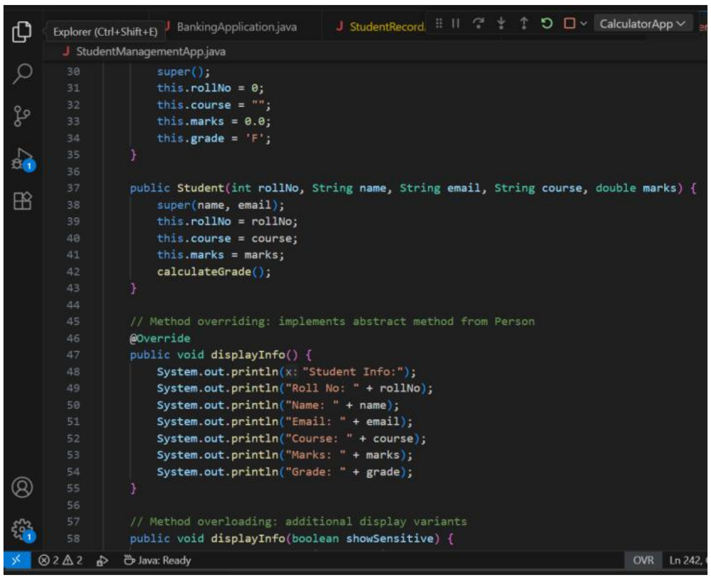
Java Lab Assignment 2

Name: Aman

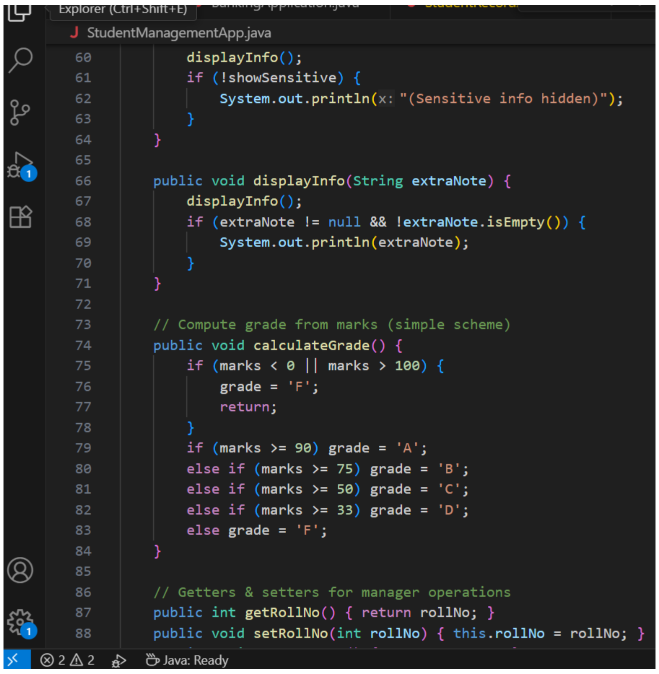
Roll no: 2401201115

Course: BCA (AI&DS)

