



K.R. MANGALAM UNIVERSITY
THE COMPLETE WORLD OF EDUCATION

School of Engineering and Technology



Java Programming 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 synchronization.
CO6	Performing file I/O, work with Java Collections Framework and manipulate data using collections.

Contents	
INSTITUTION VISION & MISSION	i
SCHOOL VISION & MISSION	ii
COURSE OUTCOMES	iii
ASSIGNMENT SCHEDULE	vi,vii
Assignment 1 - Student Class Design & Basic Operations	8

Java Programming Lab

Create a Student Record Management system that allows the user to input, display, and calculate grades for students. Implement a classbased 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.	8
Assignment 2 - Inheritance, Interfaces, and Modular Design	12
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.	12
Assignment 3 - Exception Handling, Multithreading, and Wrapper Classes	16
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	16
Assignment 4 - File Handling and Collections	20
Implement a Student Record Management System with persistent	20

<p>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</p>	
<p>Assignment 5 - Capstone Project – Student Record Management System</p> <p>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.</p>	24

31

Assignment Schedule

Sr. No.	Java Lab Assignments	COs	Release After	Duration
1	<p>ASSIGNMENT 1: Student Class Design & Basic Operations</p> <p>Problem Statement:</p> <p>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.</p>	CO1, CO6	Session 5	2 weeks

Java Programming Lab

2	<p>ASSIGNMENT 2: Inheritance, Interfaces, and Modular Design</p> <p>Problem Statement:</p> <p>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.</p>	CO1, CO2	Session 20	2 weeks
3	<p>ASSIGNMENT 3: Exception Handling, Multithreading, and Wrapper Classes</p> <p>Problem Statement:</p> <p>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</p>	CO3, CO4	Session 35	2 weeks

Java Programming Lab

	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	<p>ASSIGNMENT 4: File Handling and Collections</p> <p>Problem Statement:</p> <p>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.</p>	CO6	Session 45	2 weeks
5	<p>ASSIGNMENT 5: Capstone Project – Student Record Management System</p> <p>Problem Statement:</p> <p>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.</p>	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

Page viii of

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.

