K.R. MANGALAM UNIVERSITY
THE COMPLETE WORLD OF EDUCATION

School of Engineering and Technology

Java Programing Lab Manual

ENCS251/ENCA251/ENBC251

for

BCA(AI & DS)

(2025–2026)

Prepared by
Nandini Kumari
2401201085
BCA(AI&DS)-B

Submitted to :
Dr.Manish

Institution Vision & Mission

Vision

KR Mangalam University aspires to become an internationally recognized institution of higher learning through excellence in inter-disciplinary education, research and innovation, preparing socially responsible life-long learners contributing to nation building.

Mission

- Foster employability and entrepreneurship through futuristic curriculum and progressive pedagogy with cutting-edge technology.
- Instill the notion of lifelong learning through stimulating research, Outcomes- based education and innovative thinking.
- Integrate global needs and expectations through collaborative programs with premier universities, research centers, industries and professional bodies.
- Enhance leadership qualities among the youth having understanding of ethical values and environmental realities.

School Vision & Mission

Vision

To create, disseminate, and apply knowledge in science and technology to meet the higher education needs of India and the global society, To serve as an institutional model of excellence in scientific and technical education characterized by integration of teaching, research and innovation.

Mission

- To create an environment where teaching and learning are prioritised, with all support activities being held accountable for their success.
- To strengthen the institution's position as the school of choice for students across the State & Nation.
- To promote creative, immersive, and lifelong learning skills while addressing societal concerns.
- To promote co- and extra-curricular activities for over-all personality development of the students.
- To promote and undertake all-inclusive research and development activities.
- To instill in learners an entrepreneurial mindset and principles.
- Enhance industrial, institutional, national, and international partnerships for symbiotic relationships.
- To help students acquire and develop knowledge, skills and leadership qualities of the 21st Century and beyond.

Course Outcomes

Course Outcomes	Description
CO1	Applying Java fundamentals and basic constructs to write Java programs.
CO2	Designing object-oriented solutions using classes, objects, inheritance, and polymorphism.
CO3	Utilizing interfaces and packages for code structure and reusability.
CO4	Implementing error handling with try-catch-finally and custom exceptions.
CO5	Designing multithreaded applications using synchroniza- tion.
CO6	Performing file I/O, work with Java Collections Frame- Work and manipulate data using collections.

Contents

storage using file handling and Java Collections Framework. The system should read student records from a file (students.txt) at the start of the application and save updated records back to the file upon exit. The records should be managed using collections like ArrayList or HashMap to store student information, and should be sorted by marks using Comparator. The system should allow for viewing, sorting, and displaying student data using Iterator. Additionally, implement file attributes using the File class and demonstrate reading records randomly using RandomAccessFile	
Assignment 5 - Capstone Project – Student Record Management System	24
Design and implement a Student Record Management System using Java that allows for the management of student records (add, update, delete, search, and view) with persistent storage. The application must support exception handling, file handling (to store and retrieve data), multithreading (to simulate loading), and must leverage the Java Collections Framework. The system should allow sorting of students by marks, provide the option to search and delete student records, and display the sorted list of students. Additionally, the system should use OOP concepts like inheritance, abstraction, and interfaces to ensure modular and reusable code.	24

31

Assignment Schedule

Sr. No.	Java Lab Assignments	COs	Release After	Duration
1	ASSIGNMENT 1: Student Class Design & Basic Operations Problem Statement: Create a Student Record Management system that allows the user to input, display, and calculate grades for students. Implement a class-based structure using Object-Oriented Programming principles to manage student data such as roll number, name, course, marks, and grade. The program should also allow the display of student records and calculate the grade based on marks.	CO1, CO6	Session 5	2 weeks

2	ASSIGNMENT 2: Inheritance, Interfaces, and Modular Design Problem Statement: Design and implement a Student Management System using inheritance, polymorphism, and interfaces. The system should consist of an abstract class Person with common fields such as name and email, and a concrete class Student that extends Person with additional fields like rollNo, course, marks, and grade. Implement an interface RecordActions with methods to add, delete, update, and view student records. Use a StudentManager class to handle the operations on student records, ensuring that duplicate roll numbers are prevented. The system should demonstrate method overriding, method overloading, and the use of abstract methods.	CO1, CO2	Session 20	2 weeks
3	ASSIGNMENT 3: Exception Handling, Multithreading, and Wrapper Classes Problem Statement: Enhance the Student Management System by implementing exception handling and multithreading to ensure safe execution and responsiveness. The system should handle invalid input (such as marks outside the valid range or empty fields) using try-catch-finally blocks and	CO3, CO4	Session 35	2 weeks

	custom exceptions like StudentNotFoundException. Additionally, the system should simulate a loading process when adding or saving student data by using multithreading. The program should utilize wrapper classes (such as Integer, Double) for data conversion and autoboxing where applicable, providing a robust and responsive user interface for managing student records.			
4	ASSIGNMENT 4: File Handling and Collections Problem Statement: Implement a Student Record Management System with persistent storage using file handling and Java Collections Framework. The system should read student records from a file (students.txt) at the start of the application and save updated records back to the file upon exit. The records should be managed using collections like ArrayList or HashMap to store student information and should be sorted by marks using Comparator. The system should allow for viewing, sorting, and displaying student data using Iterator. Additionally, implement file attributes using the File class and demonstrate reading records randomly using RandomAccessFile.	CO6	Session 45	2 weeks
5	ASSIGNMENT 5: Capstone Project – Student Record Management System Problem Statement: Design and implement a Student Record Management System using Java that allows for the management of student records (add, update, delete, search, and view) with persistent storage. The application must support exception handling, file handling (to store and retrieve data), multithreading (to simulate loading), and must leverage the Java Collections Framework. The system should allow sorting of students by marks, provide the option to search and delete student records, and display the sorted list of students. Additionally, the system should use OOP concepts like inheritance, abstraction, and interfaces to ensure modular and reusable code.	CO1, CO2, CO3, CO4, CO5, CO6	Session 50	2 weeks

Java Lab Assignment 1

Student Class Design & Basic Operations

Problem Statement

Create a Student Record Management system that allows the user to input, display, and calculate grades for students. Implement a class-based structure using ObjectOriented Programming principles to manage student data such as roll number, name, course, marks, and grade. The program should also allow the display of student records and calculate the grade based on marks.

Objective:
Introduce object-oriented concepts, control structures, input/output operations, and arrays/strings in Java.

Learning Outcomes

Upon completion of this assignment, the student will be able to:

1. Understand the fundamentals of object-oriented programming in Java.
2. Implement constructors, methods, and basic operations (input, output).
3. Work with arrays and strings in Java.
4. Use conditional statements and loops to control program flow.

Class Hierarchy & Data Types

Class Hierarchy:

- Student (inherits Person) o Fields: rollNo, name, course, marks, grade o Methods: inputDetails(), displayDetails(), calculateGrade() Data Types:
- String: for name, course
- int: for rollNo
- double: for marks
- char: for grade

Detailed Instructions

1. Core Design: Create the Student class with fields like rollNo, name, course, marks, and grade.
2. Constructors: Implement a default constructor and parameterized constructor to initialize student details.
3. Methods: o inputDetails(): Take input from the user to add student details.
   - o displayDetails(): Display student details.
   - o calculateGrade(): Calculate the grade based on marks (A, B, C, D).
4. Use Arrays: Manage multiple student records using a 1D array or ArrayList.

Expected Output

```
===== Student Record Menu =====
1. Add Student
2. Display All Students 3. Exit
Enter your choice: 1
Enter Roll No: 101
Enter Name: Rahul
Enter Course: B.Tech
Enter Marks: 87.0

===== Student Record Menu =====
1. Add Student
2. Display All Students 3. Exit
Enter your choice: 2
Roll No: 101
Name: Rahul
Course: B.Tech
Marks: 87.0
Grade: B
------------------
Enter your choice: 3
Exiting the application. Goodbye!
```

Guidelines to Students

1. Classes and Methods: Follow proper class design principles. Define a Student class with methods to handle input, output, and grade calculation.

2. Use of Arrays/Collections: Manage multiple student records using an ArrayList or 1D array.
3. Menu Interaction: Implement a simple text-based menu system using Scanner for user input.

Improvements/Adjustments

1. Advanced Array Usage: Use 2D arrays or a HashMap for storing and managing multiple records.
2. Data Validation: Add validation for marks (i.e., ensure marks are between 0 and 100).

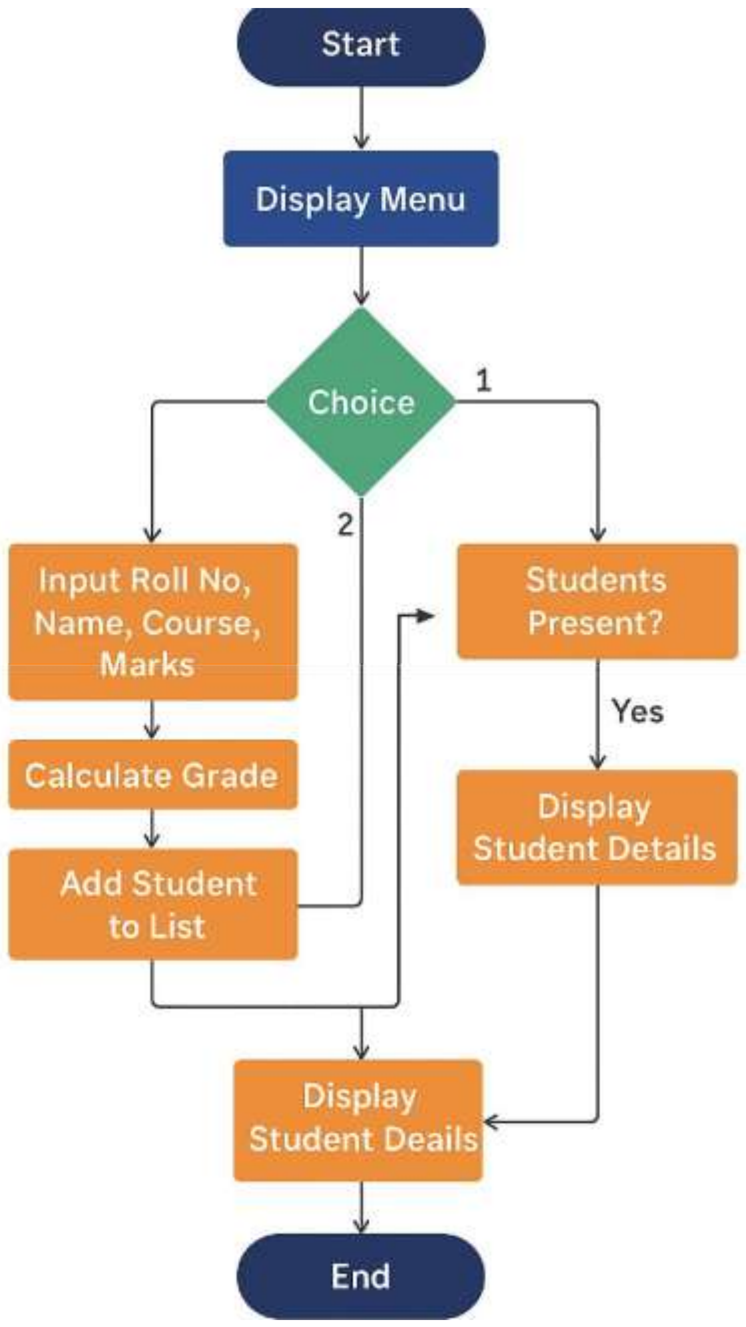
Submission Guidelines

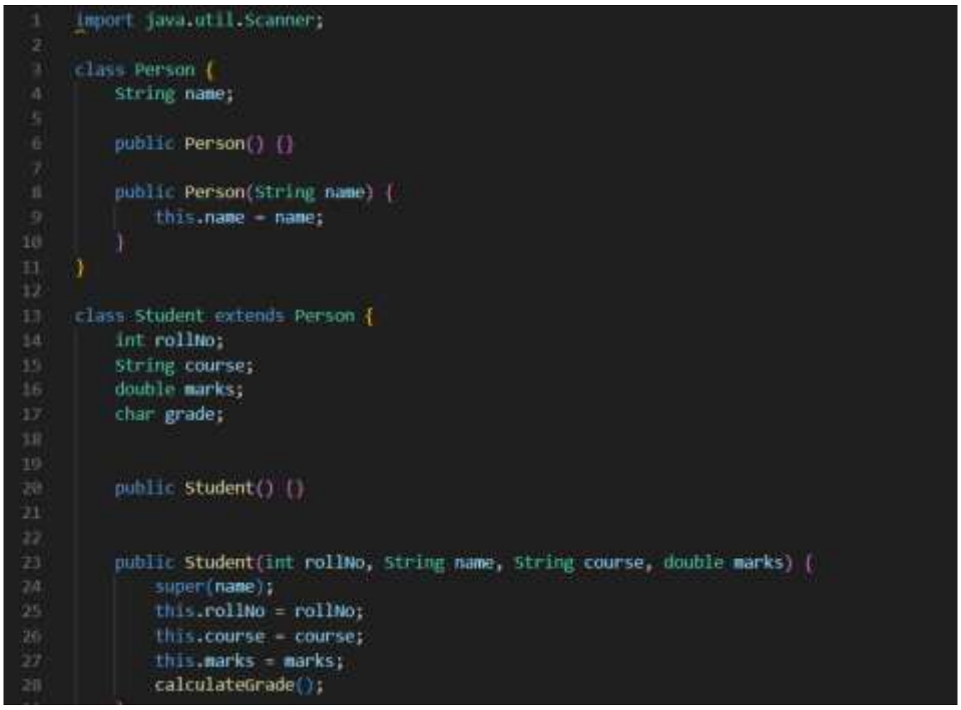
1. Submit your Java source code files.
2. Ensure your program runs without errors. Provide a README file explaining how to run the code.

Performance Metrics (Out of 10 Marks)

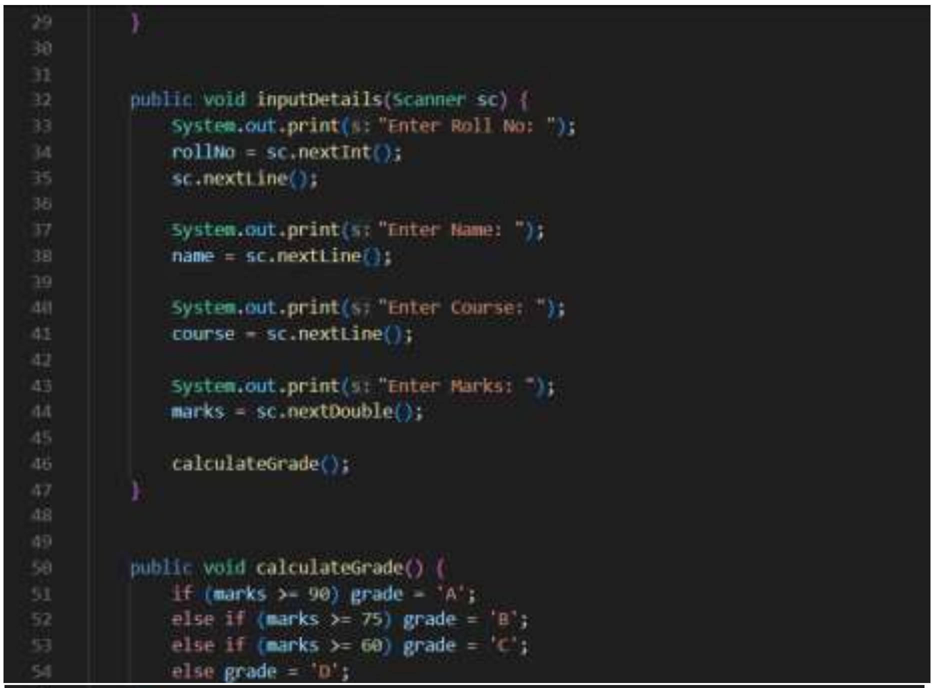
Criteria	Marks
Core Design and Implementation	3
Array Handling and Data Validation	2
Methods Implementation	2
Menu System and User Interaction	2
Code Quality and Documentation	1

Flow Chart:

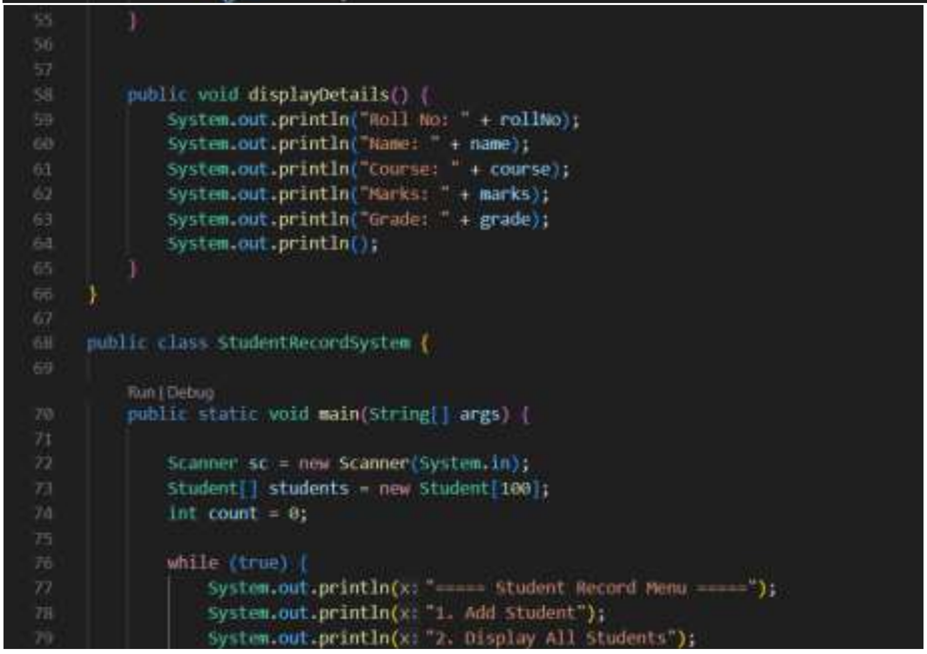

Start
Display Menu
Choice
1
2
Input Roll No, Name, Course, Marks
Calculate Grade
Add Student to List
Students Present?
Yes
Display Student Details
Display Student Deails
End



