

**Problem Statement:** This assignment focuses on building a complete Student Record Management System using Java by applying OOP concepts such as inheritance, abstraction, and interfaces. The system supports adding, updating, deleting, searching, sorting, and viewing students using the Java Collections Framework. Exception handling is implemented to validate user inputs and manage errors using custom exceptions. File handling is used to store and retrieve student records from a text file, ensuring data persistence. A Loader thread is included to simulate loading using multithreading. Overall, the assignment demonstrates strong programming concepts, including OOP design, error handling, multithreading, collections, and file I/O in a single integrated application.

## 1. Purpose of the Program

The program demonstrates how file handling, collections, random access, and OOP concepts work together to create a student record management system.

It stores data permanently in text files and allows operations like adding, viewing, searching, deleting, and sorting records.

**2. Student Class**  
A Student class represents each student. It stores roll number, name, course, email, and marks. It also contains a display method to show the details.

This shows encapsulation, where related data and methods are combined in one class.

## File Handling (Reading & Writing)

### a. Loading Data

- Uses `BufferedReader` to read the `students.txt` file.
- Each line represents one student record.
- Data is split and converted into a `Student` object.
- Records are added to an `ArrayList`.

### b. Saving Data

- Uses `BufferedWriter` to write all student records back into the file at exit.
- Ensures permanent storage of data.

This demonstrates persistent storage using a text file.

## 4. ArrayList for Storing Students

`ArrayList` is used to store multiple `Student` objects dynamically.

It allows easy:

- Adding
- Deleting
- Searching

- Sorting

This makes the system flexible and scalable.

## 5. StudentManager Class

This class controls all operations.

It includes methods for:

- Adding students
- Viewing all students
- Searching students
- Deleting records
- Sorting records

It separates the application logic from data handling.

## 6. Sorting Using Comparator

Students are sorted by marks using a Comparator.

This shows how Java collections can sort objects based on custom criteria.

## 7. Random Access File Reading

RandomAccessFile is used to read selected characters from any position in the file.

It demonstrates:

- Jumping to specific positions
- Reading only required bytes

Useful for partial reading and indexing.

## 8. Displaying File Properties

Using the File class, the program displays:

- File name
- File path
- File size
- Read/write permissions

This shows how Java interacts with the operating system's file system.

## 9. Menu-Based Interaction

A loop displays a menu with options like:

- Add

- View
- Search
- Delete
- Sort
- File Properties
- Random Access
- Exit

Each option calls the corresponding method in StudentManager.  
This provides clear, user-friendly interaction.

## 10. Scanner Handling and Closing

One Scanner object is used for all input.

It is closed only when the program exits to prevent warnings and errors.

## Overall Concepts Demonstrated

Concept	Description
Classes & Objects	Student, StudentManager, FileUtil
Encapsulation	Student data grouped inside a class
Collections	ArrayList for storing Student objects
File Handling	BufferedReader, BufferedWriter
Random Access	Reading specific file positions
Comparator	Sorting by marks
Exception Handling	try-catch for file and input errors
User Input	Scanner for all menu operations
Menu-Driven Program	Organized and interactive workflow

## Input:

```
J Main.java > ...
1  import java.io.*;
2  import java.util.*;
3
4  // ===== Student Class =====
5  class Student {
6      int rollNo;
7      String name, email, course;
8      double marks;
9
10     public Student(int rollNo, String name, String email, String course, double marks) {
11         this.rollNo = rollNo;
12         this.name = name;
13         this.email = email;
14         this.course = course;
15         this.marks = marks;
16     }
17
18     @Override
19     public String toString() {
20         return "Roll No: " + rollNo + "\nName: " + name +
21             "\nEmail: " + email + "\nCourse: " + course +
22             "\nMarks: " + marks + "\n";
23     }
24 }
25
26 // ===== FileUtil Class =====
27 class FileUtil {
28
29     public static ArrayList<Student> readStudents(String fileName) {
30         ArrayList<Student> list = new ArrayList<>();
31
32         try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
```

```

33     String line;
34
35     while ((line = br.readLine()) != null) {
36         String[] parts = line.split(regex: ",");
37
38         int roll = Integer.parseInt(parts[0]);
39         String name = parts[1];
40         String email = parts[2];
41         String course = parts[3];
42         double marks = Double.parseDouble(parts[4]);
43
44         list.add(new Student(roll, name, email, course, marks));
45     }
46
47     } catch (FileNotFoundException e) {
48         System.out.println(x: "students.txt not found – starting with empty list.");
49     } catch (Exception e) {
50         System.out.println("Error reading file: " + e.getMessage());
51     }
52
53     return list;
54 }
55
56 public static void writeStudents(String fileName, ArrayList<Student> list) {
57     try (BufferedWriter bw = new BufferedWriter(new FileWriter(fileName))) {
58         for (Student s : list) {
59             bw.write(s.rollNo + "," + s.name + "," + s.email + "," + s.course + "," + s.m
60             bw.newLine();
61         }
62     }