Java Programming Lab

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).
-

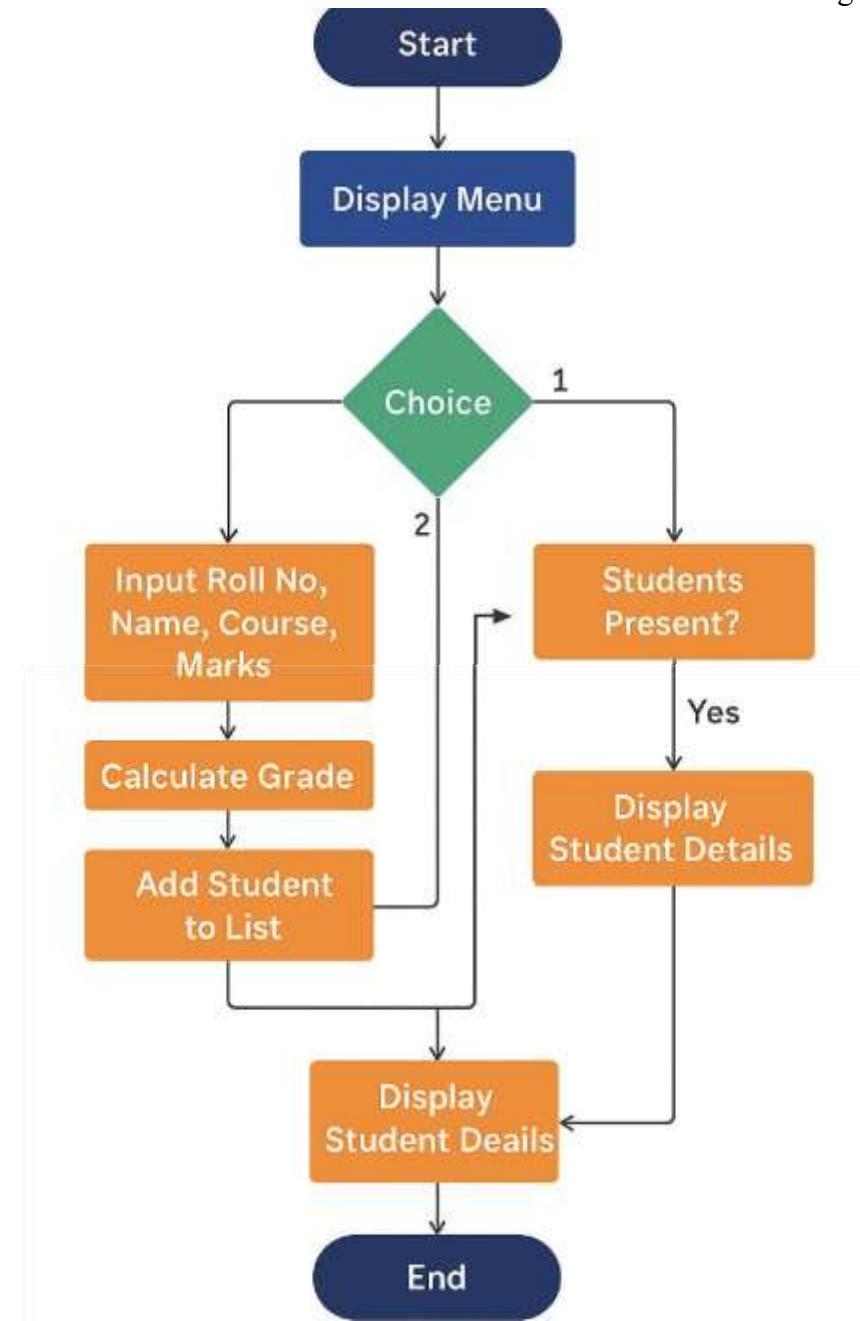
Submission Guidelines

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:



```
1 import java.util.Scanner;
2
3 class Person {
4     String name;
5
6     public Person() {}
7
8     public Person(String name) {
9         this.name = name;
10    }
11 }
12
13 class Student extends Person {
14     int rollNo;
15     String course;
16     double marks;
17     char grade;
18
19     public Student() {}
20
21
22     public Student(int rollNo, String name, String course, double marks) {
23         super(name);
24         this.rollNo = rollNo;
25         this.course = course;
26         this.marks = marks;
27         calculateGrade();
28     }
29 }
```

```
29     }
30
31
32     public void inputDetails(Scanner sc) {
33         System.out.print("Enter Roll No: ");
34         rollNo = sc.nextInt();
35         sc.nextLine();
36
37         System.out.print("Enter Name: ");
38         name = sc.nextLine();
39
40         System.out.print("Enter Course: ");
41         course = sc.nextLine();
42
43         System.out.print("Enter Marks: ");
44         marks = sc.nextDouble();
45
46         calculateGrade();
47     }
48
49
50     public void calculateGrade() {
51         if (marks >= 90) grade = 'A';
52         else if (marks >= 75) grade = 'B';
53         else if (marks >= 60) grade = 'C';
54         else grade = 'D';
55     }
56
57
58     public void displayDetails() {
59         System.out.println("Roll No: " + rollNo);
60         System.out.println("Name: " + name);
61         System.out.println("Course: " + course);
62         System.out.println("Marks: " + marks);
63         System.out.println("Grade: " + grade);
64         System.out.println();
65     }
66 }
67
68 public class StudentRecordsSystem {
69
70     Run|Debug
71     public static void main(String[] args) {
72
73         Scanner sc = new Scanner(System.in);
74         Student[] students = new Student[100];
75         int count = 0;
76
77         while (true) {
78             System.out.println("----- Student Record Menu -----");
79             System.out.println("1. Add Student");
80             System.out.println("2. Display All Students");
```

```

80     system.out.println(x: "3. Exit");
81     system.out.print(s: "Enter your choice: ");
82     int choice = sc.nextInt();
83     sc.nextLine();
84
85     switch (choice) {
86
87         case 1:
88             if (count < 100) {
89                 Student s = new Student();
90                 s.inputDetails(sc);
91                 students[count] = s;
92                 count++;
93             } else {
94                 system.out.println(x: "Student limit reached!");
95             }
96             break;
97
98         case 2:
99             if (count == 0) {
100                 system.out.println(x: "No student records available.");
101             } else {
102                 for (int i = 0; i < count; i++) {
103                     students[i].displayDetails();
104                 }
105             }
106             break;
107
108         case 3:
109             System.out.println(x: "Exiting the application. Goodbye!");
110             sc.close();
111             return;
112
113         default:
114             System.out.println(x: "Invalid choice! Please try again.");
115     }
116 }
117 }
118 }
119

```

```

===== 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)
 - o 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.
-

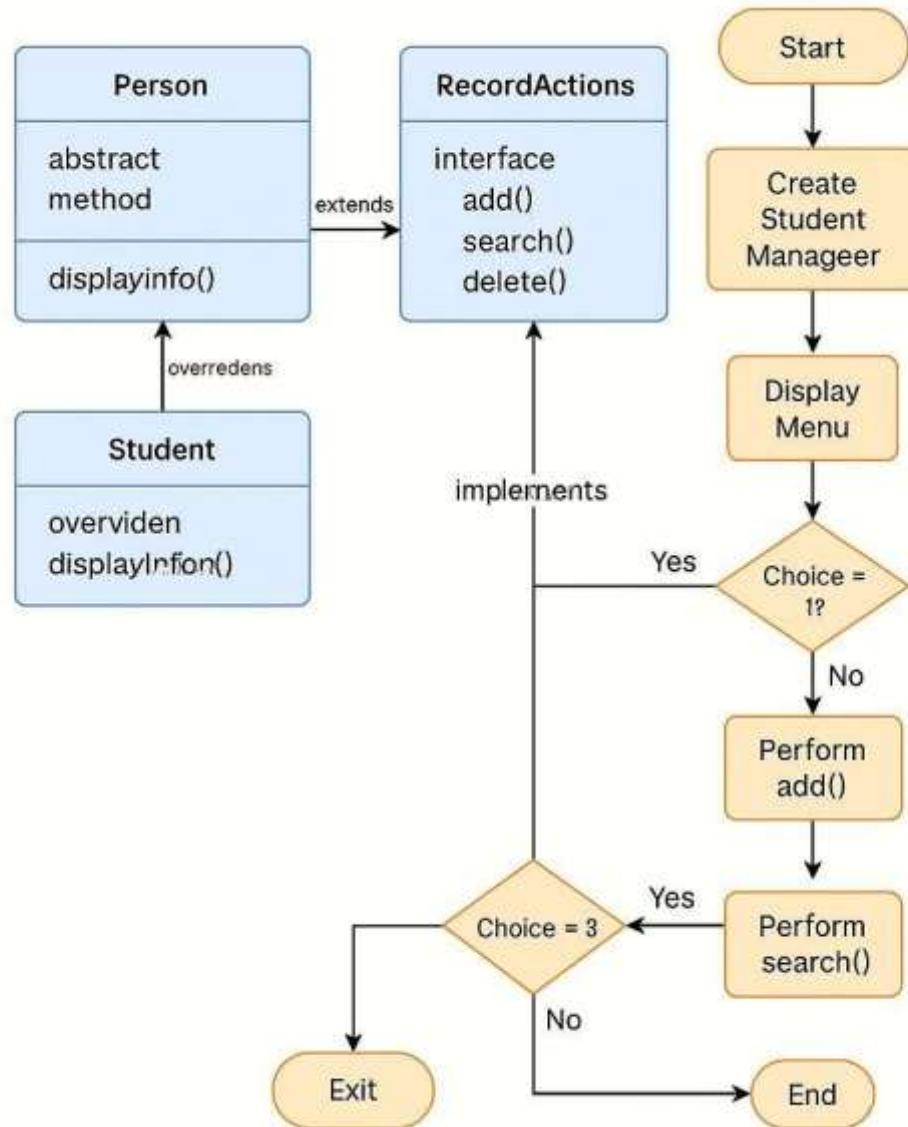
Submission Guidelines

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:



