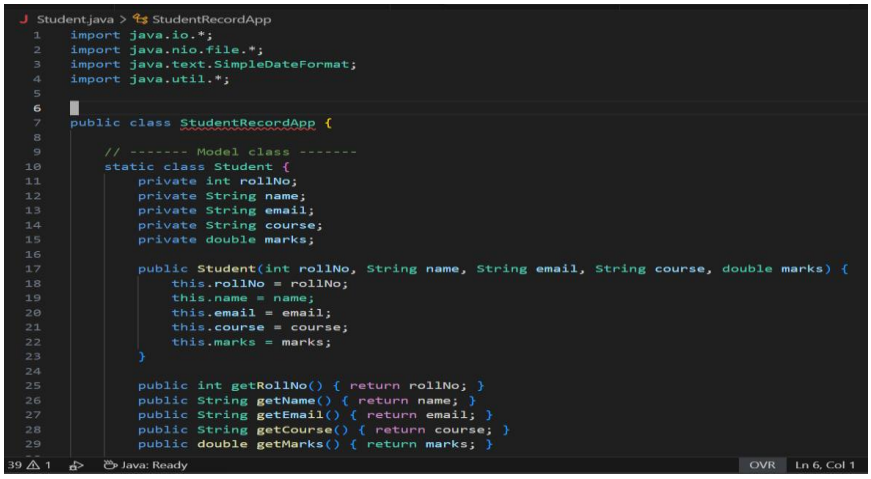
Lab Assignment Number: 04

Name: Aman

Roll no: 2401201115

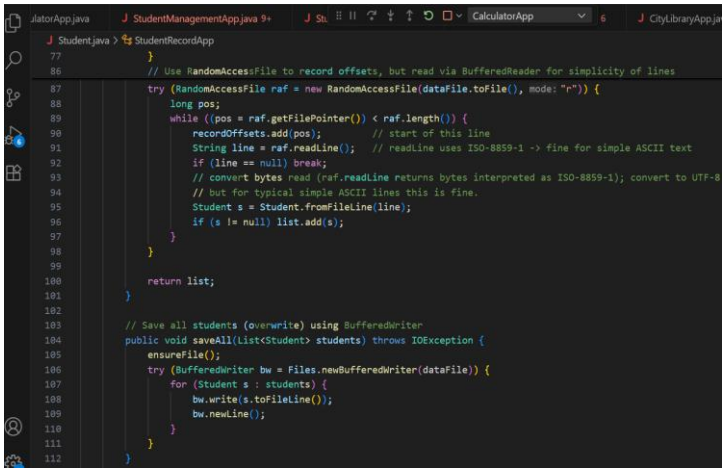
Course: BCA (AI&DS)

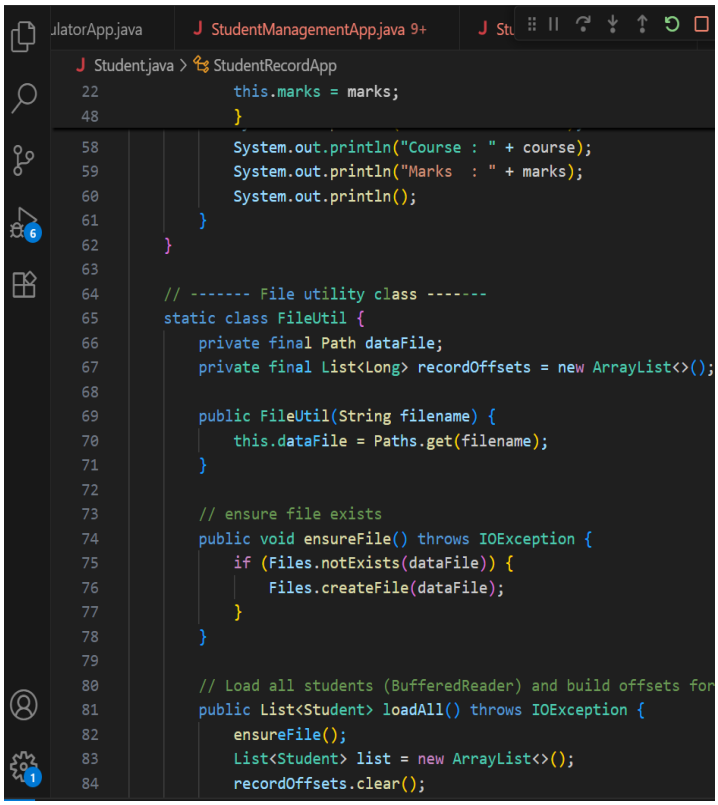
INPUT:

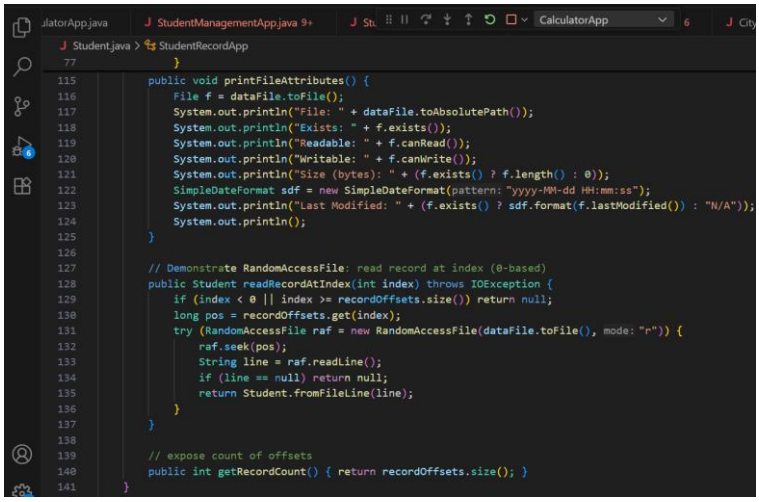


```java
import java.io.*;
import java.nio.file.*;
import java.text.SimpleDateFormat;
import java.util.*;

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; }
```

```java
        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);
```

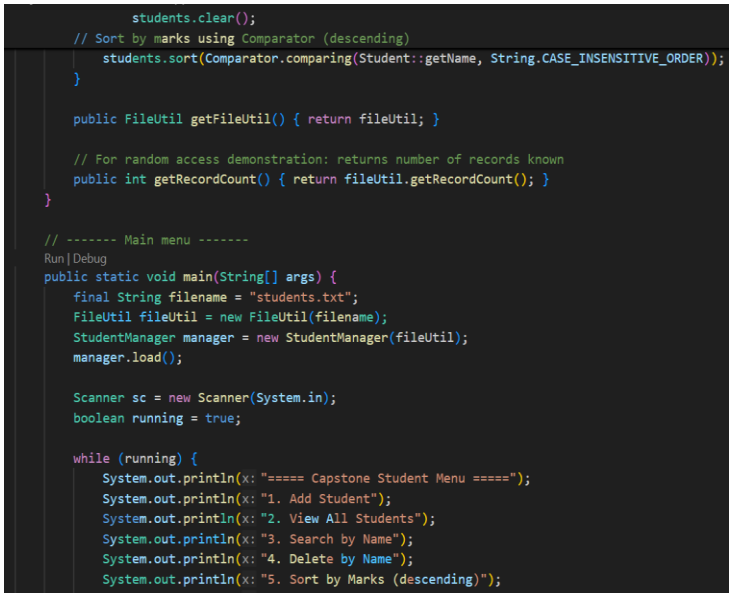

StudentManagementApp.java 9+
CalculatorApp
Student.java > StudentRecordApp
// Use RandomAccessFile to record offsets, but read via BufferedReader for simplicity of lines
try (RandomAccessFile raf = new RandomAccessFile(dataFile.toFile(), mode: "r")) {
long pos;
while ((pos = raf.getFilePointer()) < raf.length()) {
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();


StudentManagementApp.java 9+
Student.java > StudentRecordApp
this.marks = marks;
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<>();
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
public List<Student> loadAll() throws IOException {
ensureFile();
List<Student> list = new ArrayList<>();
recordOffsets.clear();


JlatorApp.java
StudentManagementApp.java 9+
CalculatorApp
Student.java > StudentRecordApp
}
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.java > StudentRecordApp
}
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() {


Student.java > StudentRecordApp
students.clear();
System.out.println("Error loading students: " + e.getMessage());
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();



```
            students.clear();
        // Sort by marks using Comparator (descending)
            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(); }
    }

    // ------- 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)");
```

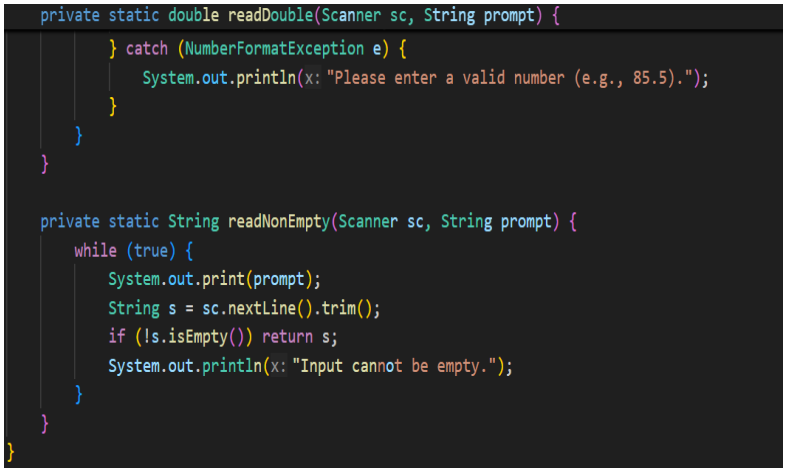
```
            students.clear();
            if (s.getName().equalsIgnoreCase(name)) {
                it.remove();
                studentMap.remove(s.getRollNo());
                removed = true;
            }
        }
        return removed;
    }

    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() {
```

```java
        System.out.println(x: "2. View All Students");
                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.java > StudentRecordApp