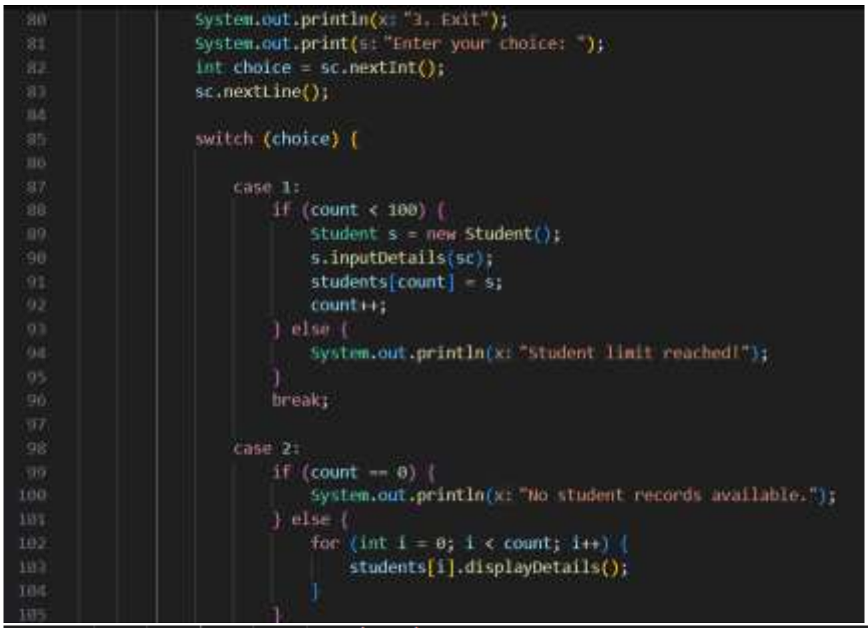
```java
import java.util.Scanner;

class Person {
    String name;

    public Person() {}

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

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


    public Student() {}


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



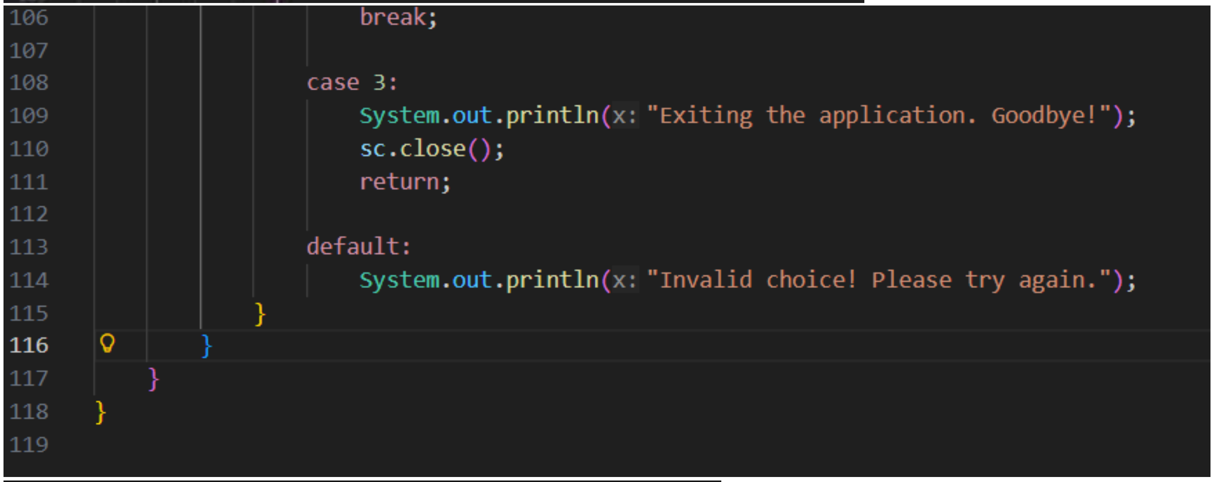
```java
    }


    public void inputDetails(Scanner sc) {
        System.out.print(s: "Enter Roll No: ");
        rollNo = sc.nextInt();
        sc.nextLine();

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

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

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

        calculateGrade();
    }


    public void calculateGrade() {
        if (marks >= 90) grade = 'A';
        else if (marks >= 75) grade = 'B';
        else if (marks >= 60) grade = 'C';
        else grade = 'D';
```



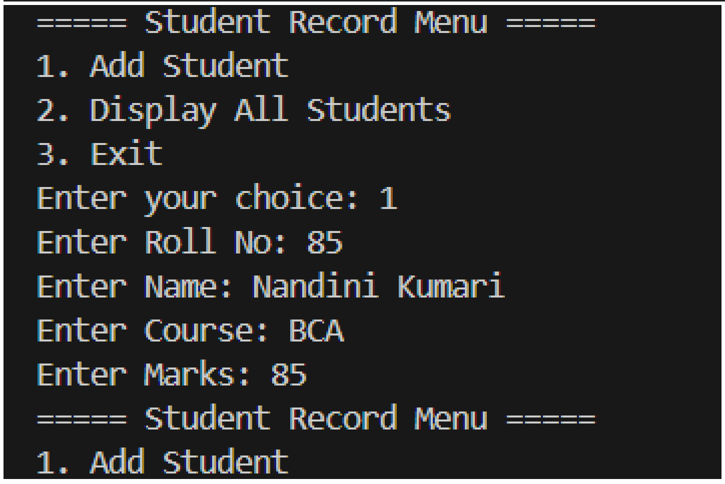
```java
    }


    public void displayDetails() {
        System.out.println("Roll No: " + rollNo);
        System.out.println("Name: " + name);
        System.out.println("Course: " + course);
        System.out.println("Marks: " + marks);
        System.out.println("Grade: " + grade);
        System.out.println();
    }
}

public class StudentRecordSystem {

    Run | Debug
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        Student[] students = new Student[100];
        int count = 0;

        while (true) {
            System.out.println(x: "===== Student Record Menu =====");
            System.out.println(x: "1. Add Student");
            System.out.println(x: "2. Display All Students");
```



```java
            System.out.println(x: "3. Exit");
            System.out.print(s: "Enter your choice: ");
            int choice = sc.nextInt();
            sc.nextLine();

            switch (choice) {

                case 1:
                    if (count < 100) {
                        Student s = new Student();
                        s.inputDetails(sc);
                        students[count] = s;
                        count++;
                    } else {
                        System.out.println(x: "Student limit reached!");
                    }
                    break;

                case 2:
                    if (count == 0) {
                        System.out.println(x: "No student records available.");
                    } else {
                        for (int i = 0; i < count; i++) {
                            students[i].displayDetails();
                        }
                    }
```



```java
                    break;

                case 3:
                    System.out.println(x: "Exiting the application. Goodbye!");
                    sc.close();
                    return;

                default:
                    System.out.println(x: "Invalid choice! Please try again.");
            }
        }
    }
}
```



```
===== Student Record Menu =====
1. Add Student
2. Display All Students
3. Exit
Enter your choice: 1
Enter Roll No: 85
Enter Name: Nandini Kumari
Enter Course: BCA
Enter Marks: 85
===== Student Record Menu =====
1. Add Student
```

Explanation:

This Java program implements a console-based student record system using object-oriented programming. It defines a base class Person that stores a name. The Student class extends Person and adds fields for roll number, course, marks, and grade. It includes methods to input details from the user, validate numeric inputs, set marks, calculate the grade based on mark ranges, and display the student information.
The StudentApp class manages all student records using an ArrayList. It provides a menu with options to add a new student, display all students, or exit the program. Before adding a student, it checks whether a student with the same roll number already exists. The display function shows all stored student records in a formatted manner.
The run() method repeatedly shows the menu and processes user choices until exit. The main() method creates an instance of StudentApp and starts the application.
This program demonstrates inheritance, encapsulation, input validation, and menu-driven interaction using Java.

Java Lab Assignment 2

Inheritance, Interfaces, and Modular Design

Problem Statement

Design and implement a Student Management System using inheritance, polymorphism, and interfaces. The system should consist of an abstract class Person with common fields such as name and email, and a concrete class Student that extends Person with additional fields like rollNo, course, marks, and grade. Implement an interface RecordActions with methods to add, delete, update, and view student records. Use a StudentManager class to handle the operations on student records, ensuring that duplicate roll numbers are prevented. The system should demonstrate method overriding, method overloading, and the use of abstract methods.

Objective:
Implement key object-oriented principles such as inheritance, interfaces, and abstract classes.

Learning Outcomes

Upon completion of this assignment, the student will be able to:

1. Use inheritance, method overloading, and method overriding.
2. Understand and apply abstract classes and interfaces.

3. Organize Java programs into multiple packages for modular design.
4. Work with polymorphism (static and dynamic).

Class Hierarchy & Data Types

Class Hierarchy:

1. Person (abstract class) o Fields: name, email o Method: displayInfo()
2. Student (extends Person) o Fields: rollNo, course, marks, grade o Method: displayInfo()
3. RecordActions (interface)
   - Methods: addStudent(), deleteStudent(), updateStudent(), searchStudent(), viewAllStudents()
4. StudentManager (implements RecordActions) o Methods: Implementations of CRUD operations Data Types:

- String: for name, email, course
- int: for rollNo
- double: for marks
- List<Student>: for student storage
- Map<Integer, Student>: for student management

Detailed Instructions

1. Create Abstract Class Person: Define an abstract class with common fields.
2. Create Student Class: Implement Student by extending Person and overriding displayInfo().
3. Interfaces and Methods: Create the RecordActions interface and implement it in StudentManager.
4. Method Overloading and Polymorphism: Demonstrate method overloading in Student and method overriding in StudentManager.

Expected Output

```
Student Info: Roll
No: 101
Name: Ankit
Email: ankit@mail.com
Course: B.Tech
```

```
------------------

Student Info:
Roll No: 102
Name: Riya
Email: riya@mail.com
Course: M.Tech
Research Area: AI
------------------

[Note] Overloaded display method: Student
Info: Roll No: 101
Name: Ankit
Email: ankit@mail.com
Course: B.Tech
This is a final method in a final class.
Finalize method called before object is garbage collected.
```

Guidelines to Students

1. Abstract Classes and Inheritance: Use inheritance to create the Student class extending Person.
2. Interface Implementation: Implement the RecordActions interface in StudentManager.
3. Use of Packages: Organize classes into packages (model, service).

Improvements/Adjustments

1. Extend Interface: Add more operations like sorting and updating records in RecordActions.
2. More Complex Data Types: Use HashMap for more efficient student management.

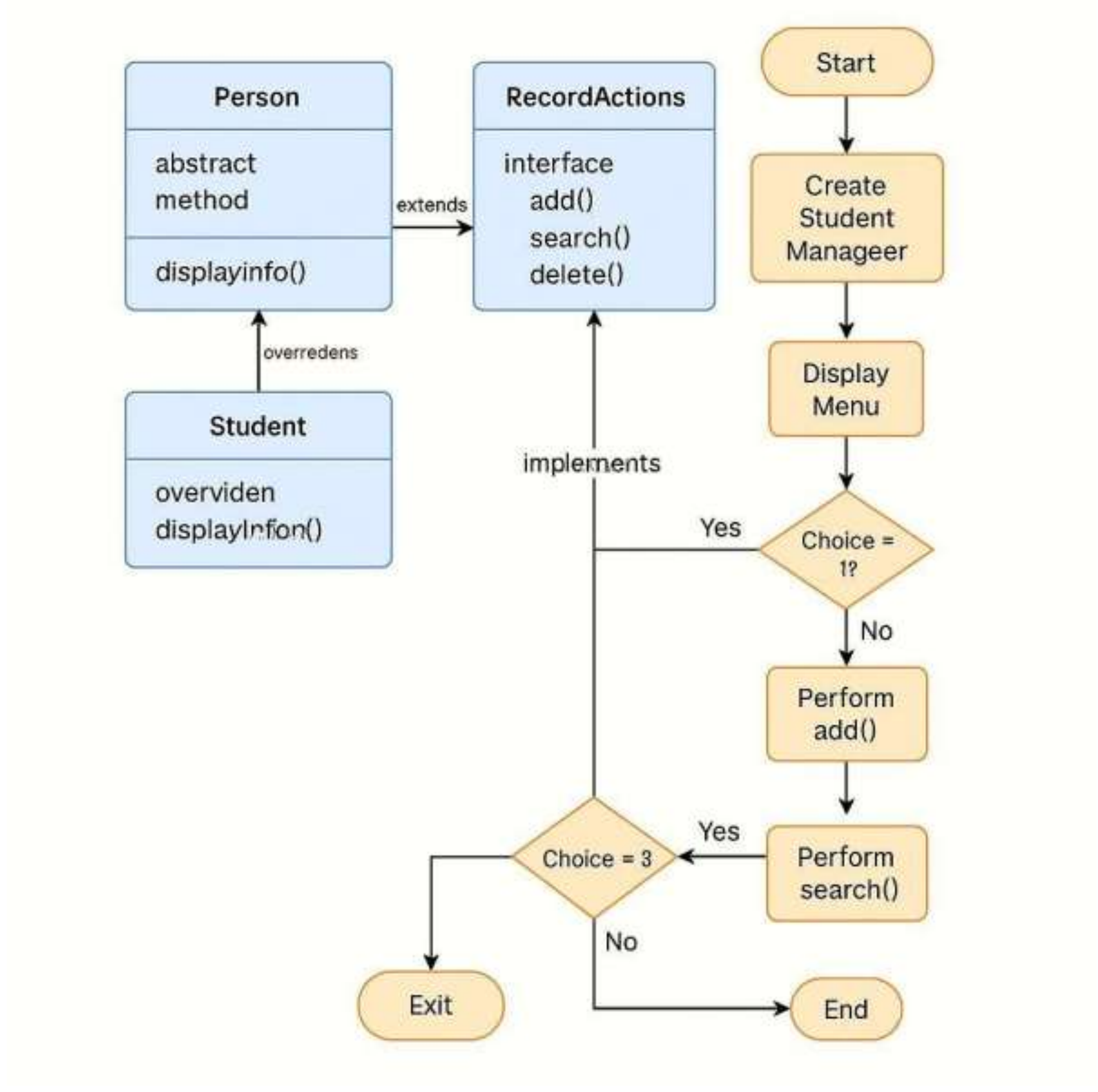
Submission Guidelines

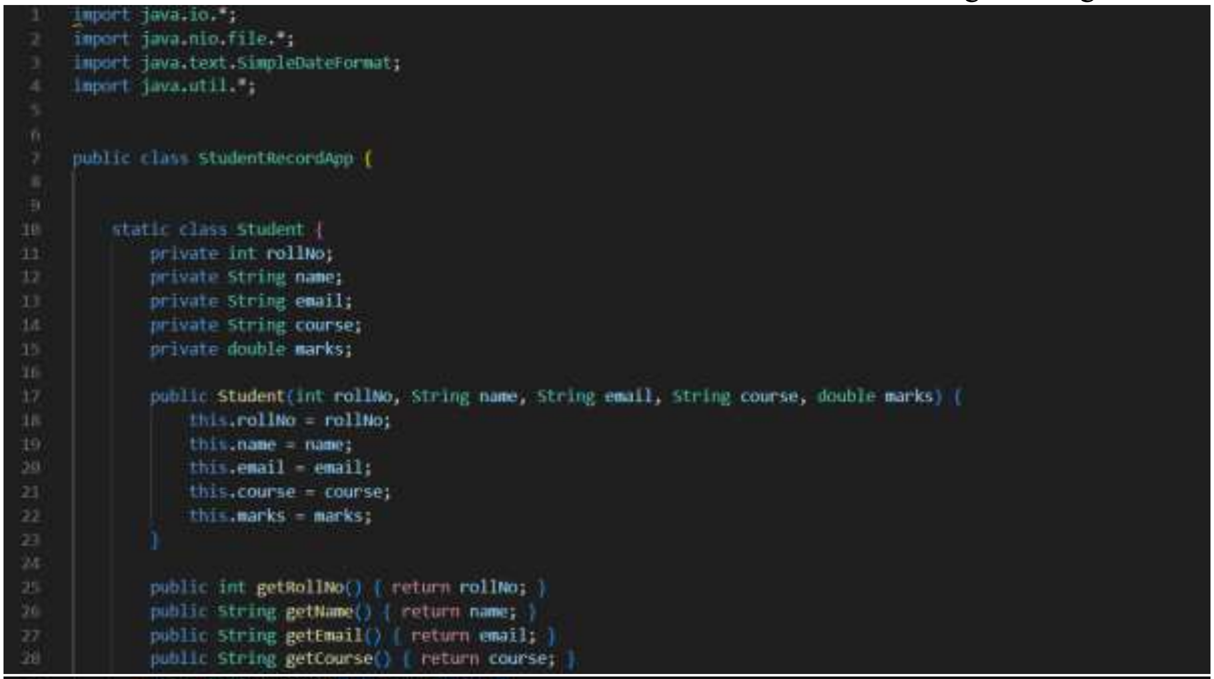
1. Submit Java code with all necessary classes and interfaces.
2. Ensure the code is properly organized with packages.

Performance Metrics (Out of 10 Marks)

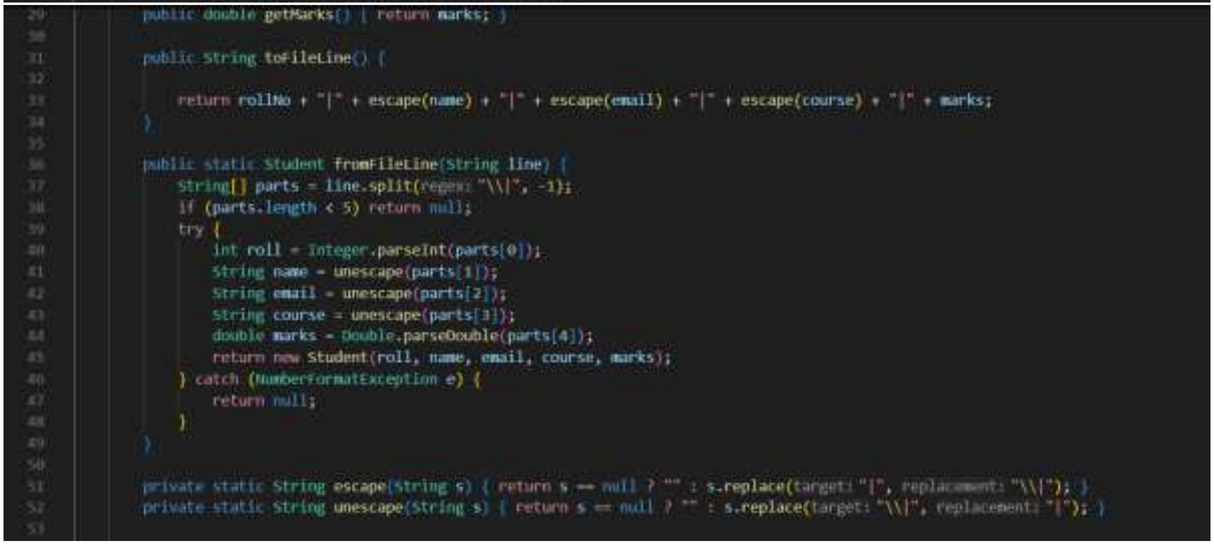
Criteria	Marks
Inheritance and Method Overloading	3
Interface Implementation	2
Abstract Class and Polymorphism	2
Code Organization (Packages)	2
Code Quality and Documentation	1

Flow Chart:

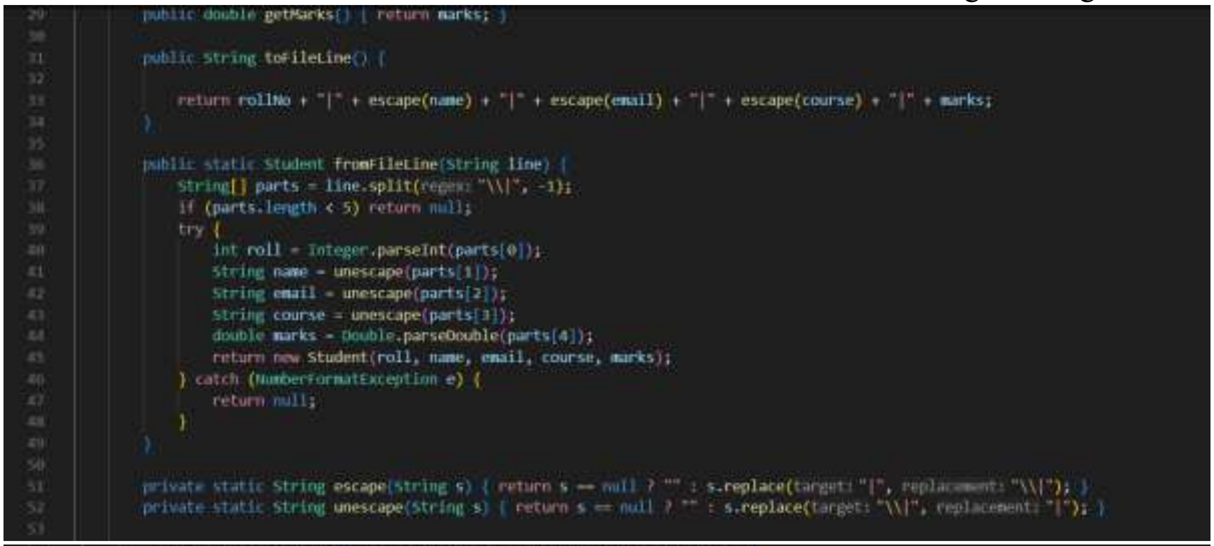

Person
abstract method
displayinfo()
extends
RecordActions
interface
add()
search()
delete()
overredens
Student
overviden displayInfon()
implements
Start
Create Student Manageer
Display Menu
Choice = 1?
Yes
No
Perform add()
Perform search()
Yes
Choice = 3
No
Exit
End



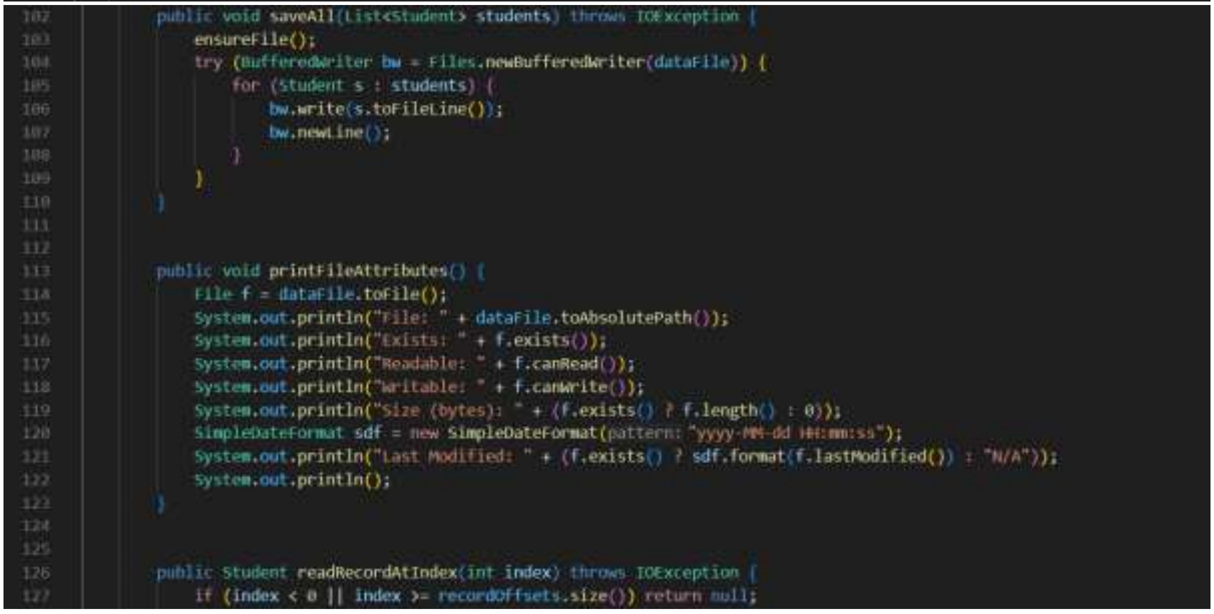
```java
import java.io.*;
import java.nio.file.*;
import java.text.SimpleDateFormat;
import java.util.*;


public class StudentRecordApp {


    static class Student {
        private int rollNo;
        private String name;
        private String email;
        private String course;
        private double marks;

