K.R. MANGALAM UNIVERSITY
THE COMPLETE WORLD OF EDUCATION

School of Engineering and Technology

Java Programing Lab Manual

ENCS251/ENCA251/ENBC251

NAME : ARYAN

COURSE : B.TECH CSE(FSD)

ROLL NO : 2401350001

CONTENT

Assignment 1 :

Problem Statement - 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

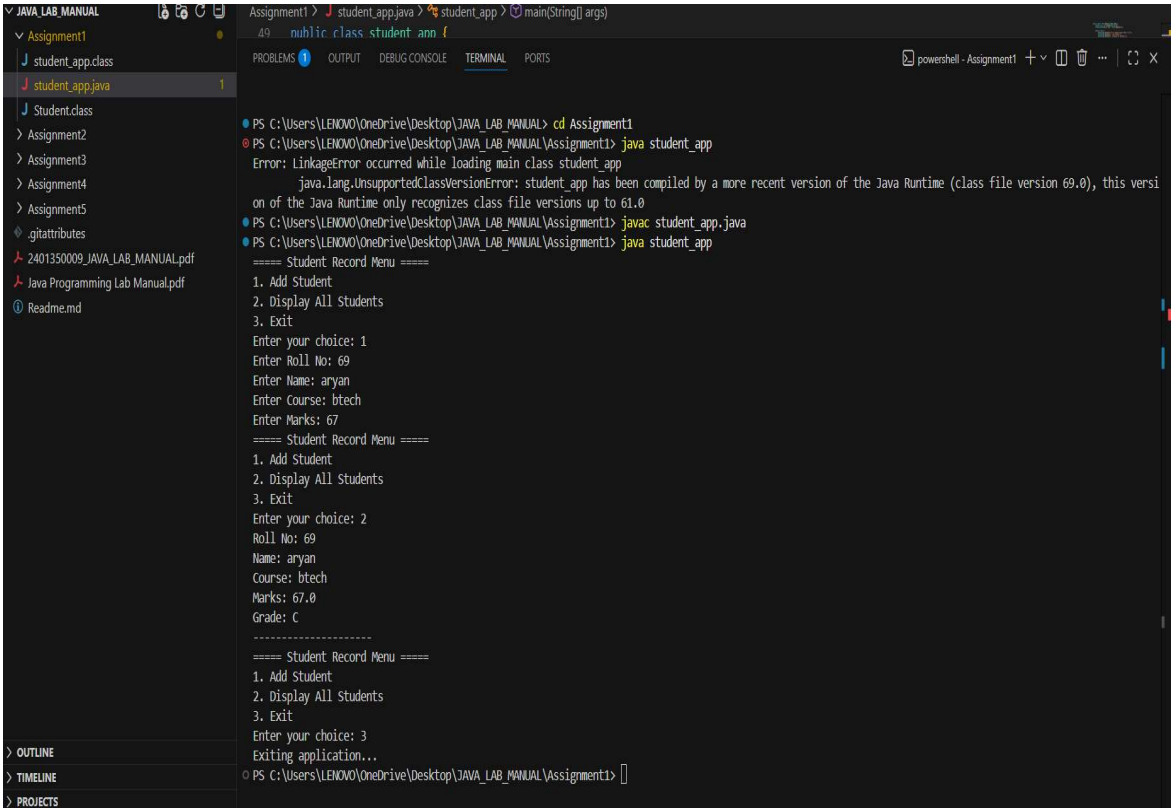
Code:

```
import java.util.*;

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

    // Input student details
    void inputDetails() {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter Roll No: ");
        rollNo = sc.nextInt();
        sc.nextLine();

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

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

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

        calculateGrade();
    }
```

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

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

public class student_app {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ArrayList<Student> list = new ArrayList<>();

        while (true) {
            System.out.println("===== Student Record Menu =====");
            System.out.println("1. Add Student");
            System.out.println("2. Display All Students");
            System.out.println("3. Exit");
            System.out.print("Enter your choice: ");
            int ch = sc.nextInt();

            switch (ch) {
                case 1:
                    Student st = new Student();
                    st.inputDetails();
                    list.add(st);
                    break;

                case 2:
                    for (Student s : list) {
                        s.displayDetails();
                    }
```

