INFORMATICS INSTITUTE OF TECHNOLOGY

Software Development II

Coursework Report 2023/2024

Student Name : M.G.G.W. Tharaki Dimithri

UOW Number : W2081980

IIT Number : 20232778

Abstract

I have developed a Student Management System tailored for universities. This system, implemented in Java, features a command-line interface for managing student records efficiently. It allows administrators to check available seats, register new students, delete existing ones, and find specific students by their ID. The system also supports storing student details in a CSV file and loading them back, ensuring data persistence. Additionally, administrators can add module marks for students and view the list of students sorted by name. The project focuses on scalability and modularity through object-oriented programming (OOP) principles, providing a robust tool for university administrators to manage student data effectively.

Acknowledgment

I want to extend my heartfelt appreciation to Mr. Dilshad Ahamed, our lecturer and module leader, and Mr.Saadh Jawwadh, and Ms.Sapna Kumarapathirage , our tutorial lecturers whose unwavering support and guidance helped me complete this assignment with excellence. I am also grateful for the prompt response to my queries and his guidance and support was invaluable and greatly appreciated. In addition, I would like to express my gratitude to my colleagues for their constant encouragement and support in completing this assignment.

Task 01 – Source Code

```
import java.util.Scanner;

public class Main {
   public static void main(String[] args) {
      StudentManagementSystem  sms = new StudentManagementSystem (); // make   a new  studen t
management  system
      Scanner scanner = new Scanner(System.in); // get the user input

      while (true) { // menu loop
         System.out.println("Menu:");
         System.out.println("1. Check available seats");
         System.out.println("2. Register student");
         System.out.println("3. Delete student");
         System.out.println("4. Find a student");
         System.out.println("5. Store student details in a CSV file");
         System.out.println("6. Load student details from the CSV file");
         System.out.println("7. View the list of students based on their names");
         System.out.println("8. Add module marks for a student");
         System.out.println("9. Exit");
         System.out.println("C: Generate summary report");
         System.out.println("D: Generate complete report");
         System.out.println("Welcome to Student Management system");
         System.out.println("Please enter your choice here : ");

         String choice = scanner.nextLine().toUpperCase();

         switch (choice) {
            case "1":
               sms.checkAvailableSeats();
               break;
            case "2":
               String id;
               while (true) {
                  System.out.print("Enter student ID: ");
                  id = scanner.nextLine();
                  if (!Student.isValidStudentId(id)) {
                     System.out.println("Invalid Student ID. Please try again.");
                  } else {
```

```
            break;
        }
    }
    System.out.print("Enter student name: ");
    String name = scanner.nextLine();
    sms.registerNewStudent(id, name);
    break;
case "3":
    System.out.print("Enter student ID to delete: ");
    id = scanner.nextLine();
    sms.deleteStudent(id);
    break;
case "4":
    System.out.print("Enter student ID to find: ");
    id = scanner.nextLine();
    sms.findStudent(id);
    break;
case "5":
    sms.storeStudentDetails();
    break;
case "6":
    sms.loadStudentDetails();
    break;
case "7":
    sms.viewStudentsSortedByName();
    break;
case "8":
    System.out.print("Enter student ID to add module marks: ");
    id = scanner.nextLine();
    System.out.print("Enter marks for Module 1: ");
    int marks1 = scanner.nextInt();
    System.out.print("Enter marks for Module 2: ");
    int marks2 = scanner.nextInt();
    System.out.print("Enter marks for Module 3: ");
    int marks3 = scanner.nextInt();
    scanner.nextLine(); // Consume newline
    sms.addModuleMarks(id, marks1, marks2, marks3);
    break;
case "9":
    System.out.println("Exiting...");
    return;
case "C":
```

```
                AddReport summaryReport = new AddReport(sms.getStudents());
                summaryReport.generateSummary();
                break;
            case "D":
                AddReport completeReport = new AddReport(sms.getStudents());
                completeReport.generateCompleteReport();
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
  }
}
```