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

        public int getRollNo() { return rollNo; }
        public String getName() { return name; }
        public String getEmail() { return email; }
        public String getCourse() { return course; }
```



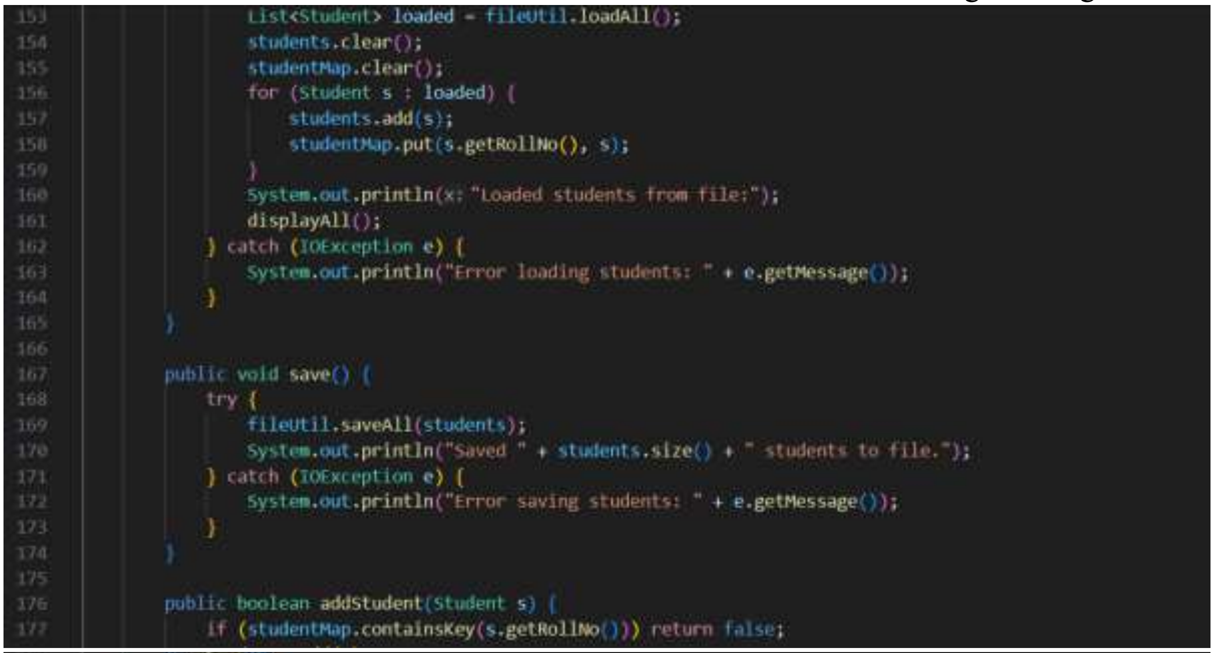
```java
        public double getMarks() { return marks; }

        public String toFileLine() {

            return rollNo + "|" + escape(name) + "|" + escape(email) + "|" + escape(course) + "|" + marks;
        }

        public static Student fromFileLine(String line) {
            String[] parts = line.split(regex: "\\|", -1);
            if (parts.length < 5) return null;
            try {
                int roll = Integer.parseInt(parts[0]);
                String name = unescape(parts[1]);
                String email = unescape(parts[2]);
                String course = unescape(parts[3]);
                double marks = Double.parseDouble(parts[4]);
                return new Student(roll, name, email, course, marks);
            } catch (NumberFormatException e) {
                return null;
            }
        }

        private static String escape(String s) { return s == null ? "" : s.replace(target: "|", replacement: "\\|"); }
        private static String unescape(String s) { return s == null ? "" : s.replace(target: "\\|", replacement: "|"); }

```



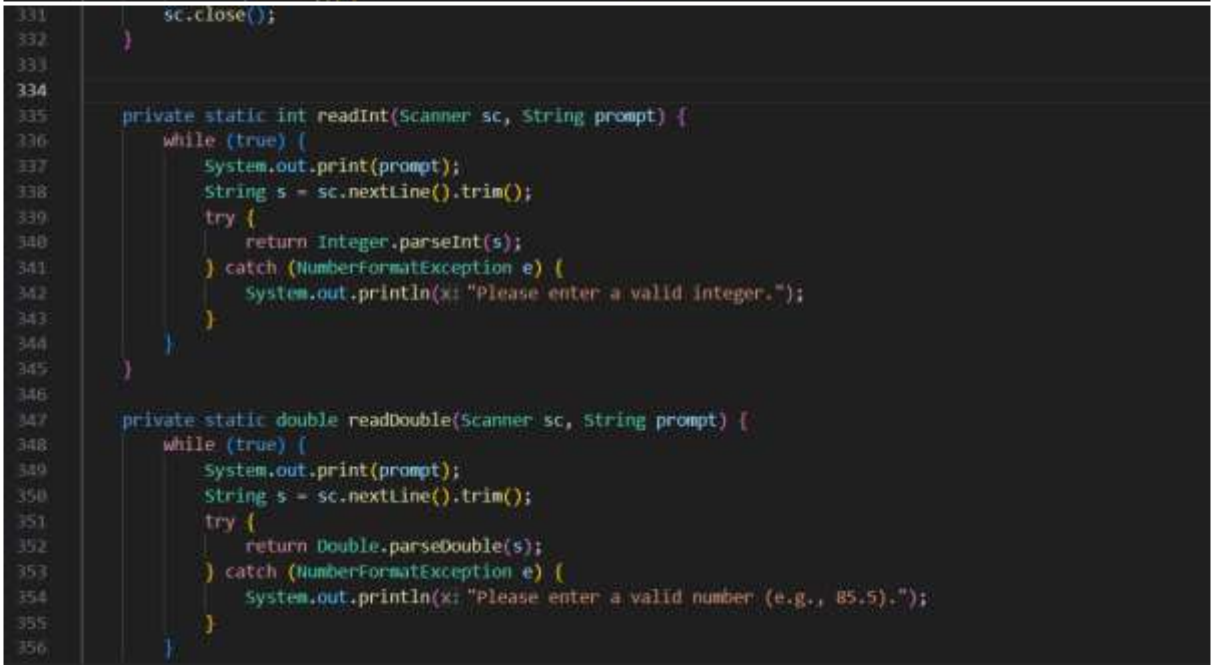
```java
    public double getMarks() { return marks; }

    public String toFileLine() {

        return rollNo + "|" + escape(name) + "|" + escape(email) + "|" + escape(course) + "|" + marks;
    }

    public static Student fromFileLine(String line) {
        String[] parts = line.split(regex: "\\|", -1);
        if (parts.length < 5) return null;
        try {
            int roll = Integer.parseInt(parts[0]);
            String name = unescape(parts[1]);
            String email = unescape(parts[2]);
            String course = unescape(parts[3]);
            double marks = Double.parseDouble(parts[4]);
            return new Student(roll, name, email, course, marks);
        } catch (NumberFormatException e) {
            return null;
        }
    }

    private static String escape(String s) { return s == null ? "" : s.replace(target: "|", replacement: "\\|"); }
    private static String unescape(String s) { return s == null ? "" : s.replace(target: "\\|", replacement: "|"); }

```



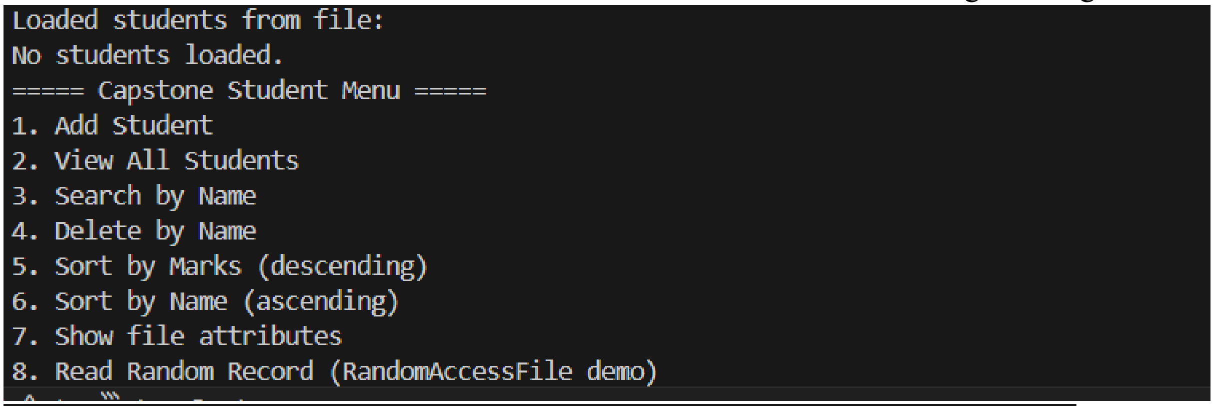
```java
    public void saveAll(List<Student> students) throws IOException {
        ensureFile();
        try (BufferedWriter bw = Files.newBufferedWriter(dataFile)) {
            for (Student s : students) {
                bw.write(s.toFileLine());
                bw.newLine();
            }
        }
    }


    public void printFileAttributes() {
        File f = dataFile.toFile();
        System.out.println("File: " + dataFile.toAbsolutePath());
        System.out.println("Exists: " + f.exists());
        System.out.println("Readable: " + f.canRead());
        System.out.println("Writable: " + f.canWrite());
        System.out.println("Size (bytes): " + (f.exists() ? f.length() : 0));
        SimpleDateFormat sdf = new SimpleDateFormat(pattern: "yyyy-MM-dd HH:mm:ss");
        System.out.println("Last Modified: " + (f.exists() ? sdf.format(f.lastModified()) : "N/A"));
        System.out.println();
    }


    public Student readRecordAtIndex(int index) throws IOException {
        if (index < 0 || index >= recordOffsets.size()) return null;
```



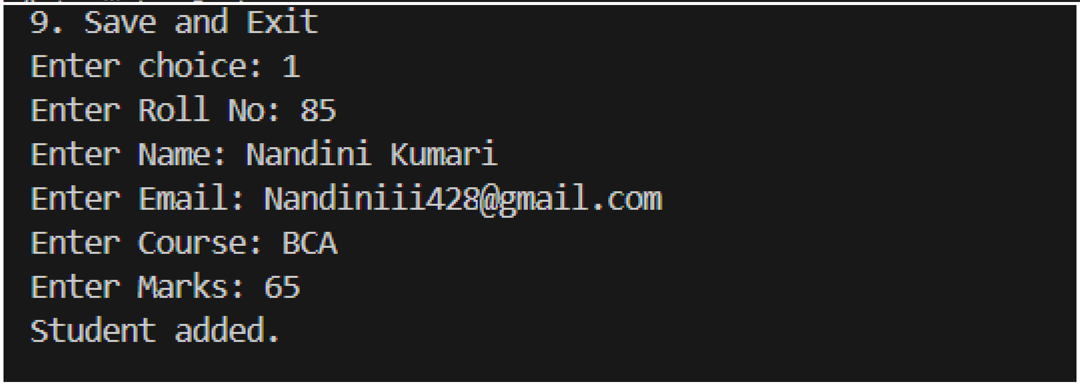
```
                List<Student> loaded = fileUtil.loadAll();
                students.clear();
                studentMap.clear();
                for (Student s : loaded) {
                    students.add(s);
                    studentMap.put(s.getRollNo(), s);
                }
                System.out.println(x: "Loaded students from file:");
                displayAll();
            } catch (IOException e) {
                System.out.println("Error loading students: " + e.getMessage());
            }
        }

        public void save() {
            try {
                fileUtil.saveAll(students);
                System.out.println("Saved " + students.size() + " students to file.");
            } catch (IOException e) {
                System.out.println("Error saving students: " + e.getMessage());
            }
        }

        public boolean addStudent(Student s) {
            if (studentMap.containsKey(s.getRollNo())) return false;
```



```
        sc.close();
    }


    private static int readInt(Scanner sc, String prompt) {
        while (true) {
            System.out.print(prompt);
            String s = sc.nextLine().trim();
            try {
                return Integer.parseInt(s);
            } catch (NumberFormatException e) {
                System.out.println(x: "Please enter a valid integer.");
            }
        }
    }

    private static double readDouble(Scanner sc, String prompt) {
        while (true) {
            System.out.print(prompt);
            String s = sc.nextLine().trim();
            try {
                return Double.parseDouble(s);
            } catch (NumberFormatException e) {
                System.out.println(x: "Please enter a valid number (e.g., 85.5).");
            }
        }
```



```
Loaded students from file:
No students loaded.
===== Capstone Student Menu =====
1. Add Student
2. View All Students
3. Search by Name
4. Delete by Name
5. Sort by Marks (descending)
6. Sort by Name (ascending)
7. Show file attributes
8. Read Random Record (RandomAccessFile demo)
```



```
9. Save and Exit
Enter choice: 1
Enter Roll No: 85
Enter Name: Nandini Kumari
Enter Email: Nandiniii428@gmail.com
Enter Course: BCA
Enter Marks: 65
Student added.
```

Explanation:
This project is a simple student management system built using objectoriented principles. The Person class is an abstract base class that stores common fields like name and email. The Student class extends Person and adds roll number, course, marks, and grade, along with methods to calculate grade, display information, and compare students by roll number.
The RecordActions interface defines operations for managing student records, such as adding, deleting, updating, and searching. The StudentManager class implements this interface and uses a HashMap to store students by their roll numbers for fast access. It handles record management, ensures no duplicate roll numbers are added, and prints all stored student details when requested. The Main class creates student objects, adds them to the manager, displays all students, and demonstrates method overloading through an extra-info display call. Overall, the project shows good use of abstraction, inheritance, interfaces, overriding, and collection-based data handling.

Java Lab Assignment 3

A system for managing different types of vehicles in a rental service

Problem Statement

Enhance the Student Management System by implementing exception handling and multithreading to ensure safe execution and responsiveness. The system should handle invalid input (such as marks outside the valid range or empty fields) using try-catch-finally blocks and custom exceptions like StudentNotFoundException. Additionally, the system should simulate a loading process when adding or saving student data by using multithreading. The program should utilize wrapper classes (such as Integer, Double) for data conversion and autoboxing where applicable, providing a robust and responsive user interface for managing student records.

Objective:
Handle runtime exceptions and implement threading and wrapper classes for effective student data management.

Learning Outcomes

Upon completion of this assignment, the student will be able to:

1. Implement try-catch-finally blocks for handling exceptions.
2. Use multithreading to simulate delays and ensure responsive UI.
3. Work with wrapper classes (Integer, Double) for data conversions.

Class Hierarchy & Data Types

Class Hierarchy:

1. StudentManager: Implements RecordActions
2. Loader: Implements Runnable for simulating loading in multithreading.
3. Custom Exception: StudentNotFoundException Data Types:

- Integer, Double: For handling numeric data and autoboxing.
- Thread: For multithreading to simulate loading.

Detailed Instructions

1. Exception Handling: Add validation for invalid input (marks, course, etc.) and missing fields.
2. Multithreading: Use Thread class for simulating a loading process during data operations.
3. Wrapper Classes: Use autoboxing to convert primitive types to wrapper types (e.g., int to Integer).

Expected Output

