INFORMATICS INSTITUTE OF TECHNOLOGY

Software Development II

Coursework Report 2023/2024

Geethaka Sankalpa Karunathilaka

UoW Number: w2084412

IIT Number: 20231659

Table of Contents

Task 01 – Source Code

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.*;

/**
 * Student Management System is a system for managing student activities
including their results.
 * It provides features for checking available seats, registering students,
deleting students, finding students, storing student details,
 * loading student details, and viewing the list of students.
 */
public class StudentManagementSystem {
    private static int seats = 100;
    private static int studentCount = 0;
    private static Student[] students = new Student[seats];
    private static boolean detailsLoaded = false;
    private static char[] specialChars = {'!', '@', '#', '$', '%', '^', '&',
'*', '(', ')', '{', '}', '+', '=', '-', '/', '.', ',', '?', '|', '\\', '"',
'\'', '`', '~', '>', '<', ':', ';'};

    public static void main(String[] args) {
        mainMenu();
    }

    /**
     * Displays the main Menu of the Student Management System.
     * Prompts user to enter an option and navigates to relevant function
based on the option.
     * Load student details first to enable other operations.
     */
    public static void mainMenu() {
        Scanner scan = new Scanner(System.in);
        while (true) {
            System.out.println("------------Student Activity Management
System----------- \n");
            System.out.println("""
                    1. Check available seats
                    2. Register student (with ID)
                    3. Delete student
                    4. Find student (with student ID)
                    5. Store student details into a file
                    6. Load student details from the file to the system
                    7. View the list of students based on their names
                    8. Generate Report
                    0. Exit""");
            try {
                System.out.print("\nSelect your choice : ");
                String strChoice = scan.nextLine();
                if (strChoice.trim().isBlank()) {   // checks an empty input
                    System.err.print("\n" + "Input cannot be an empty
value");
                    continue;
```

```
                }
                int choice = Integer.parseInt(strChoice);    // converts
string input to integer input
                switch (choice) {
                    case 1:
                        checkAvailableSeats();
                        break;

                    case 6:
                        if (!detailsLoaded) {     // checks if details are
loaded
                            loadDetails();
                            detailsLoaded = true;

                            break;
                        } else {     // executes when details are loaded
                            System.err.println("The Data already exists ");
                            System.out.println('\n');
                        }
                        break;

                    case 0:
                        System.out.println("\nExiting Program.....");
                        System.out.println("Good Bye !!!");
                        return;

                    default:    // handles other options
                        if (!detailsLoaded) {    // checks student details
are loaded initially.
                            System.err.println("Load Details First To Perform
Other Functions.");
                            System.out.println('\n');
                        } else {
                            switch (choice) {

                                case 2:
                                    registerStudents();
                                    break;

                                case 3:
                                    deleteStudent();
                                    break;

                                case 4:
                                    findStudent();
                                    break;

                                case 5:
                                    storeDetails();
                                    break;

                                case 7:
                                    viewStudentsByName(students);
                                    break;
                                case 8:
                                    generateReport();
                                    break;
```

```
                                default:
                                    System.err.println("Input Out of range");
                                    System.out.println();
                            }
                        }
                    }
                } catch (Exception e) {
                    System.err.println("\n" + "Please provide a valid Input.");
                }
            }
        }

        /**
         * Checks the availability of seats in the system
         * Number of available seats are initially declared to 100.
         */
        public static void checkAvailableSeats() {
            System.out.println("------Check Available Seats--------");
            System.out.println("-------------------------------------");
            if (seats <= studentCount) {
                System.out.println("All seats are Full");
            } else {
                System.out.println("Number of available seats : " + (seats -
studentCount) + "\n");
            }
        }

        /**
         * Validates a String input to ensure it does not contain special
characters.
         *
         * @param getInput     the input string to validate.
         * @param specialChars the array which contains special characters to
check against the input.
         * @return true if the input contains special characters, false
otherwise.
         */
        public static boolean validateSpecial(String getInput, char[]
specialChars) {
            for (char special : specialChars) {
                if (getInput.indexOf(special) != -1) {
                    return true;
                }
            }
            return false;
        }

        /**
         * Prompts the user to input a student name and validates the input to
ensure it does not contain special characters or numbers.
         *
         * @param specialChars the array which contains special characters to
check against the input.
         * @return the validated student name.
         */
        public static String inputName(char[] specialChars) {
```

