



Southeast University

Department of Computer Science and Engineering (CSE)

School of Sciences and Engineering

Semester: (Summer, Year: 2025)

LAB REPORT NO: 05

Course Title: Introduction to Programming Language II (Java) Lab

Course Code: CSE282.2

Batch: 65

Lab Experiment Name: Implementation of constructor overloading,
method overloading and overriding.

Student Details

	Name	ID
1.	Md. Mahmud Hossain	2023200000799

Submission Date : 24-08-25

Course Teacher's Name : Dr. Mohammed Ashikur Rahman

Lab Report Status

Marks:

Comments:.....

Signature:.....

Date:.....

Lab Task 5: Implementation of constructor overloading, method overloading and overriding

OBJECTIVES:

- To be familiar with different types of constructors
- To be familiar with the concept of overloading (Compile time polymorphism)
- To be familiar with the 'this' keyword

PROBLEM:

1. Create a class called Person with properties such as name, age, gender, address. Use constructor overloading, method overloading and the 'this keyword'..
2. Create a class called Employee with properties such as name, id, salary, designation. Use constructor overloading, method overloading and the 'this keyword'..

Solution:

1.

Problem Analysis:

The purpose of this program is to illustrate the concepts of constructor overloading, inheritance, method overriding, and method overloading in Java. The Person class encapsulates common attributes such as name, age, gender, and address, and provides multiple constructors to initialize objects with varying levels of detail. The Student class extends Person, adding attributes like studentId and department. It overrides the displayInfo() method to present both Person and Student details, while also offering an overloaded version to display only student-specific information. The MainPerson class demonstrates the use of inheritance, constructor chaining via the super keyword, and method overriding by creating Person and Student objects and displaying their information in different ways. The task requires designing a class to represent a person with key attributes that are kept private for protection. Encapsulation ensures these attributes are only accessed or changed via specific methods, maintaining data consistency and preventing direct tampering. By adding

inheritance, we create a base class for the properties and methods, allowing the Person class to extend it. This approach makes the code more modular and extensible, as the base could potentially be reused for similar entities. User input isn't directly involved here, but the main method demonstrates setting and displaying properties for multiple objects to verify functionality.

Background Theory:

In Object Variables: Each object has its own set of properties (name, age, etc.), stored as instance variables.

Constructors: Special methods used to initialize objects. Here, the Person class uses constructor overloading to allow flexible initialization (default values, partial info, full info).

Inheritance: The Student class extends Person, inheriting its fields and methods. This avoids code duplication and promotes reusability.

super Keyword: Used to call the parent class constructor and methods. In Student, it initializes the inherited properties.

Method Overriding: Student overrides displayInfo() to extend functionality by printing student-specific details.

Method Overloading: Both Person and Student provide overloaded versions of displayInfo() that accept different parameters, showcasing polymorphism.

main() Method: Acts as the entry point, creating objects and demonstrating constructor overloading, inheritance, method overriding, and overloading.

Algorithm Design:

1. Define a Person class with instance variables: name, age, gender, address.
2. Implement three constructors in Person:
 - A default constructor that sets default values.
 - A constructor with only name and age.
 - A constructor with all properties (name, age, gender, address).
3. Implement an displayInfo() method in Person to print details.
4. Overload displayInfo() in Person to include a custom message.
5. Create a Student class that extends Person and adds studentId and department.

6. Provide two constructors in Student:
 - A default constructor initializing default values.
 - A parameterized constructor that calls the Person constructor using super.
7. Override the displayInfo() method in Student to include both Person and Student details.
8. Overload displayInfo() in Student to optionally show only student details.
9. In the main() method:
 - Create a Person object using the default constructor.
 - Create another Person object using the partial constructor.
 - Create a Student object using the full constructor.
 - Demonstrate displayInfo() calls to show different outputs.

Code:

```
class Person {
    protected String name;
    protected int age;
    protected String gender;
    protected String address;

    public Person() {
        this("Unknown", 0, "Not specified", "Not specified");
    }

    public Person(String name, int age) {
        this(name, age, "Not specified", "Not specified");
    }

    public Person(String name, int age, String gender, String address) {
        this.name = name;
        this.age = age;
        this.gender = gender;
        this.address = address;
    }

    public void displayInfo() {
        System.out.println("Name: " + this.name);
        System.out.println("Age: " + this.age);
        System.out.println("Gender: " + this.gender);
        System.out.println("Address: " + this.address);
        System.out.println("-----");
    }
}
```

```

    }

    public void displayInfo(String message) {
        System.out.println(message);
        displayInfo();
    }
}

class Student extends Person {
    private String studentId;
    private String department;

    public Student() {
        super();
        this.studentId = "Not assigned";
        this.department = "Not assigned";
    }

    public Student(String name, int age, String gender, String address,
        String studentId, String department) {
        super(name, age, gender, address);
        this.studentId = studentId;
        this.department = department;
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Student ID: " + this.studentId);
        System.out.println("Department: " + this.department);
        System.out.println("=====");
    }

    public void displayInfo(String message, boolean showStudentOnly) {
        System.out.println(message);
        if (showStudentOnly) {
            System.out.println("Student ID: " + this.studentId);
            System.out.println("Department: " + this.department);
        } else {
            displayInfo();
        }
        System.out.println("-----");
    }
}

