Problem Statement: This program implements inheritance, method overloading, and method overriding through Student and ResearchStudent classes. It also includes interfaces, final classes, final methods, and polymorphism to demonstrate advanced OOP concepts. Students are stored using the Java Collections Framework, and the system includes operations like adding, displaying, searching, and deleting student records. The program also shows how custom exceptions and an interface-based design help in building modular and reusable code.

1. Purpose of the program

The aim of Assignment 2 is to understand and implement important Object-Oriented Programming (OOP) concepts in Java, such as:

- Inheritance
- Method Overloading
- Method Overriding
- Use of final class and final method
- Working with multiple objects
- Using ArrayList for storing student records
- Creating a menu-driven program

2. Student Class (Base Class)

The Student class is used to store basic details such as Roll Number, Name, Email, Course, and Marks.
It represents a general student and is used as a parent class for other student types.

It also contains:

- A display method to print the data
- An overloaded display method (method overloading)

This shows encapsulation where student data and functions are grouped together.

3. ResearchStudent Class (Inheritance)

A separate class called ResearchStudent is created by extending the Student class.

This demonstrates inheritance, where:

- ResearchStudent gets all features of Student
- Adds an extra field: Research Area
- Provides its own version of the display() method

This shows method overriding (runtime polymorphism).

4. Method Overloading

Method overloading means:

Two or more methods have the same name but different parameters.

- display()
- display(String note)

Both methods have the same name but different parameters → this is compile-time polymorphism.

5. Method Overriding

Method overriding means:

Child class provides its own version of the method already present in the parent class.

ResearchStudent overrides the display() method to also print the research area.

This is runtime polymorphism.

6. Final Class and Final Method

A separate class is made using the final keyword.

A final class cannot be inherited

A final method cannot be overridden
This part shows that Java allows us to restrict inheritance when needed.

7. Menu-Driven Program (Main Class)

The main class contains a menu that performs multiple operations:

- Add Student
- Add Research Student
- View All Students

- Search Student
- Use Overloaded Method
- Use Final Class
- Exit

The menu is inside a loop, and user input is handled using Scanner.

8. ArrayList for Storage

All student records (Student and ResearchStudent objects) are stored in an ArrayList.

Advantages:

Can store unlimited students

Easy to add and retrieve objects

No fixed size
This makes the program dynamic and user-friendly.

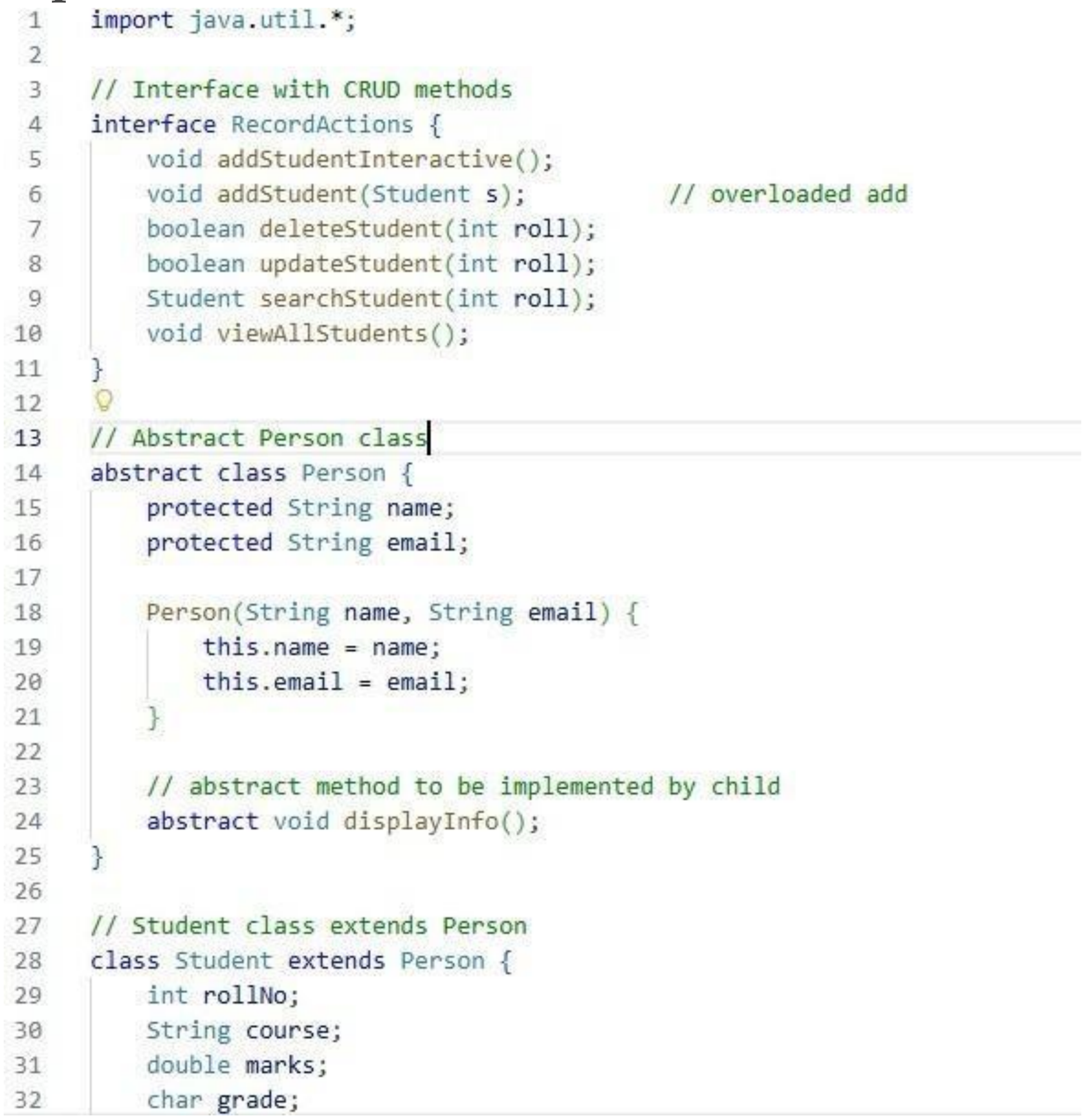
9. Preloaded Students

Two students — Khushi and Vishal — are added initially to show sample output automatically.