```
        Scanner scan = new Scanner(System.in);
        String name;
        while (true) {
            System.out.println("Enter Student Name: ");
            name = scan.nextLine().strip();
            if (name.isBlank()) {       // checks for empty inputs
                System.err.println("\nInput cannot be empty.");
            } else if (validateSpecial(name, specialChars)) {   // checks for
special characters in name
                System.err.println("Input Contains Special Characters. \n");
            } else if (name.matches(".*\\d.*")) { // checks for numbers in
name
                System.err.println("Input Contains Numbers. ");
            } else {
                break;
            }
        }
        return name;
    }

    /**
     * Prompts the user to input the student ID and validates the input.
     * Student ID should be 8-characters long.
     * Student ID should start with 'w' followed by 7 numbers.
     *
     * @param specialChars the array which contains special characters to
check against the input.
     * @return the validated Student ID.
     */
    public static String inputId(char[] specialChars) {

        Scanner scan = new Scanner(System.in);
        String id;

        while (true) {
            System.out.println("Enter Student ID : ");
            id = scan.nextLine().strip();

            if (id.isBlank()) {
                System.err.println("Input cannot be empty.\n");
            } else if (id.length() != 8) {
                System.err.println("Student ID must be 8-character long.
\n");
            } else if (validateSpecial(id, specialChars)) {
                System.err.println("Input Contains Special Characters. \n");
            } else if (id.charAt(0) != 'w') {     // checks for first
character of ID
                System.err.println("Student ID must start with 'w'. \n");
            } else {
                boolean numberExist = false;
                for (int i = 1; i < id.length(); i++) {
                    if (Character.isDigit(id.charAt(i))) {    // checks if
the remaining characters contains numbers.
                        numberExist = true;
                        break;
                    }
```

```
                }
                if (!numberExist) {
                    System.err.println("Student ID must contains atleast one
number(besides the first letter)\n");
                } else {
                    break;
                }
            }
        }
        return id;
    }

    /**
     * Validates the user provided mark to ensure that it is within the range
of 0 to 100.
     *
     * @param mark the user provided mark to validate.
     * @return true if the mark is within the range of 0 to 100, false
otherwise.
     */
    public static boolean validateMark(double mark) {
        if (mark >= 0 && mark <= 100) {
            return true;
        }
        return false;
    }

    /**
     * Prompts user to enter student marks for three modules and validates
the input.
     *
     * @return an array of validated marks.
     */
    public static double[] getMarks() {
        Scanner scan = new Scanner(System.in);
        double[] marks = new double[3];
        for (int i = 0; i < marks.length; i++) {
            double temp;
            while (true) {
                System.out.print("Enter Mark " + (i + 1) + " :");
                System.out.println();
                try {
                    temp = scan.nextInt();
                    if (validateMark(temp)) {   // checks whether user
entered marks within the range
                        marks[i] = temp;
                        break;
                    } else {
                        System.err.println("Mark " + (i + 1) + " is out of
range. Please Enter a valid Mark between 0 and 100. ");
                    }
                } catch (InputMismatchException e) {
                    System.err.println("Invalid Input.");
                    scan.next(); // consume invalid input
                }
            }
        }
```

```
        return marks;
    }

    /**
     * Takes Student Name with ID.
     * Takes the 3 marks for 3 modules.
     * Registers a new student in the system with Student Name, ID and Marks
for 3 modules.
     */
    public static void registerStudents() {
        String name;
        String id;

        if (seats <= studentCount) {
            System.out.println("All Seats are Full. ");
        } else {
            name = inputName(specialChars);
            id = inputId(specialChars);

            double[] marks = getMarks();
            students[studentCount] = new Student(name, id, marks[0],
marks[1], marks[2]);
            studentCount++;     // increment number of students
        }
    }

    /**
     * Deletes a student from the system based on the user entered ID.
     */
    public static void deleteStudent() {

        System.out.println("---------------Delete Student Here-----------
\n");
        String deleteId = inputId(specialChars);    // gets user Id
        boolean idFound = false;
        for (int i = 0; i < students.length; i++) {
            if (students[i] != null && students[i].getId().equals(deleteId))
{  // finds student in student array with the given ID.
                students[i] = null;
                idFound = true;
                break;
            }
        }
        System.out.println("\nStudent Deleted\n");
        if (!idFound) {
            System.err.println("Student with the ID : " + deleteId + " is not
existing.\n");
        }
    }

    /**
     * Finds a student in the system based on their ID.
     * Displays Student Details correspond to the entered ID.
     */
    public static void findStudent() {

