LAB ASSIGNMENT-4

Name- Rajat Rao

Roll no.-2401201067

Course-BCA(AI&DS)

Input-

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


public class StudentRecordApp {

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

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

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

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

    public String toFileLine() {
        // Escape any '|' in text (very simple)
        return rollNo + "|" + escape(name) + "|" + escape(email) + "|" + escape(course) + "|" + marks;
    }

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

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

    public void display() {
```

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

    // ------- File utility class -------
    static class FileUtil {
        private final Path dataFile;
        private final List<Long> recordOffsets = new ArrayList<>(); // for RandomAccess demo

        public FileUtil(String filename) {
            this.dataFile = Paths.get(filename);
        }

        // ensure file exists
        public void ensureFile() throws IOException {
            if (Files.notExists(dataFile)) {
                Files.createFile(dataFile);
            }
        }

```

```
        // Load all students (BufferedReader) and build offsets for random access
        public List<Student> loadAll() throws IOException {
            ensureFile();
            List<Student> list = new ArrayList<>();
            recordOffsets.clear();

            // Use RandomAccessFile to record offsets, but read via BufferedReader for simplicity of lines
            try (RandomAccessFile raf = new RandomAccessFile(dataFile.toFile(), mode: "r")) {
                long pos;
                while ((pos = raf.getFilePointer()) < raf.length()) {
                    recordOffsets.add(pos);         // start of this line
                    String line = raf.readLine();   // readLine uses ISO-8859-1 -> fine for simple ASCII text
                    if (line == null) break;
                    // convert bytes read (raf.readLine returns bytes interpreted as ISO-8859-1); convert to UTF-8
                    // but for typical simple ASCII lines this is fine.
                    Student s = Student.fromFileLine(line);
                    if (s != null) list.add(s);
                }
            }

            return list;
        }

        // Save all students (overwrite) using BufferedWriter
        public void saveAll(List<Student> students) throws IOException {
            ensureFile();
```

```
        try (BufferedWriter bw = Files.newBufferedWriter(dataFile)) {
            for (Student s : students) {
                bw.write(s.toFileLine());
                bw.newLine();
            }
        }
    }

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

    // Demonstrate RandomAccessFile: read record at index (0-based)
    public Student readRecordAtIndex(int index) throws IOException {
        if (index < 0 || index >= recordOffsets.size()) return null;
        long pos = recordOffsets.get(index);
```

```
        try (RandomAccessFile raf = new RandomAccessFile(dataFile.toFile(), mode: "r")) {
            raf.seek(pos);
            String line = raf.readLine();
            if (line == null) return null;
            return Student.fromFileLine(line);
        }
    }

    // expose count of offsets
    public int getRecordCount() { return recordOffsets.size(); }
}

// ------- Student manager -------
static class StudentManager {
    private final List<Student> students = new ArrayList<>();
    private final Map<Integer, Student> studentMap = new HashMap<>();
    private final FileUtil fileUtil;

    public StudentManager(FileUtil fileUtil) {
        this.fileUtil = fileUtil;
    }

    public void load() {
        try {
            List<Student> loaded = fileUtil.loadAll();
```

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

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

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

```
        studentMap.put(s.getRollNo(), s);
        return true;
    }

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

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

```

```
    public void displayAll() {
        if (students.isEmpty()) {
            System.out.println(x: "No students loaded.");
            return;
        }
        Iterator<Student> it = students.iterator();
        while (it.hasNext()) {
            Student s = it.next();
            s.display();
        }
    }

    // Sort by marks using Comparator (descending)
    public void sortByMarksDescending() {
        students.sort(Comparator.comparingDouble(Student::getMarks).reversed());
    }

    // Sort by name ascending
    public void sortByName() {
        students.sort(Comparator.comparing(Student::getName, String.CASE_INSENSITIVE_ORDER));
    }

    public FileUtil getFileUtil() { return fileUtil; }

    // For random access demonstration: returns number of records known
    public int getRecordCount() { return fileUtil.getRecordCount(); }
```

```java
    }

    // ------- Main menu -------
    Run | Debug
    public static void main(String[] args) {
        final String filename = "students.txt";
        FileUtil fileUtil = new FileUtil(filename);
        StudentManager manager = new StudentManager(fileUtil);
        manager.load();

        Scanner sc = new Scanner(System.in);
        boolean running = true;

        while (running) {
            System.out.println(x: "===== Capstone Student Menu =====");
            System.out.println(x: "1. Add Student");
            System.out.println(x: "2. View All Students");
            System.out.println(x: "3. Search by Name");
            System.out.println(x: "4. Delete by Name");
            System.out.println(x: "5. Sort by Marks (descending)");
            System.out.println(x: "6. Sort by Name (ascending)");
            System.out.println(x: "7. Show file attributes");
            System.out.println(x: "8. Read Random Record (RandomAccessFile demo)");
            System.out.println(x: "9. Save and Exit");
            System.out.print(s: "Enter choice: ");
            String line = sc.nextLine().trim();
```

