SNS COLLEGE OF ENGINEERING

Coimbatore-107

An Autonomous Institution

Redesigning Common Mind & Business Towards Excellence

Build an Entrepreneurial Mindset Through Our Design Thinking FrameWork

COURSE NAME : 23CSB201 & Object Oriented Programming

II YEAR/ III SEMESTER

UNIT – II INHERITANCE, PACKAGES, INTERFACE

Topic: Method Overloading & Object as Argument

Dr.P.Poonkodi

Assistant Professor(SG)

Department of Computer Science and Technology

Introduction

- Method overloading is a feature in Object-Oriented Programming (OOP) where multiple methods in the same class share the same name but have different parameters (different number, type, or order of parameters).
- It allows flexibility and improves code readability

Scenario

- A university enrollment system needs a way to register students. Students can register using different information:
- **Basic registration** – Only with Student ID.
- **Registration with name** – Student ID and Name.
- **Full registration** – Student ID, Name, and Course.
- **Registration with scholarship** – Student ID, Name, Course, and Scholarship details.

Scenario

- A university enrollment system needs a way to register students. Students can register using different information:
- **Basic registration** – Only with Student ID.
- **Registration with name** – Student ID and Name.
- **Full registration** – Student ID, Name, and Course.
- **Registration with scholarship** – Student ID, Name, Course, and Scholarship details.

Example

```
class Student
{
        private int studentId;
        private String name;
        private String course;
        private double scholarshipAmount;
// Method 1: Register with only student ID
        public void register(int studentId)
        {
                this.studentId = studentId;
                System.out.println("Student registered with ID: " + studentId);
        }
```

Example

```
// Method 2: Register with student ID and Name
public void register(int studentId, String name)
{
        this.studentId = studentId;
        this.name = name;
        System.out.println("Student registered: " + name + " (ID: " +
                                        studentId + ")");
}
```

Example

```
// Method 3: Register with student ID, Name, and Course
public void register(int studentId, String name, String course)
{
        this.studentId = studentId;
        this.name = name;
        this.course = course;
        System.out.println("Student registered: " + name + " (ID: " +
                                studentId + ") for course: " + course);
}
```

Example

```
// Method 4: Register with all details including Scholarship
public void register(int studentId, String name, String course, double
scholarshipAmount)
{
        this.studentId = studentId;
        this.name = name;
        this.course = course;
        this.scholarshipAmount = scholarshipAmount;
        System.out.println("Student registered: " + name + " (ID: " +
        studentId + ") for course: " + course+ " with scholarship of $" +
        scholarshipAmount);
}
}
```

Example

```
public class StudentRegistration
{
        public static void main(String[] args)
        {
                Student student1 = new Student();
                student1.register(101);
                Student student2 = new Student();
                student2.register(102, "Alice");
                Student student3 = new Student();
                student3.register(103, "Bob", "Computer Science");
                Student student4 = new Student();
                student4.register(104, "Charlie", "Engineering", 2000.0);
        }
}
```

Advantages

- **Flexibility**:

  Students can register using different sets of details

- **Improved Readability**:

  The method name register remains the same, making it easier to understand

- **Code Reusability**:

  No need to create multiple method names like registerWithID(), registerWithName(), etc

- **Better Maintainability**:

  If logic changes in registration, updating one method is easier than modifying multiple methods

Object as Argument

- objects can be passed as arguments to methods
- This allows methods to work on entire objects rather than just individual values
- It helps in **data encapsulation, code reusability, and modular programming**

Scenario

- A university wants a system where:
  - A student’s details (ID, Name, and Marks) are passed to a method.
  - A method calculates and displays the student’s grade based on marks.

Example

```
class Student
        {
                int studentId;
                String name;
                double marks;
                // Constructor
                Student(int studentId, String name, double marks)
                {
                        this.studentId = studentId;
                        this.name = name;
                        this.marks = marks;
                }
```

Example

```
// Method to display student details
        void display()
        {
                System.out.println("Student ID: " + studentId + ", Name: "
                                        + name + ", Marks: " + marks);
        }
```

Example

```
// Method that accepts a Student object as an argument and calculates grade
void calculateGrade(Student student)
{
        char grade;
        if (student.marks >= 90)
        {
                grade = 'A';
        }
        else if (student.marks >= 80)
        {
                grade = 'B';
        }
```

Example

```
// Method that accepts a Student object as an argument and calculates grade
        else if (student.marks >= 70)
        {
            grade = 'C';
        }
        else if (student.marks >= 60)
        {
            grade = 'D';
        }
        else
        {
            grade = 'F';
        }
    System.out.println(student.name + " received grade: " + grade);
    }
```

Example

```
public class StudentGradeSystem
{
    public static void main(String[] args)
    {
        // Creating Student objects
        Student student1 = new Student(101, "Alice", 85);
        Student student2 = new Student(102, "Bob", 92);
        // Displaying student details
        student1.display();
        student2.display();
        // Calculating grade by passing object as an argument
        student1.calculateGrade(student1);
        student2.calculateGrade(student2);
    }
}
```

Key Takeaways

1. **Passing Objects as Arguments**:

The calculateGrade(Student student) method takes a Student object as an argument and determines the grade

2. **Encapsulation & Code Reusability**:

Instead of writing separate methods for each student, we pass student objects, making the code more reusable

3. **Real-World Use Cases**:

This approach is widely used in banking applications (passing account objects), e-commerce (passing order objects), and healthcare systems (passing patient objects)

References

- Java : the complete Reference (Eleventh Edition), Herbert Schildt, 2018.


Thank You!