Correction Code

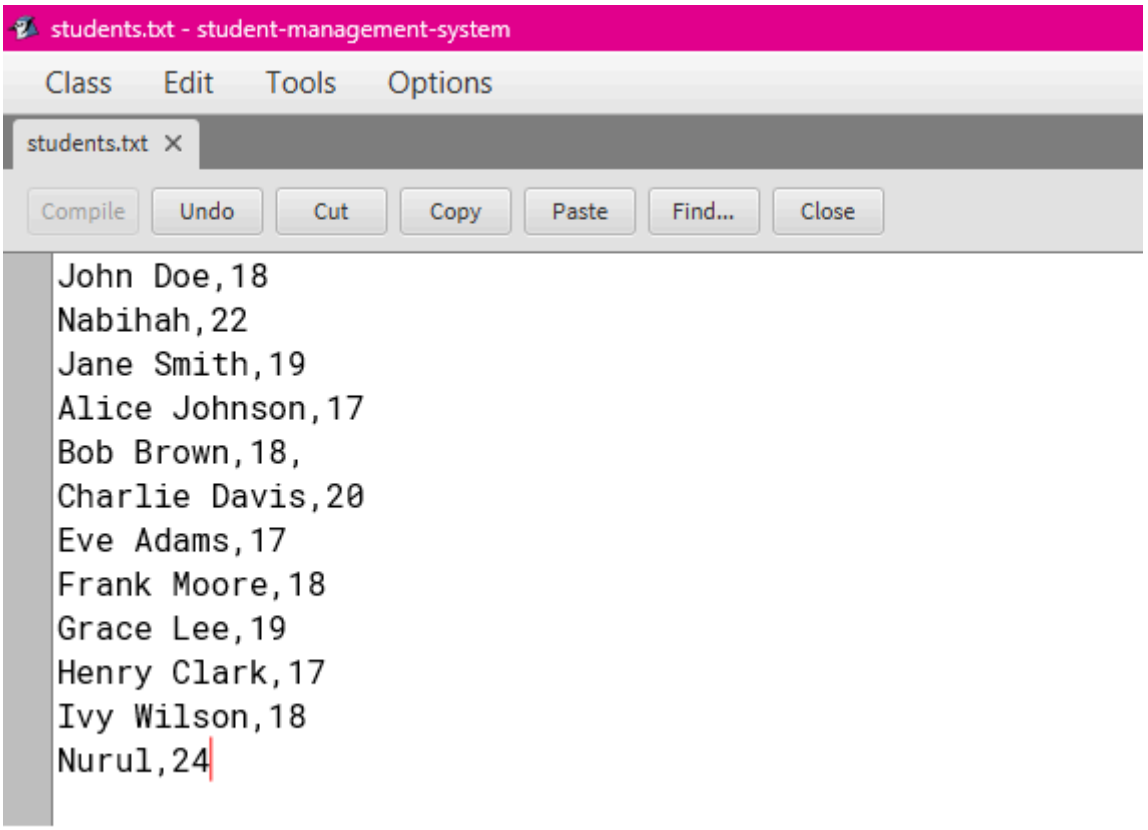
Pick a random number 1-100 for a student score.

```
// Nested class representing a Student
    public static class Student {
        String name; // Student's name
        int age; // Student's age
        int[] scores = new int[5]; // Array to hold scores for 5 subjects
        private double averageScore; // Average score of the student
        private static final Random random = new Random(); // Random number generator

        // Constructor to initialize Student object
        public Student(String name, int age) {
            this.name = name; // Assign name
            this.age = age; // Assign age
            generateRandomScores(); // Generate random scores
            calculateAverage(); // Calculate average score upon creation
            University.totalStudents++; // Increment total student count
        }

        // Constructor to initialize Student object with provided scores
        public Student(String name, int age, int[] scores) {
            this.name = name; // Assign name
            this.age = age; // Assign age
            this.scores = scores; // Assign scores
            calculateAverage(); // Calculate average score upon creation
            University.totalStudents++; // Increment total student count
        }

        // Generate random scores between 1 and 100
        private void generateRandomScores() {
            for (int i = 0; i < scores.length; i++) {
                scores[i] = random.nextInt(100) + 1; // Random number between 1 and 100
            }
        }

        // Method to calculate average score based on the scores array
        public void calculateAverage() {
            int sum = Arrays.stream(scores).sum(); // Calculate sum of scores
            this.averageScore = sum / 5.0; // Calculate average
        }

        // Getter method to retrieve the average score
```

```
    public double getAverageScore() {
        return averageScore; // Return average score
    }

    // Method to print student details to the console
    public void printDetails() {
        String scoresString = Arrays.stream(scores)
                                    .mapToObj(String::valueOf)
                                    .reduce((a, b) -> a + ", " + b)
                                    .orElse("No Scores");

        System.out.printf("Name: %s, Age: %d, Scores: [%s], Average: %.2f%n",
                          name, age, scoresString, averageScore);
    }

    // Method to convert student details to a CSV format string for saving
    public String toDataString() {
        String scoresString = Arrays.stream(scores)
                                    .mapToObj(String::valueOf)
                                    .reduce((a, b) -> a + "," + b)
                                    .orElse("");

        return String.format("%s,%d,%s", name, age, scoresString);
    }
}

// Method to load student data from a file
public void loadStudentData() {
    try (BufferedReader reader = new BufferedReader(new FileReader("students.txt"))) {
        String line;
        while ((line = reader.readLine()) != null) {
            String[] parts = line.split(",");
            if (parts.length < 2) {
                System.out.println("Invalid data format: " + line);
                continue;
            }

            String name = parts[0];
            int age = Integer.parseInt(parts[1]);
            int[] scores;

            if (parts.length >= 7) {
                scores = Arrays.stream(parts, 2, 7).mapToInt(Integer::parseInt).toArray();
            } else {
                // Generate random scores if not provided
                scores = new int[5];
                for (int i = 0; i < 5; i++) {
                    scores[i] = Student.random.nextInt(100) + 1;
```

```
                }
            }

            students[studentCount++] = new Student(name, age, scores);
        }
    } catch (IOException e) {
        System.out.println("Error loading student data: " + e.getMessage());
    }
}

// Method to save student data to a file named "students.txt"
public void saveStudentData() {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter("students.txt"))) {
        for (int i = 0; i < studentCount; i++) {
            writer.write(students[i].toDataString());
            writer.newLine();
        }
        System.out.println("Student data saved successfully.");
    } catch (IOException e) {
        System.out.println("Error saving student data: " + e.getMessage());
    }
}
```