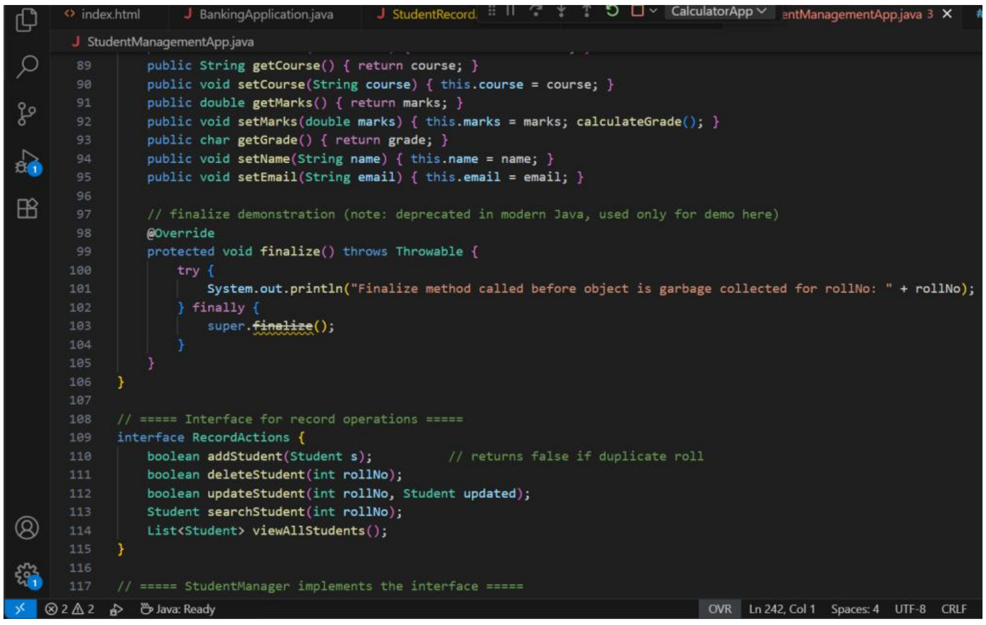
INPUT:



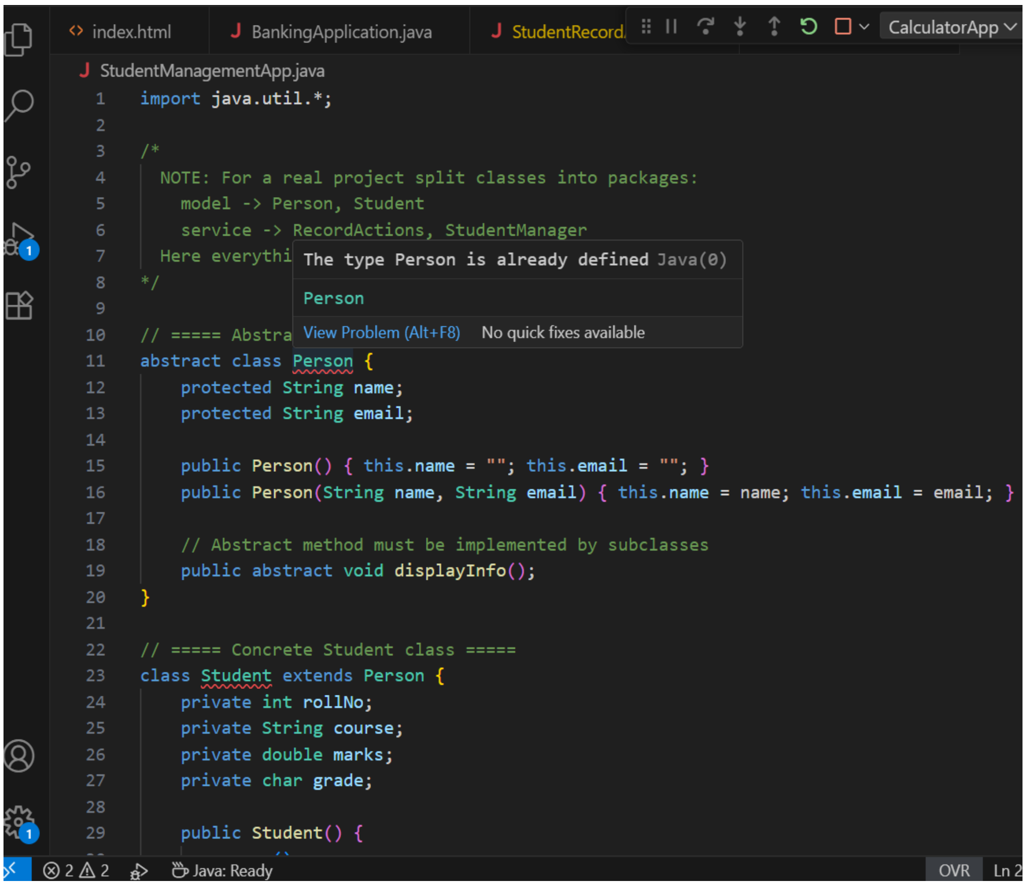
```java
        super();
        this.rollNo = 0;
        this.course = "";
        this.marks = 0.0;
        this.grade = 'F';
    }

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

    // Method overriding: implements abstract method from Person
    @Override
    public 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);
        System.out.println("Marks: " + marks);
        System.out.println("Grade: " + grade);
    }

    // Method overloading: additional display variants
    public void displayInfo(boolean showSensitive) {
```



```java
        displayInfo();
        if (!showSensitive) {
            System.out.println(x: "(Sensitive info hidden)");
        }
    }

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

    // Compute grade from marks (simple scheme)
    public void calculateGrade() {
        if (marks < 0 || marks > 100) {
            grade = 'F';
            return;
        }
        if (marks >= 90) grade = 'A';
        else if (marks >= 75) grade = 'B';
        else if (marks >= 50) grade = 'C';
        else if (marks >= 33) grade = 'D';
        else grade = 'F';
    }

    // Getters & setters for manager operations
    public int getRollNo() { return rollNo; }
    public void setRollNo(int rollNo) { this.rollNo = rollNo; }
```



```java
    public String getCourse() { return course; }
    public void setCourse(String course) { this.course = course; }
    public double getMarks() { return marks; }
    public void setMarks(double marks) { this.marks = marks; calculateGrade(); }
    public char getGrade() { return grade; }
    public void setName(String name) { this.name = name; }
    public void setEmail(String email) { this.email = email; }

    // finalize demonstration (note: deprecated in modern Java, used only for demo here)
    @Override
    protected void finalize() throws Throwable {
        try {
            System.out.println("Finalize method called before object is garbage collected for rollNo: " + rollNo);
        } finally {
            super.finalize();
        }
    }
}

// ===== Interface for record operations =====
interface RecordActions {
    boolean addStudent(Student s);          // returns false if duplicate roll
    boolean deleteStudent(int rollNo);
    boolean updateStudent(int rollNo, Student updated);
    Student searchStudent(int rollNo);
    List<Student> viewAllStudents();
}

// ===== StudentManager implements the interface =====
```