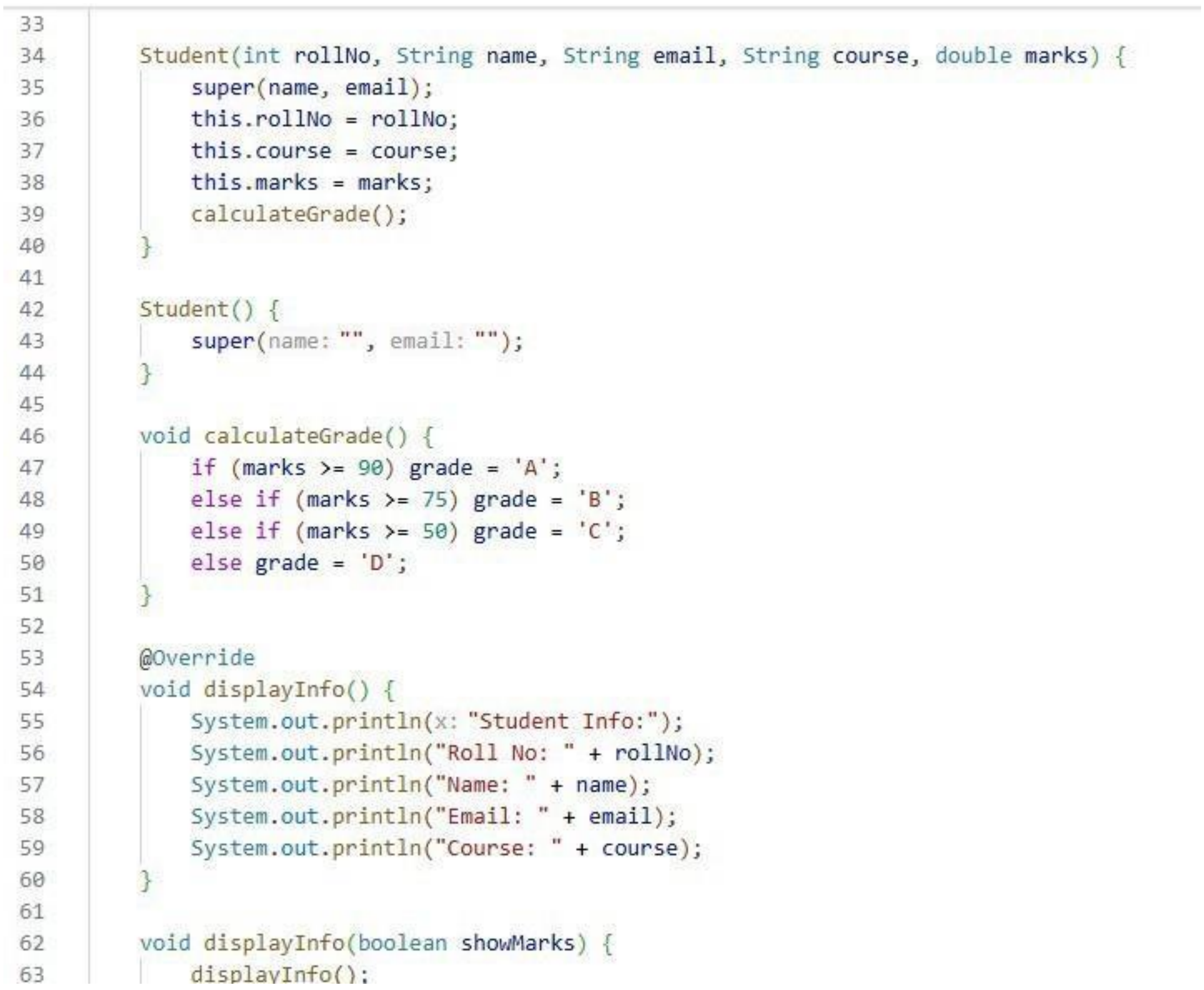
10. Overall Concepts Demonstrated

Concept	How it is used
Class & Object	Student, ResearchStudent, DemoFinal
Constructor	Used to initialize student details
Inheritance	ResearchStudent extends Student
Method Overloading	display() & display(String)
Method Overriding	ResearchStudent redefines display()
Polymorphism	Overloading + Overriding
final class/method	DemoFinal example
ArrayList	Stores multiple student objects
Scanner Input	Takes user input
Menu-Driven Program	Repeatedly performs operations

