K.R. MANGALAM UNIVERSITY
THE COMPLETE WORLD OF EDUCATION

School of Engineering and Technology

Java Programing Lab Manual

ENCS251/ENCA251/ENBC251

for

B.Tech. CSE/B.Tech. CSE (AL & ML)/B.Tech. CSE (Cyber Security)/B.Tech. (Data Sc.)
B.Sc.(CS)/B.Sc.(Cyber Security)/B.Sc. (DS)/ BCA(AI & DS)

(2025–2026)

Prepared By:
Dr. Manish Kumar
Java Course Coordinator

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

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 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.

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**)
  - Fields: rollNo, name, course, marks, grade
  - 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**:
    - inputDetails(): Take input from the user to add student details.
    - displayDetails(): Display student details.
    - 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).

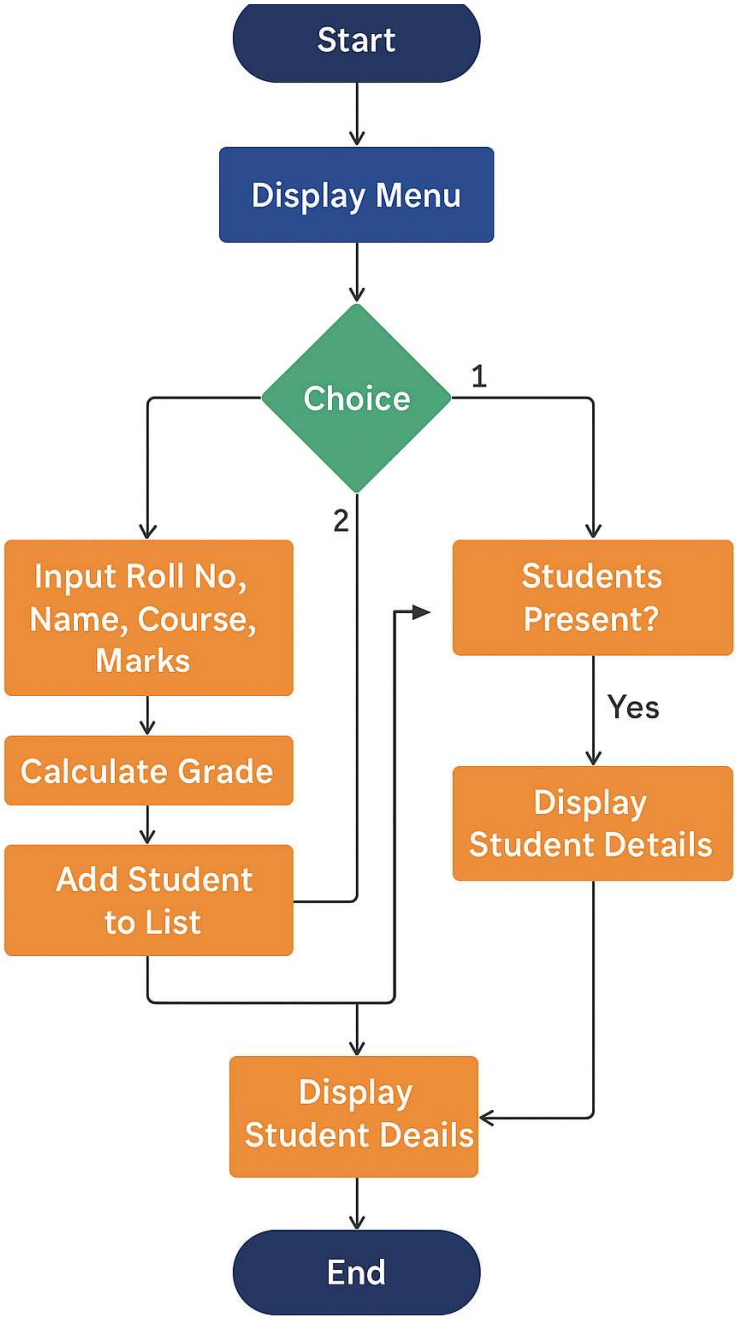
Submission Guidelines

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 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)
   - Fields: name, email
   - Method: displayInfo()
2. **Student** (extends **Person**)
   - Fields: rollNo, course, marks, grade
   - Method: displayInfo()
3. **RecordActions** (interface)

- Methods: addStudent(), deleteStudent(), updateStudent(), searchStudent(), viewAllStudents()

4. **StudentManager** (implements **RecordActions**)
   - 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.