```
                    break;

                case 3:
                    System.out.println("Exiting application...");
                    return;

                default:
                    System.out.println("Invalid choice!");
            }
        }
    }
}
```

Output Screenshot :



Assignment 2 :

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.

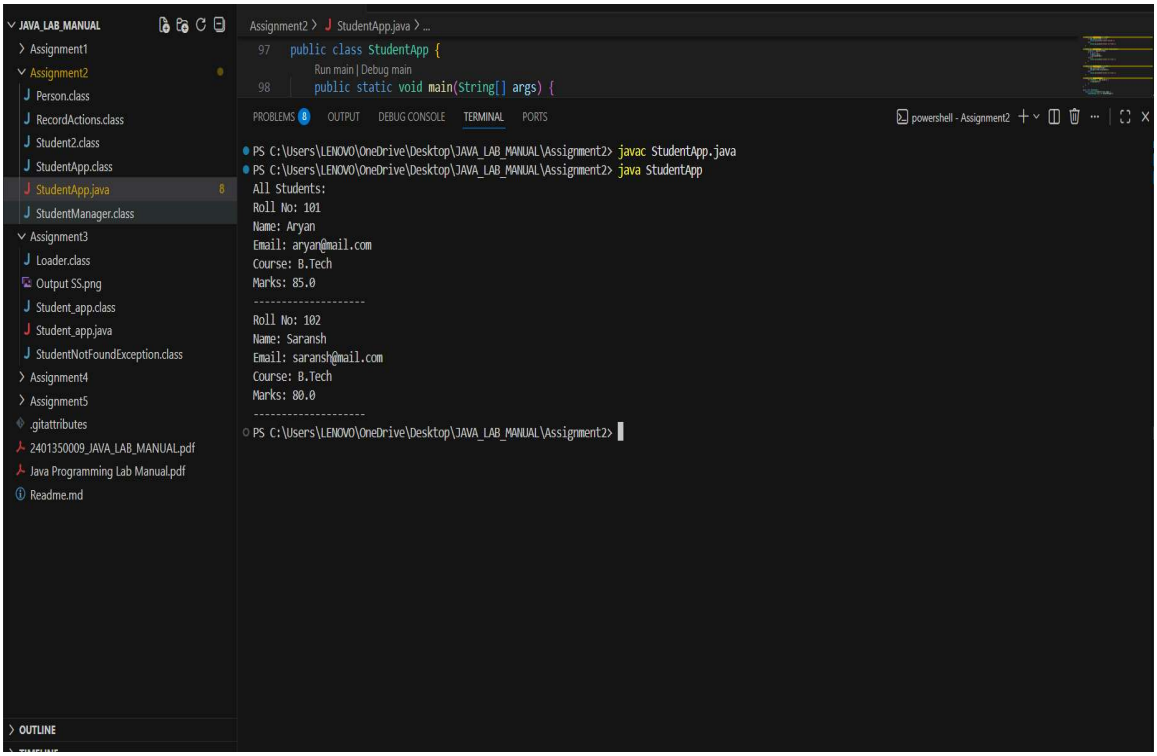
Code :

```
import java.util.*;

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

    abstract void displayInfo();
}

// Student class extending Person
class Student2 extends Person {
    int rollNo;
    String course;
    double marks;
    char grade;

    Student2(String name, String email, int rollNo, String course,
double marks) {
        this.name = name;
        this.email = email;
        this.rollNo = rollNo;
        this.course = course;
        this.marks = marks;
```

```
        calculateGrade();
    }

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

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

// Interface
interface RecordActions {
    void addStudent(Student2 s);
    void deleteStudent(int rollNo);
    void updateStudent(int rollNo, double marks);
    void viewAllStudents();
    void searchStudent(int rollNo);
}