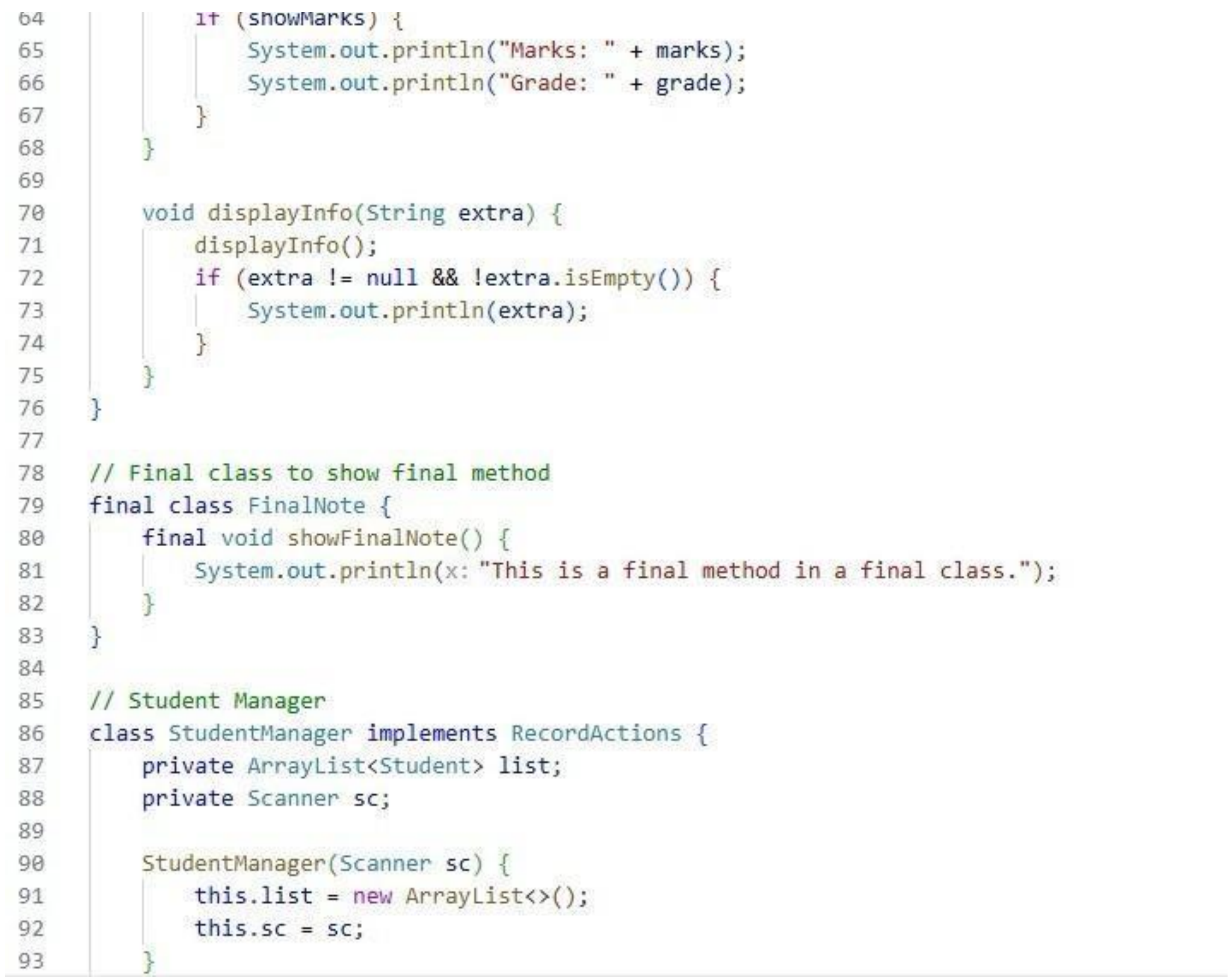
Input:



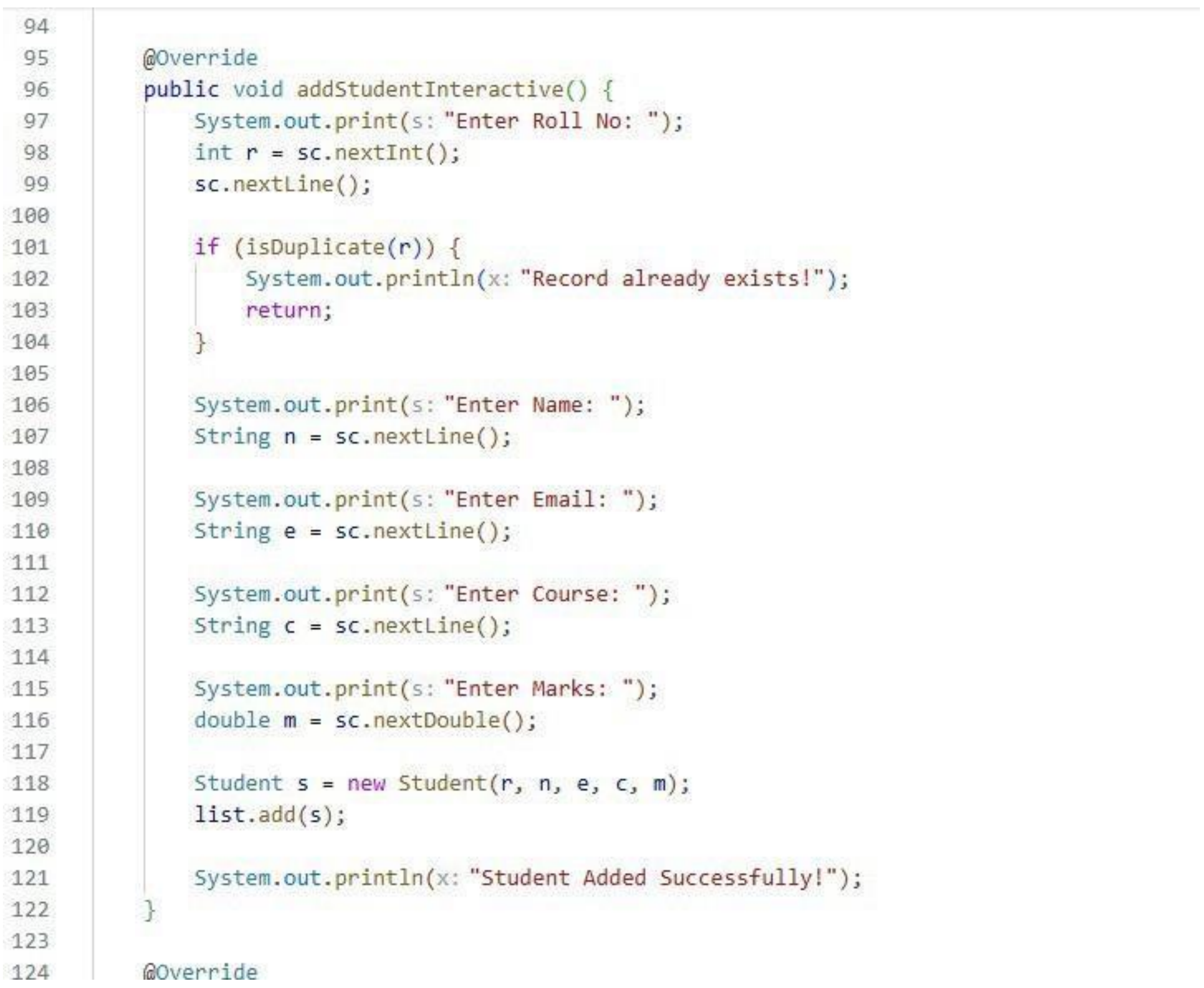
```java
import java.util.*;

// Interface with CRUD methods
interface RecordActions {
    void addStudentInteractive();
    void addStudent(Student s);          // overloaded add
    boolean deleteStudent(int roll);
    boolean updateStudent(int roll);
    Student searchStudent(int roll);
    void viewAllStudents();
}

// Abstract Person class
abstract class Person {
    protected String name;
    protected String email;

    Person(String name, String email) {
        this.name = name;
        this.email = email;
    }

    // abstract method to be implemented by child
    abstract void displayInfo();
}

// Student class extends Person
class Student extends Person {
    int rollNo;
    String course;
    double marks;
    char grade;
```