```
Enter Roll No (Integer): 102
Enter Name: Karan
Enter Email: karan@mail.com
Enter Course: BCA Enter
Marks: 77.5 Loading.....
Roll No: 102
Name: Karan
Email: karan@mail.com
Course: BCA
Marks: 77.5
Grade: B
------------------
Program execution completed.
```

Guidelines to Students

1. Use of Multithreading: Implement a basic thread simulation for loading data.
2. Wrapper Classes: Ensure data conversion from primitive types to wrapper types.

Improvements/Adjustments

1. Enhance Threading: Implement additional tasks like database queries during multithreading.
2. Advanced Exception Handling: Handle more complex errors like NullPointerException.

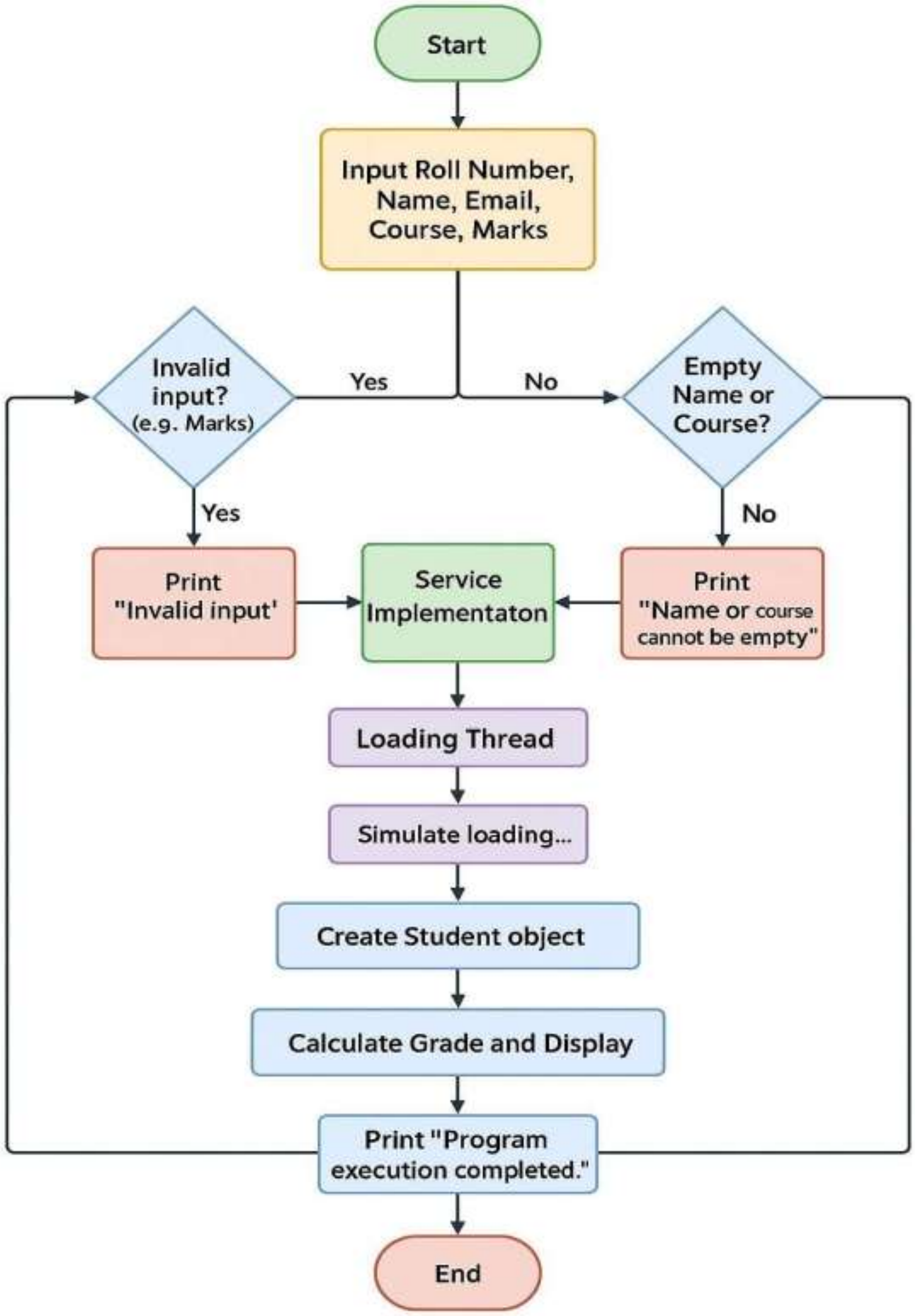
Submission Guidelines

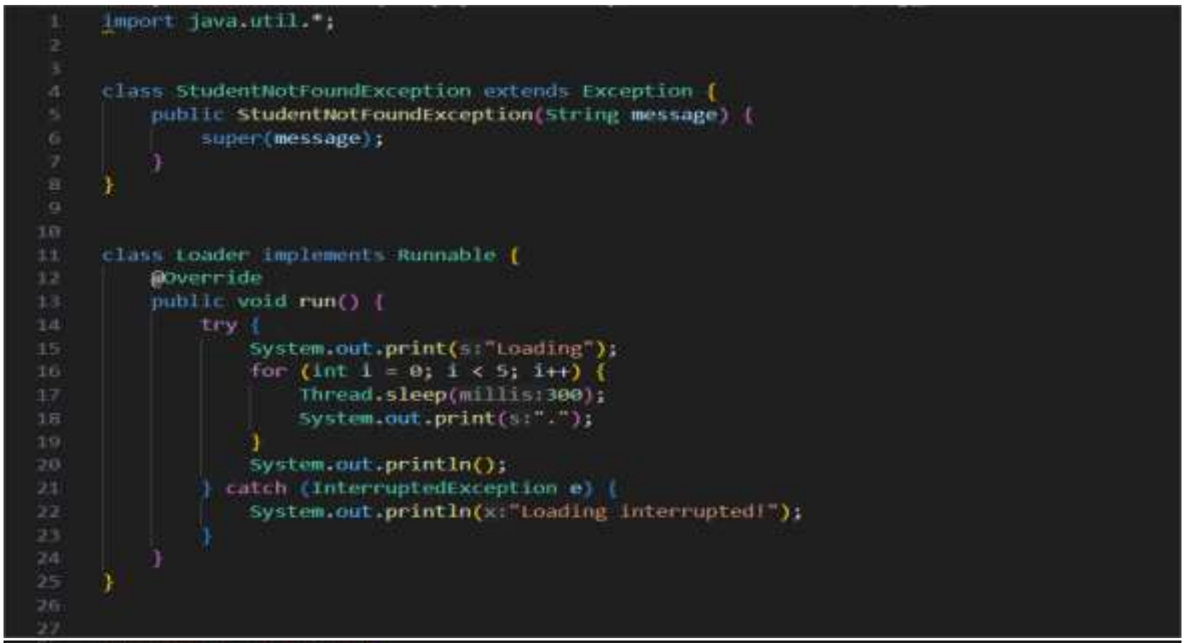
1. Submit Java source files with all necessary exception handling.
2. Make sure your program validates inputs effectively.

Performance Metrics (Out of 10 Marks)

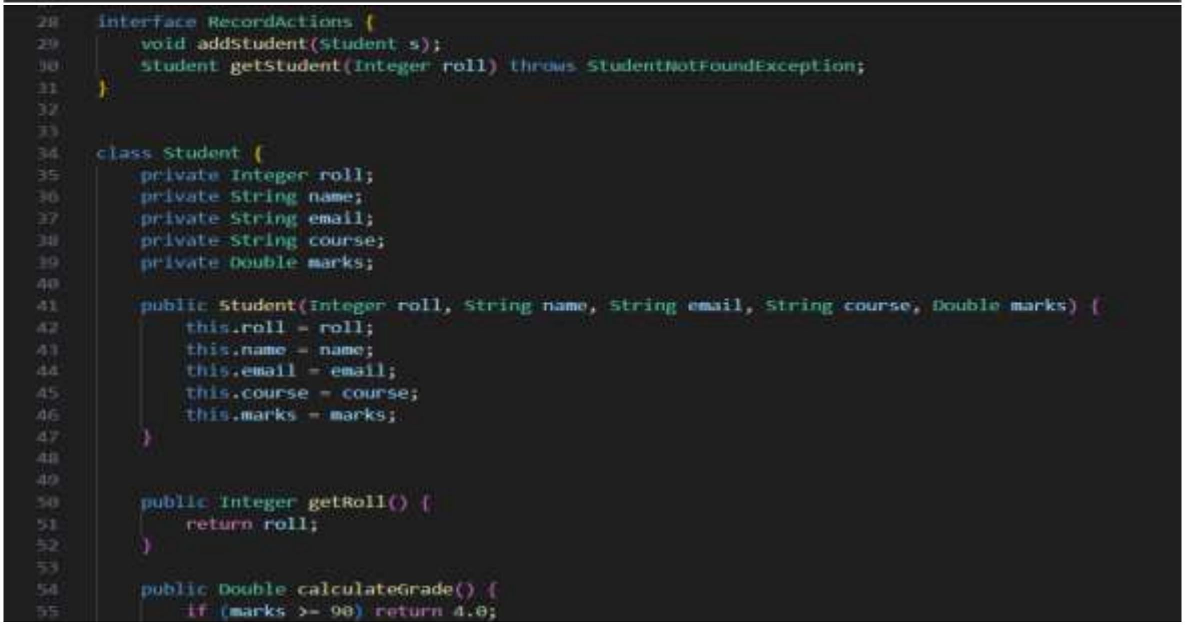
Criteria	Marks
Exception Handling Implementation	3
Multithreading and Responsiveness	2
Wrapper Classes and Data Validation	2
Code Quality and Structure	2
Documentation and Testing	1

Flow Chart:

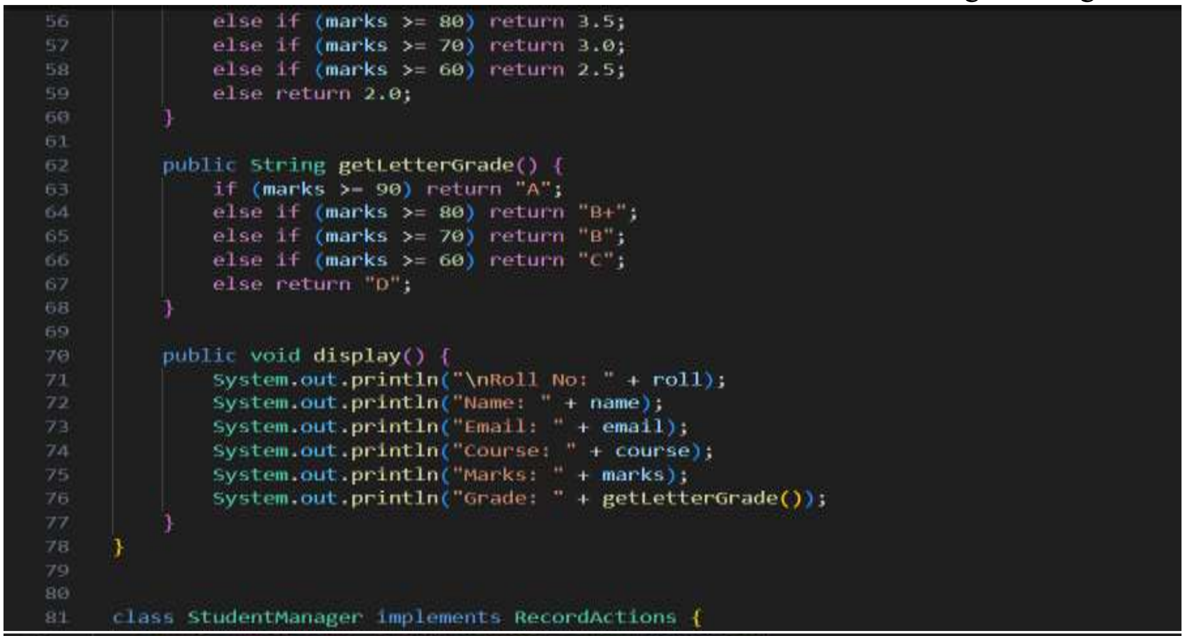

Start
Input Roll Number, Name, Email, Course, Marks
Invalid input? (e.g. Marks)
Yes
No
Empty Name or Course?
Yes
No
Print "Invalid input'
Service Implementaton
Print "Name or course cannot be empty"
Loading Thread
Simulate loading...
Create Student object
Calculate Grade and Display
Print "Program execution completed."
End



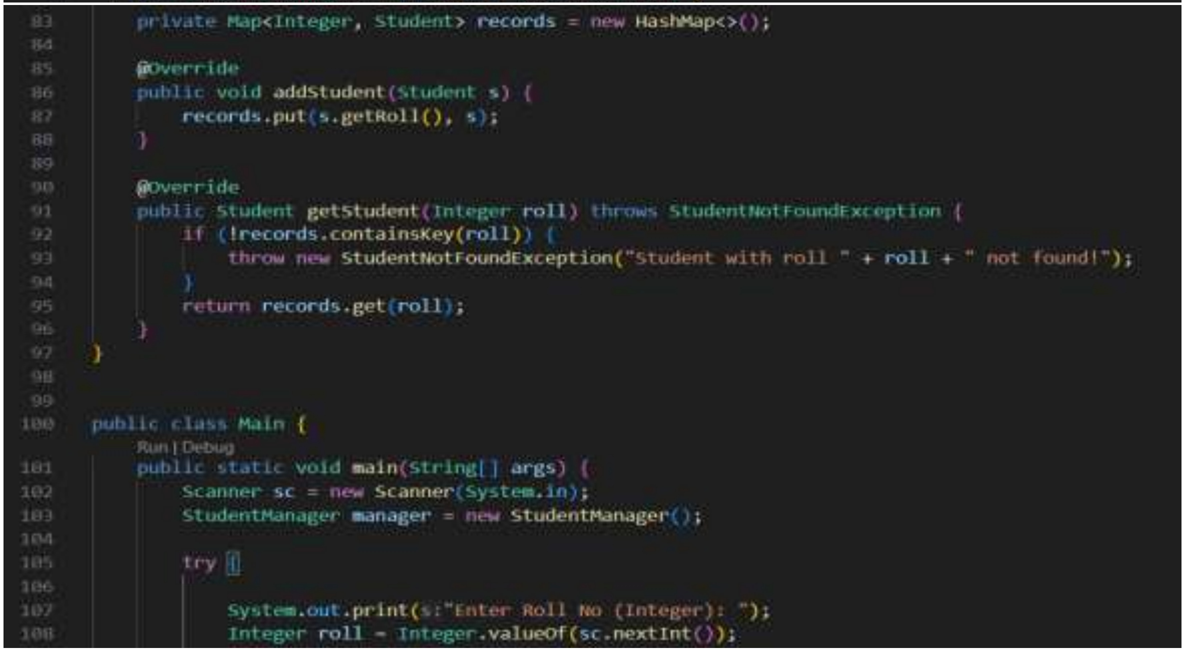
```java
import java.util.*;


class StudentNotFoundException extends Exception {
    public StudentNotFoundException(String message) {
        super(message);
    }
}


class Loader implements Runnable {
    @Override
    public void run() {
        try {
            System.out.print(s:"Loading");
            for (int i = 0; i < 5; i++) {
                Thread.sleep(millis:300);
                System.out.print(s:".");
            }
            System.out.println();
        } catch (InterruptedException e) {
            System.out.println(x:"Loading interrupted!");
        }
    }
}


```



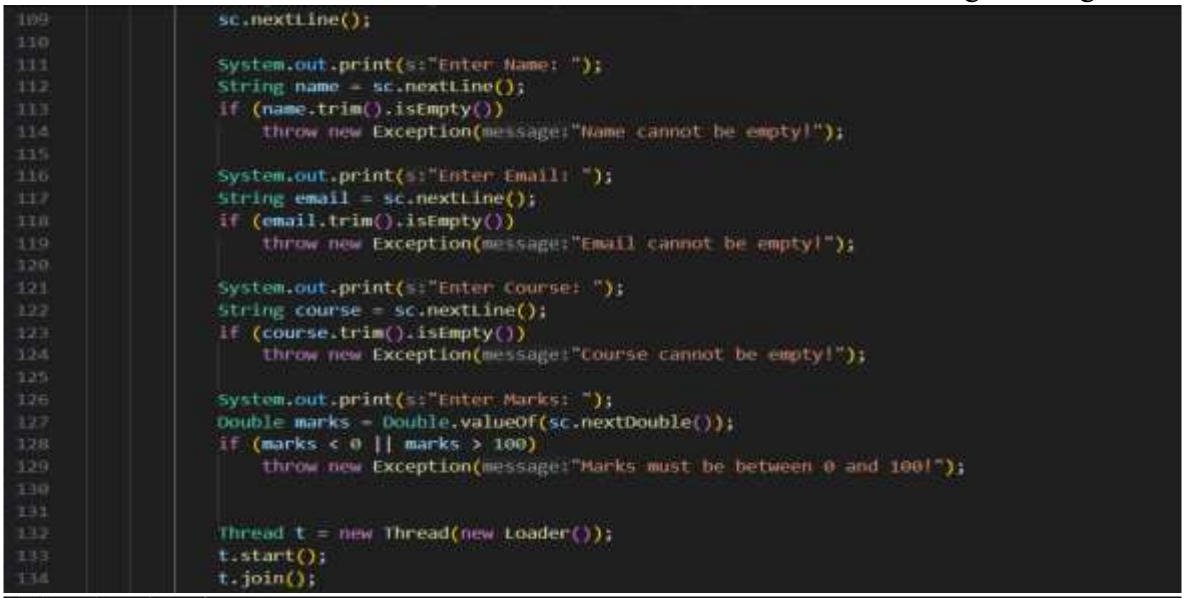
```java
interface RecordActions {
    void addStudent(Student s);
    Student getStudent(Integer roll) throws StudentNotFoundException;
}


class Student {
    private Integer roll;
    private String name;
    private String email;
    private String course;
    private Double marks;

    public Student(Integer roll, String name, String email, String course, Double marks) {
        this.roll = roll;
        this.name = name;
        this.email = email;
        this.course = course;
        this.marks = marks;
    }


    public Integer getRoll() {
        return roll;
    }

    public Double calculateGrade() {
        if (marks >= 90) return 4.0;
```



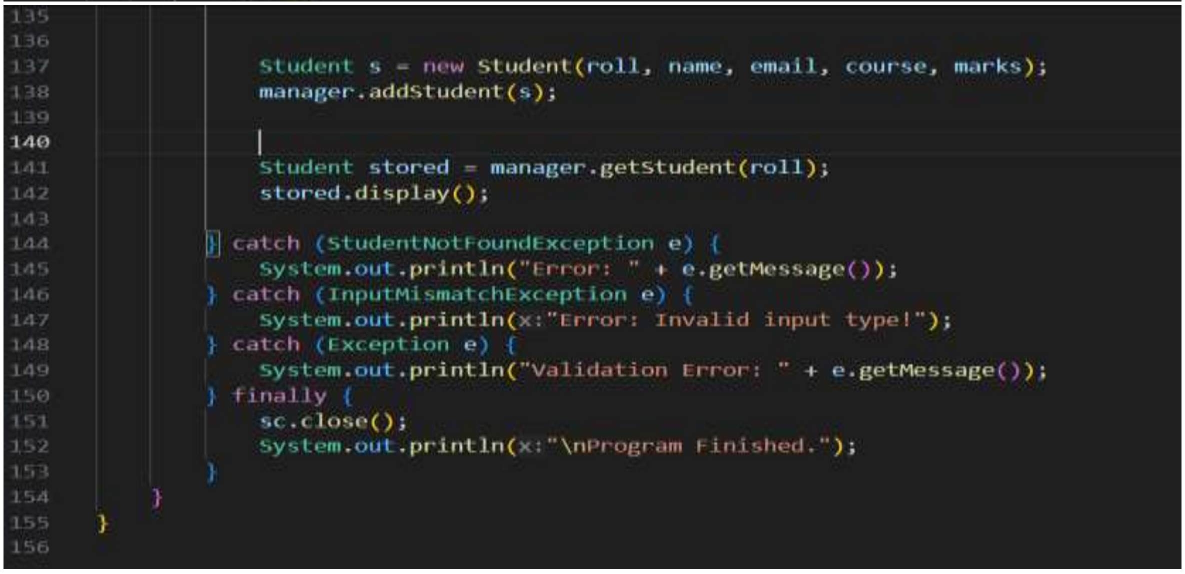
```java
        else if (marks >= 80) return 3.5;
        else if (marks >= 70) return 3.0;
        else if (marks >= 60) return 2.5;
        else return 2.0;
    }

    public String getLetterGrade() {
        if (marks >= 90) return "A";
        else if (marks >= 80) return "B+";
        else if (marks >= 70) return "B";
        else if (marks >= 60) return "C";
        else return "D";
    }

    public void display() {
        System.out.println("\nRoll No: " + roll);
        System.out.println("Name: " + name);
        System.out.println("Email: " + email);
        System.out.println("Course: " + course);
        System.out.println("Marks: " + marks);
        System.out.println("Grade: " + getLetterGrade());
    }
}


class StudentManager implements RecordActions {
```



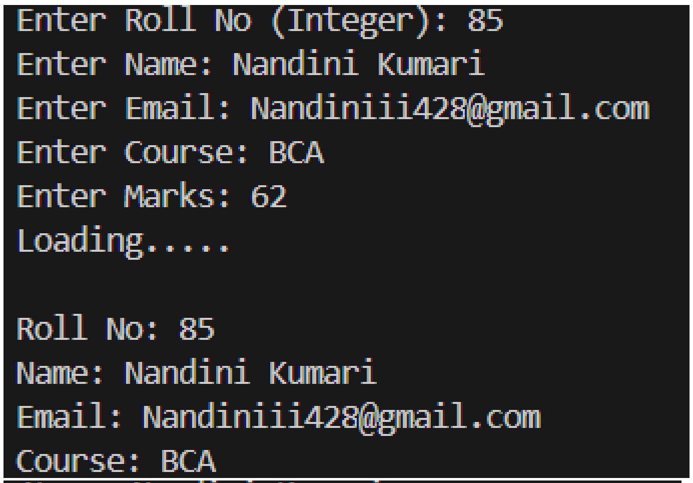
```java
    private Map<Integer, Student> records = new HashMap<>();