// Student Manager implementing interface
class StudentManager implements RecordActions {
    Map<Integer, Student2> map = new HashMap<>();

    public void addStudent(Student2 s) {
        if (map.containsKey(s.rollNo)) {
            System.out.println("Duplicate Roll Number Not Allowed!");
            return;
        }
        map.put(s.rollNo, s);
    }

    public void deleteStudent(int rollNo) {
```

```
        if (map.remove(rollNo) != null)
            System.out.println("Student deleted.");
        else
            System.out.println("Student not found.");
    }

    public void updateStudent(int rollNo, double marks) {
        Student2 s = map.get(rollNo);
        if (s != null) {
            s.marks = marks;
            s.calculateGrade();
        } else {
            System.out.println("Student not found.");
        }
    }

    public void searchStudent(int rollNo) {
        if (map.containsKey(rollNo))
            map.get(rollNo).displayInfo();
        else
            System.out.println("Student not found.");
    }

    public void viewAllStudents() {
        for (Student2 s : map.values()) {
            s.displayInfo();
        }
    }
}

public class StudentApp {
    public static void main(String[] args) {
        StudentManager sm = new StudentManager();

        sm.addStudent(new Student2("krish", "krish@mail.com", 101,
"B.Tech", 85));
        sm.addStudent(new Student2("Aryan", "aryan@mail.com", 102,
"B.Tech", 42));

        System.out.println("All Students:");
        sm.viewAllStudents();
    }
}
```

Output:


JAVA_LAB_MANUAL
Assignment1
Assignment2
Person.class
RecordActions.class
Student2.class
StudentApp.class
StudentApp.java
StudentManager.class
Assignment3
Loader.class
Output SS.png
Student_app.class
Student_app.java
StudentNotFoundException.class
Assignment4
Assignment5
.gitattributes
2401350009_JAVA_LAB_MANUAL.pdf
Java Programming Lab Manual.pdf
Readme.md
OUTLINE
Assignment2 > StudentApp.java > ...
97 public class StudentApp {
Run main | Debug main
98 public static void main(String[] args) {
PROBLEMS 8
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
powershell - Assignment2
PS C:\Users\LENOVO\OneDrive\Desktop\JAVA_LAB_MANUAL\Assignment2> javac StudentApp.java
PS C:\Users\LENOVO\OneDrive\Desktop\JAVA_LAB_MANUAL\Assignment2> java StudentApp
All Students:
Roll No: 101
Name: Aryan
Email: aryan@mail.com
Course: B.Tech
Marks: 85.0
Roll No: 102
Name: Saransh
Email: saransh@mail.com
Course: B.Tech
Marks: 80.0
PS C:\Users\LENOVO\OneDrive\Desktop\JAVA_LAB_MANUAL\Assignment2>

Assignment 3:

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.

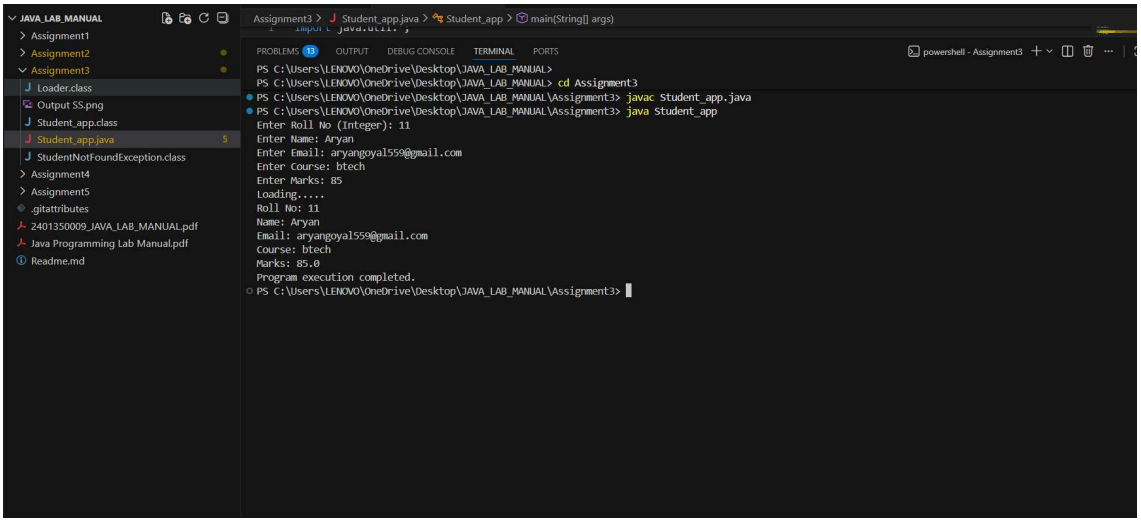
Code :