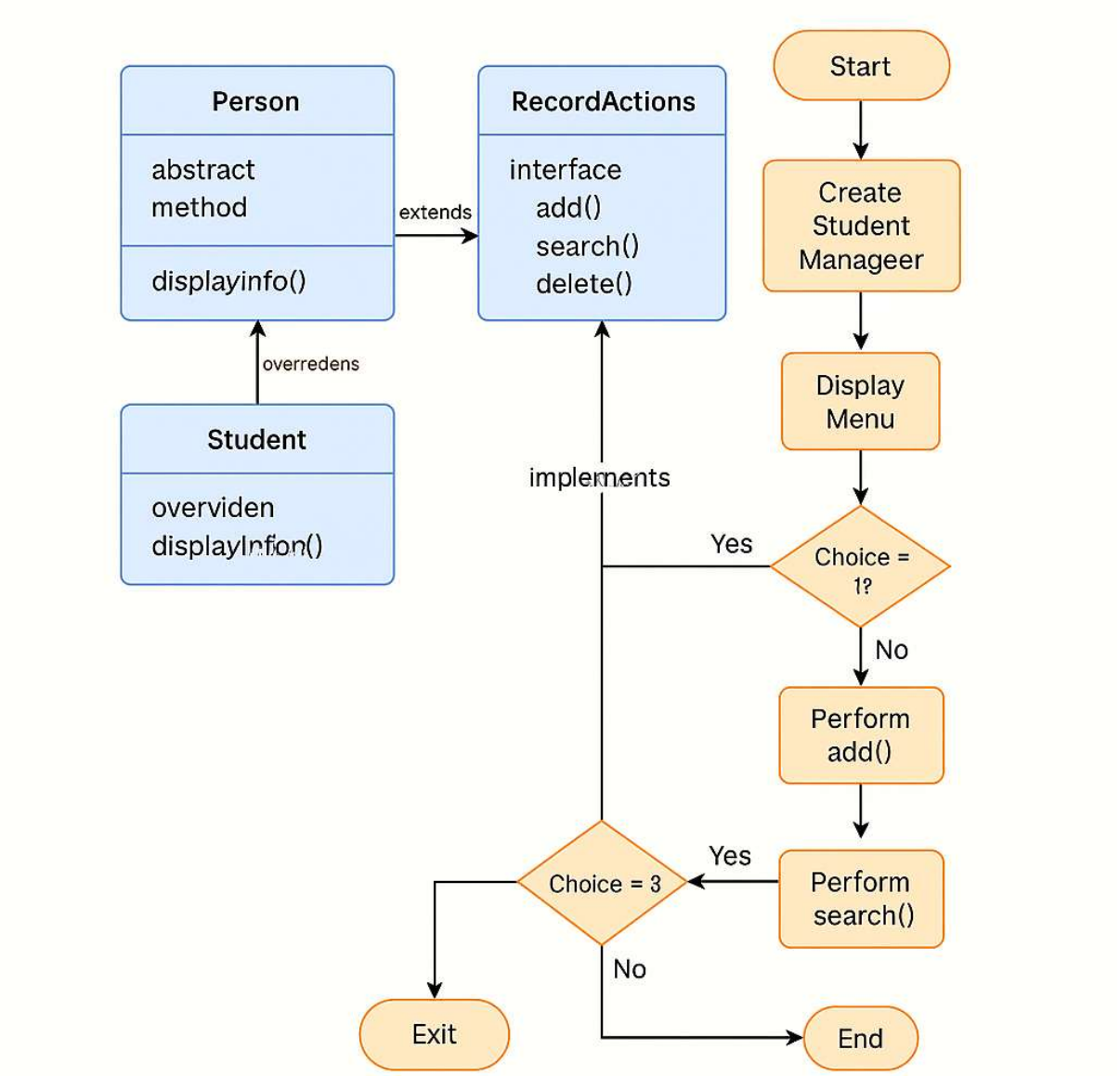
Submission Guidelines

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()
Choice = 3
Yes
Perform search()
No
Exit
End

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.