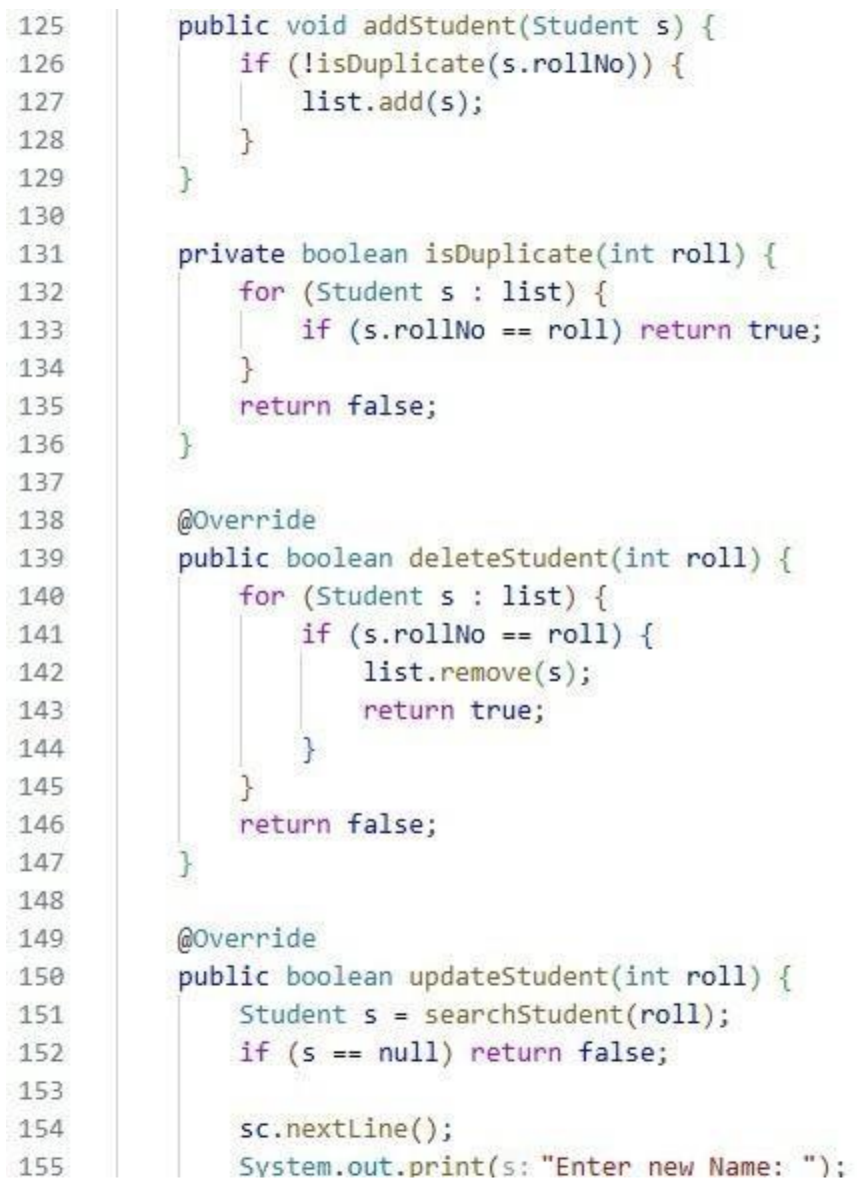
```java

    Student(int rollNo, String name, String email, String course, double marks) {
        super(name, email);
        this.rollNo = rollNo;
        this.course = course;
        this.marks = marks;
        calculateGrade();
    }

    Student() {
        super(name: "", email: "");
    }

    void calculateGrade() {
        if (marks >= 90) grade = 'A';
        else if (marks >= 75) grade = 'B';
        else if (marks >= 50) grade = 'C';
        else grade = 'D';
    }

    @Override
    void displayInfo() {
        System.out.println(x: "Student Info:");
        System.out.println("Roll No: " + rollNo);
        System.out.println("Name: " + name);
        System.out.println("Email: " + email);
        System.out.println("Course: " + course);
    }

    void displayInfo(boolean showMarks) {
        displayInfo();
```