```
import java.util.*;

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

// Loader thread
class Loader implements Runnable {
    public void run() {
        System.out.print("Loading");
        try {
            for (int i = 0; i < 5; i++) {
                Thread.sleep(300);
```

```
                System.out.print(".");
            }
        } catch (Exception e) {}
        System.out.println();
    }
}

public class Student_app {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        try {
            System.out.print("Enter Roll No (Integer): ");
            Integer roll = sc.nextInt();

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

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

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

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

            if (marks < 0 || marks > 100)
                throw new Exception("Invalid marks range!");

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

            System.out.println("Roll No: " + roll);
            System.out.println("Name: " + name);
            System.out.println("Email: " + email);
            System.out.println("Course: " + course);
            System.out.println("Marks: " + marks);

        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
```

```
            }
            finally {
                System.out.println("Program execution completed.");
            }
        }
}
```

Output :



Assignment 4:

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.

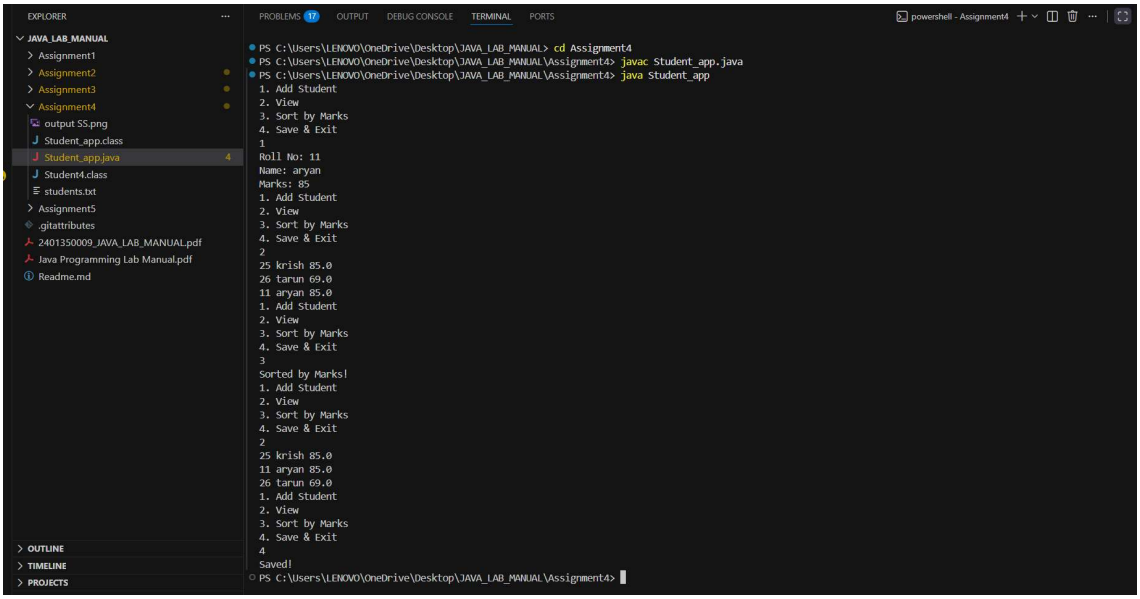
Code :

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

class Student4 {
    int rollNo;
    String name;
    double marks;

    Student4(int r, String n, double m) {
        rollNo = r;
        name = n;
        marks = m;
    }
```

```
    public String toString() {
        return rollNo + "," + name + "," + marks;
    }
}

public class Student_app {
    static ArrayList<Student4> list = new ArrayList<>();

    public static void loadFile() throws Exception {
        File file = new File("students.txt");
        if (!file.exists()) return;

        BufferedReader br = new BufferedReader(new FileReader(file));
        String line;

        while ((line = br.readLine()) != null) {
            String[] data = line.split(",");
            list.add(new Student4(
                Integer.parseInt(data[0]),
                data[1],
                Double.parseDouble(data[2])
            ));
        }
        br.close();
    }

    public static void saveFile() throws Exception {
        BufferedWriter bw = new BufferedWriter(new
FileWriter("students.txt"));
        for (Student4 s : list) {
            bw.write(s.toString());
            bw.newLine();
        }
        bw.close();
    }

    public static void main(String[] args) throws Exception {
        loadFile();

        Scanner sc = new Scanner(System.in);

        while (true) {
            System.out.println("1. Add Student");
```

```
            System.out.println("2. View");
            System.out.println("3. Sort by Marks");
            System.out.println("4. Save & Exit");
            int ch = sc.nextInt();

            switch (ch) {
                case 1:
                    System.out.print("Roll No: ");
                    int r = sc.nextInt();
                    sc.nextLine();

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

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

                    list.add(new Student4(r, n, m));
                    break;

                case 2:
                    for (Student4 s : list) {
                        System.out.println(s.rollNo + " " + s.name + "
" + s.marks);
                    }
                    break;

                case 3:
                    list.sort((a, b) -> Double.compare(b.marks,
a.marks));
                    System.out.println("Sorted by Marks!");
                    break;

                case 4:
                    saveFile();
                    System.out.println("Saved!");
                    return;
            }
        }
    }
}
```

