Experiment Number : 07 **Date:**

METHOD OVERLOADING AND METHOD OVERRIDING

Aim:

To understand and implement method overloading and method overriding.

PRE LAB EXERCISE

QUESTIONS

- ✓ What is method overloading?

Method overloading is a feature in Java where **multiple methods in the same class have the same name but different parameters** (different number, type, or order of parameters).

It is also called **compile-time polymorphism** because it is resolved at compile time.

- ✓ What is method overriding?

Method overriding occurs when a **subclass provides a specific implementation for a method that is already defined in its parent class with the same name, same parameters, and same return type**.

It is also called **runtime polymorphism** because it is resolved at runtime.

- ✓ Difference between overloading and overriding.

Method Overloading	Method Overriding
Occurs within the same class	Occurs in different classes (parent & child)
Same method name, different parameters	Same method name and same parameters

Compile-time polymorphism	Runtime polymorphism
Return type may or may not change	Return type must be same or covariant
No inheritance required	Inheritance is required

IN LAB EXERCISE

Objective:

To demonstrate compile-time and runtime polymorphism.

PROGRAMS:

1.Student Result System (Method Overriding)

Description:

- Base class Student has method displayResult().
- Subclasses UGStudent and PGStudent override the method to show different grading systems.

Code :

```
import java.util.Scanner;

// Base class
class Student {
   String name;

   void displayResult() {
      System.out.println("Student Result");
   }
}

// UG Student subclass
class UGStudent extends Student {
```

```
    int marks;

    UGStudent(String n, int m) {
        name = n;
        marks = m;
    }

    @Override
    void displayResult() {
        double percentage = (marks / 100.0) * 100;
        System.out.println("UG Student: " + name);
        System.out.println("Marks: " + marks);
        System.out.println("Percentage: " + percentage + "%");
    }
}

// PG Student subclass
class PGStudent extends Student {
    double gpa;

    PGStudent(String n, double g) {
        name = n;
        gpa = g;
    }

    @Override
    void displayResult() {
        System.out.println("PG Student: " + name);
        System.out.println("GPA: " + gpa + " / 10");
    }
}
```

```
// Main class
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // Input for UG student
        System.out.print("Enter UG Student Name: ");
        String ugName = sc.nextLine();
        System.out.print("Enter UG Student Marks (out of 100): ");
        int ugMarks = sc.nextInt();
        sc.nextLine(); // consume newline

        // Input for PG student
        System.out.print("Enter PG Student Name: ");
        String pgName = sc.nextLine();
        System.out.print("Enter PG Student GPA (0-10): ");
        double pgGpa = sc.nextDouble();

        // Create objects
        Student s1 = new UGStudent(ugName, ugMarks);
        Student s2 = new PGStudent(pgName, pgGpa);

        System.out.println("\n--- Student Results ---");
        s1.displayResult();
        System.out.println();
        s2.displayResult();

        sc.close();
    }
}
```

OUTPUT:

Sample Input:

Enter UG Student Name: Ruddhida

Enter UG Student Marks (out of 100): 85

Enter PG Student Name: Rudd

Enter PG Student GPA (0-10): 9.2

Output:

--- Student Results ---

UG Student: Ram

Marks: 85

Percentage: 85.0%

PG Student: Ravi

GPA: 9.2 / 10

```
Enter UG Student Name: Ruddhida
Enter UG Student Marks (out of 100): 85
Enter PG Student Name: Rudd
Enter PG Student GPA (0-10): 9.2

--- Student Results ---
UG Student: Ruddhida
Marks: 85
Percentage: 85.0%

PG Student: Rudd
GPA: 9.2 / 10
```

2. Calculator Program (Method Overloading)

Description:
Create a Calculator class with multiple add() methods to calculate:

- Addition of 2 integers
- Addition of 3 integers
- Addition of 2 double numbers

Code:

```
import java.util.Scanner;
class Calculator {
    int add(int a, int b) {
        return a + b;
    }

    int add(int a, int b, int c) {
        return a + b + c;
    }

    double add(double a, double b) {
        return a + b;
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Calculator calc = new Calculator();

        System.out.print("Enter two integers: ");
        int x = sc.nextInt();
        int y = sc.nextInt();
        System.out.println("Sum of two integers: " + calc.add(x, y));

        System.out.print("Enter three integers: ");
        int p = sc.nextInt();
        int q = sc.nextInt();
        int r = sc.nextInt();
        System.out.println("Sum of three integers: " + calc.add(p, q, r));
```

```
        System.out.print("Enter two decimal numbers: ");
        double a = sc.nextDouble();
        double b = sc.nextDouble();
        System.out.println("Sum of two doubles: " + calc.add(a, b));

        sc.close();
    }
}
```

Output:

Sample Input:

Enter two integers: 10 20

Enter three integers: 5 10 15

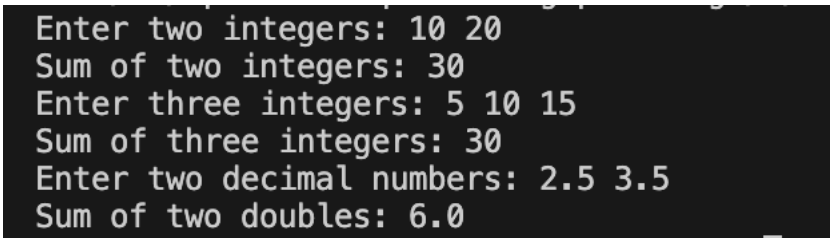
Enter two decimal numbers: 2.5 3.5

Output:

Sum of two integers: 30

Sum of three integers: 30

Sum of two doubles: 6.0



POST LAB EXERCISE

- ✓ Is return type important in method overloading and method overriding?

  In method overloading:

Return type is **not important**. Overloading depends only on the number, type, or order of parameters, not on return type.

In method overriding:

Return type **is important**. It must be the same as the parent method or a **covariant return type** (i.e., a subclass of the parent's return type).

- ✓ Can you overload a method by changing only the return type?

No. You cannot overload a method by changing only the return type.

If two methods have the same name and same parameters but different return types, it will cause a **compile-time error**.

- ✓ Can static methods be overridden? Can they be overloaded?

Can static methods be overridden? — No.

Static methods belong to the class, not to the object, so they cannot be overridden. If a child class defines a static method with the same name, it is called **method hiding**, not overriding.

Can static methods be overloaded? — Yes.

Static methods **can be overloaded** because overloading is resolved at compile time based on parameters.

- ✓ Can a method be overridden if the parameter list is different?

No.

If the parameter list is different, it is **method overloading**, not overriding.

For overriding, the method name **and** parameter list must be exactly the same as in the parent class.

Result:

Thus the method overloading and overriding concepts were implemented and executed successfully.

ASSESSMENT

Description	Max Marks	Marks Awarded
Pre Lab Exercise	**5**	
In Lab Exercise	**10**	
Post Lab Exercise	**5**	
Viva	**10**	
Total	**30**	
	Faculty Signature	