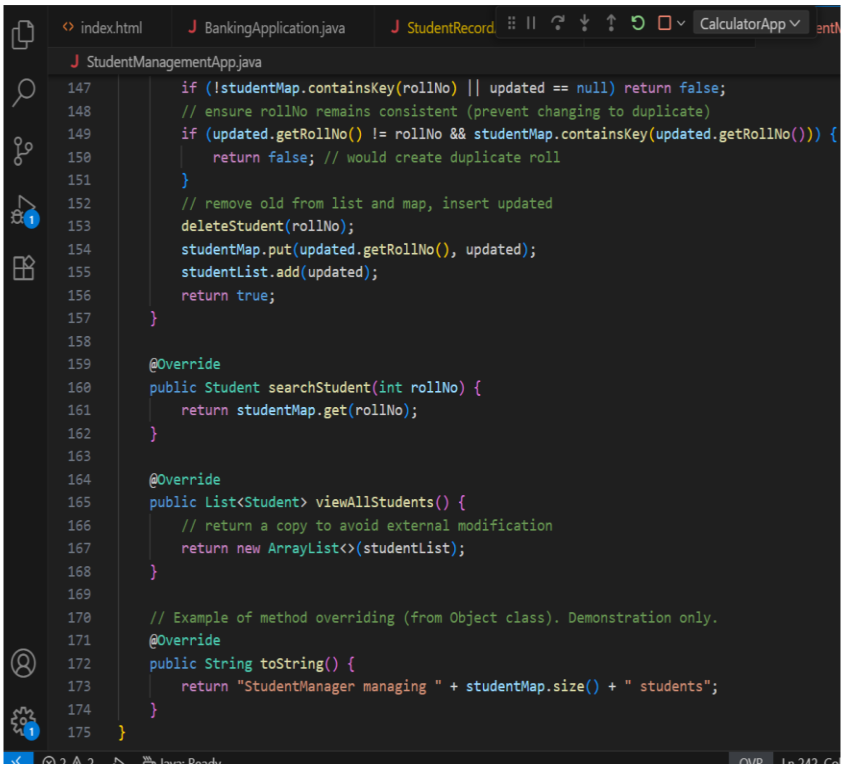


```java
import java.util.*;

/*
  NOTE: For a real project split classes into packages:
    model -> Person, Student
    service -> RecordActions, StudentManager
  Here everythi
*/

// ===== Abstra
abstract class Person {
    protected String name;
    protected String email;

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

    // Abstract method must be implemented by subclasses
    public abstract void displayInfo();
}

// ===== Concrete Student class =====
class Student extends Person {
    private int rollNo;
    private String course;
    private double marks;
    private char grade;

    public Student() {
```