    @Override
    public void addStudent(Student s) {
        records.put(s.getRoll(), s);
    }

    @Override
    public Student getStudent(Integer roll) throws StudentNotFoundException {
        if (!records.containsKey(roll)) {
            throw new StudentNotFoundException("Student with roll " + roll + " not found!");
        }
        return records.get(roll);
    }
}


public class Main {
    Run | Debug
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        StudentManager manager = new StudentManager();

        try {

            System.out.print(s:"Enter Roll No (Integer): ");
            Integer roll = Integer.valueOf(sc.nextInt());
```



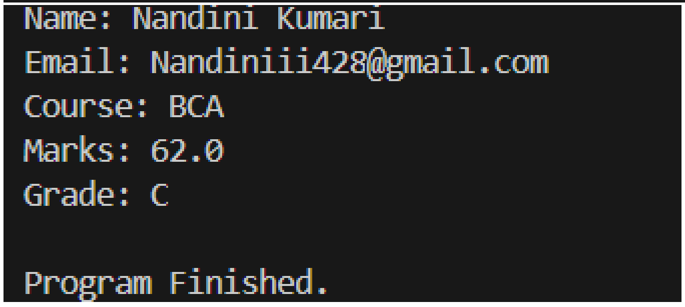
```
            sc.nextLine();

            System.out.print(s:"Enter Name: ");
            String name = sc.nextLine();
            if (name.trim().isEmpty())
                throw new Exception(message:"Name cannot be empty!");

            System.out.print(s:"Enter Email: ");
            String email = sc.nextLine();
            if (email.trim().isEmpty())
                throw new Exception(message:"Email cannot be empty!");

            System.out.print(s:"Enter Course: ");
            String course = sc.nextLine();
            if (course.trim().isEmpty())
                throw new Exception(message:"Course cannot be empty!");

            System.out.print(s:"Enter Marks: ");
            Double marks = Double.valueOf(sc.nextDouble());
            if (marks < 0 || marks > 100)
                throw new Exception(message:"Marks must be between 0 and 100!");


            Thread t = new Thread(new Loader());
            t.start();
            t.join();
```



```


            Student s = new Student(roll, name, email, course, marks);
            manager.addStudent(s);


            Student stored = manager.getStudent(roll);
            stored.display();

        } catch (StudentNotFoundException e) {
            System.out.println("Error: " + e.getMessage());
        } catch (InputMismatchException e) {
            System.out.println(x:"Error: Invalid input type!");
        } catch (Exception e) {
            System.out.println("Validation Error: " + e.getMessage());
        } finally {
            sc.close();
            System.out.println(x:"\nProgram Finished.");
        }
    }
}
```



```
Enter Roll No (Integer): 85
Enter Name: Nandini Kumari
Enter Email: Nandiniii428@gmail.com
Enter Course: BCA
Enter Marks: 62
Loading.....

Roll No: 85
Name: Nandini Kumari
Email: Nandiniii428@gmail.com
Course: BCA
```



```
Name: Nandini Kumari
Email: Nandiniii428@gmail.com
Course: BCA
Marks: 62.0
Grade: C

Program Finished.
```

Pupil Records Manager – Detailed Explanation

Pupil Records Manager – Detailed Explanation

This Java program is designed to manage pupil records using object-oriented programming, file handling, custom exceptions, and menu-driven console interaction. It is structured into several classes, each handling a specific responsibility.

1. Pupil Class

The Pupil class represents a single student's record. It stores attributes such as roll number, name, email, course, and marks. The constructor trims empty inputs, and the validate() method ensures that all fields are filled and marks fall between 0 and 100. It also provides a toCSV() method for file storage and toString() for displaying formatted output.

2. PupilManager Class

This class manages all pupil records and handles loading/saving data from a text file. It keeps an ArrayList of Pupil objects and performs: • Adding records (while preventing duplicate roll numbers)

- Searching pupils by name
- Deleting pupils by name
- Viewing all pupils
- Saving the list back to the file in CSV format It also calculates grades based on marks.

3. Custom Exception – PupilNotFoundException

This exception is thrown when a search or delete request is made for a pupil that does not exist.

4. RecordOps Interface
Defines the required operations such as addRecord(), searchByName(), deleteByName(), viewAll(), and saveToFile(). The PupilManager class implements this interface.

5. AppRunner (Main Program)
This class creates a menu-driven loop that takes user input and performs the requested actions. It displays: 1. Add Pupil
2. View All
3. Search by Name
4. Delete by Name
5. Save & Exit
For each operation, the program reads user input, runs a Spinner thread (for a loading animation), and calls the appropriate PupilManager method.
The program handles various exceptions including invalid numbers, validation errors, missing records, and thread interruptions.

6. Overall Workflow
When the program starts, it loads past records from the file. The user can then continuously add, search, delete, or view data until they choose to save and exit. Upon exit, all records are written back to the file.

This program demonstrates object-oriented design, exception handling, multithreading, and file-based persistence in Java.

40

Java Lab Assignment 4

A Java Application for a basic shape drawing application

Problem Statement

Implement a Student Record Management System with persistent storage using file handling and Java Collections Framework. The system should read student records from a file (students.txt) at the start of the application and save updated records back to the file upon exit. The records should be managed using collections like ArrayList or HashMap to store student information, and should be sorted by marks using Comparator. The system should allow for viewing, sorting, and displaying student data using Iterator. Additionally, implement file attributes using the File class and demonstrate reading records randomly using RandomAccessFile.

Objective:
Implement file handling and use the collections API to manage student records efficiently.

Learning Outcomes

Upon completion of this assignment, the student will be able to:

1. Implement file handling for storing and retrieving student records.
2. Use collections (List, Map) to manage and manipulate records.
3. Sort and display records using Comparator, Comparable, and Iterator.

Class Hierarchy & Data Types

Class Hierarchy:

1.FileUtil: Contains methods for reading and writing to file.
2.StudentManager: Manages student records. Data Types:

- ArrayList<Student>: For managing student records.
- BufferedReader, BufferedWriter: For file handling.

Detailed Instructions

1. File Handling: Use BufferedReader and BufferedWriter to read and write student data to a file.
2. Sorting: Sort students by marks using Comparator.
3. Displaying: Use Iterator to display student data.

Expected Output

```
Loaded students from file:
Roll No: 101
Name: Ankit
Email: ankit@mail.com
Course: B.Tech
Marks: 85.5
----------------
Roll No: 102
Name: Riya
Email: riya@mail.com
Course: M.Tech
```

```
Marks: 91.0
----------------

===== Capstone Student Menu =====
1.      Add Student
2.      View All Students
3.      Search by Name
4.      Delete by Name
5.      Sort by Marks 6. Save and Exit Enter choice: 1
Enter Roll No: 103
Enter Name: Karan
Enter Email: karan@mail.com
Enter Course: BCA
Enter Marks: 76.2

Sorted Student List by Marks:
Roll No: 102
Name: Riya
Email: riya@mail.com
Course: M.Tech
Marks: 91.0
```

Guidelines to Students

1. File Handling: Ensure proper error handling for file read/write operations.
2. Collections: Use Map and List for managing and displaying student data.
3. Sorting: Implement sorting both by marks and by name using Comparator.

Improvements/Adjustments

1. Queue Implementation: Use PriorityQueue for storing and retrieving students based on different criteria.
2. File Format Enhancement: Consider saving records in a more structured format like CSV or JSON.

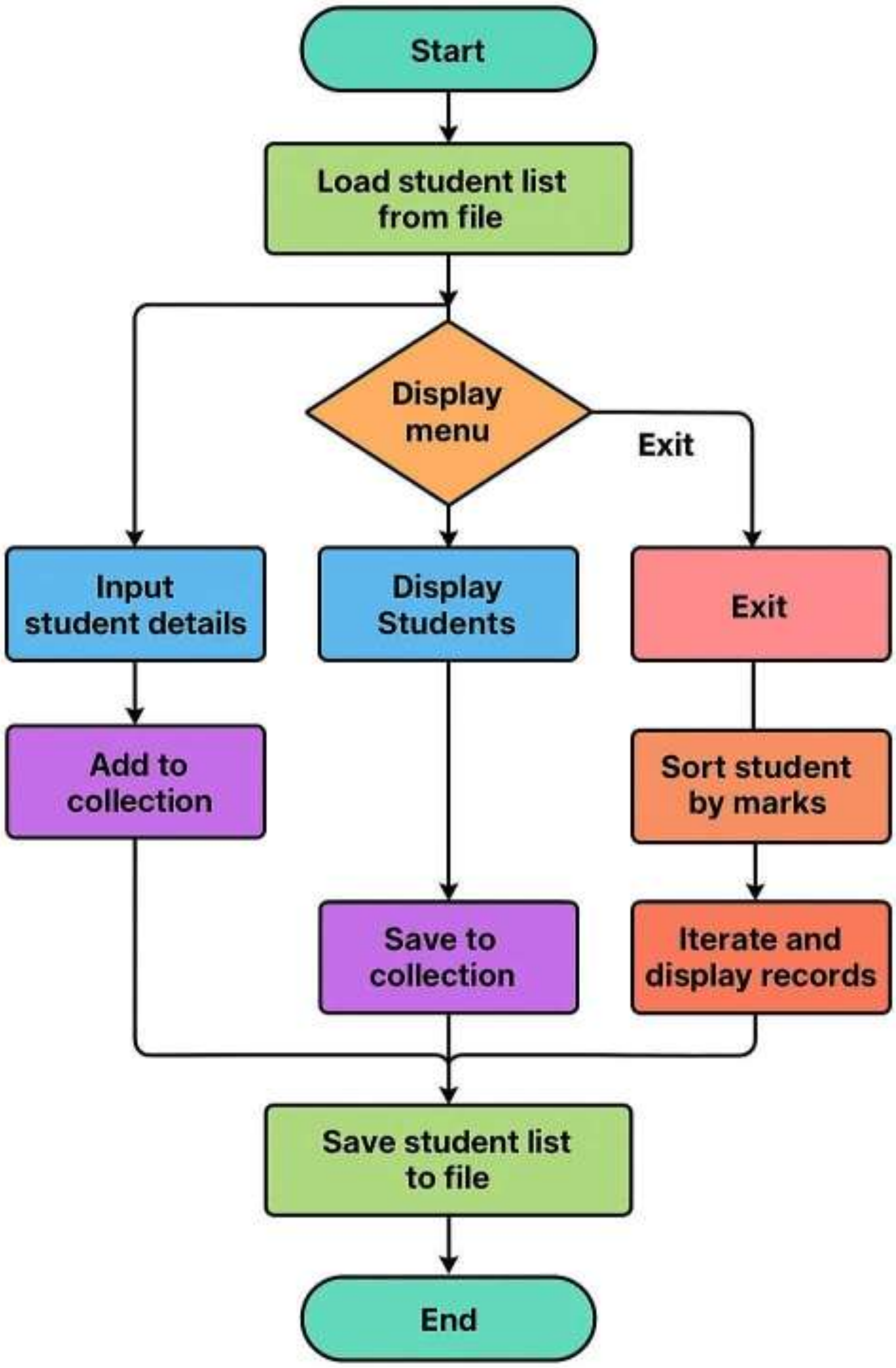
Submission Guidelines

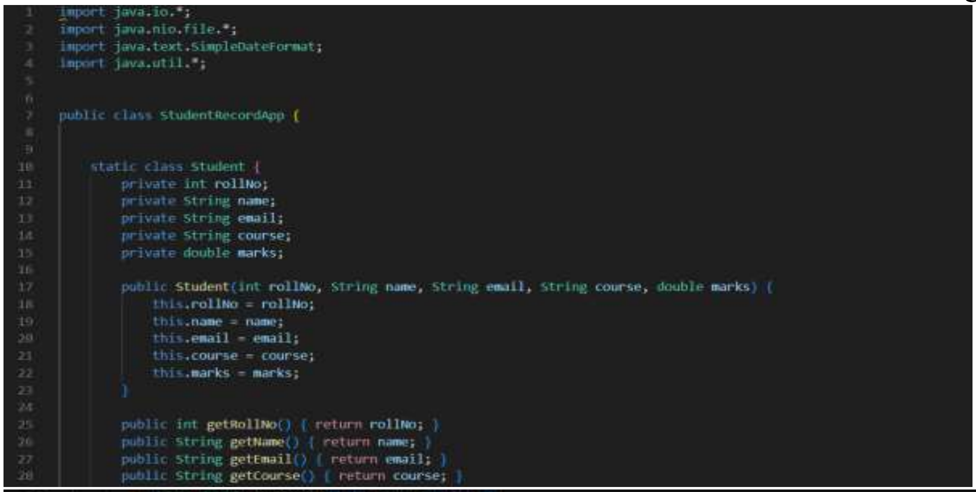
1. Submit all source files along with necessary libraries.
2. Ensure proper formatting and comments in your code.

Performance Metrics (Out of 10 Marks)

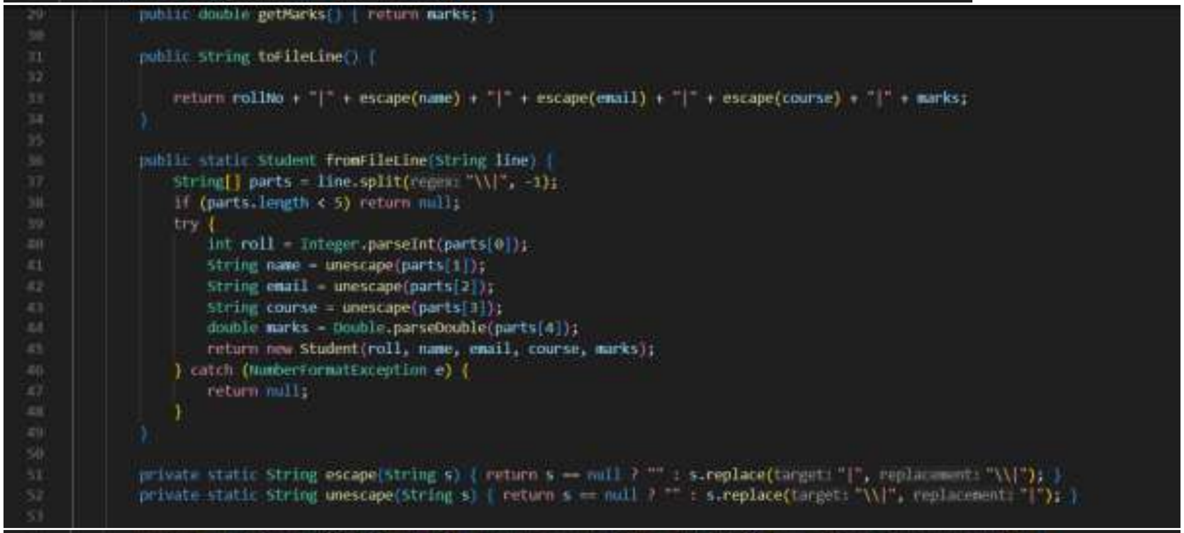
Criteria	Marks
File Handling and Persistence	3
Sorting and Display with Collections	2
Iterator and Data Management	2
Code Structure and Comments	2
Testing and Output Validation	1

Flow Chart:

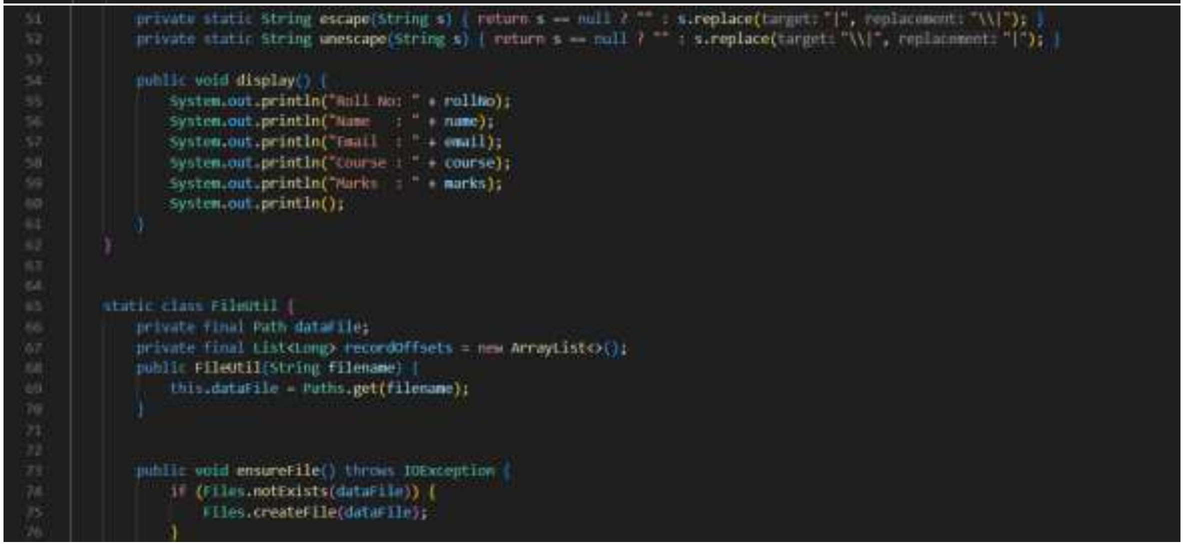

Start
Load student list from file
Display menu
Exit
Input student details
Display Students
Exit
Add to collection
Sort student by marks
Save to collection
Iterate and display records
Save student list to file
End



```java
import java.io.*;
import java.nio.file.*;
import java.text.SimpleDateFormat;
import java.util.*;


public class StudentRecordApp {


    static class Student {
        private int rollNo;
        private String name;
        private String email;
        private String course;
        private double marks;

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

        public int getRollNo() { return rollNo; }
        public String getName() { return name; }
        public String getEmail() { return email; }
        public String getCourse() { return course; }
```

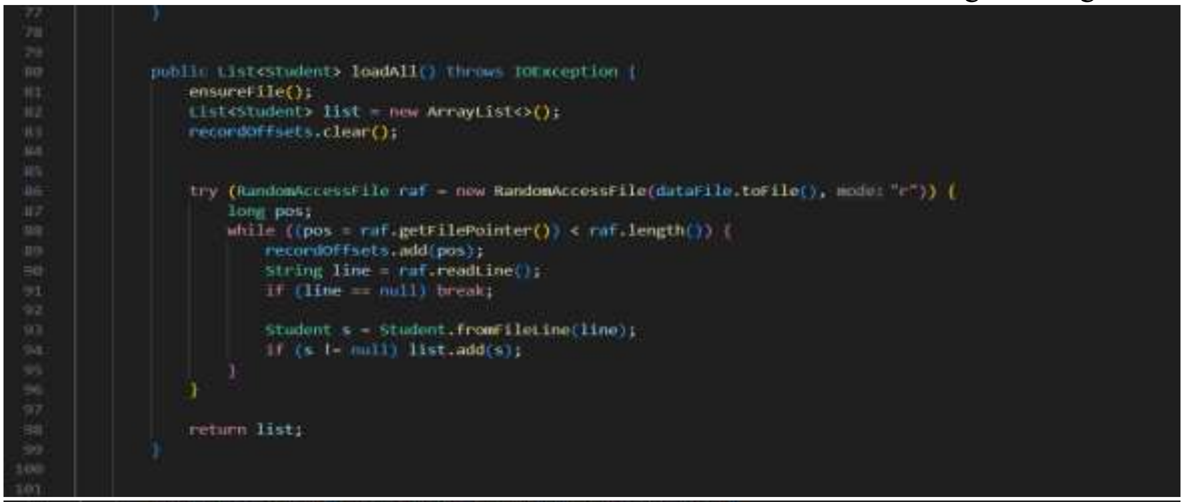


