LAB PROBLEM 1: Comparing Objects Using equals() and == (Any Four)
Topic: Object Class Methods – equals() vs ==
Problem Statement:
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.

```java
// File: BookComparison.java
class Book {
    private String title;
    private String author;

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

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

        // Check if obj is an instance of Book
        if (obj instanceof Book) {
            Book other = (Book) obj;
            // Compare title and author
            return this.title.equals(other.title) &&
this.author.equals(other.author);
        }

        return false;
    }
}

public class BookComparison {
    public static void main(String[] args) {
        // Create two separate objects with same content
        Book b1 = new Book("Java Programming", "James Gosling");
        Book b2 = new Book("Java Programming", "James Gosling");

        // Create a reference copy
        Book b3 = b1;

        System.out.println("Comparing b1 and b2 using == : " + (b1 == b2));
// false (different objects)
        System.out.println("Comparing b1 and b2 using equals(): " +
b1.equals(b2)); // true (same content)

        System.out.println("Comparing b1 and b3 using == : " + (b1 == b3));
// true (same reference)
        System.out.println("Comparing b1 and b3 using equals(): " +
b1.equals(b3)); // true (same object)
    }
}
```

```
Comparing b1 and b2 using == : false
Comparing b1 and b2 using equals(): true
Comparing b1 and b3 using == : true
Comparing b1 and b3 using equals(): true
```

LAB PROBLEM 2: toString() and getClass() Usage
Topic: Object Class Methods – toString(), getClass()
Problem Statement:
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().

```
// File: CarInfo.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 object details
```

```java
    @Override
    public String toString() {
        return "Car Details: " +
               "Brand = " + brand +
               ", Model = " + model +
               ", Price = ₹" + price;
    }
}

public class CarInfo {
    public static void main(String[] args) {
        // Create Car object
        Car car1 = new Car("Tesla", "Model S", 79999.99);

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

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

```
Car Details: Brand = Tesla, Model = Model S, Price = ₹79999.99
Class Name: Car
```

LAB PROBLEM 3: hashCode() and equals() Contract
Topic: Object Class Methods – hashCode() and equals() Relationship
Problem Statement:
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.

```java
// File: StudentHashDemo.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 students by id
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj instanceof Student) {
            Student other = (Student) obj;
            return this.id == other.id;
        }
        return false;
    }

    // Override hashCode() to match the equality rule
    @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 StudentHashDemo {
    public static void main(String[] args) {
        // Create HashSet of students
        HashSet<Student> students = new HashSet<>();

        // Add students
        students.add(new Student(101, "Alice"));
        students.add(new Student(102, "Bob"));
        students.add(new Student(101, "Alice (Duplicate ID)"));

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

        System.out.println("\nNote: Student with duplicate ID is not added again!");
    }
}
```

```
Students in HashSet:
Student{id=101, name='Alice'}
Student{id=102, name='Bob'}

Note: Student with duplicate ID is not added again!
```

LAB PROBLEM 4: Cloning an Object
Topic: Object Cloning – Shallow Copy vs Deep Copy
Problem Statement:
Create a Person class with name and Address object as fields. Implement Cloneable and demonstrate both shallow and deep cloning.

```java
// File: CloningDemo.java

class Address {
    String city;
    String state;

    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;

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

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

    // Deep clone
    protected Person deepClone() throws CloneNotSupportedException {
        Person cloned = (Person) super.clone();
        // Create a new Address object for deep copy
        cloned.address = new Address(this.address.city,
this.address.state);
        return cloned;
    }

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

public class CloningDemo {
    public static void main(String[] args) throws
CloneNotSupportedException {

        Address addr = new Address("Hyderabad", "Telangana");
        Person p1 = new Person("Rahul", addr);

        // Shallow copy
        Person p2 = (Person) p1.clone();

        // Deep copy
        Person p3 = p1.deepClone();

        System.out.println("Before modification:");
        System.out.println("Original: " + p1);
        System.out.println("Shallow Clone: " + p2);
        System.out.println("Deep Clone: " + p3);
        // Modify original address
        p1.address.city = "Bangalore";

        System.out.println("\nAfter modifying original address city:");
        System.out.println("Original: " + p1);
        System.out.println("Shallow Clone: " + p2);
        System.out.println("Deep Clone: " + p3);
    }
}
```

```
Before modification:
Original: Person{name='Rahul', address=Hyderabad, Telangana}
Shallow Clone: Person{name='Rahul', address=Hyderabad, Telangana}
Deep Clone: Person{name='Rahul', address=Hyderabad, Telangana}

After modifying original address city:
Original: Person{name='Rahul', address=Bangalore, Telangana}
Shallow Clone: Person{name='Rahul', address=Bangalore, Telangana}
Deep Clone: Person{name='Rahul', address=Hyderabad, Telangana}
```