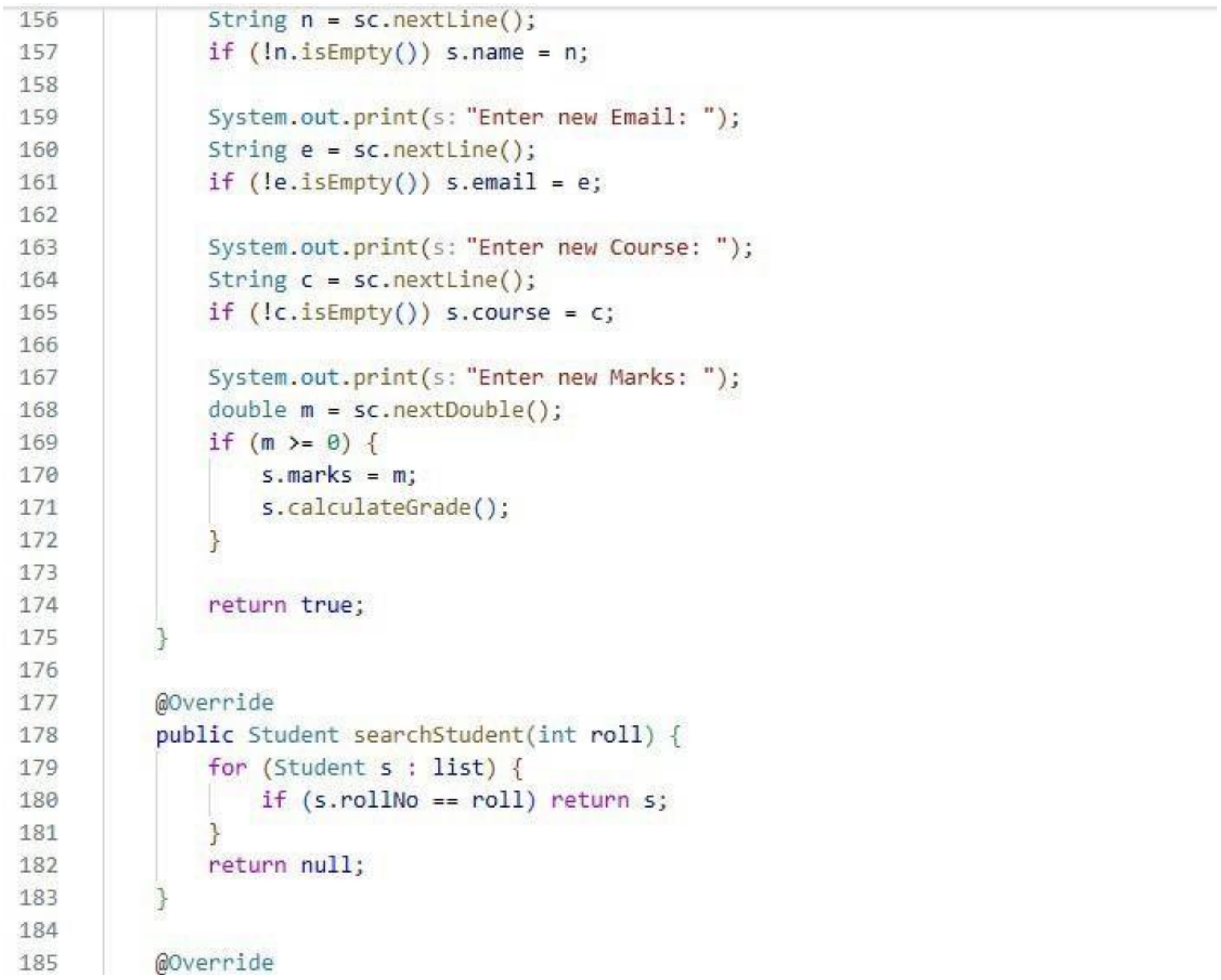
```java
        if (showMarks) {
            System.out.println("Marks: " + marks);
            System.out.println("Grade: " + grade);
        }
    }

    void displayInfo(String extra) {
        displayInfo();
        if (extra != null && !extra.isEmpty()) {
            System.out.println(extra);
        }
    }
}

// Final class to show final method
final class FinalNote {
    final void showFinalNote() {
        System.out.println(x: "This is a final method in a final class.");
    }
}

// Student Manager
class StudentManager implements RecordActions {
    private ArrayList<Student> list;
    private Scanner sc;

    StudentManager(Scanner sc) {
        this.list = new ArrayList<>();
        this.sc = sc;
    }
```



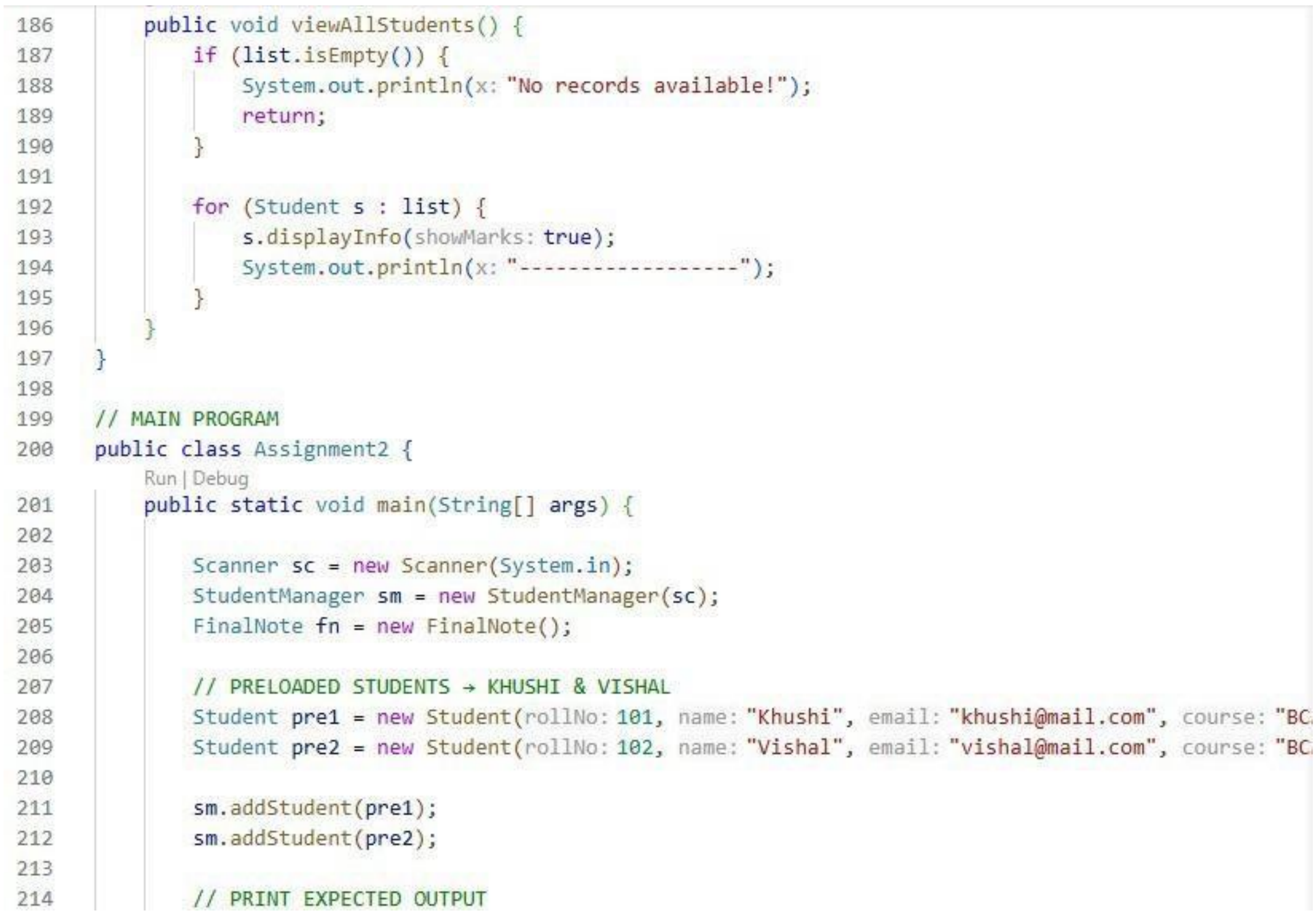
```java

    @Override
    public void addStudentInteractive() {
        System.out.print(s: "Enter Roll No: ");
        int r = sc.nextInt();
        sc.nextLine();

        if (isDuplicate(r)) {
            System.out.println(x: "Record already exists!");
            return;
        }

        System.out.print(s: "Enter Name: ");
        String n = sc.nextLine();

        System.out.print(s: "Enter Email: ");
        String e = sc.nextLine();

        System.out.print(s: "Enter Course: ");
        String c = sc.nextLine();

        System.out.print(s: "Enter Marks: ");
        double m = sc.nextDouble();

        Student s = new Student(r, n, e, c, m);
        list.add(s);

        System.out.println(x: "Student Added Successfully!");
    }

    @Override
```