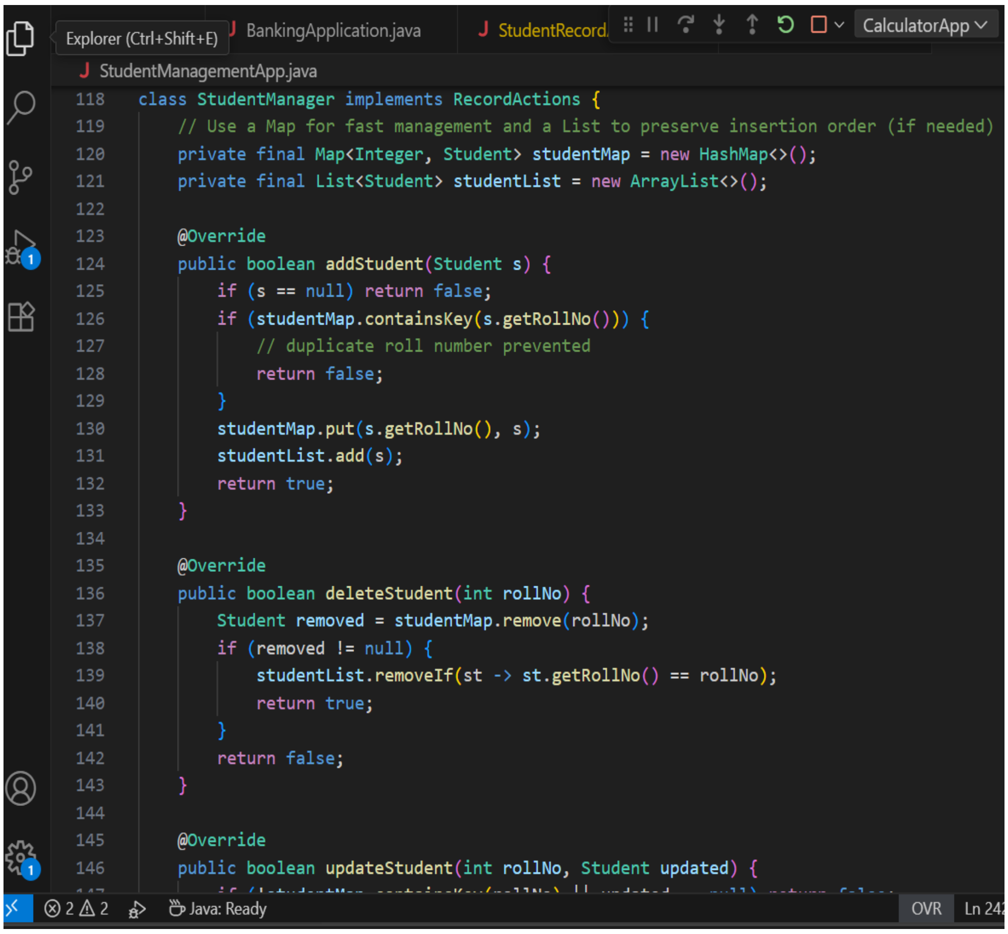
The type Person is already defined Java(0)