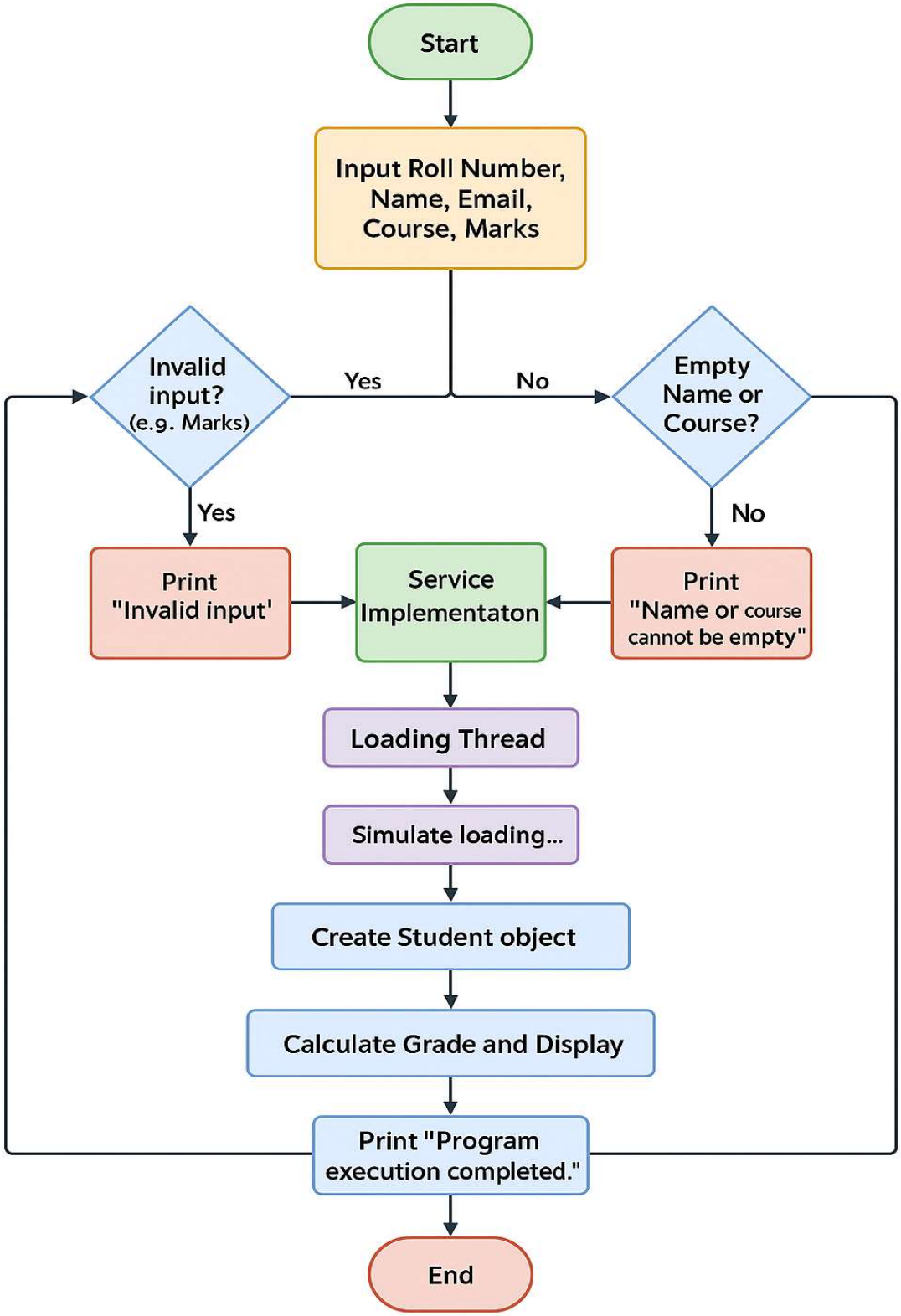
Submission Guidelines

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 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**.