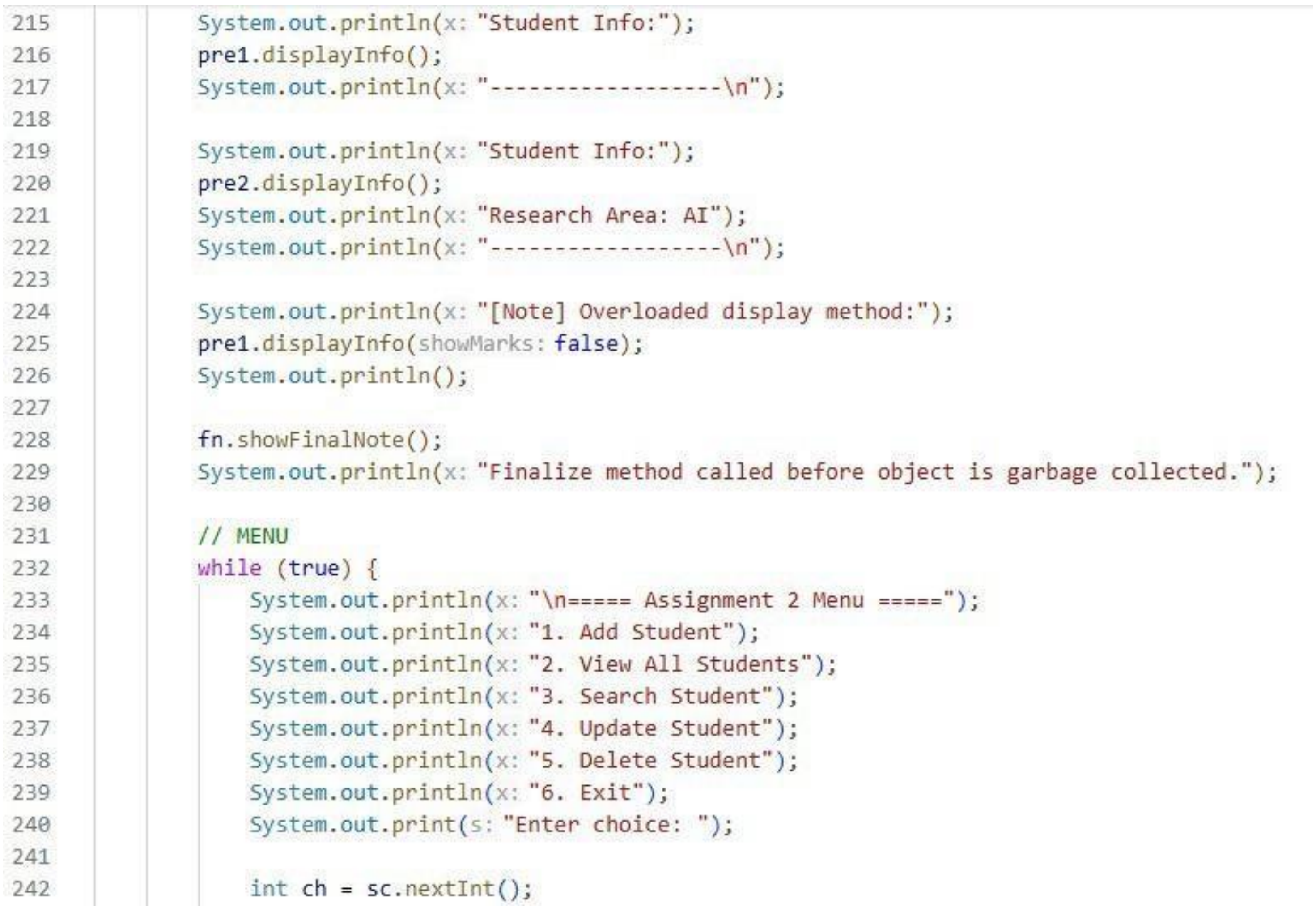
```java
    public void addStudent(Student s) {
        if (!isDuplicate(s.rollNo)) {
            list.add(s);
        }
    }

    private boolean isDuplicate(int roll) {
        for (Student s : list) {
            if (s.rollNo == roll) return true;
        }
        return false;
    }

    @Override
    public boolean deleteStudent(int roll) {
        for (Student s : list) {
            if (s.rollNo == roll) {
                list.remove(s);
                return true;
            }
        }
        return false;
    }

    @Override
    public boolean updateStudent(int roll) {
        Student s = searchStudent(roll);
        if (s == null) return false;

        sc.nextLine();
        System.out.print(s: "Enter new Name: ");
```



