($?) { java CloneDemo }
Original: Person{name='Ramesh', address=Chennai, Tamil Nadu}
Shallow Copy: Person{name='Ramesh', address=Chennai, Tamil Nadu}
Deep Copy: Person{name='Ramesh', address=Chennai, Tamil Nadu}

=== After modifying original address ===
Original: Person{name='Ramesh', address=Bangalore, Tamil Nadu}
Shallow Copy: Person{name='Ramesh', address=Bangalore, Tamil Nadu}
Deep Copy: Person{name='Ramesh', address=Chennai, Tamil Nadu}
```