Person

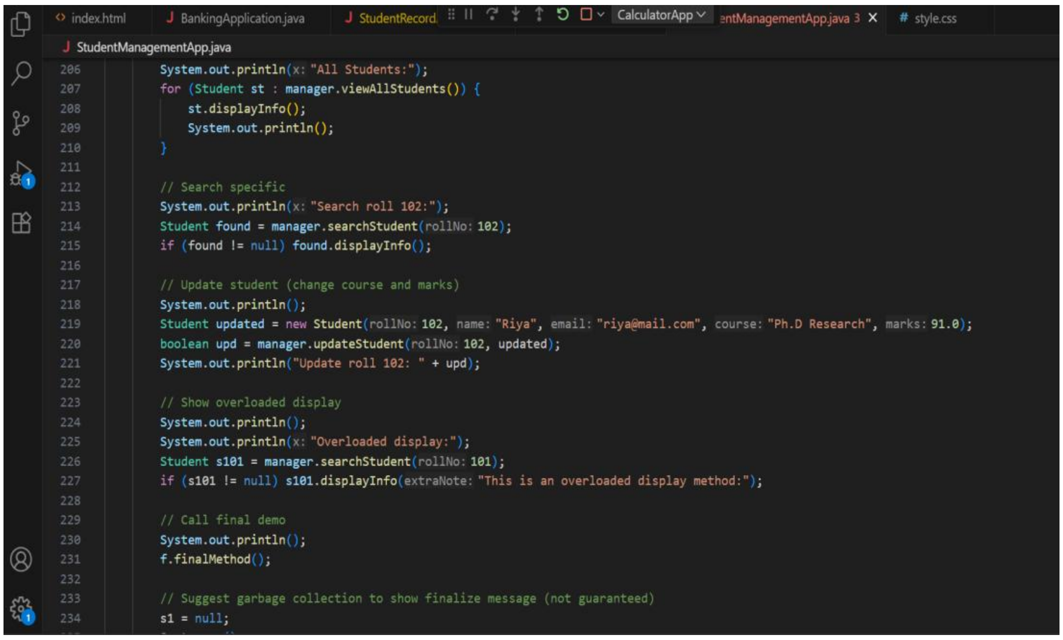
View Problem (Alt+F8)   No quick fixes available