Output :



Assignment 5:

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.

Output :

```
// Project: Student Record Management System (Capstone)
// Structure (files below separated by headers):

/* === File: model/Person.java === */
package model;

public abstract class Person {
    protected String name;
    protected String email;

    public Person() {}

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

    public String getName() { return name; }
    public String getEmail() { return email; }

    public void setName(String name) { this.name = name; }
    public void setEmail(String email) { this.email = email; }

    public abstract void displayInfo();
}


/* === File: model/Student.java === */
package model;
```

```java
public class Student extends Person implements Comparable<Student> {
    private int rollNo;
    private String course;
    private double marks;
    private char grade;

    public Student() {}

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

    public int getRollNo() { return rollNo; }
    public String getCourse() { return course; }
    public double getMarks() { return marks; }
    public char getGrade() { return grade; }

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

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

    @Override
    public void displayInfo() {
        System.out.println("Roll No: " + rollNo);
        System.out.println("Name: " + name);
        System.out.println("Email: " + email);
        System.out.println("Course: " + course);
        System.out.println("Marks: " + marks);
        System.out.println("Grade: " + grade);
        System.out.println("----------------------------");
    }
```

```
    @Override
    public String toString() {
        // CSV line
        return rollNo + "," + name.replace(',', ' ') + "," +
email.replace(',', ' ') + "," + course.replace(',', ' ') + "," + marks;
    }

    public static Student fromCSV(String line) {
        if (line == null || line.trim().isEmpty()) return null;
        String[] parts = line.split(",");
        if (parts.length < 5) return null;
        try {
            int r = Integer.parseInt(parts[0].trim());
            String n = parts[1].trim();
            String e = parts[2].trim();
            String c = parts[3].trim();
            double m = Double.parseDouble(parts[4].trim());
            return new Student(r, n, e, c, m);
        } catch (Exception ex) {
            return null;
        }
    }

    @Override
    public int compareTo(Student other) {
        // descending by marks
        return Double.compare(other.marks, this.marks);
    }
}


/* === File: service/RecordActions.java === */
package service;

import model.Student;
import java.util.List;

public interface RecordActions {
    boolean addStudent(Student s);
    boolean deleteStudent(int rollNo);
    boolean updateStudentMarks(int rollNo, double marks);
    Student searchByRoll(int rollNo);
```

```
    List<Student> viewAllStudents();
}


/* === File: service/StudentManager.java === */
package service;

import model.Student;
import util.FileUtil;

import java.util.*;
import java.io.IOException;

public class StudentManager implements RecordActions {
    private Map<Integer, Student> map = new HashMap<>();

    public StudentManager() {}

    // load existing students from file
    public void loadFromFile(String path) throws IOException {
        List<Student> list = FileUtil.readStudents(path);
        if (list != null) {
            for (Student s : list) map.put(s.getRollNo(), s);
        }
    }

    // save current students to file
    public void saveToFile(String path) throws IOException {
        FileUtil.writeStudents(path, new ArrayList<>(map.values()));
    }

    @Override
    public boolean addStudent(Student s) {
        if (map.containsKey(s.getRollNo())) return false;
        map.put(s.getRollNo(), s);
        return true;
    }

    @Override
    public boolean deleteStudent(int rollNo) {
        return map.remove(rollNo) != null;
    }
```

```
    @Override
    public boolean updateStudentMarks(int rollNo, double marks) {
        Student s = map.get(rollNo);
        if (s == null) return false;
        s.setMarks(marks);
        return true;
    }

    @Override
    public Student searchByRoll(int rollNo) {
        return map.get(rollNo);
    }

    @Override
    public List<Student> viewAllStudents() {
        return new ArrayList<>(map.values());
    }

    // utility: return sorted list by marks desc
    public List<Student> getSortedByMarksDesc() {
        List<Student> list = viewAllStudents();
        Collections.sort(list);
        return list;
    }
}


/* === File: util/FileUtil.java === */
package util;

import model.Student;

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

public class FileUtil {
    // read students from a CSV file (one student per line)
    public static List<Student> readStudents(String path) throws
IOException {
        File file = new File(path);
        List<Student> list = new ArrayList<>();
        if (!file.exists()) return list;
```

```
        try (BufferedReader br = new BufferedReader(new
FileReader(file))) {
            String line;
            while ((line = br.readLine()) != null) {
                Student s = Student.fromCSV(line);
                if (s != null) list.add(s);
            }
        }
        return list;
    }

    public static void writeStudents(String path, List<Student> list)
throws IOException {
        try (BufferedWriter bw = new BufferedWriter(new
FileWriter(path))) {
            for (Student s : list) bw.write(s.toString() +
System.lineSeparator());
        }
    }
}


/* === File: util/Loader.java === */
package util;

public class Loader implements Runnable {
    private final String message;

    public Loader(String message) { this.message = message; }

    @Override
    public void run() {
        System.out.print(message);
        try {
            for (int i = 0; i < 6; i++) {
                Thread.sleep(200);
                System.out.print('.');
            }
        } catch (InterruptedException ignored) {}
        System.out.println();
    }
}
```

```java
/* === File: app/Main.java === */
package app;

import model.Student;
import service.StudentManager;
import util.Loader;

import java.util.*;
import java.io.IOException;

public class Main {
    private static final String DATA_FILE = "students.txt"; // project
root

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

        // load from file
        try {
            manager.loadFromFile(DATA_FILE);
            System.out.println("Loaded students from file: " +
DATA_FILE);
        } catch (IOException e) {
            System.out.println("No existing data or error reading
file.");
        }

        while (true) {
            printMenu();
            System.out.print("Enter choice: ");
            String in = sc.nextLine();
            int choice = -1;
            try { choice = Integer.parseInt(in.trim()); } catch
(Exception ignored) {}

            switch (choice) {
                case 1:
                    addStudentFlow(sc, manager);
                    break;
                case 2:
                    viewAll(manager);
```