```
1 import java.io.*;
2 import java.nio.file.*;
3 import java.text.SimpleDateFormat;
4 import java.util.*;
5
6
7 public class StudentRecordApp {
8
9
10     static class Student {
11         private int rollNo;
12         private String name;
13         private String email;
14         private String course;
15         private double marks;
16
17         public Student(int rollNo, String name, String email, String course, double marks) {
18             this.rollNo = rollNo;
19             this.name = name;
20             this.email = email;
21             this.course = course;
22             this.marks = marks;
23         }
24
25         public int getRollNo() { return rollNo; }
26         public String getName() { return name; }
27         public String getEmail() { return email; }
28         public String getCourse() { return course; }
29
30         public double getMarks() { return marks; }
31
32         public String toFileLine() {
33
34             return rollNo + "|" + escape(name) + "|" + escape(email) + "|" + escape(course) + "|" + marks;
35         }
36
37     }
38
39     public static Student fromFileLine(String line) {
40         String[] parts = line.split(regex:"\\|", -1);
41         if (parts.length < 5) return null;
42         try {
43             int roll = Integer.parseInt(parts[0]);
44             String name = unescape(parts[1]);
45             String email = unescape(parts[2]);
46             String course = unescape(parts[3]);
47             double marks = Double.parseDouble(parts[4]);
48             return new Student(roll, name, email, course, marks);
49         } catch (NumberFormatException e) {
50             return null;
51         }
52     }
53
54     private static String escape(String s) { return s == null ? "" : s.replace(target: "!", replacement: "\\\\""); }
55     private static String unescape(String s) { return s == null ? "" : s.replace(target: "\\\\"", replacement: "!"); }
```

```

29     public double getMarks() { return marks; }
30
31     public String toFileLine() {
32
33         return rollNo + "|" + escape(name) + "|" + escape(email) + "|" + escape(course) + "|" + marks;
34     }
35
36     public static Student fromFileLine(String line) {
37         String[] parts = line.split(regex("\\|", -1));
38         if (parts.length < 5) return null;
39         try {
40             int roll = Integer.parseInt(parts[0]);
41             String name = unescape(parts[1]);
42             String email = unescape(parts[2]);
43             String course = unescape(parts[3]);
44             double marks = Double.parseDouble(parts[4]);
45             return new Student(roll, name, email, course, marks);
46         } catch (NumberFormatException e) {
47             return null;
48         }
49     }
50
51     private static String escape(String s) { return s == null ? "" : s.replace(target: "\\", replacement: "\\\""); }
52     private static String unescape(String s) { return s == null ? "" : s.replace(target: "\\\"", replacement: "\""); }
53
54
55     public void saveAll(List<Student> students) throws IOException {
56         ensureFile();
57         try (BufferedWriter bw = Files.newBufferedWriter(dataFile)) {
58             for (Student s : students) {
59                 bw.write(s.toFileLine());
60                 bw.newLine();
61             }
62         }
63     }
64
65
66     public void printFileAttributes() {
67         File f = dataFile.toFile();
68         System.out.println("File: " + dataFile.getAbsolutePath());
69         System.out.println("Exists: " + f.exists());
70         System.out.println("Readable: " + f.canRead());
71         System.out.println("Writable: " + f.canWrite());
72         System.out.println("Size (bytes): " + (f.exists() ? f.length() : 0));
73         SimpleDateFormat sdf = new SimpleDateFormat(pattern: "yyyy-MM-dd HH:mm:ss");
74         System.out.println("Last Modified: " + (f.exists() ? sdf.format(f.lastModified()) : "N/A"));
75         System.out.println();
76     }
77
78
79     public Student readRecordAtIndex(int index) throws IOException {
80         if (index < 0 || index >= recordOffsets.size()) return null;
81     }

```

```
153     List<Student> loaded = fileUtil.loadAll();
154     students.clear();
155     studentMap.clear();
156     for (Student s : loaded) {
157         students.add(s);
158         studentMap.put(s.getRollNo(), s);
159     }
160     System.out.println("Loaded students from file:");
161     displayAll();
162 } catch (IOException e) {
163     System.out.println("Error loading students: " + e.getMessage());
164 }
165
166
167 public void save() {
168     try {
169         fileUtil.saveAll(students);
170         System.out.println("Saved " + students.size() + " students to file.");
171     } catch (IOException e) {
172         System.out.println("Error saving students: " + e.getMessage());
173     }
174 }
175
176 public boolean addStudent(Student s) {
177     if (studentMap.containsKey(s.getRollNo())) return false;
178     sc.close();
179 }
180
181
182
183
184
185 private static int readInt(Scanner sc, String prompt) {
186     while (true) {
187         System.out.print(prompt);
188         String s = sc.nextLine().trim();
189         try {
190             return Integer.parseInt(s);
191         } catch (NumberFormatException e) {
192             System.out.println("Please enter a valid integer.");
193         }
194     }
195 }
196
197
198
199
200
201 private static double readDouble(Scanner sc, String prompt) {
202     while (true) {
203         System.out.print(prompt);
204         String s = sc.nextLine().trim();
205         try {
206             return Double.parseDouble(s);
207         } catch (NumberFormatException e) {
208             System.out.println("Please enter a valid number (e.g., 85.5).");
209         }
210     }
211 }
```

```
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 object-oriented 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