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

public class StudentManagementSystem {
   private static final int UOW_STUDENTS = 100;
   private static final String FILENAME = "data.csv";
   private List<Student> students;

   public StudentManagementSystem() {
       students  = new  ArrayList <>();
   }

   /**
    * prints number of available seats from this
    */
   public void  checkAvailableSeats () {
      System.out.println("Available seats : " + (UOW_STUDENTS - students.size()));
   }

   /**
    * registers a student if the ID is valid and there are available seats.
    * @param id   The student's ID.
    * @param name The student's name.
    */
   public void  registerNewStudent (String id, String name) {
      if  (!Student.isValidStudentId(id))  {
         System.out.println("Invalid Student ID. Registration failed.");
         return;
      }
      if (students. size() >= UOW_STUDENTS) {
         System.out.println("No  available  seats.");
         return;
      }
      students.add ( new Student(id, name));
      System.out.println("Student registered successfully.");
   }
```

```
/*
 * adds module marks for a student.
 *
 * @param id     The student's ID.
 * @param marks1 Marks for Module 1.
 * @param marks2 Marks for Module 2.
 * @param marks3 Marks for Module 3.
 */
public void addModuleMarks (String id, int marks1, int marks2, int marks3) {
    Student student = findStudentById( id);
    if (student == null) {
        System.out.println ("Student not found.");
    } else {
        student.addModule (marks1, marks2, marks3);
        System.out.println("Module marks added successfully.");
    }
}

/**
 * delete the student by their ID.
 *
 * @param id The student's ID.
 */
public void deleteStudent(String id) {
    students.removeIf (student -> student.getId( ).equals(id));
    System.out.println("Student deleted successfully.");
}

/**
 * find and prints the details of a student by their ID.
 *
 * @param id The student's ID.
 */
public void  findStudent(String id) {
    Student student  = findStudentById (id);
    if (student != null ) {
        System.out.println(student);
    } else {
        System.out.println ("Student not found.");
    }
}
```

```
/**
 * stores the details of all students in a CSV file.
 */
public void storeStudentDetails() {
    try (PrintWriter writer = new PrintWriter(new FileWriter(FILENAME))) {
        for (Student student : students) {
            writer.println(studentToCSVString(student));
        }
        System.out.println("Student details stored successfully.");
    } catch (IOException e) {
        System.err.println("Error storing student details: " + e.getMessage());
    }
}

/**
 * loads student details from a CSV file.
 */
public void loadStudentDetails() {
    File file = new File(FILENAME);
    if (!file.exists()) {
        System.out.println("No data file found. Starting with an empty list.");
        return;
    }

    students.clear();
    try (BufferedReader reader = new BufferedReader(new FileReader(FILENAME))) {
        String line;
        while ((line = reader.readLine()) != null) {
            students.add(studentFromCSVString(line));
        }
        System.out.println("Student  details  loaded  successfully .");
    } catch (IOException e) {
        System.err.println("Error loading student details: " + e.getMessage());
    }
}

/**
 * sorts and prints the list of students by their names.
 */
public void viewStudentsSortedByName() {
    sortStudentsByName(students);
    for (Student student : students) {
```

```
            System.out.println(student);
        }
    }

    /**
     * find a student by their ID.
     *
     * @param id The student's ID.
     * @return The student if found, otherwise null.
     */
    private Student findStudentById(String id) {
        for (Student student : students) {
            if (student.getId().equals(id)) {
                return student;
            }
        }
        return null;
    }

    /**
     * converts a student object to a CSV string.
     *
     * @param student The student.
     * @return the CSV string representing the student.
     */
    private String studentToCSVString(Student student) {
        StringBuilder sb = new StringBuilder();
        sb.append(student.getId()).append(",")
                .append(student.getName()).append(",");
        for (Module module : student.getModules()) {
            sb.append(module.getMarks1()).append(",")
                    .append(module.getMarks2()).append(",")
                    .append(module.getMarks3()).append(",");
        }
        return sb.toString();
    }

    /**
     * creates student object from a CSV string.
     *
     * @param csvLine The CSV string.
     * @return The student.
```