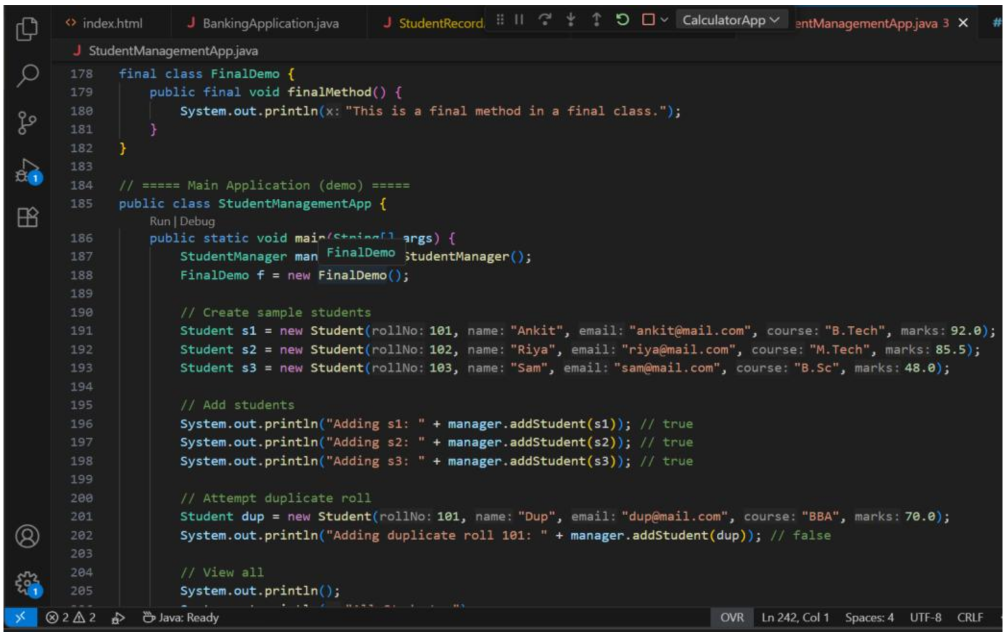
```java
        if (!studentMap.containsKey(rollNo) || updated == null) return false;
        // ensure rollNo remains consistent (prevent changing to duplicate)
        if (updated.getRollNo() != rollNo && studentMap.containsKey(updated.getRollNo())) {
            return false; // would create duplicate roll
        }
        // remove old from list and map, insert updated
        deleteStudent(rollNo);
        studentMap.put(updated.getRollNo(), updated);
        studentList.add(updated);
        return true;
    }

    @Override
    public Student searchStudent(int rollNo) {
        return studentMap.get(rollNo);
    }

    @Override
    public List<Student> viewAllStudents() {
        // return a copy to avoid external modification
        return new ArrayList<>(studentList);
    }

    // Example of method overriding (from Object class). Demonstration only.
    @Override
    public String toString() {
        return "StudentManager managing " + studentMap.size() + " students";
    }
}
```



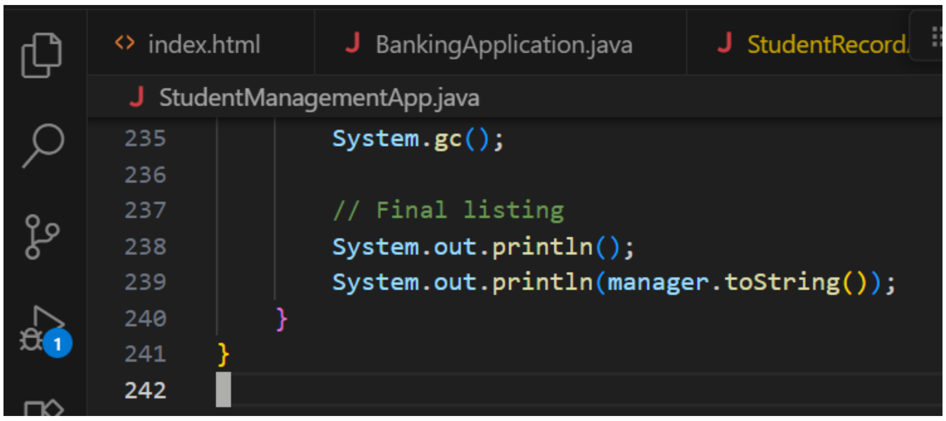
```java
class StudentManager implements RecordActions {
    // Use a Map for fast management and a List to preserve insertion order (if needed)
    private final Map<Integer, Student> studentMap = new HashMap<>();
    private final List<Student> studentList = new ArrayList<>();

    @Override
    public boolean addStudent(Student s) {
        if (s == null) return false;
        if (studentMap.containsKey(s.getRollNo())) {
            // duplicate roll number prevented
            return false;
        }
        studentMap.put(s.getRollNo(), s);
        studentList.add(s);
        return true;
    }

    @Override
    public boolean deleteStudent(int rollNo) {
        Student removed = studentMap.remove(rollNo);
        if (removed != null) {
            studentList.removeIf(st -> st.getRollNo() == rollNo);
            return true;
        }
        return false;
    }

    @Override
    public boolean updateStudent(int rollNo, Student updated) {
```



```java
        System.out.println(x: "All Students:");
        for (Student st : manager.viewAllStudents()) {
            st.displayInfo();
            System.out.println();
        }

        // Search specific
        System.out.println(x: "Search roll 102:");
        Student found = manager.searchStudent(rollNo: 102);
        if (found != null) found.displayInfo();

        // Update student (change course and marks)
        System.out.println();
        Student updated = new Student(rollNo: 102, name: "Riya", email: "riya@mail.com", course: "Ph.D Research", marks: 91.0);
        boolean upd = manager.updateStudent(rollNo: 102, updated);
        System.out.println("Update roll 102: " + upd);

        // Show overloaded display
        System.out.println();
        System.out.println(x: "Overloaded display:");
        Student s101 = manager.searchStudent(rollNo: 101);
        if (s101 != null) s101.displayInfo(extraNote: "This is an overloaded display method:");

        // Call final demo
        System.out.println();
        f.finalMethod();

        // Suggest garbage collection to show finalize message (not guaranteed)
        s1 = null;
```