        System.out.println("-----------Find Student With Student ID----------
```

```
--\n");
        String findId = inputId(specialChars);  // gets student ID

        boolean found = false;
        System.out.println("Student Relevant to " + findId + " has been found
! \n");
        for (Student student : students) {
            if (student != null && student.getId().equals(findId)) { // finds
student in student array with the given ID.
                System.out.println("----------------------------------------
");
                System.out.println("Student Name            : " +
student.getName());
                System.out.println("Student ID              : " +
student.getId());
                System.out.println("Marks for First Module  : " +
student.getModule().getMark1());
                System.out.println("Marks for Second Module : " +
student.getModule().getMark2());
                System.out.println("Marks for Third Module  : " +
student.getModule().getMark3());
                System.out.println("----------------------------------------
");
                found = true;
                break;
            }
        }
        if (!found) {
            System.err.println("Student with the ID : " + findId + " does not
exist.");
        }
    }

    /**
     * Stores the student details from the system to a text file.
     * Student details includes Student Name, ID and Marks of 3 modules.
     */
    public static void storeDetails() {
        try {
            File file = new
File("F:\\Java\\SD_2\\src\\Coursework\\Details.txt");
            if (file.createNewFile()) {
                System.out.println("File created: " + file.getName());
            } else {
                System.out.println("File already exists in the system.");
            }
        } catch (IOException e) {
            System.err.println("An error occurred.");
            e.printStackTrace();
        }

        try {
            FileWriter writer = new
FileWriter("F:\\Java\\SD_2\\src\\Coursework\\Details.txt");
            for (Student student : students) {
                if (student != null) { // Check if the student object is not
null
```

```
                    writer.write(student.getName() + ", " + student.getId() +
", " + student.getModule().getMark1() +
                            ", " + student.getModule().getMark2() + ", " +
student.getModule().getMark3() + "\n");
                }
            }
            writer.close();
            System.out.println("\nStudent Details have been successfully
added to the file.\n");
        } catch (IOException e) {
            System.err.println("An error occurred.");
            e.printStackTrace();
        }
    }

    /**
     * Loads stored student details from the text file into the system.
     * Five student details expected in a specific format : "name, id, mark1,
mark2, mark3"
     */
    public static void loadDetails() {
        int studentIndex = 0;
        try {
            File file = new
File("F:\\Java\\SD_2\\src\\Coursework\\Details.txt");
            Scanner reader = new Scanner(file);

            while (reader.hasNextLine()) {
                String[] details = reader.nextLine().split(", ");

                if (details.length < 5) {   // skip lines with invalid format
                    System.out.println("Invalid line format: " +
Arrays.toString(details));
                    continue;
                }
                String loadName = details[0];
                String loadId = details[1];
                double loadMark1 = Double.parseDouble(details[2]);
                double loadMark2 = Double.parseDouble(details[3]);
                double loadMark3 = Double.parseDouble(details[4]);

                Student loadStudent = new Student(loadName, loadId,
loadMark1, loadMark2, loadMark3);

                if (studentIndex < seats - 1) {   // since studentCount
starts from zero, seats-1 to get a range from 0-99
                    students[studentIndex] = loadStudent;
                    studentIndex++;
                } else {
                    System.out.println("No more seats available.");
                    break;
                }
            }
            System.out.println("\nStudent Details Loaded to the System
Successfully!");
        } catch (FileNotFoundException e) {
            System.out.println("An error occurred while loading details from
```

```
file.");
            e.printStackTrace();
        }
        studentCount = studentIndex;
    }

    /**
     * Views the students list in alphabetical order by name.
     *
     * @param students the array of students to view.
     */
    public static void viewStudentsByName(Student[] students) {
        System.out.println("------------------View Students By Name----------
-----\n");

        for (int i = 0; i < studentCount - 1; i++) { // bubble sort by name
            for (int j = 0; j < studentCount - i - 1; j++) {
                if (students[j].getName().charAt(0) > students[j +
1].getName().charAt(0)) {
                    Student temp = students[j];
                    students[j] = students[j + 1];
                    students[j + 1] = temp;
                }

            }
        }
        for (int count = 0; count < students.length; count++) {    //
displays sorted array.
            Student student = students[count];
            if (student != null) {
                System.out.println((count + 1) + ". " + student.getName());
            }
        }
    }