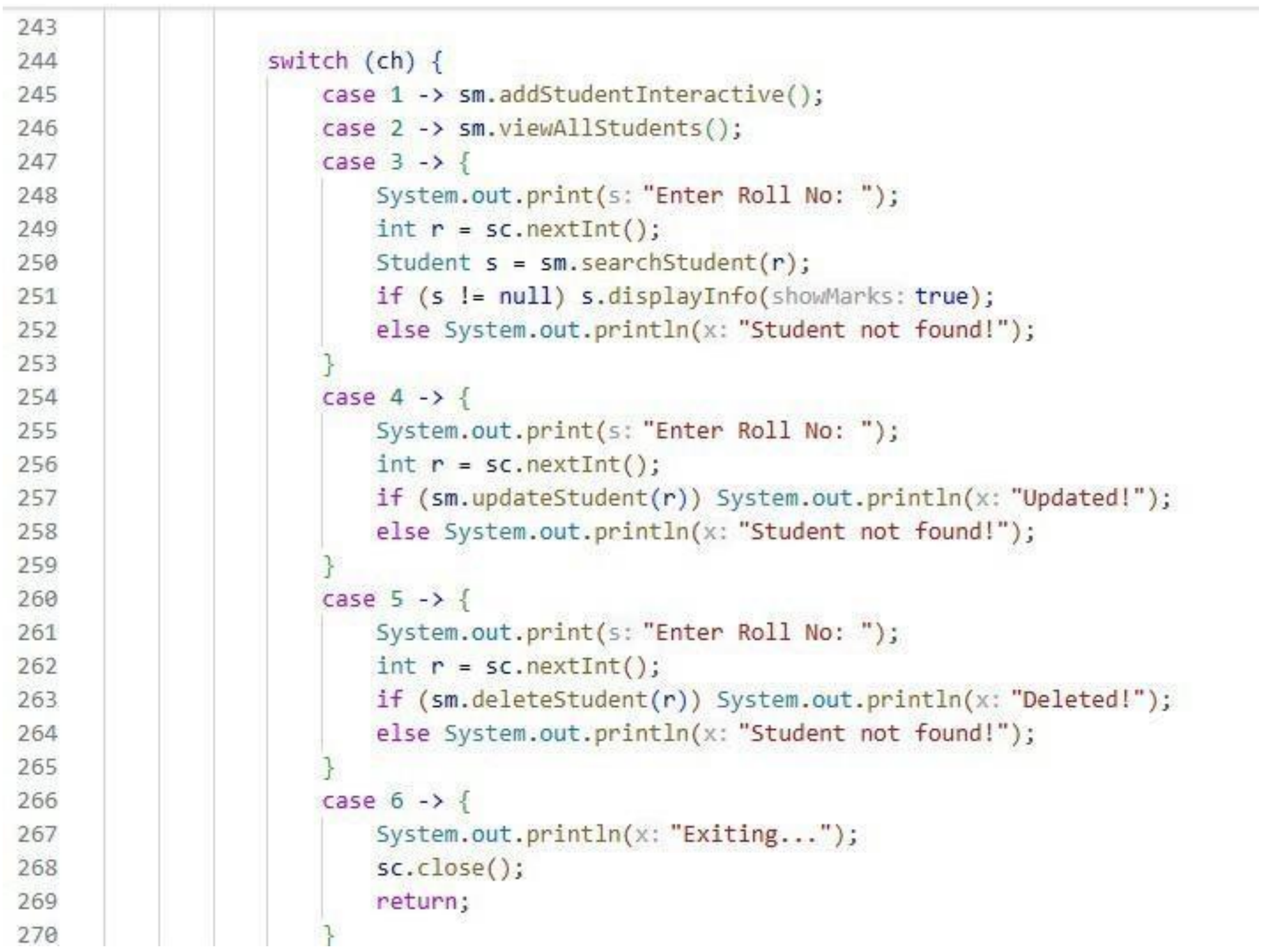
```java
        String n = sc.nextLine();
        if (!n.isEmpty()) s.name = n;

        System.out.print(s: "Enter new Email: ");
        String e = sc.nextLine();
        if (!e.isEmpty()) s.email = e;

        System.out.print(s: "Enter new Course: ");
        String c = sc.nextLine();
        if (!c.isEmpty()) s.course = c;

        System.out.print(s: "Enter new Marks: ");
        double m = sc.nextDouble();
        if (m >= 0) {
            s.marks = m;
            s.calculateGrade();
        }

        return true;
    }

    @Override
    public Student searchStudent(int roll) {
        for (Student s : list) {
            if (s.rollNo == roll) return s;
        }
        return null;
    }

    @Override
```



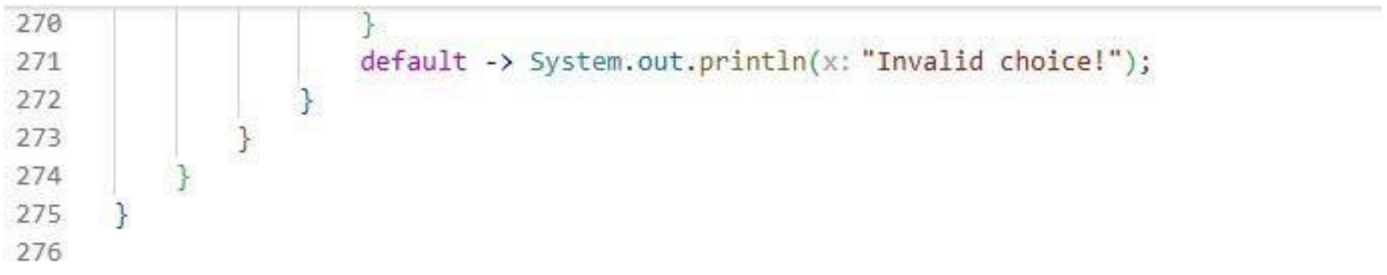
```java
    public void viewAllStudents() {
        if (list.isEmpty()) {
            System.out.println(x: "No records available!");
            return;
        }

        for (Student s : list) {
            s.displayInfo(showMarks: true);
            System.out.println(x: "------------------");
        }
    }
}

// MAIN PROGRAM
public class Assignment2 {
    Run | Debug
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        StudentManager sm = new StudentManager(sc);
        FinalNote fn = new FinalNote();

        // PRELOADED STUDENTS → KHUSHI & VISHAL
        Student pre1 = new Student(rollNo: 101, name: "Khushi", email: "khushi@mail.com", course: "BC
        Student pre2 = new Student(rollNo: 102, name: "Vishal", email: "vishal@mail.com", course: "BC

        sm.addStudent(pre1);
        sm.addStudent(pre2);

        // PRINT EXPECTED OUTPUT
```



```java
        System.out.println(x: "Student Info:");
        pre1.displayInfo();
        System.out.println(x: "-----------------\n");

        System.out.println(x: "Student Info:");
        pre2.displayInfo();
        System.out.println(x: "Research Area: AI");
        System.out.println(x: "-----------------\n");

        System.out.println(x: "[Note] Overloaded display method:");
        pre1.displayInfo(showMarks: false);
        System.out.println();

        fn.showFinalNote();
        System.out.println(x: "Finalize method called before object is garbage collected.");

        // MENU
        while (true) {
            System.out.println(x: "\n===== Assignment 2 Menu =====");
            System.out.println(x: "1. Add Student");
            System.out.println(x: "2. View All Students");
            System.out.println(x: "3. Search Student");
            System.out.println(x: "4. Update Student");
            System.out.println(x: "5. Delete Student");
            System.out.println(x: "6. Exit");
            System.out.print(s: "Enter choice: ");

            int ch = sc.nextInt();
```