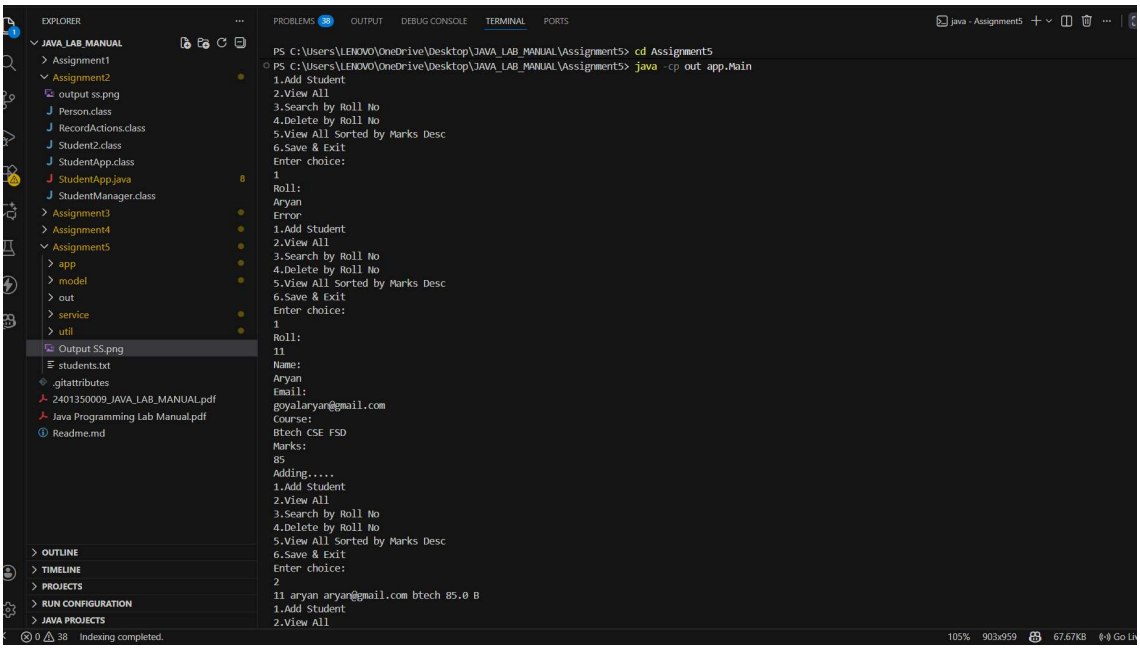
```
                    break;
                case 3:
                    searchByRoll(sc, manager);
                    break;
                case 4:
                    deleteByRoll(sc, manager);
                    break;
                case 5:
                    sortByMarks(manager);
                    break;
                case 6:
                    // save and exit
                    try {
                        Thread t = new Thread(new Loader("Saving"));
                        t.start();
                        t.join();
                        manager.saveToFile(DATA_FILE);
                        System.out.println("Saved to " + DATA_FILE + ".
Exiting.");
                    } catch (IOException | InterruptedException e) {
                        System.out.println("Error saving file: " +
e.getMessage());
                    }
                    return;
                default:
                    System.out.println("Invalid choice. Enter 1-6.");
            }
        }
    }

    private static void printMenu() {
        System.out.println("===== Capstone Student Menu =====");
        System.out.println("1. Add Student");
        System.out.println("2. View All Students");
        System.out.println("3. Search by Roll No");
        System.out.println("4. Delete by Roll No");
        System.out.println("5. Sort by Marks (desc)");
        System.out.println("6. Save and Exit");
    }

    private static void addStudentFlow(Scanner sc, StudentManager
manager) {
        try {
```

```
            System.out.print("Enter Roll No: ");
            int r = Integer.parseInt(sc.nextLine().trim());
            System.out.print("Enter Name: ");
            String name = sc.nextLine().trim();
            System.out.print("Enter Email: ");
            String email = sc.nextLine().trim();
            System.out.print("Enter Course: ");
            String course = sc.nextLine().trim();
            System.out.print("Enter Marks: ");
            double marks = Double.parseDouble(sc.nextLine().trim());

            if (marks < 0 || marks > 100) {
                System.out.println("Marks must be between 0 and 100.");
                return;
            }

            Thread t = new Thread(new Loader("Adding"));
            t.start();
            t.join();

            boolean ok = manager.addStudent(new Student(r, name, email,