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.
-

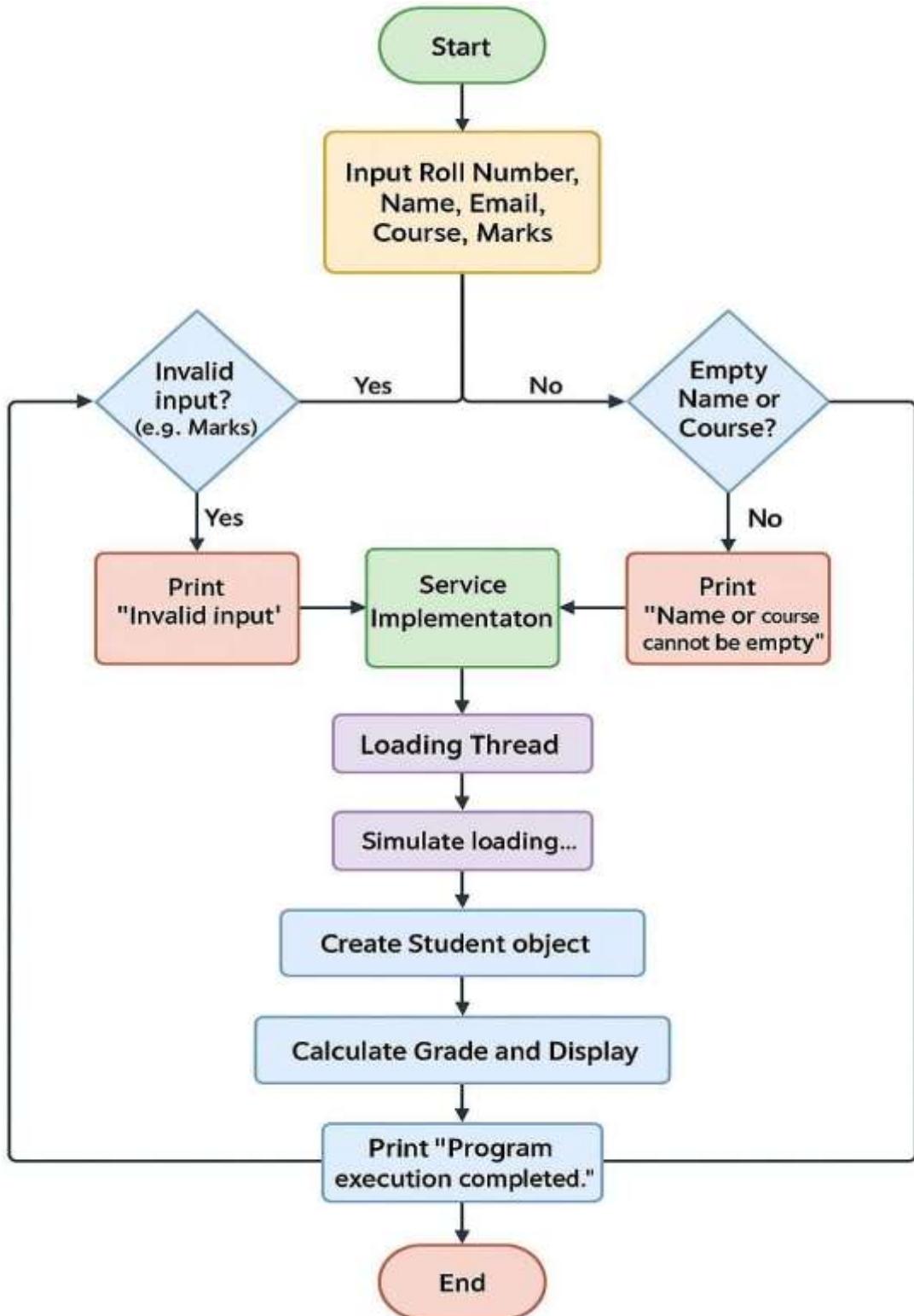
Submission Guidelines

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:



```
1 import java.util.*;
2
3
4 class StudentNotFoundException extends Exception {
5     public StudentNotFoundException(String message) {
6         super(message);
7     }
8 }
9
10
11 class Loader implements Runnable {
12     @Override
13     public void run() {
14         try {
15             System.out.print("Loading");
16             for (int i = 0; i < 5; i++) {
17                 Thread.sleep(millis:300);
18                 System.out.print(".");
19             }
20             System.out.println();
21         } catch (InterruptedException e) {
22             System.out.println(x:"Loading interrupted!");
23         }
24     }
25 }
26
27
28 interface RecordActions {
29     void addStudent(Student s);
30     Student getStudent(Integer roll) throws StudentNotFoundException;
31 }
32
33
34 class Student {
35     private Integer roll;
36     private String name;
37     private String email;
38     private String course;
39     private Double marks;
40
41     public Student(Integer roll, String name, String email, String course, Double marks) {
42         this.roll = roll;
43         this.name = name;
44         this.email = email;
45         this.course = course;
46         this.marks = marks;
47     }
48
49
50     public Integer getRoll() {
51         return roll;
52     }
53
54     public Double calculateGrade() {
55         if (marks >= 90) return 4.0;
```