```
     */
    private Student studentFromCSVString(String csvLine) {
        String[] parts = csvLine.split(",");
        String id = parts[0];
        String name = parts[1];
        Student student = new Student(id, name);
        for (int i = 2; i < parts.length; i += 3) {
            int marks1 = Integer.parseInt(parts[i]);
            int marks2 = Integer.parseInt(parts[i + 1]);
            int marks3 = Integer.parseInt(parts[i + 2]);
            student.addModule(marks1, marks2, marks3);
        }
        return student;
    }

    /**
     * sorts a list of students by their names.
     *
     * @param students The list of students.
     */
    private void sortStudentsByName(List<Student> students) {
        students.sort(Comparator.comparing(Student::getName));
    }
}
```

Task 02 – Source Code

```
import java.util.ArrayList;
import java.util.List;

public class Student {
    private String id;
    private String name;
    private List<Module> modules;

    public Student(String id, String name) {
        this.id = id;
        this.name = name;
        this.modules = new ArrayList<>();
    }

    public String getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public List<Module> getModules() {
        return modules;
    }

    /**
     * Adds a module with specified marks to the student's list of modules.
     * @param marks1 Marks for the first module.
     * @param marks2 Marks for the second module.
     * @param marks3 Marks for the third module.
     */

    public void addModule(int marks1, int marks2, int marks3) {
        Module module = new Module(marks1, marks2, marks3);
        modules.add(module);
    }
```

```
    @Override
    public String toString() {
        return "Student{" +
                "id='" + id + '\'' +
                ", name='" + name + '\'' +
                ", modules=" + modules +
                '}';
    }

    /**
     * Validates the student ID.
     * @param studentId The student ID to validate.
     * @return True if the ID is valid (length 8 and starts with 'w'), false otherwise.
     */
    public static boolean isValidStudentId(String studentId) {
        if (studentId == null || studentId.length() != 8) {
            return false;
        }

        // Check if the first character is 'w'
        if (studentId.charAt(0) != 'w') {
            return false;
        }

        // Check if the remaining characters are digits
        for (int i = 1; i < studentId.length(); i++) {
            if (!Character.isDigit(studentId.charAt(i))) {
                return false;
            }
        }

        return true;
    }
}
```

```
public class Module {
    private int marks1;
    private int marks2;
    private int marks3;
    private String grade;

    public Module(int marks1, int marks2, int marks3) {
        this.marks1 = marks1;
        this.marks2 = marks2;
        this.marks3 = marks3;
        calculateGrade(); // Calculate the grade according to creation
    }

    public int getMarks1() {
        return marks1;
    }

    public void setMarks1(int marks1) {
        this.marks1 = marks1;
        calculateGrade();
    }

    public int getMarks2() {
        return marks2;
    }

    public void setMarks2(int marks2) {
        this.marks2 = marks2;
        calculateGrade();
    }

    public int getMarks3() {
        return marks3;
    }

    public void setMarks3(int marks3) {
        this.marks3 = marks3;
        calculateGrade();
    }

    public String getGrade() {
        return grade;
```

```
    }

    private void calculateGrade() {
        double average = (marks1 + marks2 + marks3) / 3.0;

        if (average >= 80) {
            grade = "Distinction";
        } else if (average >= 70) {
            grade = "Merit";
        } else if (average >= 40) {
            grade = "Pass";
        } else {
            grade = "Fail";
        }
    }
    // Private method to calculate the grade based on average marks
    @Override
    public String toString() {
        return "Module{" +
                "marks1=" + marks1 +
                ", marks2=" + marks2 +
                ", marks3=" + marks3 +
                ", grade='" + grade + '\'' +
                '}';
    }
}
```

Task 03 – Source Code

```
import java.util.*;

public class AddReport {

   private List<Student> students;

   // initialize with the list of students
   public AddReport(List<Student> students) {
      this.students = students;
   }

   // generates a summary report with total registrations and students scoring above 40 in each module
   public void generateSummary() {
      int totalRegistrations = students.size(); // get total number of students
      // count students scoring above 40 in Module 1
      long studentsAbove40Module1 = students.stream().filter(s -> s.getModules().stream().anyMatch(m
-> m.getMarks1() > 40)).count();
      long studentsAbove40Module2 = students.stream().filter(s -> s.getModules().stream().anyMatch(m
-> m.getMarks2() > 40)).count();
      long studentsAbove40Module3 = students.stream().filter(s -> s.getModules().stream().anyMatch(m
-> m.getMarks3() > 40)).count();