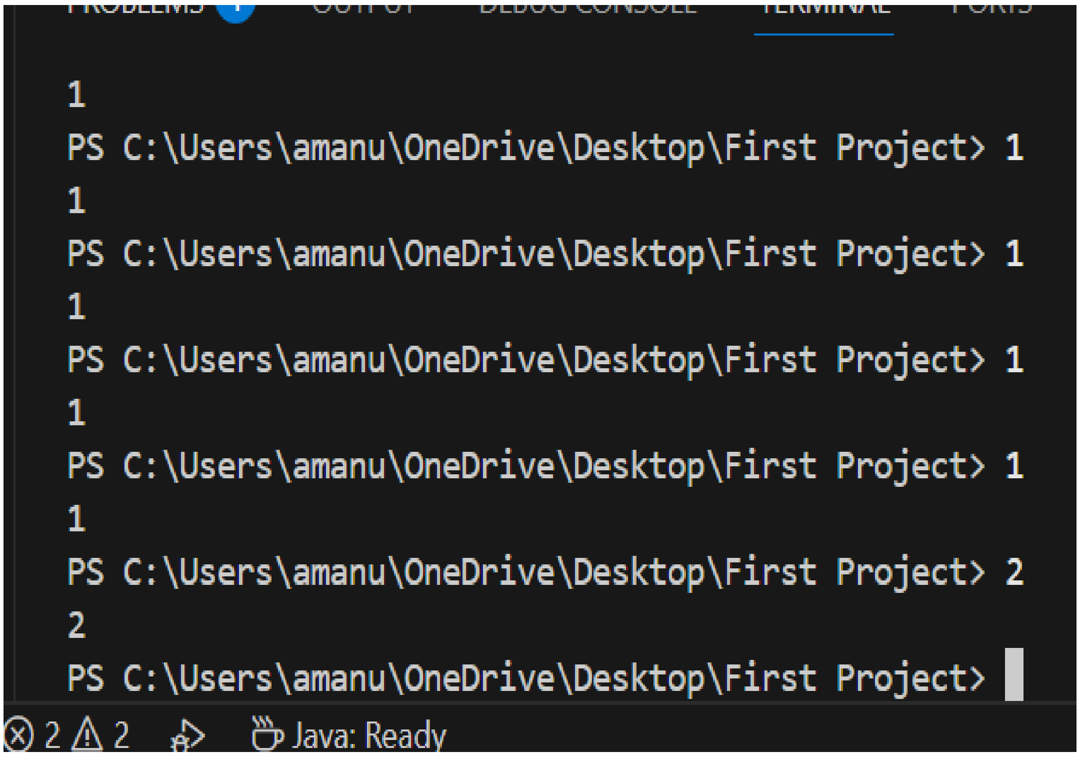


```java
final class FinalDemo {
    public final void finalMethod() {
        System.out.println(x: "This is a final method in a final class.");
    }
}

// ===== Main Application (demo) =====
public class StudentManagementApp {
    public static void main(String[] args) {
        StudentManager man                StudentManager();
        FinalDemo f = new FinalDemo();

        // Create sample students
        Student s1 = new Student(rollNo: 101, name: "Ankit", email: "ankit@mail.com", course: "B.Tech", marks: 92.0);
        Student s2 = new Student(rollNo: 102, name: "Riya", email: "riya@mail.com", course: "M.Tech", marks: 85.5);
        Student s3 = new Student(rollNo: 103, name: "Sam", email: "sam@mail.com", course: "B.Sc", marks: 48.0);

        // Add students
        System.out.println("Adding s1: " + manager.addStudent(s1)); // true
        System.out.println("Adding s2: " + manager.addStudent(s2)); // true
        System.out.println("Adding s3: " + manager.addStudent(s3)); // true

        // Attempt duplicate roll
        Student dup = new Student(rollNo: 101, name: "Dup", email: "dup@mail.com", course: "BBA", marks: 70.0);
        System.out.println("Adding duplicate roll 101: " + manager.addStudent(dup)); // false

        // View all
        System.out.println();
```



```java
        System.gc();

        // Final listing
        System.out.println();
        System.out.println(manager.toString());
    }
}
```

