Topic 1: Using toString() and getClass()
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().

```java
// File: EmployeeInfo.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() to display employee details
    @Override
    public String toString() {
        return "Employee Details: " +
               "ID = " + id +
               ", Name = " + name +
               ", Salary = ₹" + salary;
    }
}

public class EmployeeInfo {
    public static void main(String[] args) {
        // Create multiple Employee objects
        Employee e1 = new Employee(101, "John", 55000);
        Employee e2 = new Employee(102, "Mary", 60000);
        Employee e3 = new Employee(103, "Alex", 70000);

        // Print details and class name of each object
        System.out.println(e1);
        System.out.println("Class Name: " + e1.getClass().getName());
        System.out.println();

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

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

```
Employee Details: ID = 101, Name = John, Salary = ₹55000.0
Class Name: Employee

Employee Details: ID = 102, Name = Mary, Salary = ₹60000.0
Class Name: Employee

Employee Details: ID = 103, Name = Alex, Salary = ₹70000.0
Class Name: Employee
```

Topic 2: equals() vs ==
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.

```
// File: ProductComparison.java
class Product {
    private int productId;
    private String productName;

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

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

```java
        // Check if obj is of type Product
        if (obj instanceof Product) {
            Product other = (Product) obj;
            // Compare by productId (logical equality)
            return this.productId == other.productId;
        }

        return false;
    }

    @Override
    public String toString() {
        return "Product ID: " + productId + ", Name: " + productName;
    }
}

public class ProductComparison {
    public static void main(String[] args) {
        // Create two separate objects with same data
        Product p1 = new Product(101, "Laptop");
        Product p2 = new Product(101, "Laptop");

        // Create another reference pointing to same object
        Product p3 = p1;

        // Print details
        System.out.println("p1: " + p1);
        System.out.println("p2: " + p2);
        System.out.println("p3: " + p3);
        System.out.println();

        // Compare using ==
        System.out.println("p1 == p2 : " + (p1 == p2)); // false (different
memory references)
        System.out.println("p1 == p3 : " + (p1 == p3)); // true (same
reference)
        System.out.println();

        // Compare using equals()
        System.out.println("p1.equals(p2) : " + p1.equals(p2)); // true
(same productId)
        System.out.println("p1.equals(p3) : " + p1.equals(p3)); // true
(same object)
```

```
    }
}
```

```
p1: Product ID: 101, Name: Laptop
p2: Product ID: 101, Name: Laptop
p3: Product ID: 101, Name: Laptop

p1 == p2 : false
p1 == p3 : true

p1.equals(p2) : true
p1.equals(p3) : true
```

Topic 3: hashCode() and equals() Contract
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.

```java
// File: StudentHashSetDemo.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 instanceof Student) {
            Student other = (Student) obj;
            return this.rollNo == other.rollNo; // logical equality
        }
        return false;
    }

    // Override hashCode() to match equality condition
    @Override
    public int hashCode() {
        return Objects.hash(rollNo);
    }

    // For better display
    @Override
    public String toString() {
        return "Student{Roll No=" + rollNo + ", Name='" + name + "'}";
    }
}

public class StudentHashSetDemo {
    public static void main(String[] args) {
        // Create a HashSet of Student objects
        HashSet<Student> studentSet = new HashSet<>();

        // Add students
        studentSet.add(new Student(101, "Alice"));
        studentSet.add(new Student(102, "Bob"));
        studentSet.add(new Student(101, "Alicia (Duplicate Roll No)")); //
Duplicate rollNo

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

        System.out.println("\nNote: Duplicate roll numbers are not added
again due to equals() and hashCode().");
    }
}
```

```
Students stored in HashSet:
Student{Roll No=101, Name='Alice'}
```

Student{Roll No=102, Name='Bob'}

Note: Duplicate roll numbers are not added again due to equals() and hashCode().