```java
        public double getMarks() { return marks; }

        public String toFileLine() {

            return rollNo + "|" + escape(name) + "|" + escape(email) + "|" + escape(course) + "|" + marks;
        }

        public static Student fromFileLine(String line) {
            String[] parts = line.split(regex: "\\|", -1);
            if (parts.length < 5) return null;
            try {
                int roll = Integer.parseInt(parts[0]);
                String name = unescape(parts[1]);
                String email = unescape(parts[2]);
                String course = unescape(parts[3]);
                double marks = Double.parseDouble(parts[4]);
                return new Student(roll, name, email, course, marks);
            } catch (NumberFormatException e) {
                return null;
            }
        }

        private static String escape(String s) { return s == null ? "" : s.replace(target: "|", replacement: "\\|"); }
        private static String unescape(String s) { return s == null ? "" : s.replace(target: "\\|", replacement: "|"); }

```

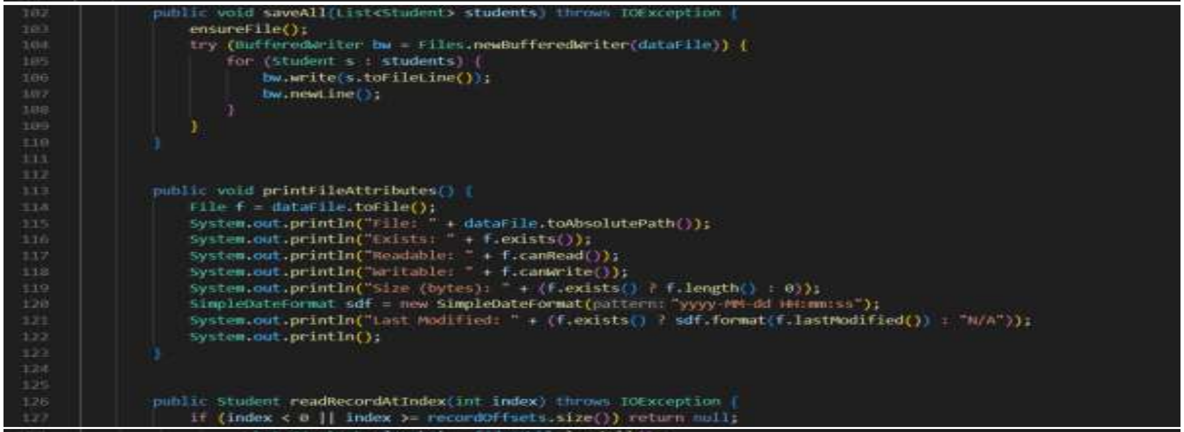


```java
        private static String escape(String s) { return s == null ? "" : s.replace(target: "|", replacement: "\\|"); }
        private static String unescape(String s) { return s == null ? "" : s.replace(target: "\\|", replacement: "|"); }
```

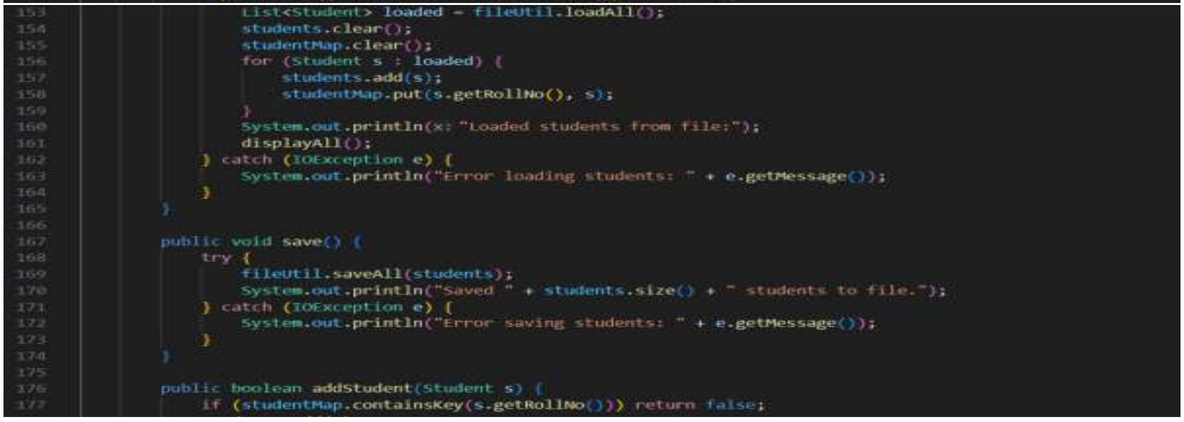
```java

        public void display() {
            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();
        }
    }


    static class FileUtil {
        private final Path dataFile;
        private final List<Long> recordOffsets = new ArrayList<>();
        public FileUtil(String filename) {
            this.dataFile = Paths.get(filename);
        }


        public void ensureFile() throws IOException {
            if (Files.notExists(dataFile)) {
                Files.createFile(dataFile);
            }
```



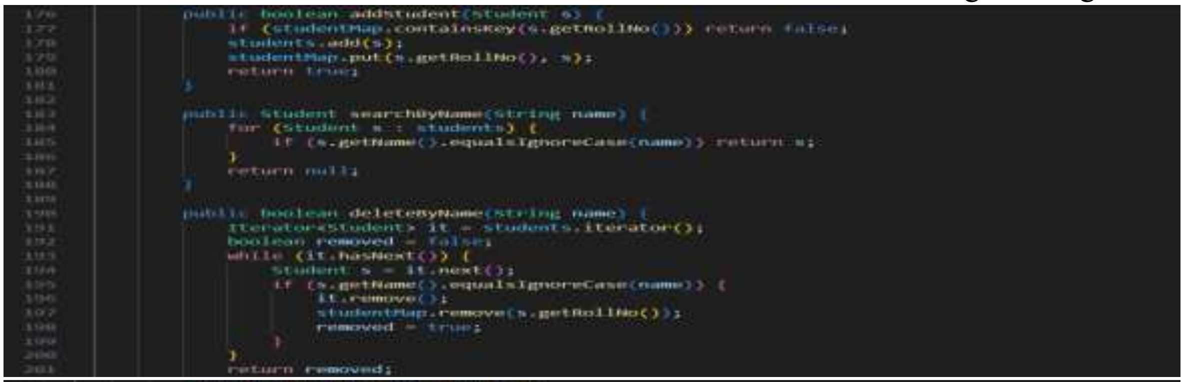
```
    }


    public List<Student> loadAll() throws IOException {
        ensureFile();
        List<Student> list = new ArrayList<>();
        recordOffsets.clear();


        try (RandomAccessFile raf = new RandomAccessFile(dataFile.toFile(), mode: "r")) {
            long pos;
            while ((pos = raf.getFilePointer()) < raf.length()) {
                recordOffsets.add(pos);
                String line = raf.readLine();
                if (line == null) break;

                Student s = Student.fromFileLine(line);
                if (s != null) list.add(s);
            }
        }

        return list;
    }

```



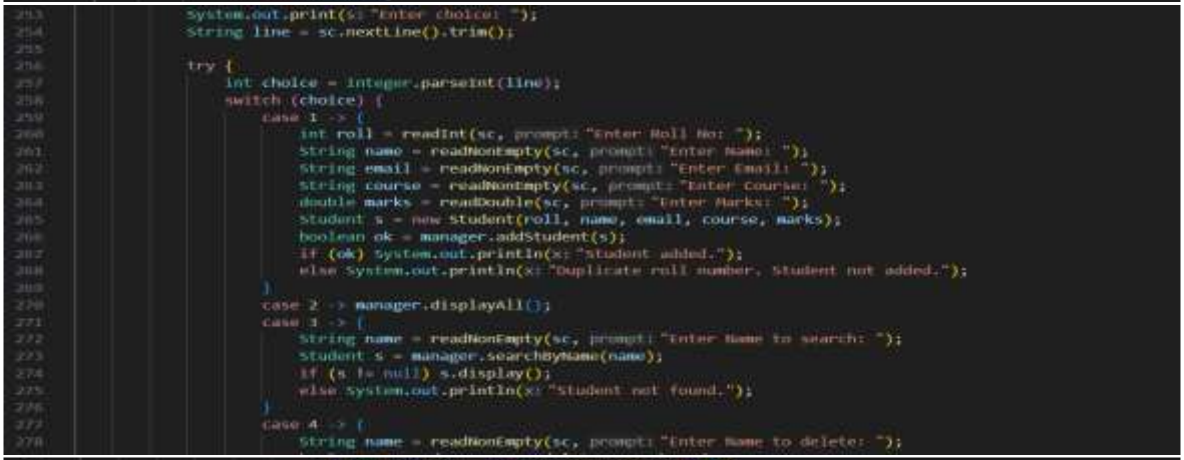
```
    public void saveAll(List<Student> students) throws IOException {
        ensureFile();
        try (BufferedWriter bw = Files.newBufferedWriter(dataFile)) {
            for (Student s : students) {
                bw.write(s.toFileLine());
                bw.newLine();
            }
        }
    }


    public void printFileAttributes() {
        File f = dataFile.toFile();
        System.out.println("File: " + dataFile.toAbsolutePath());
        System.out.println("Exists: " + f.exists());
        System.out.println("Readable: " + f.canRead());
        System.out.println("Writable: " + f.canWrite());
        System.out.println("Size (bytes): " + (f.exists() ? f.length() : 0));
        SimpleDateFormat sdf = new SimpleDateFormat(pattern: "yyyy-MM-dd HH:mm:ss");
        System.out.println("Last Modified: " + (f.exists() ? sdf.format(f.lastModified()) : "N/A"));
        System.out.println();
    }


    public Student readRecordAtIndex(int index) throws IOException {
        if (index < 0 || index >= recordOffsets.size()) return null;
```



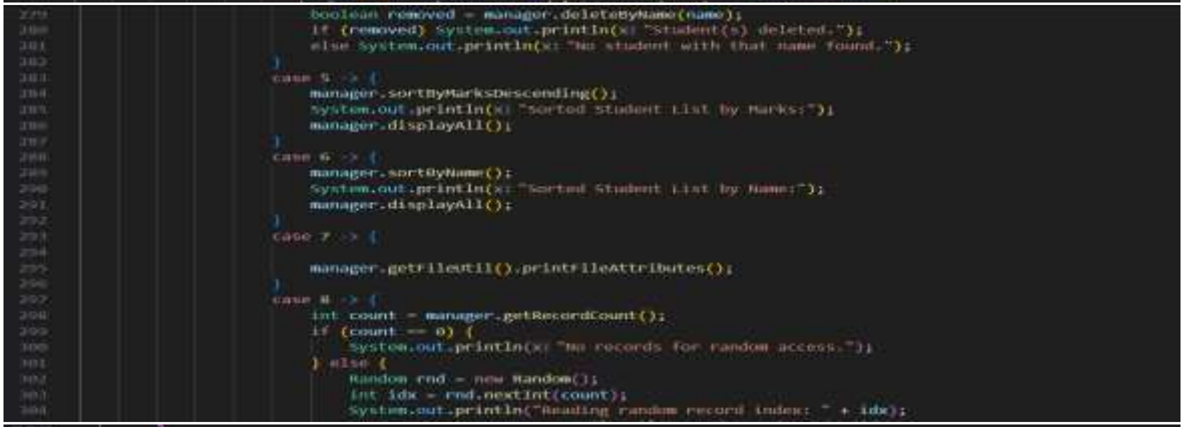
```
            List<Student> loaded = fileUtil.loadAll();
            students.clear();
            studentMap.clear();
            for (Student s : loaded) {
                students.add(s);
                studentMap.put(s.getRollNo(), s);
            }
            System.out.println(x: "Loaded students from file:");
            displayAll();
        } catch (IOException e) {
            System.out.println("Error loading students: " + e.getMessage());
        }
    }

    public void save() {
        try {
            fileUtil.saveAll(students);
            System.out.println("Saved " + students.size() + " students to file.");
        } catch (IOException e) {
            System.out.println("Error saving students: " + e.getMessage());
        }
    }

    public boolean addStudent(Student s) {
        if (studentMap.containsKey(s.getRollNo())) return false;
```



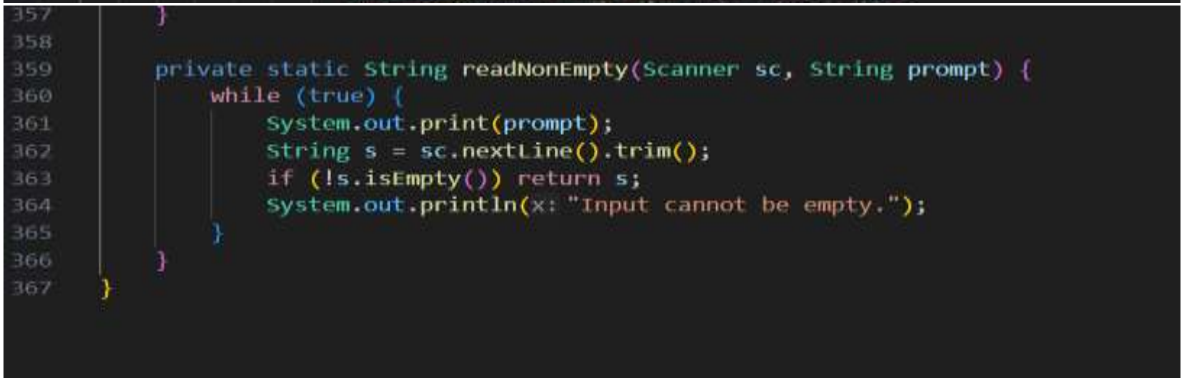
```java
    public boolean addStudent(Student s) {
        if (studentMap.containsKey(s.getRollNo())) return false;
        students.add(s);
        studentMap.put(s.getRollNo(), s);
        return true;
    }

    public Student searchByName(String name) {
        for (Student s : students) {
            if (s.getName().equalsIgnoreCase(name)) return s;
        }
        return null;
    }

    public boolean deleteByName(String name) {
        Iterator<Student> it = students.iterator();
        boolean removed = false;
        while (it.hasNext()) {
            Student s = it.next();
            if (s.getName().equalsIgnoreCase(name)) {
                it.remove();
                studentMap.remove(s.getRollNo());
                removed = true;
            }
        }
        return removed;
```



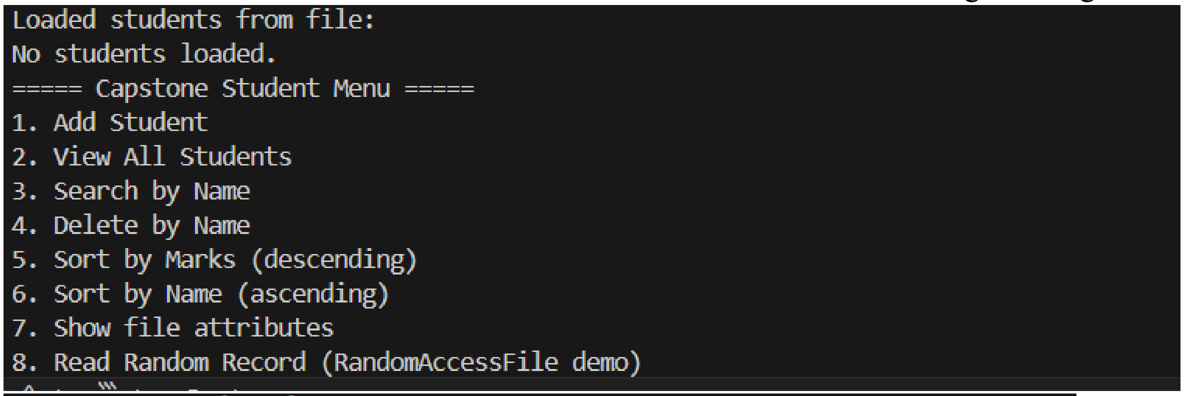
```java
            System.out.print(s: "Enter choice: ");
            String line = sc.nextLine().trim();

            try {
                int choice = Integer.parseInt(line);
                switch (choice) {
                    case 1 -> {
                        int roll = readInt(sc, prompt: "Enter Roll No: ");
                        String name = readNonEmpty(sc, prompt: "Enter Name: ");
                        String email = readNonEmpty(sc, prompt: "Enter Email: ");
                        String course = readNonEmpty(sc, prompt: "Enter Course: ");
                        double marks = readDouble(sc, prompt: "Enter Marks: ");
                        Student s = new Student(roll, name, email, course, marks);
                        boolean ok = manager.addStudent(s);
                        if (ok) System.out.println(x: "Student added.");
                        else System.out.println(x: "Duplicate roll number. Student not added.");
                    }
                    case 2 -> manager.displayAll();
                    case 3 -> {
                        String name = readNonEmpty(sc, prompt: "Enter Name to search: ");
                        Student s = manager.searchByName(name);
                        if (s != null) s.display();
                        else System.out.println(x: "Student not found.");
                    }
                    case 4 -> {
                        String name = readNonEmpty(sc, prompt: "Enter Name to delete: ");
```



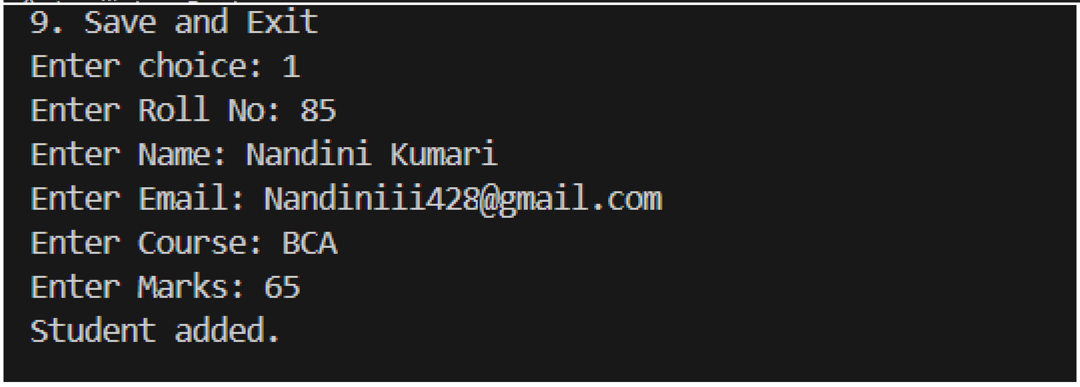
```java
                        boolean removed = manager.deleteByName(name);
                        if (removed) System.out.println(x: "Student(s) deleted.");
                        else System.out.println(x: "No student with that name found.");
                    }
                    case 5 -> {
                        manager.sortByMarksDescending();
                        System.out.println(x: "Sorted Student List by Marks:");
                        manager.displayAll();
                    }
                    case 6 -> {
                        manager.sortByName();
                        System.out.println(x: "Sorted Student List by Name:");
                        manager.displayAll();
                    }
                    case 7 -> {

                        manager.getFileUtil().printFileAttributes();
                    }
                    case 8 -> {
                        int count = manager.getRecordCount();
                        if (count == 0) {
                            System.out.println(x: "No records for random access.");
                        } else {
                            Random rnd = new Random();
                            int idx = rnd.nextInt(count);
                            System.out.println("Reading random record index: " + idx);
```



```java
    }

    private static String readNonEmpty(Scanner sc, String prompt) {
        while (true) {
            System.out.print(prompt);
            String s = sc.nextLine().trim();
            if (!s.isEmpty()) return s;
            System.out.println(x: "Input cannot be empty.");
        }
    }
}
```



```
Loaded students from file:
No students loaded.
===== Capstone Student Menu =====
1. Add Student
2. View All Students
3. Search by Name
4. Delete by Name
5. Sort by Marks (descending)
6. Sort by Name (ascending)
7. Show file attributes
8. Read Random Record (RandomAccessFile demo)
```



