Experiment Number : 07 **Date: 06/02/2026**

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 where multiple methods have the same name but different parameter lists. The difference can be in number, type, or order of parameters. It is resolved at compile time. Overloading improves code readability and flexibility. It allows performing similar operations with different inputs.

- ✓ What is method overriding?

  Method overriding occurs when a subclass provides a specific implementation of a method already defined in its parent class. The method name and parameter list must be the same. It is resolved at runtime. Overriding supports dynamic polymorphism. It allows changing behavior of inherited methods.

- ✓ Difference between overloading and overriding.

  Method overloading occurs within the same class, while overriding occurs between parent and child classes. Overloading requires different parameter lists, whereas overriding requires the same signature. Overloading is compile-time polymorphism, overriding is runtime polymorphism. Overloading increases flexibility, while overriding changes inherited behavior.

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: Shankar

Enter UG Student Marks (out of 100): 99

Enter PG Student Name: Niva

Enter PG Student GPA (0-10): 9.0

Output:

--- Student Results ---

UG Student: Shankar

Marks: 99

Percentage: 99.0%

PG Student: Niva

GPA: 9.0 / 10

```
<terminated> main [Java Application] D:\eclipse\plugi
Enter UG Student Name: Shankar
Enter UG Student Marks (out of 100): 99
Enter PG Student Name: Niva
Enter PG Student GPA (0-10): 9

--- Student Results ---
UG Student: Shankar
Marks: 99
Percentage: 99.0%

PG Student: Niva
GPA: 9.0 / 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: 5 7
Enter three integers: 2 5 5
Enter two decimal numbers: 5.7 6.3
Output:

Sum of two integers: 12
Sum of three integers: 12
Sum of two doubles: 12.0

```
Enter two integers: 5 7
Sum of two integers: 12
Enter three integers: 2 5 5
Sum of three integers: 12
Enter two decimal numbers: 5.7 6.3
Sum of two doubles: 12.0
```

POST LAB EXERCISE

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

  In method overloading, the return type alone is not considered for distinguishing methods. Overloaded methods must differ in parameter list. In method overriding, the return type must be the same or a covariant type. The method signature is mainly used to decide overriding. Hence, return type matters more in overriding than overloading.

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

  No, a method cannot be overloaded by changing only the return type. The compiler uses the method name and parameter list to differentiate methods. If only the return type is changed, it causes a compilation error. Overloading requires a change in number, type, or order of parameters. Therefore, return type alone is insufficient.

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

  Static methods cannot be overridden because they belong to the class, not the object. They are resolved at compile time, not runtime. However, static methods can be overloaded by changing the parameter list. This is allowed since overloading is decided at compile time. Hence, static methods support overloading but not overriding.

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

  No, a method cannot be overridden if the parameter list is different. Overriding requires the same method name and parameter list. Changing parameters creates a new method instead of overriding. This is considered method overloading. Therefore, parameter list must be identical for overriding.

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	