Student.txt.



- Display all student data with 5 random scores.

```
Student Menu:
1. Display All Students
2. Add New Student
3. Sort Students by Average Score
4. Get Top Scoring Student
5. Edit Student Data
6. Back to Main Menu
Choose an option: 1
All Students:
Name: John Doe, Age: 18, Scores: [7, 75, 92, 90, 53], Average: 63.40
Name: Nabihah, Age: 22, Scores: [100, 30, 50, 38, 99], Average: 63.40
Name: Jane Smith, Age: 19, Scores: [26, 5, 4, 8, 57], Average: 20.00
Name: Alice Johnson, Age: 17, Scores: [94, 64, 14, 97, 7], Average: 55.20
Name: Bob Brown, Age: 18, Scores: [88, 66, 65, 98, 59], Average: 75.20
Name: Charlie Davis, Age: 20, Scores: [72, 7, 69, 88, 30], Average: 53.20
Name: Eve Adams, Age: 17, Scores: [5, 96, 14, 91, 35], Average: 48.20
Name: Frank Moore, Age: 18, Scores: [29, 30, 37, 62, 8], Average: 33.20
Name: Grace Lee, Age: 19, Scores: [15, 32, 63, 77, 28], Average: 43.00
Name: Henry Clark, Age: 17, Scores: [84, 100, 44, 65, 25], Average: 63.60
Name: Ivy Wilson, Age: 18, Scores: [92, 63, 56, 78, 19], Average: 61.60
Name: Nurul, Age: 24, Scores: [99, 27, 15, 24, 80], Average: 49.00
```

REFLECTIVE CONCLUSION

Through this assignment I learnt that building a complete student and course management system in Java is not an easy task. Besides practising my code in file handling, data structure manipulation and encapsulation practices, this also led to a usage of the understanding of multi-user input and error handling as well as getting all around persisting data. Difficulty in implementing dynamic features such as generating random student scores and recalculating averages made me realise the importance of writing clean, maintainable code.

Additionally, developing these functionalities stressed the importance of systematic debugging and iterative development. This exercise also honed my skills in detecting logical mistakes and thinking through corner cases that might occur when the code is running for real-world problems. The challenge of changing the code itself to add new features without changing some core functionalities taught us about adaptability and careful structure.

In the future, these same lessons are going to be incredibly useful for individual projects that will need a base coding knowledge as well as building on top of existing systems. The skills obtained here as problem-solving, code structuring, and managing the user interface will certainly be used by me in all upcoming elaborate software projects. The assignment has now confirmed my understanding of how modular design and object-oriented principles can be used to develop scalable and maintainable software solutions.