course, marks));
            if (ok) System.out.println("Student added.");
            else System.out.println("Roll number already exists.");
        } catch (NumberFormatException nfe) {
            System.out.println("Invalid number format.");
        } catch (InterruptedException ie) {
            System.out.println("Interrupted.");
        }
    }

    private static void viewAll(StudentManager manager) {
        List<Student> list = manager.viewAllStudents();
        if (list.isEmpty()) System.out.println("No students.");
        else list.forEach(Student::displayInfo);
    }

    private static void searchByRoll(Scanner sc, StudentManager
manager) {
        try {
            System.out.print("Enter Roll No to search: ");
            int r = Integer.parseInt(sc.nextLine().trim());
            Student s = manager.searchByRoll(r);
```

```
            if (s != null) s.displayInfo();
            else System.out.println("Student not found.");
        } catch (NumberFormatException e) {
            System.out.println("Invalid number.");
        }
    }

    private static void deleteByRoll(Scanner sc, StudentManager
manager) {
        try {
            System.out.print("Enter Roll No to delete: ");
            int r = Integer.parseInt(sc.nextLine().trim());
            boolean ok = manager.deleteStudent(r);
            if (ok) System.out.println("Student deleted.");
            else System.out.println("Student not found.");
        } catch (NumberFormatException e) {
            System.out.println("Invalid number.");
        }
    }

    private static void sortByMarks(StudentManager manager) {
        List<Student> sorted = manager.getSortedByMarksDesc();
        if (sorted.isEmpty()) System.out.println("No students.");
        else sorted.forEach(Student::displayInfo);
    }
}


/* === File: README.md === */
# Student Record Management System (Java)

Simple console-based Java project demonstrating OOP, file I/O,
multithreading, exception handling and collections.

## How to run
1. Compile: `javac -d out $(find . -name "*.java")` (or compile by
packages)
2. Run: `java -cp out app.Main`

Data is stored in `students.txt` in project root (CSV format).


/* === File: .gitignore === */
```

```
out/
*.class
students.txt

/* === File: students.txt (sample) === */
101,krish,krish@mail.com,B.Tech,85.5
102,aryan,aryan@mail.com,M.Tech,91.0
```