Topic 4: Deep vs Shallow Cloning of Objects
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.

```java
// File: LibraryCloneDemo.java
import java.util.ArrayList;
import java.util.List;

class Book implements Cloneable {
    String title;
    String author;

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

    // Clone method for Book
    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone(); // Shallow copy (no nested objects here)
    }

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

class Library implements Cloneable {
    String libraryName;
    List<Book> books;

    public Library(String libraryName) {
        this.libraryName = libraryName;
        this.books = new ArrayList<>();
    }

    public void addBook(Book book) {
        books.add(book);
    }

    // Shallow clone: only top-level fields are copied (book references
remain shared)
    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();
    }

    // Deep clone: clone all Book objects inside the list
    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 "Library{Name='" + libraryName + "', Books=" + books + "}";
    }
}

public class LibraryCloneDemo {
    public static void main(String[] args) throws
CloneNotSupportedException {
        // Create library with books
        Library lib1 = new Library("Central Library");
        lib1.addBook(new Book("Java Basics", "James Gosling"));
        lib1.addBook(new Book("Python Programming", "Guido van Rossum"));

        // Shallow clone
        Library shallowClone = (Library) lib1.clone();
        // Deep clone
        Library deepClone = lib1.deepClone();

        System.out.println("Before modification:");
        System.out.println("Original: " + lib1);
        System.out.println("Shallow Clone: " + shallowClone);
        System.out.println("Deep Clone: " + deepClone);

        // Modify a book title in cloned library
        shallowClone.books.get(0).title = "C++ for Beginners";
        deepClone.books.get(1).title = "Advanced Python";

        System.out.println("\nAfter modifying book titles:");
        System.out.println("Original: " + lib1);
        System.out.println("Shallow Clone: " + shallowClone);
        System.out.println("Deep Clone: " + deepClone);
    }
}
```

```
Before modification:
Original: Library{Name='Central Library', Books=[Book{Title='Java Basics', Author='James Gosling'}, Book{Title='Python Programming', Author='Guido van Rossum'}]}
Shallow Clone: Library{Name='Central Library', Books=[Book{Title='Java Basics', Author='James Gosling'}, Book{Title='Python Programming', Author='Guido van Rossum'}]}
Deep Clone: Library{Name='Central Library', Books=[Book{Title='Java Basics', Author='James Gosling'}, Book{Title='Python Programming', Author='Guido van Rossum'}]}

After modifying book titles:
Original: Library{Name='Central Library', Books=[Book{Title='C++ for Beginners', Author='James Gosling'}, Book{Title='Python Programming', Author='Guido van Rossum'}]}
Shallow Clone: Library{Name='Central Library', Books=[Book{Title='C++ for Beginners', Author='James Gosling'}, Book{Title='Python Programming', Author='Guido van Rossum'}]}
Deep Clone: Library{Name='Central Library', Books=[Book{Title='Java Basics', Author='James Gosling'}, Book{Title='Advanced Python', Author='Guido van Rossum'}]}
```

Topic 5: Member and Static Inner Classes
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.

```java
// File: UniversityDemo.java
class University {
    private String universityName = "ABC University";

    // ✅ Member (non-static) inner class
    class Department {
        private String deptName;

        Department(String deptName) {
            this.deptName = deptName;
        }

        public void showDepartmentInfo() {
            // Access outer class field directly
            System.out.println("University: " + universityName);
            System.out.println("Department: " + deptName);
        }
    }

    // ✅ Static nested class
    static class ExamCell {
        public static void conductExam() {
            System.out.println("ExamCell: Exams are being conducted for all
departments.");
        }

        public void announceResults() {
            System.out.println("ExamCell: Results are published.");
        }
    }
}

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

        // 🔹 Create inner (non-static) class object using outer class
instance
        University.Department dept = uni.new Department("Computer
Science");
        dept.showDepartmentInfo();
        System.out.println();

        // 🔹 Access static nested class methods
        University.ExamCell.conductExam(); // static method -- called using
class name
        University.ExamCell examCellObj = new University.ExamCell();
        examCellObj.announceResults(); // non-static method -- called using
object
    }
}
```

```
University: ABC University
Department: Computer Science

ExamCell: Exams are being conducted for all departments.
ExamCell: Results are published.
```

Topic 6: Local and Anonymous Inner Classes
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.

```
// File: PaymentDemo.java

interface Discount {
    double apply(double amount);
}

class Payment {

    public void processTransaction(double amount) {
        // ✅ Local Inner Class (defined inside a method)
        class Validator {
            public boolean isValid(double amt) {
                return amt > 0; // simple validation rule
            }
        }
```

```java
        // Create instance of local inner class
        Validator validator = new Validator();

        if (validator.isValid(amount)) {
            System.out.println("Payment amount is valid: ₹" + amount);
        } else {
            System.out.println("Invalid payment amount!");
            return;
        }

        // ✅ Anonymous Inner Class implementing Discount interface
        Discount discount = new Discount() {
            @Override
            public double apply(double amt) {
                System.out.println("Applying 10% festival discount...");
                return amt * 0.9; // 10% off
            }
        };

        // Apply discount
        double finalAmount = discount.apply(amount);
        System.out.println("Final amount after discount: ₹" + finalAmount);
    }
}

public class PaymentDemo {
    public static void main(String[] args) {
        Payment payment = new Payment();

        System.out.println("---- Transaction 1 ----");
        payment.processTransaction(1000);

        System.out.println("\n---- Transaction 2 ----");
        payment.processTransaction(-500); // Invalid amount example
    }
}
```

```
---- Transaction 1 ----
Payment amount is valid: ₹1000.0
Applying 10% festival discount...
Final amount after discount: ₹900.0

---- Transaction 2 ----
```

Invalid payment amount!