Java Programming Lab

```
56     else if (marks >= 80) return 3.5;
57     else if (marks >= 70) return 3.0;
58     else if (marks >= 60) return 2.5;
59     else return 2.0;
60   }
61
62   public String getLetterGrade() {
63     if (marks >= 90) return "A";
64     else if (marks >= 80) return "B+";
65     else if (marks >= 70) return "B";
66     else if (marks >= 60) return "C";
67     else return "D";
68   }
69
70   public void display() {
71     System.out.println("\nRoll No: " + roll);
72     System.out.println("Name: " + name);
73     System.out.println("Email: " + email);
74     System.out.println("Course: " + course);
75     System.out.println("Marks: " + marks);
76     System.out.println("Grade: " + getLetterGrade());
77   }
78 }
79
80
81 class StudentManager implements RecordActions {
82   private Map<Integer, Student> records = new HashMap<>();
83
84   @Override
85   public void addStudent(Student s) {
86     records.put(s.getRoll(), s);
87   }
88
89
90   @Override
91   public Student getStudent(Integer roll) throws StudentNotFoundException {
92     if (!records.containsKey(roll)) {
93       throw new StudentNotFoundException("Student with roll " + roll + " not found!");
94     }
95     return records.get(roll);
96   }
97 }
98
99
100 public class Main {
101   Run|Debug
102   public static void main(String[] args) {
103     Scanner sc = new Scanner(System.in);
104     StudentManager manager = new StudentManager();
105
106     try {
107       System.out.print("Enter Roll No (Integer): ");
108       Integer roll = Integer.valueOf(sc.nextInt());
109     }
110   }
111 }
```

```
199     sc.nextLine();
200
201     System.out.print(s:"Enter Name: ");
202     String name = sc.nextLine();
203     if (name.trim().isEmpty())
204         throw new Exception(message:"Name cannot be empty!");
205
206     System.out.print(s:"Enter Email: ");
207     String email = sc.nextLine();
208     if (email.trim().isEmpty())
209         throw new Exception(message:"Email cannot be empty!");
210
211     System.out.print(s:"Enter Course: ");
212     String course = sc.nextLine();
213     if (course.trim().isEmpty())
214         throw new Exception(message:"Course cannot be empty!");
215
216     System.out.print(s:"Enter Marks: ");
217     Double marks = Double.valueOf(sc.nextDouble());
218     if (marks < 0 || marks > 100)
219         throw new Exception(message:"Marks must be between 0 and 100!");
220
221
222     Thread t = new Thread(new Loader());
223     t.start();
224     t.join();
225
226
227     Student s = new Student(roll, name, email, course, marks);
228     manager.addStudent(s);
229
230     |
231     Student stored = manager.getStudent(roll);
232     stored.display();
233
234 } catch (StudentNotFoundException e) {
235     System.out.println("Error: " + e.getMessage());
236 } catch (InputMismatchException e) {
237     System.out.println(x:"Error: Invalid input type!");
238 } catch (Exception e) {
239     System.out.println("Validation Error: " + e.getMessage());
240 } finally {
241     sc.close();
242     System.out.println(x:"\nProgram Finished.");
243 }
```

```
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.
-

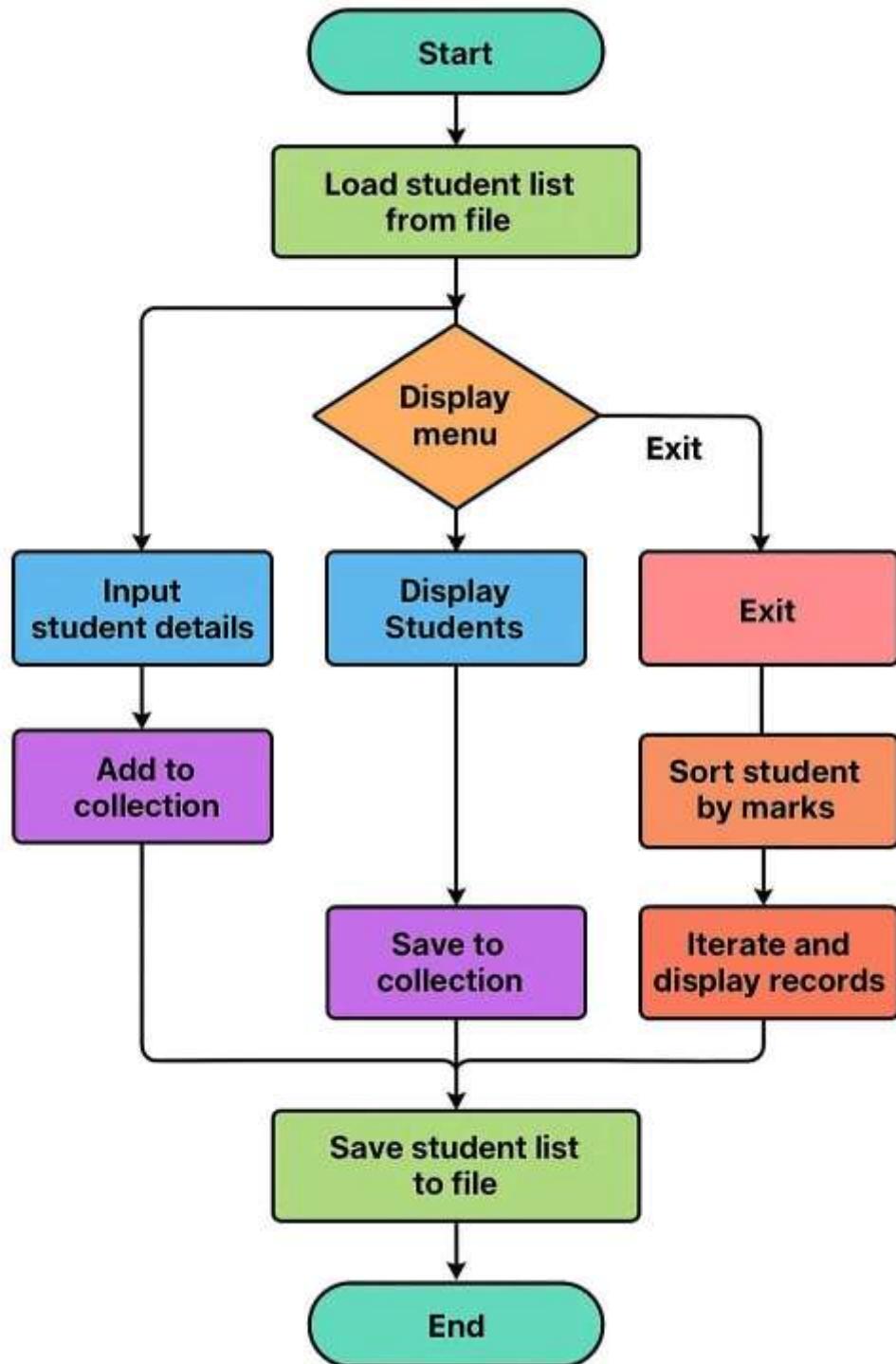
Submission Guidelines

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:



Java Programming Lab