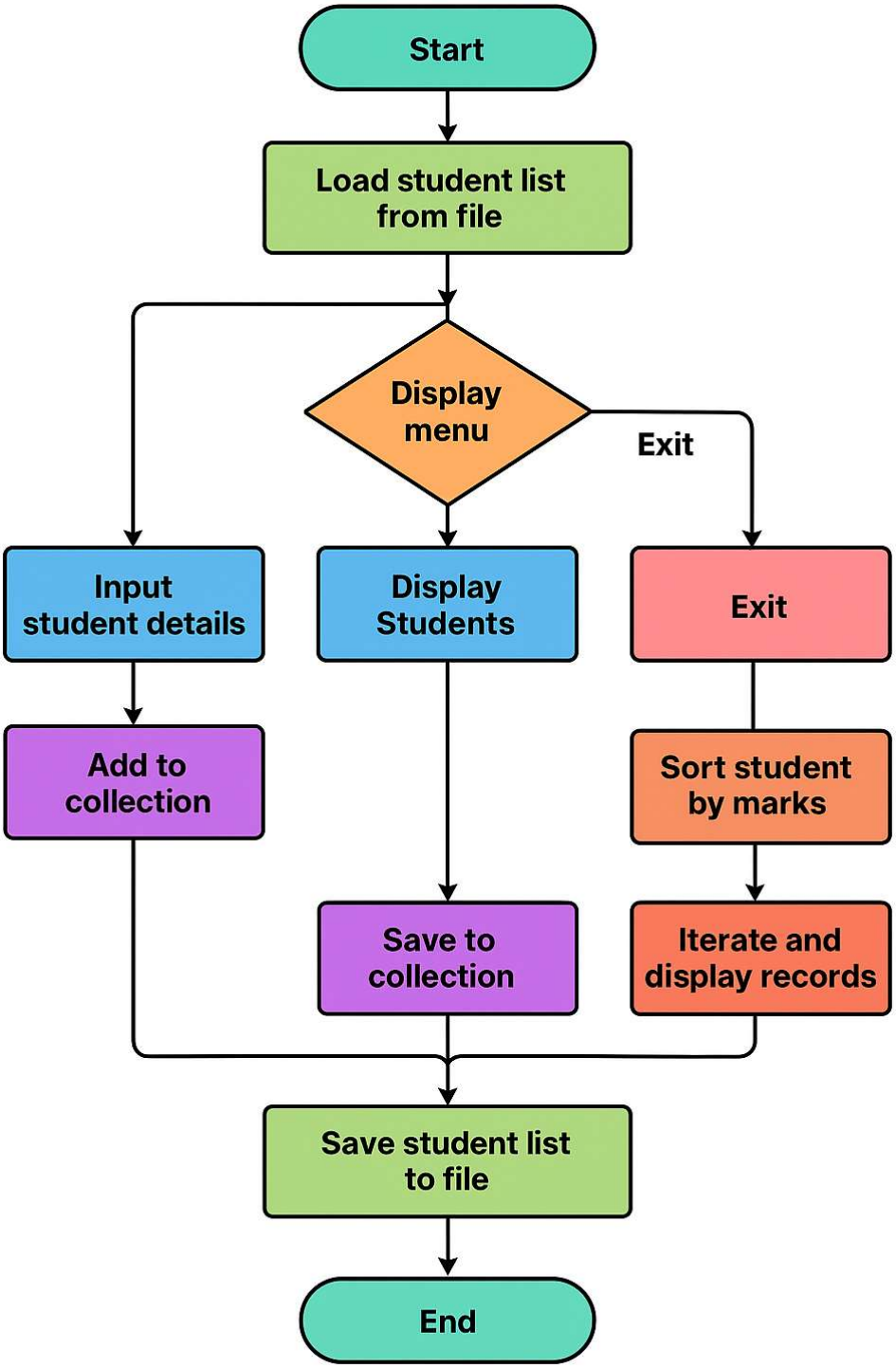
Submission Guidelines

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 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)
   - Fields: name, email
   - Methods: displayInfo() (abstract)
2. **Student** (extends **Person**)

- Fields: rollNo, course, marks, grade
- Methods: inputDetails(), displayDetails(), calculateGrade()

3. **StudentManager** (implements **RecordActions** interface)
   - Methods: addStudent(), deleteStudent(), updateStudent(), searchStudent(), viewAllStudents()
4. **Loader** (implements **Runnable**)
   - 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**:
    - Ensure all classes are correctly placed within their respective packages (model, service, util).

- Use object-oriented principles like inheritance and interfaces for clean code.

2. **Modularity**:
   - Keep methods short, clear, and reusable.
   - Handle all exceptions with meaningful messages.
3. **File Handling**:
   - Handle file reading/writing operations with **BufferedReader** and **BufferedWriter**.
   - Ensure the program can load existing student records on startup and save updated records before exiting.
4. **Multithreading**:
   - Simulate a realistic loading experience by using a Thread class.