OUTPUT:



```
1
PS C:\Users\amanu\OneDrive\Desktop\First Project> 1
1
PS C:\Users\amanu\OneDrive\Desktop\First Project> 1
1
PS C:\Users\amanu\OneDrive\Desktop\First Project> 1
1
PS C:\Users\amanu\OneDrive\Desktop\First Project> 1
1
PS C:\Users\amanu\OneDrive\Desktop\First Project> 2
2
PS C:\Users\amanu\OneDrive\Desktop\First Project>
```

Explanation —

◆ 1. Abstract Class – Person

The system begins with an **abstract class** named **Person**, which represents all human objects in the system.
It contains common fields:

- `name`
- `email`

It also declares an **abstract method** `displayInfo()`, which must be implemented by any subclass.
This demonstrates the concept of **abstraction** and **incomplete classes**.

◆ 2. Student Class – Inheritance + Method Overriding

The **Student** class **extends** the `Person` class, meaning it inherits the common fields and implements the abstract method.

Student introduces extra attributes:

- `rollNo`
- `course`
- `marks`
- `grade`

The Student class overrides `displayInfo()` from the parent class, demonstrating **method overriding** (runtime polymorphism).

It also includes **overloaded methods**:

- Multiple versions of `displayInfo()` with different parameters
  This demonstrates **method overloading** (compile-time polymorphism).

A `calculateGrade()` method assigns a grade based on marks.

◆ 3. RecordActions Interface

The `RecordActions` interface defines the operations required to manage student records:

- `addStudent()`
- `deleteStudent()`
- `updateStudent()`
- `searchStudent()`
- `viewAllStudents()`

An interface ensures **100% abstraction**, meaning the class implementing it must define these methods.

◆ 4. StudentManager Class – Interface Implementation

This class **implements** the `RecordActions` interface.
It contains the logic for all CRUD (Create, Read, Update, Delete) operations.

StudentManager uses:

- **HashMap** (rollNo → Student) to prevent duplicate roll numbers
- **ArrayList** to preserve insertion order

It performs operations like adding, updating, searching, and deleting records.

It also overrides the `toString()` method from the Object class, demonstrating **polymorphism** again.

◆ 5. Final Class and Final Method

A separate `FinalDemo` class is declared as **final**, meaning it cannot be inherited.
It contains a **final method**, which cannot be overridden.

These are included to demonstrate the **final keyword**.

◆ 6. Main Application – Demonstration of System

The main class creates several Student objects and uses StudentManager to:

- Add students
- Prevent duplicates
- Search student records
- Update student details
- Display all students
- Test overloaded and overridden methods
- Show final methods