```
1 import java.io.*;
2 import java.nio.file.*;
3 import java.text.SimpleDateFormat;
4 import java.util.*;
5
6 public class StudentRecordApp {
7
8
9     static class Student {
10         private int rollNo;
11         private String name;
12         private String email;
13         private String course;
14         private double marks;
15
16         public Student(int rollNo, String name, String email, String course, double marks) {
17             this.rollNo = rollNo;
18             this.name = name;
19             this.email = email;
20             this.course = course;
21             this.marks = marks;
22         }
23
24         public int getRollNo() { return rollNo; }
25         public String getName() { return name; }
26         public String getEmail() { return email; }
27         public String getCourse() { return course; }
28
29         public double getMarks() { return marks; }
30
31         public String toFileLine() {
32
33             return rollNo + "|" + escape(name) + "|" + escape(email) + "|" + escape(course) + "|" + marks;
34         }
35
36         public static Student fromFileLine(String line) {
37             String[] parts = line.split(regex: "\\\\", -1);
38             if (parts.length < 5) return null;
39             try {
40                 int roll = Integer.parseInt(parts[0]);
41                 String name = unescape(parts[1]);
42                 String email = unescape(parts[2]);
43                 String course = unescape(parts[3]);
44                 double marks = Double.parseDouble(parts[4]);
45                 return new Student(roll, name, email, course, marks);
46             } catch (NumberFormatException e) {
47                 return null;
48             }
49         }
50
51         private static String escape(String s) { return s == null ? "" : s.replace(target: "|", replacement: "\\\\"");}
52         private static String unescape(String s) { return s == null ? "" : s.replace(target: "\\\\"", replacement: "|");}
53
54         private static String escape(String s) { return s == null ? "" : s.replace(target: "\\", replacement: "\\\\"");}
55         private static String unescape(String s) { return s == null ? "" : s.replace(target: "\\\\", replacement: "\\");}
56
57         public void display() {
58             System.out.println("Roll No: " + rollNo);
59             System.out.println("Name : " + name);
60             System.out.println("Email : " + email);
61             System.out.println("Course : " + course);
62             System.out.println("Marks : " + marks);
63             System.out.println();
64         }
65
66     static class FileUtil {
67         private final Path datafile;
68         private final List<Long> recordOffsets = new ArrayList<>();
69         public FileUtil(String filename) {
70             this.datafile = Paths.get(filename);
71         }
72
73         public void ensureFile() throws IOException {
74             if (!Files.notExists(datafile)) {
75                 Files.createFile(datafile);
76             }
77         }
78     }
79 }
```

```

77
78
79
80    public List<Student> loadAll() throws IOException {
81        ensureFile();
82        List<Student> list = new ArrayList<>();
83        recordOffsets.clear();
84
85
86        try (RandomAccessFile raf = new RandomAccessFile(dataFile.toFile(), "r")) {
87            long pos;
88            while ((pos = raf.getFilePointer()) < raf.length()) {
89                recordOffsets.add(pos);
90                String line = raf.readLine();
91                if (line == null) break;
92
93                Student s = Student.fromFileLine(line);
94                if (s != null) list.add(s);
95            }
96        }
97
98        return list;
99    }
100
101
102    public void saveAll(List<Student> students) throws IOException {
103        ensureFile();
104        try (BufferedWriter bw = Files.newBufferedWriter(dataFile)) {
105            for (Student s : students) {
106                bw.write(s.toFileLine());
107                bw.newLine();
108            }
109        }
110
111
112    public void printFileAttributes() {
113        File f = dataFile.toFile();
114        System.out.println("file: " + dataFile.toAbsolutePath());
115        System.out.println("exists: " + f.exists());
116        System.out.println("Readable: " + f.canRead());
117        System.out.println("Writable: " + f.canWrite());
118        System.out.println("size (bytes): " + (f.exists() ? f.length() : 0));
119        SimpleDateFormat sdf = new SimpleDateFormat(pattern: "yyyy-MM-dd HH:mm:ss");
120        System.out.println("Last Modified: " + (f.exists() ? sdf.format(f.lastModified()) : "N/A"));
121        System.out.println();
122    }
123
124
125
126    public Student readRecordAtIndex(int index) throws IOException {
127        if (index < 0 || index >= recordOffsets.size()) return null;
128        List<Student> loaded = fileUtil.loadAll();
129        students.clear();
130        studentMap.clear();
131        for (Student s : loaded) {
132            students.add(s);
133            studentMap.put(s.getRollNo(), s);
134        }
135        System.out.println(x: "Loaded students from file:");
136        displayAll();
137    } catch (IOException e) {
138        System.out.println("Error loading students: " + e.getMessage());
139    }
140}
141
142
143    public void save() {
144        try {
145            fileUtil.saveAll(students);
146            System.out.println("Saved " + students.size() + " students to file.");
147        } catch (IOException e) {
148            System.out.println("Error saving students: " + e.getMessage());
149        }
150    }
151
152
153    public boolean addStudent(Student s) {
154        if (studentMap.containsKey(s.getRollNo())) return false;
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177

```

Java Programming Lab