```
9. Save and Exit
Enter choice: 1
Enter Roll No: 85
Enter Name: Nandini Kumari
Enter Email: Nandiniii428@gmail.com
Enter Course: BCA
Enter Marks: 65
Student added.
```

Student Records Manager (Lab 4) – Detailed Explanation

This Java program is a complete student record management system that uses object-oriented programming, file handling, RandomAccessFile, collections, and stream operations. It is divided into several classes, each responsible for a different part of the program.

1. FileUtil4 Class
This utility class handles all file operations:
- **readStudents(): Reads student data from a CSV text file. If the file does not exist, it creates one with a header.**
- **writeStudents(): Saves all student records back to the file in CSV format.**
- **readLineAt(): Uses RandomAccessFile to read a specific line (by index) from the data file, skipping the header. This demonstrates random access reading.**

2. Student Class
This class represents a single student record. It stores:
- **rollNo**
- **name**
- **email**
- **course**
- **marks**

It also includes:
- **Getters and setters**

• CSV conversion (toCSV())
• Comparable implementation (naturally sorted by roll number)
• equals() and hashCode() based on roll number
• Simple string representation for display

3. StudentManager4 Class
This is the main class responsible for managing student data in memory. It stores:
• A List<Student> for ordered data
• A Map<Integer, Student> (byRoll) for fast roll-based lookups The class provides functionality such as:
• Adding a student (checking for duplicate roll numbers)
• Viewing all students
• Searching by exact name
• Searching by partial name using Streams
• Deleting by roll or by name
• Sorting by marks (ascending or descending)
• Sorting by name
• Updating marks of a specific student
• Saving updates to the file

4. Main4 Class (User Interface)
This class provides a menu-driven console interface:
1) Add Student
2) Display All Students
3) Search (exact name)
4) Search (partial name)
5) Delete by Roll
6) Delete by Name
7) Sort by Marks (descending)
8) Sort by Marks (ascending)
9) Update Marks
10) Save & Exit
The program validates numerical inputs and catches exceptions to prevent crashes.

After exiting, it demonstrates RandomAccessFile by reading the second student record directly from the file.

5. Overall Workflow
• On launch, the program loads data from the file.
• Users can perform operations such as adding, searching, deleting, sorting, and updating data.
• Before exiting, the program writes all changes back to the file.
• The system uses both list and map structures for efficient operations.
• RandomAccessFile is used to show fast, line-based lookup.

This Lab 4 program demonstrates strong use of Java collections, file I/O, error handling, streams, random access, and object-oriented design principles.

Java Lab Assignment 5

A Java multithreaded application that simulates a banking system

Problem Statement

Design and implement a Student Record Management System using Java that allows for the management of student records (add, update, delete, search, and view) with persistent storage. The application must support exception handling, file handling (to store and retrieve data), multithreading (to simulate loading), and must leverage the Java Collections Framework. The system should allow sorting of students by marks, provide the option to search and delete student records, and display the sorted list of students. Additionally, the system should use OOP concepts like inheritance, abstraction, and interfaces to ensure modular and reusable code.

Learning Outcomes

Upon completion of this assignment, the students will be able to:

1. Design and implement an object-oriented system using classes, inheritance, and interfaces.
2. Use exception handling to ensure safe program execution and validation.
3. Implement file I/O for persistent data storage using Java's BufferedReader and BufferedWriter.
4. Use Java Collections (List, Map, Set) to manage and manipulate student records.
5. Sort student records using Comparator and display records via Iterator.
6. Implement and understand multithreading for responsive user interaction.
7. Apply custom exceptions and perform input validation.
8. Understand modular programming with packages for better code organization and reusability.

Class Hierarchy & Data Types

Class Hierarchy:

1. Person (abstract class) o Fields: name, email o Methods:

displayInfo() (abstract)

2. Student (extends Person)
   o Fields: rollNo, course, marks, grade o Methods: inputDetails(), displayDetails(), calculateGrade()
3. StudentManager (implements RecordActions interface) o Methods: addStudent(), deleteStudent(), updateStudent(), searchStudent(), viewAllStudents()
4. Loader (implements Runnable) o Methods: run() (for simulating loading in multithreading) Data Types:

- String: For student name, email, course.
- int: For rollNo.
- double: For marks.
- List<Student>: For storing students.
- Map<Integer, Student>: For storing students in a map with rollNo as the key.
- Thread: For multithreading to simulate a loading process.

Detailed Instructions

1. Core Design: Create the abstract class Person with basic fields like name and email, and extend it in the Student class. Include methods like inputDetails(), displayDetails(), and calculateGrade() based on marks.
2. Interface Implementation: Create a RecordActions interface and implement it in the StudentManager class. Include methods like addStudent(), deleteStudent(), updateStudent(), searchStudent(), and viewAllStudents(). Implement validations for duplicate rollNo.
3. Exception Handling: Implement appropriate try-catch-finally blocks for handling invalid input (marks outside the valid range, empty fields, invalid rollNo) and create custom exceptions like StudentNotFoundException.
4. File I/O: Implement BufferedReader and BufferedWriter to load and save student records from/to a file (students.txt). Handle file reading and writing with exception handling.
5. Multithreading: Use a Thread to simulate a delay when performing actions like adding or saving records, showing the loading state.
6. Sorting and Display: Implement sorting of student records by marks in descending order using Comparator. Use Iterator to display the records in a sorted order.

Expected Output

The program should output the following results based on user interaction:

Example Output:

```
===== Capstone Student Menu =====
1. Add Student
2. View All Students
3. Search by Name
4. Delete by Name
5. Sort by Marks 6. Save and Exit
Enter choice: 1
Enter Roll No: 101
Enter Name: Rahul
Enter Email: rahul@mail.com
Enter Course: B.Tech
Enter Marks: 85.0

===== Capstone Student Menu =====
1. Add Student
2. View All Students
3. Search by Name
4. Delete by Name
5. Sort by Marks 6. Save and Exit
Enter choice: 2
Roll No: 101
Name:   Rahul   Email:
rahul@mail.com
Course: B.Tech Marks:
85.0
------------------

===== Capstone Student Menu =====
1. Add Student
2. View All Students
3. Search by Name
4. Delete by Name
5. Sort by Marks 6. Save and Exit
Enter choice: 3
Enter name to search: Rahul Student
Info:
Roll No: 101
Name:   Rahul   Email:
rahul@mail.com
Course: B.Tech Marks:
85.0
------------------

===== Capstone Student Menu =====
1. Add Student
2. View All Students
3. Search by Name
4. Delete by Name
5. Sort by Marks
6. Save and Exit
Enter choice: 4
Enter name to delete: Rahul Student record
deleted.

===== Capstone Student Menu =====
1. Add Student
2. View All Students
3. Search by Name
4. Delete by Name
5. Sort by Marks 6. Save and Exit
Enter choice: 5
Sorted Student List by Marks:
Roll No: 101
Name:   Rahul   Email:
rahul@mail.com
Course: B.Tech Marks:
85.0
------------------

===== Capstone Student Menu =====
1.      Add Student
2.      View All Students
3.      Search by Name
```

4. Delete by Name
5. Sort by Marks 6. Save and Exit Enter choice: 6 Saved and exiting.

Guidelines to Students

1. Code Structure:
    - o Ensure all classes are correctly placed within their respective packages (model, service, util).
    - o Use object-oriented principles like inheritance and interfaces for clean code.
2. Modularity:
    - o Keep methods short, clear, and reusable.
    - o Handle all exceptions with meaningful messages.
3. File Handling:
    - o Handle file reading/writing operations with BufferedReader and BufferedWriter. o Ensure the program can load existing student records on startup and save updated records before exiting.
4. Multithreading:
    - o Simulate a realistic loading experience by using a Thread class.

Improvements/Adjustments

1. GUI Enhancement:
    - o Optional enhancement: Implement a simple GUI using JavaFX or Swing to replace the console-based interface.
2. Advanced Sorting: o Add sorting features for Student class (e.g., sorting by name or course).
3. Custom Data Validation:
    - o Use more complex validations like validating email format and proper name formatting.
4. Database Integration:
    - o (Optional) Integrate SQLite or another database for more robust data management.

Submission Guidelines

1. Code Submission: o Submit the entire project folder with all Java source files.
    - o Ensure proper indentation and readable code.
2. Documentation:

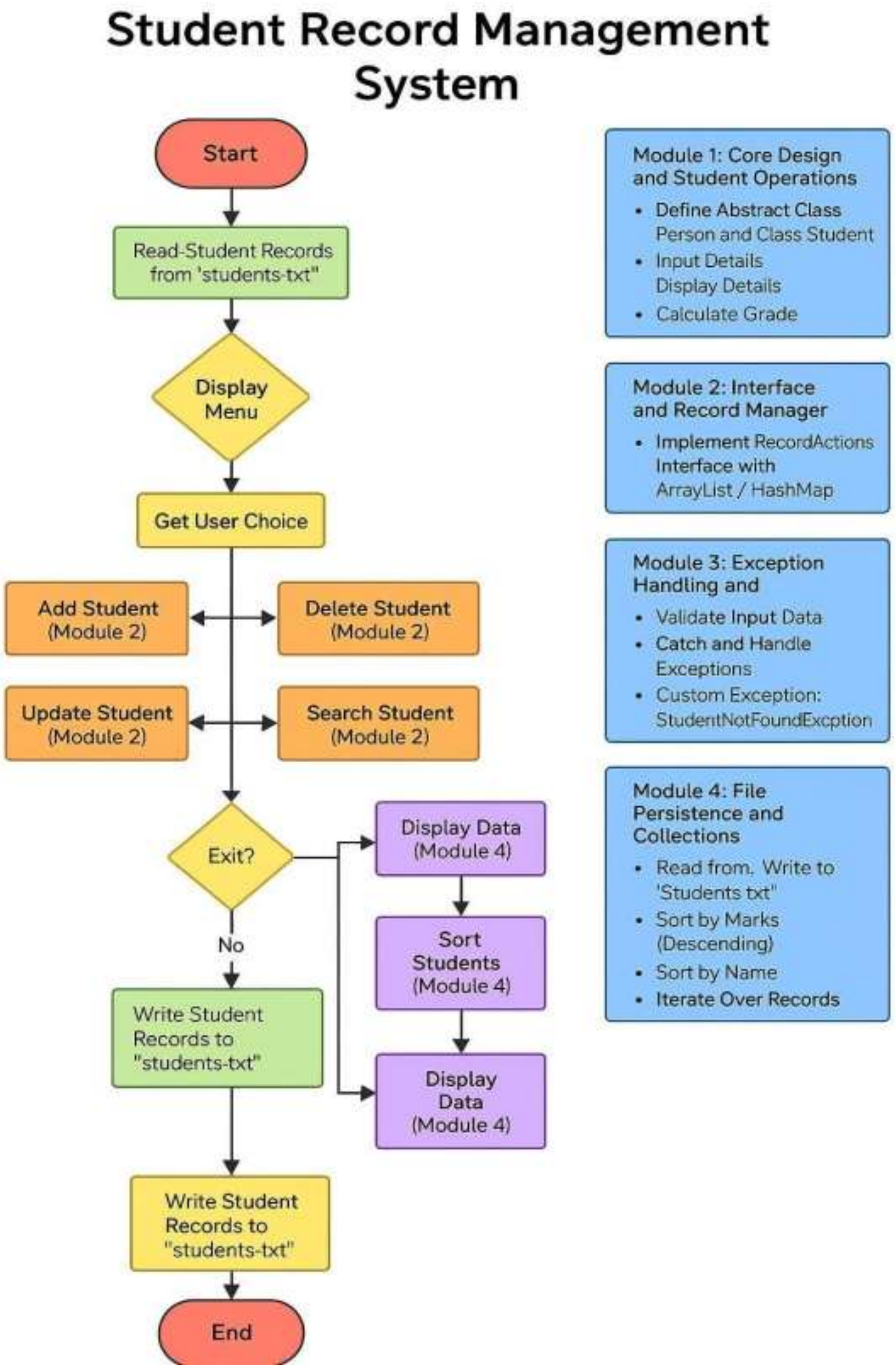
- Include a brief README explaining how to run the project and how it handles different operations.

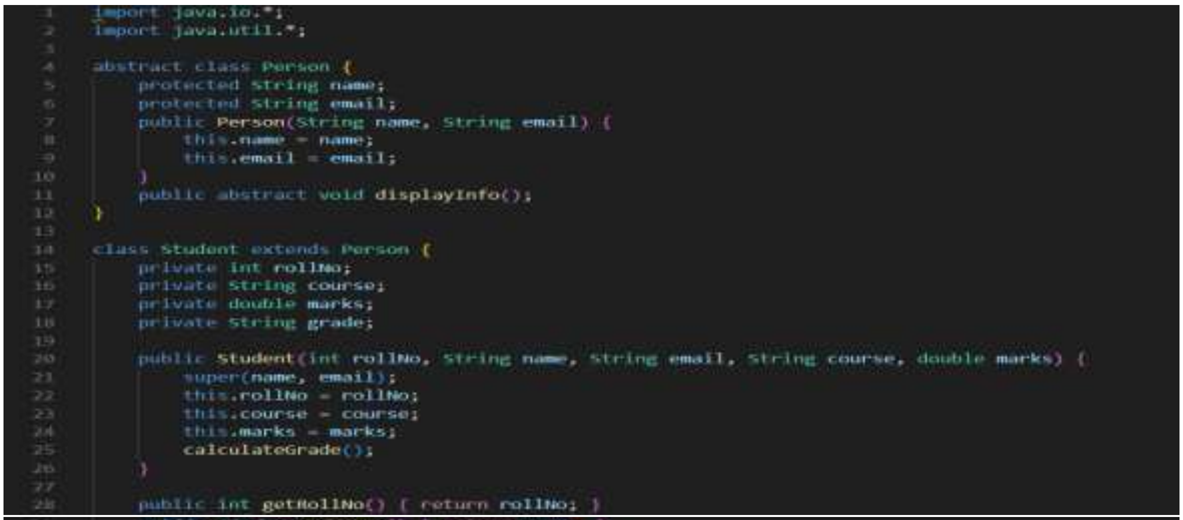
3. File Storage:
   - Make sure all student records are properly loaded from and saved to students.txt.

Performance Metrics (Out of 10 Marks)

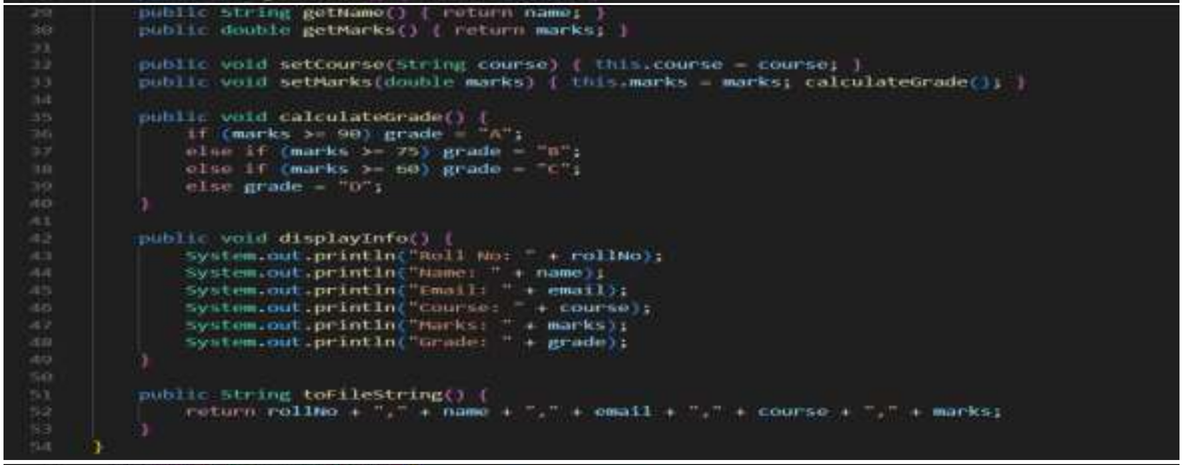
Criteria	Marks
Core Design and Implementation	3
Interface and Record Manager	2
Exception Handling and Validation	1.5
File Handling and Persistence	1.5
Sorting and Display	1
Multithreading and Responsiveness	1

Flow Chart:

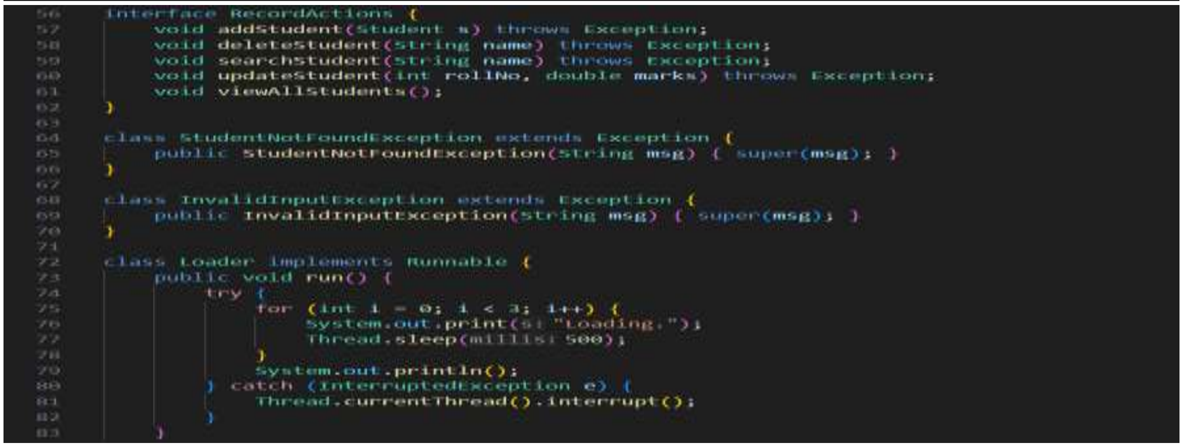

Student Record Management System
Start
Read-Student Records from 'students-txt"
Display Menu
Get User Choice
Add Student (Module 2)
Delete Student (Module 2)
Update Student (Module 2)
Search Student (Module 2)
Exit?
No
Write Student Records to "students-txt"
Write Student Records to "students-txt"
End
Display Data (Module 4)
Sort Students (Module 4)
Display Data (Module 4)
Module 1: Core Design and Student Operations
• Define Abstract Class Person and Class Student
• Input Details Display Details
• Calculate Grade
Module 2: Interface and Record Manager
• Implement RecordActions Interface with ArrayList / HashMap
Module 3: Exception Handling and
• Validate Input Data
• Catch and Handle Exceptions
• Custom Exception: StudentNotFoundExcption
Module 4: File Persistence and Collections
• Read from. Write to 'Students txt"
• Sort by Marks (Descending)
• Sort by Name
• Iterate Over Records



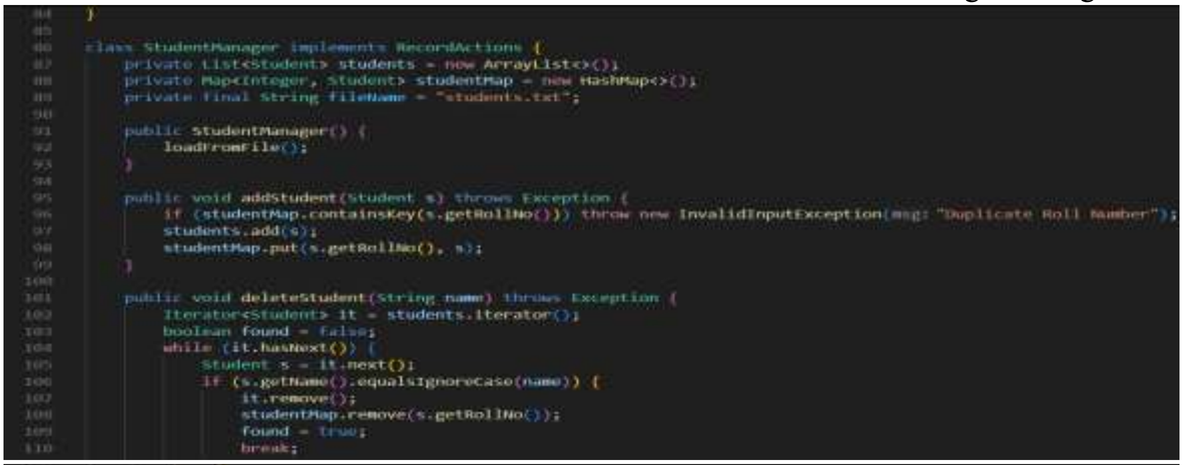
```java
import java.io.*;
import java.util.*;

abstract class Person {
    protected String name;
    protected String email;
    public Person(String name, String email) {
        this.name = name;
        this.email = email;
    }
    public abstract void displayInfo();
}

class Student extends Person {
    private int rollNo;
    private String course;
    private double marks;
    private String grade;