```java
47              System.out.println(x: "2. View All Students");
51              System.out.println(x: "6. Sort by Name (ascending)");
52              System.out.println(x: "7. Show file attributes");
53              System.out.println(x: "8. Read Random Record (RandomAccessFile demo)");
54              System.out.println(x: "9. Save and Exit");
55              System.out.print(s: "Enter choice: ");
56              String line = sc.nextLine().trim();
57
58              try {
59                  int choice = Integer.parseInt(line);
60                  switch (choice) {
61                      case 1 -> {
62                          int roll = readInt(sc, prompt: "Enter Roll No: ");
63                          String name = readNonEmpty(sc, prompt: "Enter Name: ");
64                          String email = readNonEmpty(sc, prompt: "Enter Email: ");
65                          String course = readNonEmpty(sc, prompt: "Enter Course: ");
66                          double marks = readDouble(sc, prompt: "Enter Marks: ");
67                          Student s = new Student(roll, name, email, course, marks);
68                          boolean ok = manager.addStudent(s);
69                          if (ok) System.out.println(x: "Student added.");
70                          else System.out.println(x: "Duplicate roll number. Student not added.");
71                      }
72                      case 2 -> manager.displayAll();
73                      case 3 -> {
74                          String name = readNonEmpty(sc, prompt: "Enter Name to search: ");
75                          Student s = manager.searchByName(name);
76                          if (s != null) s.display();
77                          else System.out.println(x: "Student not found.");
78                      }
```

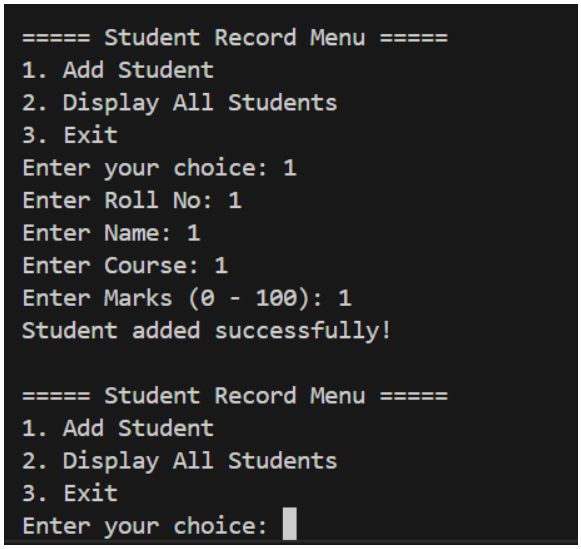
```
Student.java > StudentRecordApp
247                 System.out.println(x: "2. View All Students");
334         }
335
336         // ------- Input helpers -------
337         private static int readInt(Scanner sc, String prompt) {
338             while (true) {
339                 System.out.print(prompt);
340                 String s = sc.nextLine().trim();
341                 try {
342                     return Integer.parseInt(s);
343                 } catch (NumberFormatException e) {
344                     System.out.println(x: "Please enter a valid integer.");
345                 }
346             }
347         }
348
349         private static double readDouble(Scanner sc, String prompt) {
350             while (true) {
351                 System.  String s - StudentRecordApp.readDouble(Scanner, String)
352                 String s = sc.nextLine().trim();
353                 try {
354                     return Double.parseDouble(s);
355                 } catch (NumberFormatException e) {
356                     System.out.println(x: "Please enter a valid number (e.g., 85.5).");
357                 }
358             }
359         }
360
361         private static String readNonEmpty(Scanner sc, String prompt) {
```



```
    private static double readDouble(Scanner sc, String prompt) {
            } 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) {
            System.out.print(prompt);
            String s = sc.nextLine().trim();
            if (!s.isEmpty()) return s;
            System.out.println(x: "Input cannot be empty.");
        }
    }
}
```