```

```

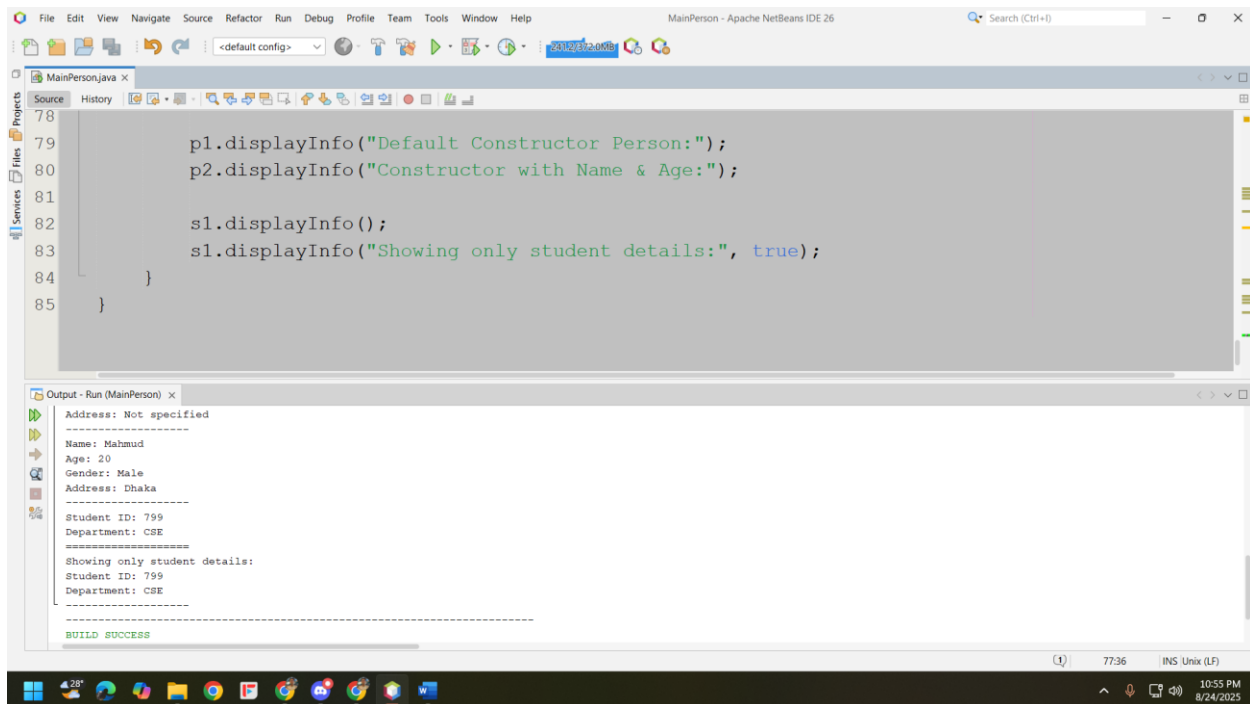
public class MainPerson {
    public static void main(String[] args) {
        Person p1 = new Person();
        Person p2 = new Person("Faujul", 21);
        Student s1 = new Student("mahmud", 20, "Male", "Dhaka", "799", "CSE");

        p1.displayInfo("Default Constructor Person:");
        p2.displayInfo("Constructor with Name & Age:");

        s1.displayInfo();
        s1.displayInfo("Showing only student details:", true);
    }
}

```

Output:



The screenshot shows the Apache NetBeans IDE interface. The main editor window displays the Java code for MainPerson.java. The output window, titled "Output - Run (MainPerson)", shows the following text:

```

Address: Not specified
Name: Mahmud
Age: 20
Gender: Male
Address: Dhaka
-----
Student ID: 799
Department: CSE
=====
Showing only student details:
Student ID: 799
Department: CSE
-----
BUILD SUCCESS

```

Figure 1: Output

2.