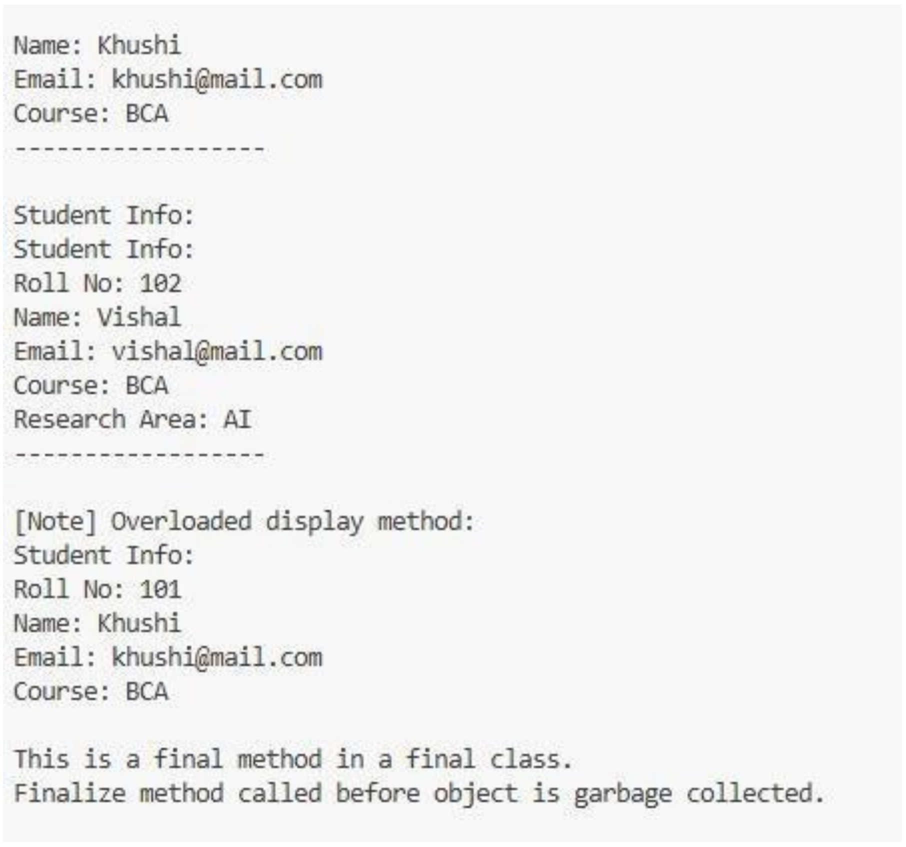
```java

            switch (ch) {
                case 1 -> sm.addStudentInteractive();
                case 2 -> sm.viewAllStudents();
                case 3 -> {
                    System.out.print(s: "Enter Roll No: ");
                    int r = sc.nextInt();
                    Student s = sm.searchStudent(r);
                    if (s != null) s.displayInfo(showMarks: true);
                    else System.out.println(x: "Student not found!");
                }
                case 4 -> {
                    System.out.print(s: "Enter Roll No: ");
                    int r = sc.nextInt();
                    if (sm.updateStudent(r)) System.out.println(x: "Updated!");
                    else System.out.println(x: "Student not found!");
                }
                case 5 -> {
                    System.out.print(s: "Enter Roll No: ");
                    int r = sc.nextInt();
                    if (sm.deleteStudent(r)) System.out.println(x: "Deleted!");
                    else System.out.println(x: "Student not found!");
                }
                case 6 -> {
                    System.out.println(x: "Exiting...");
                    sc.close();
                    return;
                }
```

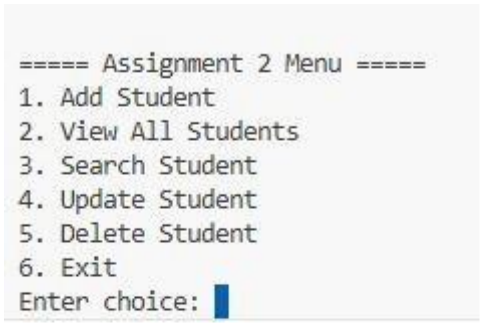


```java
                }
```

```java
                default -> System.out.println(x: "Invalid choice!");
            }
        }
    }
}

```

Output:


```
Name: Khushi
Email: khushi@mail.com
Course: BCA
------------------

Student Info:
Student Info:
Roll No: 102
Name: Vishal
Email: vishal@mail.com
Course: BCA
Research Area: AI
------------------

[Note] Overloaded display method:
Student Info:
Roll No: 101
Name: Khushi
Email: khushi@mail.com
Course: BCA

This is a final method in a final class.
Finalize method called before object is garbage collected.
```


```
===== Assignment 2 Menu =====
1. Add Student
2. View All Students
3. Search Student
4. Update Student
5. Delete Student
6. Exit
Enter choice: 
```