```

179     public boolean addStudent(Student s) {
180         if (studentMap.containsKey(s.getRollNo())) return false;
181         students.add(s);
182         studentMap.put(s.getRollNo(), s);
183         return true;
184     }
185
186     public Student searchByName(String name) {
187         for (Student s : students) {
188             if (s.getName().equalsIgnoreCase(name)) return s;
189         }
190         return null;
191     }
192
193     public boolean deleteByName(String name) {
194         Iterator<Student> it = students.iterator();
195         boolean removed = false;
196         while (it.hasNext()) {
197             Student s = it.next();
198             if (s.getName().equalsIgnoreCase(name)) {
199                 it.remove();
200                 studentMap.remove(s.getRollNo());
201                 removed = true;
202             }
203         }
204         return removed;
205     }
206
207     System.out.print("Enter choice: ");
208     String line = sc.nextLine().trim();
209
210     try {
211         int choice = Integer.parseInt(line);
212         switch (choice) {
213             case 1 -> {
214                 int roll = readInt(sc, prompt: "Enter Roll No: ");
215                 String name = readNonEmpty(sc, prompt: "Enter Name: ");
216                 String email = readNonEmpty(sc, prompt: "Enter Email: ");
217                 String course = readNonEmpty(sc, prompt: "Enter Course: ");
218                 double marks = readDouble(sc, prompt: "Enter Marks: ");
219                 Student s = new Student(roll, name, email, course, marks);
220                 boolean ok = manager.addStudent(s);
221                 if (ok) System.out.println(s + " student added.");
222                 else System.out.println(s + " Duplicate roll number, student not added.");
223             }
224             case 2 -> manager.displayAll();
225             case 3 -> {
226                 String name = readNonEmpty(sc, prompt: "Enter Name to search: ");
227                 Student s = manager.searchByName(name);
228                 if (s != null) s.display();
229                 else System.out.println(s + " student not found.");
230             }
231             case 4 -> {
232                 String name = readNonEmpty(sc, prompt: "Enter Name to delete: ");
233
234                 boolean removed = manager.deleteByName(name);
235                 if (removed) System.out.println(s + " student(s) deleted.");
236                 else System.out.println(s + " No student with that name found.");
237             }
238             case 5 -> {
239                 manager.sortByMarksDescending();
240                 System.out.println(s + " sorted student list by Marks");
241                 manager.displayAll();
242             }
243             case 6 -> {
244                 manager.sortByName();
245                 System.out.println(s + " sorted student list by Name");
246                 manager.displayAll();
247             }
248             case 7 -> {
249                 manager.getFileUtil().printFileAttributes();
250             }
251             case 8 -> {
252                 int count = manager.getRecordCount();
253                 if (count == 0) {
254                     System.out.println(s + " no records for random access.");
255                 } else {
256                     Random rnd = new Random();
257                     int idx = rnd.nextInt(count);
258                     System.out.println("Reading random record index: " + idx);
259                 }
260             }
261         }
262     }
263
264     private static String readNonEmpty(Scanner sc, String prompt) {
265         while (true) {
266             System.out.print(prompt);
267             String s = sc.nextLine().trim();
268             if (!s.isEmpty()) return s;
269             System.out.println(s + " Input cannot be empty.");
270         }
271     }
272
273 }

```

```
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. FileUtils 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

- marks

It also includes:

- #### **It also includes:**

- 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:

Java Programming Lab

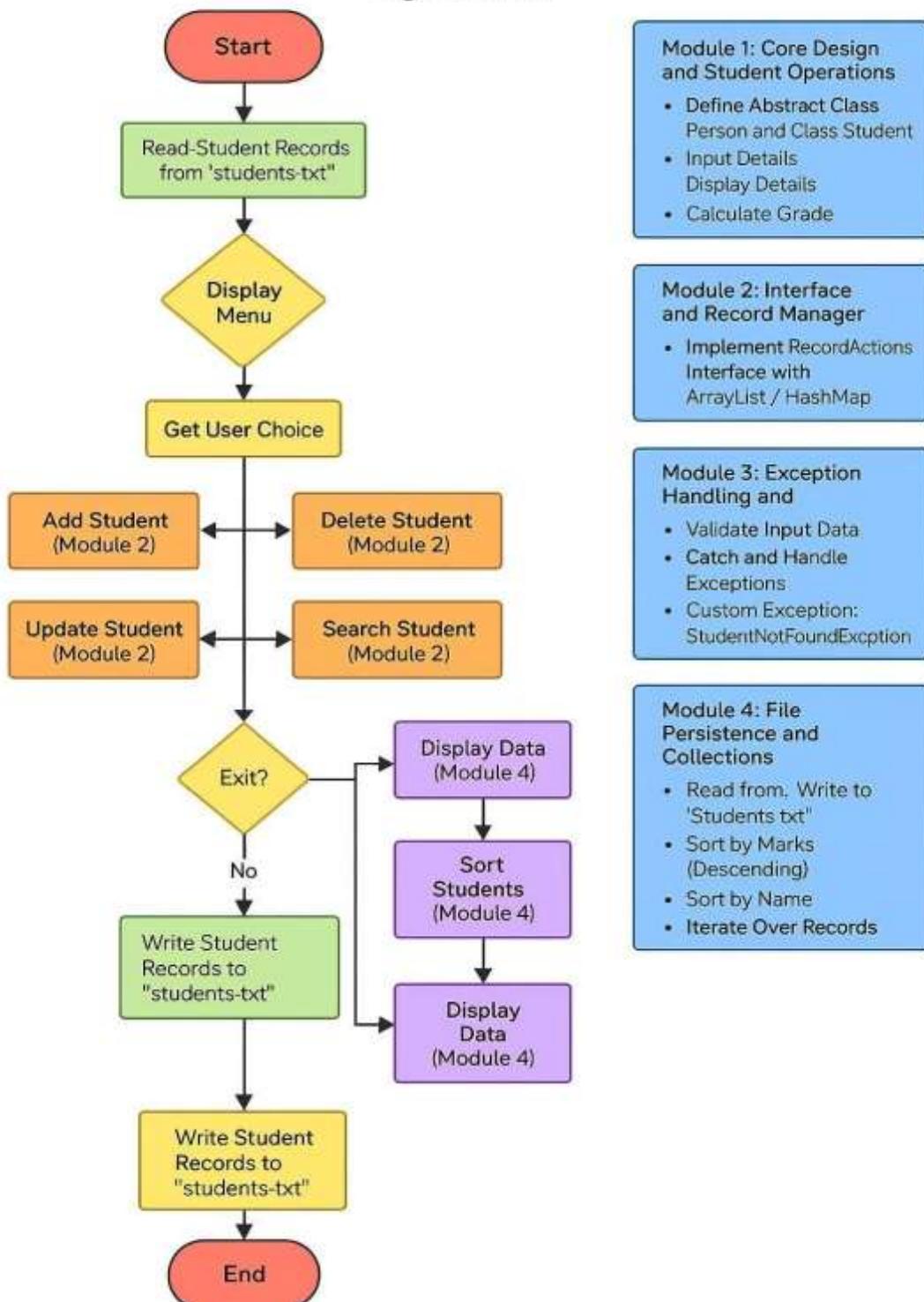
- o Include a brief README explaining how to run the project and how it handles different operations.
3. File Storage:
- o Make sure all student records are properly loaded from and saved to students.txt.
-