```

```

63         System.out.println(x: "\nRecords saved successfully!");
64
65     } catch (Exception e) {
66         System.out.println("Error writing file: " + e.getMessage());
67     }
68 }
69
70 public static void randomRead(String fileName) {
71     try (RandomAccessFile raf = new RandomAccessFile(fileName, mode: "r")) {
72         byte[] buffer = new byte[20];
73         raf.read(buffer);
74         System.out.println("\nRandom Access Read (20 bytes): " + new String(buffer));
75     } catch (Exception e) {
76         System.out.println("Random read error: " + e.getMessage());
77     }
78 }
79 }
80
81 // ===== StudentManager Class =====
82 class StudentManager {
83
84     ArrayList<Student> students = new ArrayList<>();
85     Scanner sc = new Scanner(System.in);
86     String fileName = "students.txt";
87
88     public StudentManager() {
89         students = FileUtil.readStudents(fileName);
90
91         System.out.println(x: "\nLoaded students from file:");
92         for (Student s : students) {

```

```
93         System.out.println(s);
94     }
95 }
96
97 public void addStudent() {
98     System.out.print(s: "Enter Roll No: ");
99     int roll = sc.nextInt();
100    sc.nextLine();
101
102    System.out.print(s: "Enter Name: ");
103    String name = sc.nextLine();
104
105    System.out.print(s: "Enter Email: ");
106    String email = sc.nextLine();
107
108    System.out.print(s: "Enter Course: ");
109    String course = sc.nextLine();
110
111    System.out.print(s: "Enter Marks: ");
112    double marks = sc.nextDouble();
113
114    students.add(new Student(roll, name, email, course, marks));
115    System.out.println(x: "Student Added!");
116 }
117
118 public void viewAll() {
119     System.out.println(x: "\nAll Students:");
120     Iterator<Student> it = students.iterator();
121     while (it.hasNext()) {
122         System.out.println(it.next());
123     }
124 }
```

---

```
123     }
124 }
125
126 public void searchByName() {
127     sc.nextLine();
128     System.out.print(s: "Enter name: ");
129     String name = sc.nextLine();
130
131     boolean found = false;
132     for (Student s : students) {
133         if (s.name.equalsIgnoreCase(name)) {
134             System.out.println("\nFound:\n" + s);
135             found = true;
136         }
137     }
138
139     if (!found)
140         System.out.println("No student found with name: " + name);
141 }
142
143 public void deleteByName() {
144     sc.nextLine();
145     System.out.print(s: "Enter name to delete: ");
146     String name = sc.nextLine();
147
148     boolean removed = students.removeIf(s -> s.name.equalsIgnoreCase(name));
149
150     if (removed) System.out.println(x: "Student Removed!");
151     else System.out.println(x: "No student found!");
152 }
```

```
153
154     public void sortByMarks() {
155         students.sort(Comparator.comparingDouble(s -> s.marks));
156
157         System.out.println(x: "\nSorted by Marks:");
158         for (Student s : students) {
159             System.out.println(s);
160         }
161     }
162
163     public void saveAndExit() {
164         FileUtil.writeStudents(fileName, students);
165         FileUtil.randomRead(fileName);
166         System.out.println(x: "Exiting...");
167     }
168 }
```

```
169
170 // ===== Main Class =====
```

```
171 public class Main {
172     Run | Debug
173     public static void main(String[] args) {
174
175         StudentManager sm = new StudentManager();
176         Scanner sc = new Scanner(System.in);
177
178         while (true) {
179             System.out.println(x: "\n===== MENU =====");
180             System.out.println(x: "1. Add Student");
181             System.out.println(x: "2. View All");
182             System.out.println(x: "3. Search by Name");
183             System.out.println(x: "4. Delete by Name");
```

```
183 System.out.println(x: "5. Sort by Marks");
184 System.out.println(x: "6. Save & Exit");
185 System.out.print(s: "Enter choice: ");
186
187 int ch = sc.nextInt();
188
189 switch (ch) {
190     case 1 -> sm.addStudent();
191     case 2 -> sm.viewAll();
192     case 3 -> sm.searchByName();
193     case 4 -> sm.deleteByName();
194     case 5 -> sm.sortByMarks();
195     case 6 -> {
196         sm.saveAndExit();
197         return;
198     }
199     default -> System.out.println(x: "Invalid choice!");
200 }
201 }
202 }
203 }
204
```

## Output:

```
Loaded students from file:
```

```
===== MENU =====
```

1. Add Student
2. View All
3. Search by Name
4. Delete by Name
5. Sort by Marks
6. Save & Exit

```
Enter choice: 1
```

```
Enter Roll No: 102
```

```
Enter Name: khushi
```

```
Enter Email: khushi@gmail.com
```

```
Enter Course: bca
```

```
Enter Marks: 99
```

```
Student Added!
```

```
===== MENU =====
```

1. Add Student
2. View All
3. Search by Name
4. Delete by Name
5. Sort by Marks
6. Save & Exit

```
Enter choice: 2
```

```
All Students:
```

```
Roll No: 102
```

---

Name: khushi  
Email: khushi@gmail.com  
Course: bca  
Marks: 99.0

===== MENU =====

1. Add Student
2. View All
3. Search by Name
4. Delete by Name
5. Sort by Marks
6. Save & Exit

Enter choice: 3

Enter name: khushi

Found:

Roll No: 102

Name: khushi

Email: khushi@gmail.com

Course: bca

Marks: 99.0

===== MENU =====

1. Add Student
2. View All
3. Search by Name
4. Delete by Name
5. Sort by Marks
6. Save & Exit

Enter choice: 4

khushi

Enter name to delete: khushi

Student Removed!

===== MENU =====

1. Add Student

2. View All

3. Search by Name

4. Delete by Name

5. Sort by Marks

6. Save & Exit

Enter choice: 6

Records saved successfully!

Random Access Read (20 bytes):

Exiting...

PS C:\Users\Hp\JAVA> █