    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();
    }

    public int getRollNo() { return rollNo; }
```



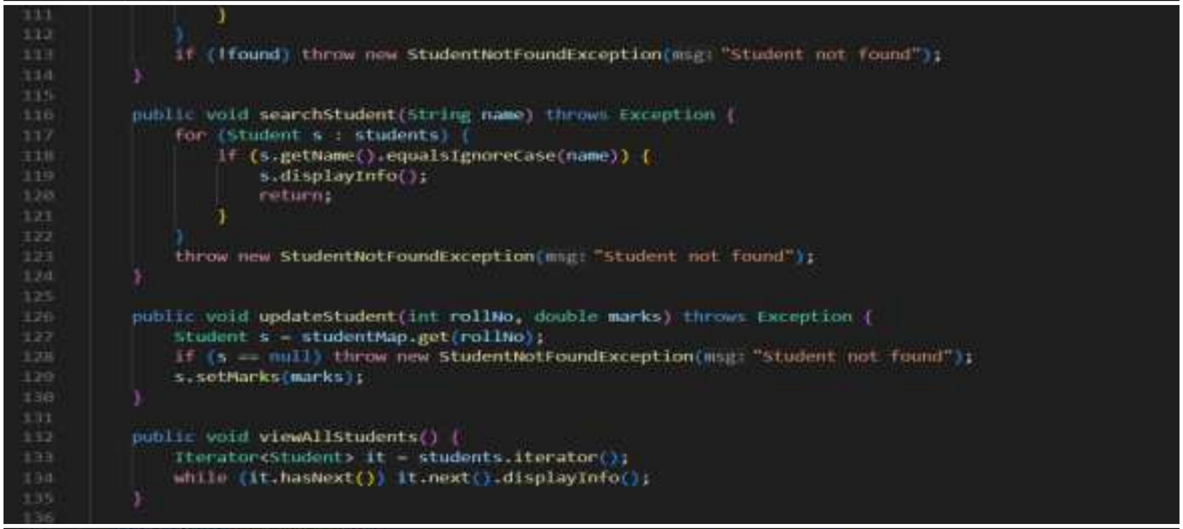
```java
    public String getName() { return name; }
    public double getMarks() { return marks; }

    public void setCourse(String course) { this.course = course; }
    public void setMarks(double marks) { this.marks = marks; calculateGrade(); }

    public void calculateGrade() {
        if (marks >= 90) grade = "A";
        else if (marks >= 75) grade = "B";
        else if (marks >= 60) grade = "C";
        else grade = "D";
    }

    public void displayInfo() {
        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);
    }

    public String toFileString() {
        return rollNo + "," + name + "," + email + "," + course + "," + marks;
    }
}
```



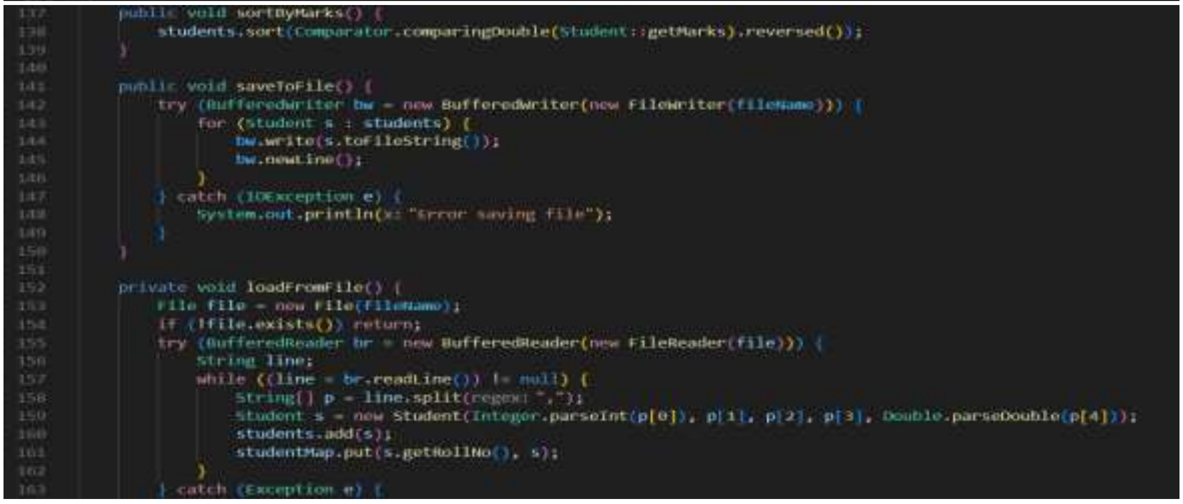
```java
interface RecordActions {
    void addStudent(Student s) throws Exception;
    void deleteStudent(String name) throws Exception;
    void searchStudent(String name) throws Exception;
    void updateStudent(int rollNo, double marks) throws Exception;
    void viewAllStudents();
}

class StudentNotFoundException extends Exception {
    public StudentNotFoundException(String msg) { super(msg); }
}

class InvalidInputException extends Exception {
    public InvalidInputException(String msg) { super(msg); }
}

class Loader implements Runnable {
    public void run() {
        try {
            for (int i = 0; i < 3; i++) {
                System.out.print(s: "Loading.");
                Thread.sleep(millis: 500);
            }
            System.out.println();
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
```



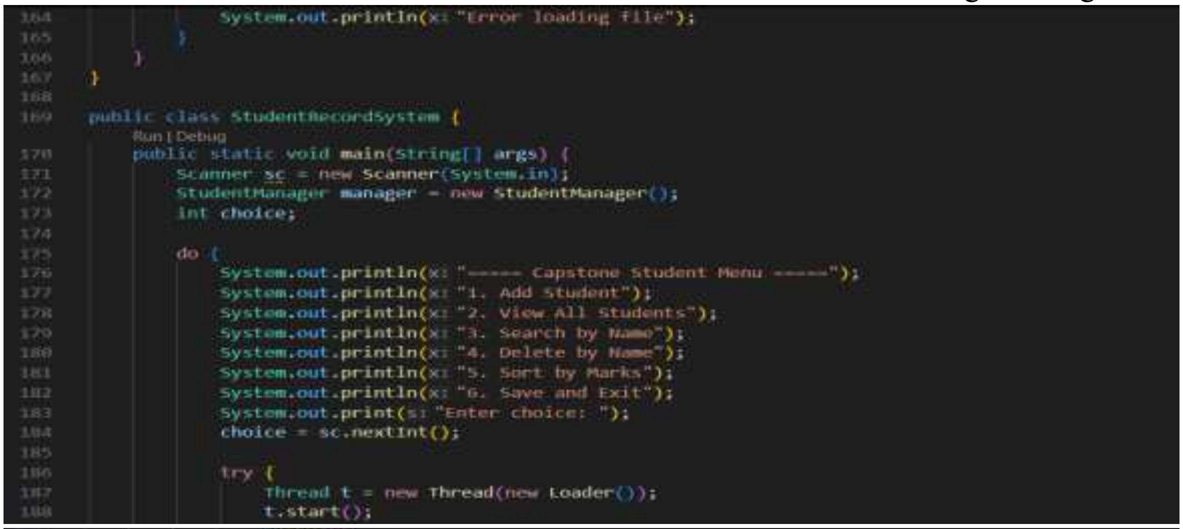
```java
}

class StudentManager implements RecordActions {
    private List<Student> students = new ArrayList<>();
    private Map<Integer, Student> studentMap = new HashMap<>();
    private final String fileName = "students.txt";

    public StudentManager() {
        loadFromFile();
    }

    public void addStudent(Student s) throws Exception {
        if (studentMap.containsKey(s.getRollNo())) throw new InvalidInputException(msg: "Duplicate Roll Number");
        students.add(s);
        studentMap.put(s.getRollNo(), s);
    }

    public void deleteStudent(String name) throws Exception {
        Iterator<Student> it = students.iterator();
        boolean found = false;
        while (it.hasNext()) {
            Student s = it.next();
            if (s.getName().equalsIgnoreCase(name)) {
                it.remove();
                studentMap.remove(s.getRollNo());
                found = true;
                break;
```



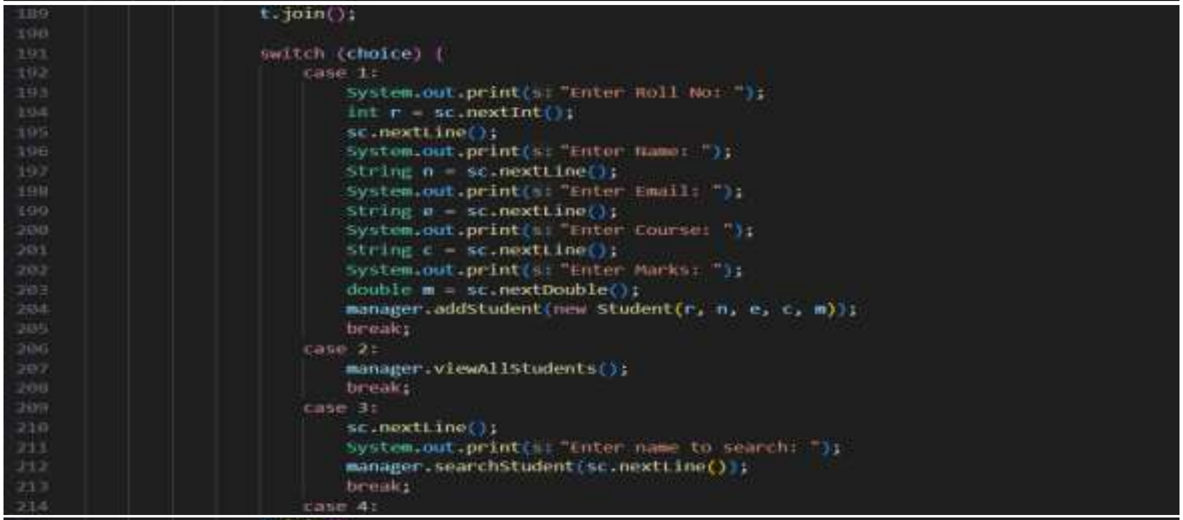
```java
            }
        }
        if (!found) throw new StudentNotFoundException(msg: "Student not found");
    }

    public void searchStudent(String name) throws Exception {
        for (Student s : students) {
            if (s.getName().equalsIgnoreCase(name)) {
                s.displayInfo();
                return;
            }
        }
        throw new StudentNotFoundException(msg: "Student not found");
    }

    public void updateStudent(int rollNo, double marks) throws Exception {
        Student s = studentMap.get(rollNo);
        if (s == null) throw new StudentNotFoundException(msg: "Student not found");
        s.setMarks(marks);
    }

    public void viewAllStudents() {
        Iterator<Student> it = students.iterator();
        while (it.hasNext()) it.next().displayInfo();
    }

```



```java
    public void sortByMarks() {
        students.sort(Comparator.comparingDouble(Student::getMarks).reversed());
    }

    public void saveToFile() {
        try (BufferedWriter bw = new BufferedWriter(new FileWriter(fileName))) {
            for (Student s : students) {
                bw.write(s.toFileString());
                bw.newLine();
            }
        } catch (IOException e) {
            System.out.println(x: "Error saving file");
        }
    }

    private void loadFromFile() {
        File file = new File(fileName);
        if (!file.exists()) return;
        try (BufferedReader br = new BufferedReader(new FileReader(file))) {
            String line;
            while ((line = br.readLine()) != null) {
                String[] p = line.split(regex: ",");
                Student s = new Student(Integer.parseInt(p[0]), p[1], p[2], p[3], Double.parseDouble(p[4]));
                students.add(s);
                studentMap.put(s.getRollNo(), s);
            }
        } catch (Exception e) {
```

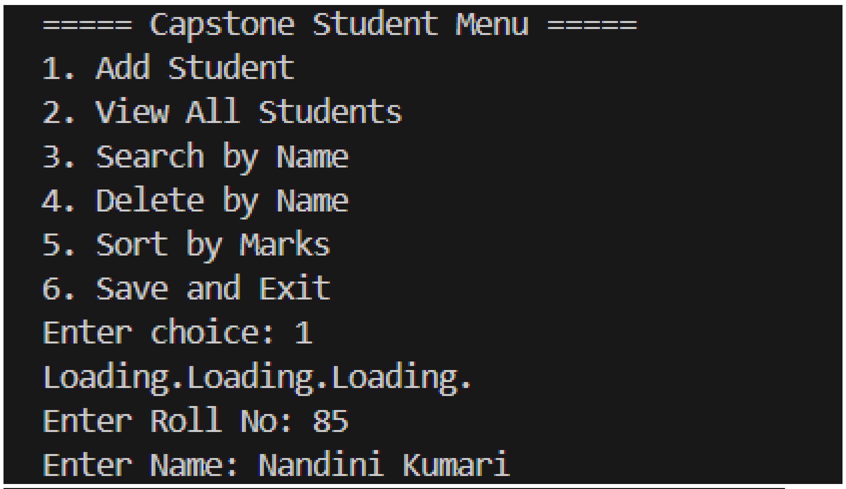


```java
            System.out.println(x: "Error loading file");
        }
    }
}

public class StudentRecordSystem {
    Run | Debug
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        StudentManager manager = new StudentManager();
        int choice;

        do {
            System.out.println(x: "----- Capstone Student Menu -----");
            System.out.println(x: "1. Add Student");
            System.out.println(x: "2. View All Students");
            System.out.println(x: "3. Search by Name");
            System.out.println(x: "4. Delete by Name");
            System.out.println(x: "5. Sort by Marks");
            System.out.println(x: "6. Save and Exit");
            System.out.print(s: "Enter choice: ");
            choice = sc.nextInt();

            try {
                Thread t = new Thread(new Loader());
                t.start();
```

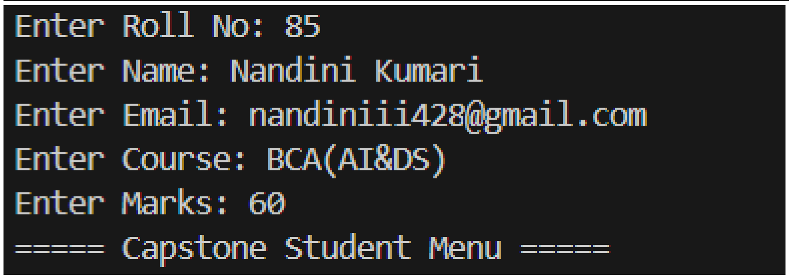


```java
                t.join();

                switch (choice) {
                    case 1:
                        System.out.print(s: "Enter Roll No: ");
                        int r = sc.nextInt();
                        sc.nextLine();
                        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();
                        manager.addStudent(new Student(r, n, e, c, m));
                        break;
                    case 2:
                        manager.viewAllStudents();
                        break;
                    case 3:
                        sc.nextLine();
                        System.out.print(s: "Enter name to search: ");
                        manager.searchStudent(sc.nextLine());
                        break;
                    case 4:
```

```java
                t.join();

                switch (choice) {
                    case 1:
                        System.out.print(s: "Enter Roll No: ");
                        int r = sc.nextInt();
                        sc.nextLine();
                        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();
                        manager.addStudent(new Student(r, n, e, c, m));
                        break;
                    case 2:
                        manager.viewAllStudents();
                        break;
                    case 3:
                        sc.nextLine();
                        System.out.print(s: "Enter name to search: ");
                        manager.searchStudent(sc.nextLine());
                        break;
                    case 4:
```



```
===== Capstone Student Menu =====
1. Add Student
2. View All Students
3. Search by Name
4. Delete by Name
5. Sort by Marks
6. Save and Exit
Enter choice: 1
Loading.Loading.Loading.
Enter Roll No: 85
Enter Name: Nandini Kumari
```



```
Enter Roll No: 85
Enter Name: Nandini Kumari
Enter Email: nandiniii428@gmail.com
Enter Course: BCA(AI&DS)
Enter Marks: 60
===== Capstone Student Menu =====
```

Loader.java — Runnable that prints a message with three animated dots. Used to show a short visual "Adding/Deleting/Updating/Saving..." animation on a separate thread. Sleeps 350ms between dots and catches (but ignores) InterruptedException. Cosmetic only; threads are started then joined so the main operation waits for the animation. Main.java — CLI entry point with a menu loop for add/view/search/delete/sort/save actions. Reads user input via Scanner, parses numbers, and delegates all data operations to StudentManager. Handles some exceptions by printing their messages and exits after saveToFile(). Sort option uses Collections.sort on the list returned by the manager (Student.compareTo sorts by marks desc).
Person.java — Abstract base class holding name and email with getters. Defines an abstract displayInfo() method that subclasses must implement. Fields are protected so subclasses (like Student) access them directly. Provides a simple shared API for personlike objects.
Student.java — Concrete Person subclass with rollNo, course, marks and derived grade. Calculates grade from marks, prints details in displayInfo(), and implements CSV serialization/parsing (toRecord/fromRecord). Implements Comparable<Student> to sort by marks in descending order (higher marks first). fromRecord uses simple split(',') so fields containing commas are not supported and there is minimal validation. StudentManager.java — Manages Student objects in

a HashMap keyed by roll number and provides CRUD + persistence. All public methods are synchronized for thread safety and spawn a Loader thread for a short animation, joining it before returning. saveToFile/loadFromFile write/read CSV-like records; HashMap iteration yields nondeterministic save order. Name-based operations search map.values(); missing entries throw StudentNotFoundException.

StudentNotFoundException.java — Small checked exception class used to signal missingstudent errors. Extends Exception and passes a message to the superclass constructor.
Thrown by StudentManager when search/update/delete operations can't find a student. Allows callers to distinguish "not found" situations from other errors.

Java Lab Assignment Submission Rubric Total: 100%

Criteria	Excellent (90-100%)	Good (7589%)	Fair (5074%)	Poor (049%)	Weightage (%)
Correctness & Output	Program produces correct output for all test cases; meets all requirements perfectly	Minor issues in output or misses some edge cases; meets most requirements	Output partially correct; meets some requirements; some test cases fail	Output incorrect or program fails to run	40
Code Implementation	Correct and efficient use of Java syntax, control structures, and data types	Mostly correct syntax and logic; some inefficiencies or minor errors	Several syntax or logic errors; inefficient code	Poorly implemented code; many syntax and logic errors	25
Use of Java Concepts	Proper and effective use of OOP concepts, exception handling, and libraries taught in lab	Uses Java concepts adequately but may miss some best practices	Basic use of Java concepts with limited understanding	Incorrect or no use of Java concepts covered in lab	15

Code Readability & Style	Wellformatted, indented code with meaningful names and comments explaining logic	Generally readable code; minor issues with style or commenting	Inconsistent formatting; few comments or unclear variable names	Poor formatting; no comments; hard to follow	10
Submission Guidelines	Submitted all required files on time with proper naming conventions	Submitted with minor issues in file naming or slight delay	Late submission or missing some required files	Missing files or not submitted	10

Total: 100%