Improvements/Adjustments

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

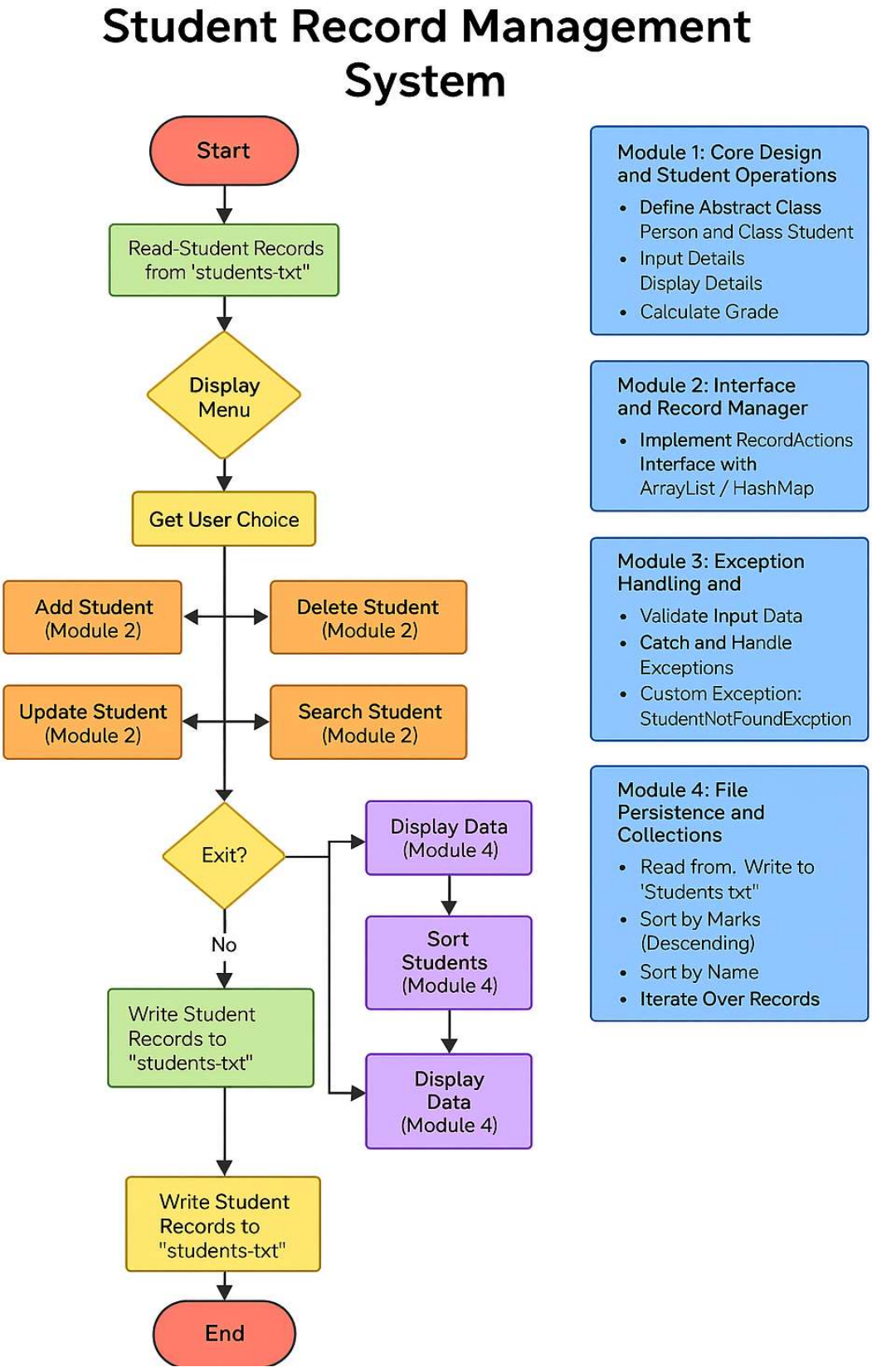
Submission Guidelines

1. **Code Submission**:
   - Submit the entire project folder with all Java source files.
   - 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 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
Code Readability & Style	Well-formatted, 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%