```
    System.out.println(x: "2. View All Students");
                    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();
```

OUTPUT:



Explanation —

1 — High-level overview

`StudentRecordApp` is a console-based Student Record Management System that:

- Loads student records from `students.txt` at startup.
- Keeps records in memory using an `ArrayList` and a `HashMap`.
- Lets the user add/view/search/delete/sort records.
- Saves updated records back to `students.txt` on exit.
- Demonstrates file attributes and random-access reads using `RandomAccessFile`.

---

2 — Main components (classes & roles)

`Student` (model)

- Fields: `rollNo`, `name`, `email`, `course`, `marks`.
- `toFileLine()` / `fromFileLine(String)` — convert between object and single-line file format (`roll|name|email|course|marks`).
- `display()` — nicely prints the student details.
- `escape` / `unescape` helpers handle `|` safe storage.

`FileUtil`

- Responsible for file operations on the data file (default `"students.txt"`).
- `ensureFile()` — makes sure the file exists.

- `loadAll()` — reads the file line-by-line and builds a `recordOffsets` list (start byte offset of each line) while creating `Student` objects.
    - Uses `RandomAccessFile` to read lines and capture offsets.
- `saveAll(List<Student>)` — overwrites file with current student list using `BufferedWriter`.
- `printFileAttributes()` — prints `exists`, `readable`, `writable`, `size`, `last modified` using `File`.
- `readRecordAtIndex(int)` — demonstrates random access by seeking to stored offset and reading that specific line.

`StudentManager`

- Holds in-memory collections:
    - `List<Student> students` — preserves order and supports sorting.
    - `Map<Integer, Student> studentMap` — fast lookup by roll no and duplicate check.
- Methods:
    - `load()` — load from file via `FileUtil` into collections.
    - `save()` — delegate to `FileUtil.saveAll`.
    - `addStudent(Student)` — add if roll no not duplicate.
    - `searchByName(String)` — linear search.
    - `deleteByName(String)` — remove with `Iterator`.
    - `displayAll()` — prints via `Iterator`.
    - `sortByMarksDescending()` / `sortByName()` — use `Comparator` / `Collections.sort`.

---

3 — File format & persistence

- **File:** `students.txt` plain text; each line:
- `roll|name|email|course|marks`

  Example:

  ```
  101|Ankit|ankit@mail.com|B.Tech|85.5
  ```

- On startup `load()` reads the file and populates memory.
- On exit option `9` the program writes the current list back to the file — persistent storage across runs.

---

4 — RandomAccessFile demonstration

- While loading, `FileUtil` records the byte offset at the start of each line (via `RandomAccessFile.getFilePointer()` before reading the line).

- Later `readRecordAtIndex(index)` calls `raf.seek(offset)` and `raf.readLine()` to re-read that exact record. This demonstrates random access (useful for fixed-position reads or implementing indexed retrieval).

---

5 — Collections and iteration

- `ArrayList<Student>` stores records for ordering and sorting.
- `HashMap<Integer, Student>` stores records keyed by roll number for O(1) lookup and duplicate detection.
- `displayAll()` uses an `Iterator` to traverse the `ArrayList` (showing usage of Iterator API).
- Sorting uses `students.sort(...)` with `Comparator.comparingDouble(...)` for marks (or name comparator).

---

6 — Menu and user flows

Main loop offers options:

1. Add Student — prompts for fields, constructs `Student`, calls `addStudent`.
2. View All Students — prints every student.
3. Search by Name — linear search; prints result if found.
4. Delete by Name — remove matching students using `Iterator.remove`.
5. Sort by Marks — sort descending and display.
6. Sort by Name — sort ascending and display.
7. Show file attributes — prints file metadata.
8. Read Random Record — picks a random index (if offsets exist) and reads that specific record using `RandomAccessFile`.
9. Save and Exit — writes file and exits.

Input is validated by helper methods (`readInt`, `readDouble`, `readNonEmpty`) that re-prompt on bad input.

---

7 — Error handling

- `try/catch` around parsing and file operations prevents the program from crashing.
- `NumberFormatException` handled when parsing numeric menu choice.
- `IOException` handled for file I/O.
- `IllegalArgumentException` and generic `Exception` have catch-all messages to indicate unexpected errors.

---

8 — Efficiency & complexity

- Loading: O(n) in file size / number of students.
- Lookup by roll number: O(1) average using `HashMap`.
- Search by name: O(n) linear scan.
- Sorting: O(n log n).
- Random access read: O(1) seek + line read (after offsets built).

---

9 — Limitations & possible improvements

- File format is simple text — consider CSV or JSON for structured data and safer parsing.
- `RandomAccessFile.readLine()` uses ISO-8859-1 interpretation; better to manage encoding if non-ASCII needed.
- Current `readRecordAtIndex` depends on offsets built at load; if file is modified externally, offsets may be stale — rebuild offsets when needed.
- Add edit/update student option, duplicate-name handling, email validation, and date-stamping.
- For concurrent access or a GUI, consider synchronization or background save threads.
- For large datasets, use a database or binary index for scalable random access.

---

10 — Quick example run

- Program starts → `load()` prints loaded students.
- Choose `1` (Add student), input fields → student added to memory.
- Choose `5` (Sort by marks) → students printed in descending marks order.
- Choose `9` → `save()` writes file and program exits.