Output :


EXPLORER
JAVA_LAB_MANUAL
Assignment1
Assignment2
output ss.png
Person.class
RecordActions.class
Student2.class
StudentApp.class
StudentApp.java 8
StudentManager.class
Assignment3
Assignment4
Assignment5
app
model
out
service
util
Output SS.png
students.txt
.gitattributes
2401350009_JAVA_LAB_MANUAL.pdf
Java Programming Lab Manual.pdf
Readme.md
OUTLINE
TIMELINE
PROJECTS
RUN CONFIGURATION
JAVA PROJECTS
PROBLEMS 38
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
java - Assignment5
PS C:\Users\LENOVO\OneDrive\Desktop\JAVA_LAB_MANUAL\Assignment5> cd Assignment5
PS C:\Users\LENOVO\OneDrive\Desktop\JAVA_LAB_MANUAL\Assignment5> java -cp out app.Main
1.Add Student
2.View All
3.Search by Roll No
4.Delete by Roll No
5.View All Sorted by Marks Desc
6.Save & Exit
Enter choice:
1
Roll:
Aryan
Error
1.Add Student
2.View All
3.Search by Roll No
4.Delete by Roll No
5.View All Sorted by Marks Desc
6.Save & Exit
Enter choice:
1
Roll:
11
Name:
Aryan
Email:
goyalaryan@gmail.com
Course:
Btech CSE FSD
Marks:
85
Adding.....
1.Add Student
2.View All
3.Search by Roll No
4.Delete by Roll No
5.View All Sorted by Marks Desc
6.Save & Exit
Enter choice:
2
11 aryan aryan@gmail.com btech 85.0 B
1.Add Student
2.View All
0 38 Indexing completed.
105% 903x959 67.67KB