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



```
1 import java.io.*;
2 import java.util.*;
3
4 abstract class Person {
5     protected String name;
6     protected String email;
7     public Person(String name, String email) {
8         this.name = name;
9         this.email = email;
10    }
11    public abstract void displayInfo();
12 }
13
14 class Student extends Person {
15     private int rollNo;
16     private String course;
17     private double marks;
18     private String grade;
19
20     public Student(int rollNo, String name, String email, String course, double marks) {
21         super(name, email);
22         this.rollNo = rollNo;
23         this.course = course;
24         this.marks = marks;
25         calculateGrade();
26     }
27
28     public int getRollNo() { return rollNo; }
29     public String getName() { return name; }
30     public double getMarks() { return marks; }
31
32     public void setCourse(String course) { this.course = course; }
33     public void setMarks(double marks) { this.marks = marks; calculateGrade(); }
34
35     public void calculateGrade() {
36         if (marks >= 90) grade = "A";
37         else if (marks >= 75) grade = "B";
38         else if (marks >= 60) grade = "C";
39         else grade = "D";
40     }
41
42     public void displayInfo() {
43         System.out.println("Roll No: " + rollNo);
44         System.out.println("Name: " + name);
45         System.out.println("Email: " + email);
46         System.out.println("Course: " + course);
47         System.out.println("Marks: " + marks);
48         System.out.println("Grade: " + grade);
49     }
50
51     public String toFileString() {
52         return rollNo + "," + name + "," + email + "," + course + "," + marks;
53     }
54 }
55
56 interface RecordActions {
57     void addStudent(Student s) throws Exception;
58     void deleteStudent(String name) throws Exception;
59     void searchStudent(String name) throws Exception;
60     void updateStudent(int rollNo, double marks) throws Exception;
61     void viewAllStudents();
62 }
63
64 class StudentNotFoundException extends Exception {
65     public StudentNotFoundException(String msg) { super(msg); }
66 }
67
68 class InvalidInputException extends Exception {
69     public InvalidInputException(String msg) { super(msg); }
70 }
71
72 class Loader implements Runnable {
73     public void run() {
74         try {
75             for (int i = 0; i < 3; i++) {
76                 System.out.print(s1 + "Loading..");
77                 Thread.sleep(1000);
78             }
79             System.out.println();
80         } catch (InterruptedException e) {
81             Thread.currentThread().interrupt();
82         }
83     }
84 }
```



```

364     System.out.println("Error loading file");
365
366 }
367
368
369 public class StudentRecordsystem {
370     Run(Debug
371     public static void main(String[] args) {
372         Scanner sc = new Scanner(System.in);
373         StudentManager manager = new StudentManager();
374         int choice;
375
376         do {
377             System.out.println("----- Capstone Student Menu -----");
378             System.out.println("1. Add Student");
379             System.out.println("2. View All Students");
380             System.out.println("3. Search by Name");
381             System.out.println("4. Delete by Name");
382             System.out.println("5. Sort by Marks");
383             System.out.println("6. Save and Exit");
384             System.out.print("Enter choice: ");
385             choice = sc.nextInt();
386
387             try {
388                 Thread t = new Thread(new Loader());
389                 t.start();
390                 t.join();
391
392                 switch (choice) {
393                     case 1:
394                         System.out.print("Enter Roll No: ");
395                         int r = sc.nextInt();
396                         sc.nextLine();
397                         System.out.print("Enter Name: ");
398                         string n = sc.nextLine();
399                         System.out.print("Enter Email: ");
400                         string e = sc.nextLine();
401                         System.out.print("Enter Course: ");
402                         string c = sc.nextLine();
403                         System.out.print("Enter Marks: ");
404                         double m = sc.nextDouble();
405                         manager.addStudent(new Student(r, n, e, c, m));
406                         break;
407                     case 2:
408                         manager.viewAllStudents();
409                         break;
410                     case 3:
411                         sc.nextLine();
412                         System.out.print("Enter name to search: ");
413                         manager.searchStudent(sc.nextLine());
414                         break;
415                 }
416                 t.join();
417
418                 switch (choice) {
419                     case 1:
420                         System.out.print("Enter Roll No: ");
421                         int r = sc.nextInt();
422                         sc.nextLine();
423                         System.out.print("Enter Name: ");
424                         string n = sc.nextLine();
425                         System.out.print("Enter Email: ");
426                         string e = sc.nextLine();
427                         System.out.print("Enter Course: ");
428                         string c = sc.nextLine();
429                         System.out.print("Enter Marks: ");
430                         double m = sc.nextDouble();
431                         manager.addStudent(new Student(r, n, e, c, m));
432                         break;
433                     case 2:
434                         manager.viewAllStudents();
435                         break;
436                     case 3:
437                         sc.nextLine();
438                         System.out.print("Enter name to search: ");
439                         manager.searchStudent(sc.nextLine());
440                         break;
441                     case 4:
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2296
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2396
2397
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2449
2450
```

```
===== 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

Java Programming Lab

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 (75-89%)	Fair (50-74%)	Poor (0-49%)	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

Java Programming Lab

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%