      // Print out the summary
      System.out.println("Total student registrations: " + totalRegistrations);
      System.out.println("Total students scoring more than 40 marks in Module 1: " +
studentsAbove40Module1);
      System.out.println("Total students scoring more than 40 marks in Module 2: " +
studentsAbove40Module2);
      System.out.println("Total students scoring more than 40 marks in Module 3: " +
studentsAbove40Module3);
   }

   //  a complete report with detailed student information
   public void generateCompleteReport() {
      List<StudentReport> studentReports = new ArrayList<>();
      for (Student student : students) {
```

```
            for (Module module : student.getModules()) {
                int total = module.getMarks1() + module.getMarks2() + module.getMarks3(); // calculate total
marks
                double average = total / 3.0; // calculate average marks
                // create a new StudentReport and add it to the list
                studentReports.add(new StudentReport(student.getId(), student.getName(),
module.getMarks1(), module.getMarks2(), module.getMarks3(), total, average, module.getGrade()));
            }
        }

        // sort student reports by average marks in descending order
        studentReports.sort(Comparator.comparingDouble(StudentReport::getAverage).reversed());

        // print out each student's report
        for (StudentReport report : studentReports) {
            System.out.println(report);
        }
    }

    // Inner class to represent a student's report
    private class StudentReport {
        private String id;
        private String name;
        private int marks1;
        private int marks2;
        private int marks3;
        private int total;
        private double average;
        private String grade;

        //  initialize the report details
        public StudentReport(String id, String name, int marks1, int marks2, int marks3, int total, double
average, String grade) {
            this.id = id;
            this.name = name;
            this.marks1 = marks1;
            this.marks2 = marks2;
            this.marks3 = marks3;
            this.total = total;
            this.average = average;
            this.grade = grade;
        }
```

```
    // Getter for average marks
    public double getAverage() {
        return average;
    }