Problem Analysis:

The purpose of this program is to illustrate constructor overloading, inheritance, method overriding, and method overloading in Java through an employee-management scenario. The Employee class encapsulates common employee attributes such as name, ID, salary, and designation, and provides multiple constructors to initialize objects with varying levels of detail. It also includes overloaded methods to display either standard employee information or a custom message alongside it. The Manager class extends Employee, adding attributes like department and team size. It overrides the showEmployee() method to include manager-specific details and provides an overloaded version to optionally display only manager-related information. The MainTest class demonstrates these concepts by creating Employee and Manager objects with different constructors and calling the various showEmployee() methods to display information in multiple formats.

Background Theory:

Object Variables: Each Employee or Manager object maintains its own set of properties, such as name, ID, salary, designation, department, and team size.

Constructors: Constructors are special methods used to initialize objects. The Employee class demonstrates constructor overloading with a default constructor, a partial constructor (name and ID), and a full constructor (all properties). The Manager class also includes overloaded constructors, using the super keyword to initialize inherited properties.

Inheritance: The Manager class inherits from Employee, reusing common fields and methods, which avoids code duplication and promotes maintainability.

super Keyword: Used in Manager constructors and overridden methods to invoke the parent class (Employee) constructor and methods, ensuring proper initialization and reuse of code.

Method Overriding: The Manager class overrides showEmployee() to extend the functionality of Employee's method, adding manager-specific details.

Method Overloading: Both Employee and Manager classes provide overloaded versions of showEmployee() to demonstrate polymorphism, allowing methods to handle different parameters or display formats.

main() Method: Acts as the entry point of the program, creating Employee and Manager objects and demonstrating constructor overloading, inheritance, method overriding, and overloading.

Algorithm Design:

1. Define an Employee class with instance variables: name, ID, salary, and designation.
2. Implement three constructors in Employee: a default constructor, a partial constructor with name and ID, and a full constructor with all properties.
3. Implement a showEmployee() method in Employee to display object details.
4. Overload showEmployee() in Employee to include a custom message alongside the details.
5. Define a Manager class that extends Employee and adds department and teamSize attributes.
6. Implement two constructors in Manager: a default constructor and a parameterized constructor that calls Employee's constructor using super.
7. Override the showEmployee() method in Manager to display both Employee and Manager details.
8. Overload showEmployee() in Manager to optionally display only manager-specific information.
9. In the main() method, create Employee objects using different constructors.
10. Create a Manager object and demonstrate calling all versions of showEmployee() to display information in different formats.

Code:

```
class information{
    private String name;
    private int id;
    private double salary;
    private String designation;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getId() {
```



```

        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    public String getDesignation() {
        return designation;
    }

    public void setDesignation(String designation) {
        this.designation = designation;
    }
}

public class Employee extends information{
    public static void main(String[] args) {
        Employee e1 = new Employee();
        e1.setName("Rakib");
        e1.setId(101);
        e1.setSalary(1000.00);
        e1.setDesignation("Manager");

        Employee e2 = new Employee();
        e2.setName("Faujul");
        e2.setId(201);
        e2.setSalary(2000.00);
        e2.setDesignation("IT specialist");

        System.out.println("Employee 1:");
    }
}

```

```

        System.out.println("Name: "+e1.getName());
        System.out.println("Id: "+e1.getId());
        System.out.println("Salary: "+e1.getSalary());
        System.out.println("Designation: "+e1.getDesignation());
        System.out.println("");
        System.out.println("Employee 2:");
        System.out.println("Name: "+e2.getName());
        System.out.println("Id: "+e2.getId());
        System.out.println("Salary: "+e2.getSalary());
        System.out.println("Designation: "+e2.getDesignation());
        System.out.println("");
    }
}

```

Output:

The screenshot shows the Apache NetBeans IDE with the following content:

Source View:

```

class Employee {
    protected String name;
    protected int id;
    protected double salary;
    protected String designation;

    public Employee() {
        this("Not Assigned", 0, 0.0, "Not Assigned");
    }
}

```

Output View (Run - MainTest):

```

-----
Designation: Project Manager
Department: IT
Team Size: 10
=====
Showing only Manager details:
Department: IT
Team Size: 10
-----

BUILD SUCCESS

Total time: 0.968 s
Finished at: 2025-08-24T22:57:41+06:00
-----

```

The IDE interface includes a menu bar (File, Edit, View, etc.), a toolbar, and a status bar at the bottom showing system information like temperature (28°) and time (10:58 PM 8/24/2025).

Figure 2: Output