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

```
                }
                case 5 -> {
                    manager.sortByMarksDescending();
                    System.out.println(x: "Sorted Student List by Marks:");
                    manager.displayAll();
                }
                case 6 -> {
                    manager.sortByName();
                    System.out.println(x: "Sorted Student List by Name:");
                    manager.displayAll();
                }
                case 7 -> {
                    // File attributes
                    manager.getFileUtil().printFileAttributes();
                }
                case 8 -> {
                    int count = manager.getRecordCount();
                    if (count == 0) {
                        System.out.println(x: "No records for random access.");
                    } else {
                        Random rnd = new Random();
                        int idx = rnd.nextInt(count);
                        System.out.println("Reading random record index: " + idx);
                        Student rs = manager.getFileUtil().readRecordAtIndex(idx);
                        if (rs != null) {
                            System.out.println(x: "Random record read via RandomAccessFile:");
```

```
                            rs.display();
                        } else {
                            System.out.println(x: "Unable to read random record.");
                        }
                    }
                }
                case 9 -> {
                    manager.save();
                    System.out.println(x: "Exiting... Goodbye!");
                    running = false;
                }
                default -> System.out.println(x: "Invalid choice.");
            }
        } catch (NumberFormatException e) {
            System.out.println(x: "Please enter numeric choice.");
        } catch (IOException ioe) {
            System.out.println("File I/O error: " + ioe.getMessage());
        } catch (Exception ex) {
            System.out.println("Unexpected error: " + ex.getMessage());
        }
        System.out.println();
    }

    sc.close();
}
```

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

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

    private static String readNonEmpty(Scanner sc, String prompt) {
        while (true) {
```


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

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

    private static String readNonEmpty(Scanner sc, String prompt) {
        while (true) {
```


Output-

```
PS C:\Users\rajat> cd "c:\Users\rajat\OneDrive\Desktop\rajat java\" ; if ($?) { javac StudentRecordApp.java } ; if ($?) { java StudentRecordApp }
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)
```

```
7. Show file attributes
8. Read Random Record (RandomAccessFile demo)
9. Save and Exit
Enter choice: 1
Enter Roll No: 67
Enter Name: Rajat Rao
Enter Email: rajatrao404@gmail.com
Enter Course: BCA(AI&DS)
Enter Marks: 80
Student added.
```

EXPLANATION-

1. Student Class
This class represents a single student record.
It stores:
Roll number
Name
Email
Course
Marks
Key functions:
To String()
Returns the student details in readable format.
To File String()
Converts data into a comma-separated string to store in file.
From File String()
Reads one line from file and converts it back into a Student object.
✓ This is used during file loading.

🔹 2. File Util Class
This class handles all file operations.
Read Students(filename)
Uses Buffered Reader
Reads each line
Converts it into Student objects
Adds them to an ArrayList
It also prints:
File path
File size
If file exists
Write Students(filename, list)
Uses Buffered Writer
Writes all student records back into the text file

Saves data permanently
readRandom(filename)
Demonstrates Random Access File:
Reads the first 50 bytes
Shows how to read file content from any position
✓ This is required in your assignment.

🔹 3. Main Program (Menu System)
Uses:
ArrayList<Student> to store records
Scanner for input
Iterator for displaying data
Comparator for sorting by marks
Steps the program performs:

Step 1: Load Students from File
ArrayList<Student> students = FileUtil.readStudents(FILE);
Reads students.txt
Loads existing records
Displays them

Step 2: Show Menu Repeatedly
Menu options:
Add Student
View All Students
Search by Name
Delete by Name
Sort by Marks
Save & Exit

🔹 4. Menu Operations Explanation
(1) Add Student
Takes user input (roll, name, email, course, marks)
Creates a new Student object
Adds it to ArrayList

---

(2) View All Students
Uses Iterator:

```
Iterator<Student> it = students.iterator();
while (it.hasNext()) {
   System.out.println(it.next());
}
```

✓ Fulfills requirement: Displaying with Iterator

---

(3) Search by Name
Compares name (case-insensitive)
Prints matching record

---

(4) Delete by Name
Uses:
students.removeIf(...)
Removes all students with given name.

---

(5) Sort by Marks
Uses Comparator:
students.sort(Comparator.comparingDouble(Student::getMarks).reversed());
✓ Sorts students from highest to lowest marks.

---

(6) Save and Exit
Writes all students back to file using FileUtil.writeStudents().
✓ Ensures persistent storage.