    // toString method to format the report details for printing
    @Override
    public String toString() {
        return "Student ID: " + id + ", Name: " + name + ", Marks1: " + marks1 + ", Marks2: " + marks2 + ",
Marks3: " + marks3 +
                ", Total: " + total + ", Average: " + average + ", Grade: " + grade;
    }
  }
}
```

Task 04 – Testing

Test Case	Expected Result	Actual Result	Pass/Fail
(Run the program) Display the main menu	When run the program, the main menu pops up right away.	Main menu displays properly	Pass
Check whether there were available seats to register	When user input '1' as choice Displays available Seats	Displays available Seats	Pass
Register Student	Type '2' to sign up a student with their ID.	Able to Register a student with ID	Pass
Register Student with Invalid ID	Enter Invalid ID without in 8 digit range and starting from "w", Display "Invalid student Id Enter again : "	Display Invalid Student ID. Please try again.	Pass
Delete student	Type "3" to delete, checked the latest list by pressing "7", listed without deleted student	Press "3" to delete, checked the latest list by pressing "7", listed without deleted student	Pass
Find Student with ID	Type '4' to look up a student by their ID. Finds the right person every time.	Display the correct person as per ID	Pass
Find Student with Invalid ID	Type '4' enter invalid Id , and should display student not found	Displays "student not found"	Pass
Store student details into a file	Press '5' to save all the data to a text file.	Stored the data to the text file properly displays "Student details stored successfully"	Pass
Verify that storing a list with one student writes the correct data to the file.	The file should contain one line with the student's data.	The file contains one line with the student's data.	Pass
Load student details from the file to the system	Press '6' to load the data back from the file. Shows all the saved info perfectly.	Display the correct data which were stored previously	Pass

View the list of students based on their names	Enter ‘7' to see a list of all registered students. Shows everyone's info.	Displays entire data of the registered students	Pass
Register Student	Type '2' to sign up a student with their ID and name.	should sign up a student with their ID and name. But, if user put in a full name without underscores, it throws an error.	Fail
Add student name and module marks to the system	Press ‘8’ to User can update their marks and user data	update students' marks and info. Works as usual.	Pass
Generate a summary of the system	Press ‘C’ to see how many students scored more than 40. Gives you the total number.	Generated total number of students who scored more than 40	Pass
Generate complete report with list of students	Press 'D'' to get a detailed report of all students. Successfully creates the report.	Successfully Generated Full report of the system	Pass
Exit	Press '0' to exit the program safely. Closes without any problems.	Safely exiting from the system	Pass

Task 04 – Testing - Discussion

To guarantee thorough coverage of the Student Management System's functionality, a series of test cases were developed to validate various scenarios, encompassing both typical and edge cases.

- ***Checking Available Seats:*** *Tests were performed with no vacant seats, partially filled seats, etc., to verify the system's accuracy in counting available seats.*
- ***Registering Students:*** *Tested by registering students at different stages, with valid IDs, names, and module marks to ensure proper registration.*
- ***Deleting Students:*** *Various scenarios were tested, including deleting students from different positions (beginning, middle, end) and attempting to delete non-existent students to confirm the list updates correctly. These tests were aimed at verifying the system's robustness in managing deletions.*
- ***Searching for Students:*** *Covered searches for both valid and invalid student IDs, ensuring expected retrieval outcomes and correct error handling.*
- ***Storing Student Details:*** *Tested by saving records for different numbers of students, followed by validating the contents of the generated file to ensure accuracy.*
- ***Loading Student Details:*** *Conducted tests with files that were empty, correctly formatted, and incorrectly formatted to enhance the system's file processing capabilities.*

Class Implementation vs. Array Implementation

Class Implementation:

- ***Readability and Understandability:*** *The class-based implementation is more readable and understandable. By storing student data in the Student class, it is clear what operations are being performed on user records, and the methods become more modular.*
- ***Data Structure Modifiability:*** *The class-based structure is easily modifiable. If additional information is needed from the Student class, it can be added without affecting the rest of the system.*
- ***Scalability:*** *The class-based approach is more scalable, allowing for better management and extension of functionality in larger systems. This results in cleaner, more maintainable code compared to a pure array-based approach, which can become cumbersome and unmanageable as the system grows.*

Self-Evaluation form

Criterion	Maximum Mark	Student Mark	Tutor Mark	Student Comment
Check available seats				Fully implemented and working. Checks the number of available seats accurately based on the student limit.
Register student				Fully implemented and working. Registers new students and handles input validation.
Delete student				Fully implemented and working. Deletes students based on ID and updates the list correctly.
Find student				Fully implemented and working. Finds and displays student details based on ID.
Store student details to file				Fully implemented and working. Stores student details into a file accurately.
Load student details from file				Fully implemented and working. Loads student details from a file correctly, including handling file errors.
View students by name				Fully implemented and working. Displays a sorted list of students by name.
Add student name and module marks				Fully implemented and working. Updates student names and module marks accurately.
	24	23		
Student class works correctly	**14**	13		Fully implemented and working
Module class works correctly	**10**	8		Fully implemented and working
Sub menu (A and B works well)	**6**	5		Fully implemented and working
Menu works correctly	**6**	5		Fully implemented and working. Displays properly
Generate summary	**7**	5		Fully implemented and working. Generates a summary of the student performance accurately.
Generate complete report	**10**	6		Fully implemented and working. Generates a complete report with accurate totals, averages, and grades.
Implementation of Bubble sort	**3**	2		Fully implemented and working. Displays properly
Test case coverage and reasons	**6**	4		Fully implemented. Coverage properly

Writeup on which version is better and why	**4**	2		The class-based implementation is better because it offers improved readability, easier maintainability, and better scalability compared to the array-based implementation.
Coding Style (Comments, indentation, style)	**7**	5		Fully implemented. Code is well-organized and clear, with appropriate use of comments for readability.
self-evaluation	**3**	2		Overall the program is fully implemented.it runs properly as expected
Criterion	**Maximum Mark**	**Student Mark**	**Tutor Mark**	**Student Comment**
Overall functionality	**100**	80		Fully implemented and working. The system meets all specified requirements and performs all functionalities correctly.

References

- **https://www.w3schools.com/java/java_files_create.asp**
- https://stackoverflow.com/
- https://www.geeksforgeeks.org/java /