Invertis University Bareilly

Java Mini Project Title: Student Management Systems

Name: Aditya Verma

Student id: LBCS2024011

Course: B.Tech (CSE) Vth Sem.

Java Trainer: Chetan Dhakad

Objective

To create a small Java program that stores and displays information about students and teachers, while demonstrating all major Object-Oriented Programming (OOPs) concepts.

Concept Overview

Concept	Used In
Class & Object	Person, Student, Teacher, Main
Constructor (Default, Parameterized, Copy)	Implemented in all classes
Encapsulation	Private fields with getters/setters
Inheritance	Student and Teacher inherit from Person
Polymorphism	Method overriding (displayInfo())
Abstraction	Abstract class Person defines base structure

Classes Used in Project

We have three main classes and one main program:

1. Person (Abstract Class) → base class
2. Student (Child Class) → inherits from Person
3. Teacher (Child Class) → inherits from Person
4. Main (Driver Class) → program execution starts here

Detailed Concept Explanation

1️⃣ Class & Object

- Class is a blueprint that defines the structure (variables and methods).
- Objects are instances (real examples) of the class.

👉 In this project:

- Person, Student, and Teacher are classes.
- We create objects like Student s1 = new Student(...) and Teacher t1 = new Teacher(...).

2️⃣ Constructor (Default, Parameterized, Copy)

Constructors initialize objects when they are created.

Type	Description	Example in Project
Default Constructor	No parameters; used for initialization	Person() and Student()
Parameterized Constructor	Takes arguments to initialize variables	Student(String name, int age, int rollNo, double marks)
Copy Constructor	Creates a new object by copying values from another object	Student(Student s)

Example

```
Student s1 = new Student("Aditya", 20, 101, 88.5);   // Parameterized
Student s2 = new Student(s1);                        // Copy
```

3️⃣ Encapsulation (Data Hiding)

- Data (variables) in each class is private.
- Access is given using getter and setter methods.

Example:

```
private int rollNo;
```

public int getRollNo() { return rollNo; }

public void setRollNo(int rollNo) { this.rollNo = rollNo; }

4️⃣ Inheritance (Reusability)

- Inheritance allows one class to use properties and methods of another class.
- It improves reusability and code organization.

Example:

class Student extends Person { ... }

class Teacher extends Person { ... }

5️⃣ Polymorphism (Many Forms)

- Polymorphism means one method behaves differently depending on the object that calls it.
- Implemented using method overriding.

Example 1:

void displayInfo() { ... } // in Person (abstract)

- Student class overrides displayInfo() to show student details.
- Teacher class overrides the same method to show teacher details.

Example 2:

Person p1 = new Student("Riya", 19, 103, 95.0);

Person p2 = new Teacher("Mr. Verma", 40, "Maths");

p1.displayInfo(); // Calls Student's version

p2.displayInfo(); // Calls Teacher's version

6️⃣ Abstraction (Hiding Implementation Details)

- Abstraction focuses on what an object does, not how it does it.
- Achieved using abstract classes or interfaces.

👉 Example:

```
abstract class Person {
    abstract void displayInfo();
}
```

✅ Here, Person is abstract and cannot be directly instantiated.
It defines the structure (name, age) and enforces that all subclasses (Student, Teacher) must implement displayInfo().

⚙️ How It Works

1. The program starts from the main() method.
2. A Student object (s1) is created using the parameterized constructor → it prints student details.
3. A second student (s2) is created using the copy constructor.
4. A Teacher object is created and displayed.
5. Two polymorphic references (p1 and p2) of type Person are used to show dynamic method overriding.

💻 Java Code Implementation

```
// Abstract class for Abstraction
abstract class Person {
    private String name;
    private int age;

    // Default Constructor
    Person() {
        System.out.println("Default Person created");
    }
```

```
    // Parameterized Constructor
    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Copy Constructor
    Person(Person p) {
        this.name = p.name;
        this.age = p.age;
    }

    // Encapsulation - Getters & Setters
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }

    // Abstract method
    abstract void displayInfo();
}

// Child Class 1 - Student (Inheritance)
class Student extends Person {
    private int rollNo;
    private double marks;
```

```
// Default Constructor
Student() {
    super(); // calls Person default
    System.out.println("Default Student created");
}

// Parameterized Constructor
Student(String name, int age, int rollNo, double marks) {
    super(name, age); // calling parent constructor
    this.rollNo = rollNo;
    this.marks = marks;
}

// Copy Constructor
Student(Student s) {
    super(s); // copy Person attributes
    this.rollNo = s.rollNo;
    this.marks = s.marks;
}

// Encapsulation
public int getRollNo() { return rollNo; }
public void setRollNo(int rollNo) { this.rollNo = rollNo; }

public double getMarks() { return marks; }
public void setMarks(double marks) { this.marks = marks; }
```

```
    // Polymorphism (method overriding)
    @Override
    void displayInfo() {
        System.out.println(" 🎓 Student Details:");
        System.out.println("Name: " + getName());
        System.out.println("Age: " + getAge());
        System.out.println("Roll No: " + rollNo);
        System.out.println("Marks: " + marks);
        System.out.println("----------------------------");
    }
}

// Child Class 2 - Teacher (Inheritance)
class Teacher extends Person {
    private String subject;

    // Default Constructor (added to avoid error)
    Teacher() {
        super();
        this.subject = "Not Assigned";
    }

    // Parameterized Constructor
    Teacher(String name, int age, String subject) {
        super(name, age);
        this.subject = subject;
```

```
    }

    // Polymorphism (method overriding)
    @Override
    void displayInfo() {
        System.out.println("👩‍🏫 Teacher Details:");
        System.out.println("Name: " + getName());
        System.out.println("Age: " + getAge());
        System.out.println("Subject: " + subject);
        System.out.println("----------------------------");
    }
}

// Main Class
public class Main {
    public static void main(String[] args) {

        // Using Parameterized Constructor
        Student s1 = new Student("Aditya", 20, 101, 88.5);
        s1.displayInfo();

        // Using Copy Constructor
        Student s2 = new Student(s1);
        s2.setRollNo(102);
        s2.setMarks(91.2);
        s2.displayInfo();
```

```
        // Teacher Object
        Teacher t1 = new Teacher("Mrs. Sharma", 35, "Computer Science");
        t1.displayInfo();

        // Demonstrate Polymorphism
        Person p1 = new Student("Riya", 19, 103, 95.0);
        Person p2 = new Teacher("Mr. Verma", 40, "Mathematics");

        // Dynamic Method Dispatch
        p1.displayInfo();
        p2.displayInfo();
    }
}
```

📄 Expected Output

```
Default Person created
Default Student created
 Student Details:
Name: Aditya
Age: 20
Roll No: 101
Marks: 88.5
----------------------------
 Student Details:
Name: Aditya
Age: 20
Roll No: 102
Marks: 91.2
```

Teacher Details:

Name: Mrs. Sharma

Age: 35

Subject: Computer Science

Student Details:

Name: Riya

Age: 19

Roll No: 103

Marks: 95.0

Teacher Details:

Name: Mr. Verma

Age: 40

Subject: Mathematics

Thank You...