    /**
     * Sorts the array of students highest to lowest based on their average
marks.
     *
     * @param students the array of students to be sorted.
     */
    public static void sortStudentsByAverage(Student[] students) {
        Student temp;   // holds student during swapping

        for (int a = 0; a < studentCount - 1; a++) { // access every object
            for (int b = 0; b < studentCount - a - 1; b++) {    // compare
average

                // calculating total and average for comparison
                double total1 = students[b].getModule().getMark1() +
                        students[b].getModule().getMark2() +
                        students[b].getModule().getMark3();
                double average1 = total1 / 3;

                double total2 = students[b + 1].getModule().getMark1() +
                        students[b + 1].getModule().getMark2() +
                        students[b + 1].getModule().getMark3();
```

```
                double average2 = total2 / 3;

                if (average1 < average2) {  // swaps students if the average
of next student is higher
                    temp = students[b];
                    students[b] = students[b + 1];
                    students[b + 1] = temp;
                }
            }
        }
    }

    /**
     * Generates a report.
     * option 1 - summary report of student registrations and total number of
students who scored more than 40 for all modules
     * option 2 - detailed report including student details, module marks,
total marks, average and grade based on average.
     * option 0 - back to main menu
     */
    public static void generateReport() {
        int option;
        Scanner scan = new Scanner(System.in);
        if (seats <= studentCount) {
            System.out.println("All Seats are Full. ");
        } else {
            while (true) {
                System.out.println("\n---------Generate Report---------");
                System.out.println("""
                        1. GENERATE SUMMARY.
                        2. GENERATE COMPLETE REPORT.
                        0. RETURN TO MAIN MENU.""");

                while (true) {
                    System.out.println("\n Choose your option : ");
                    try {
                        option = scan.nextInt();
                        break;
                    } catch (InputMismatchException e) {
                        System.err.println("Invalid Option");
                        scan.next();
                    }
                }
                switch (option) {
                    case 1:     // summary report
                        int gradeCount1 = 0;
                        int gradeCount2 = 0;
                        int gradeCount3 = 0;
                        System.out.println("----------------------------------
-----Summary-------------------------------------\n");
                        System.out.println("Total Student Registrations : " +
studentCount + " registrations.");
                        for (Student student : students) {
                            if (student != null) {
                                if (student.getModule().getMark1() >= 40) {
                                    gradeCount1++;
                                }
```

```
                                if (student.getModule().getMark2() >= 40) {
                                    gradeCount2++;
                                }
                                if (student.getModule().getMark3() >= 40) {
                                    gradeCount3++;
                                }

                            }
                        }
                        System.out.println("No of students who scored more
than 40 marks in Module 1        : " + gradeCount1 + " students.");
                        System.out.println("No of students who scored more
than 40 marks in Module 2        : " + gradeCount2 + " students.");
                        System.out.println("No of students who scored more
than 40 marks in Module 3        : " + gradeCount3 + " students.");
                        System.out.println("--------------------------------
------------------------------------------------------------");
                        System.out.println("Total No of students who scored
more than 40 marks in 3 Modules : " + (gradeCount1 + gradeCount2 +
gradeCount3) + " students.");
                        break;

                    case 2:     // detailed report
                        double total;
                        double average;
                        sortStudentsByAverage(students);
                        System.out.println("---------------Complete Student
Report---------------\n");
                        for (Student student : students) {
                            if (student != null) {
                                System.out.println("Student ID     : " +
student.getId());
                                System.out.println("Student Name   : " +
student.getName());
                                System.out.println("Module 1 marks : " +
student.getModule().getMark1());
                                System.out.println("Module 2 marks : " +
student.getModule().getMark2());
                                System.out.println("Module 3 marks : " +
student.getModule().getMark3());
                                total = student.getModule().getMark1() +
student.getModule().getMark2() + student.getModule().getMark3();
                                System.out.println("Total          : " +
total);
                                average = total / 3;
                                System.out.println("Average        : " +
average);
                                System.out.println("Grade          : " +
student.getModule().Grade(average));

System.out.println("_____________________________________________");
                            }
                        }
                        break;
                    case 0:
                        return;
                    default:
```

```
                        System.err.println("Input out of range.\n");
                    }

                }
            }
        }
}
```

Task 02 – Source Code

Student Class

```
/**
 * Represents the Student in the Student Management System, which contains
Student name, Student ID, and module marks.
 */
public class Student {
    private String name;
    private String id;
    private Module module;

    /**
     * Constructs a new Student object with the given name,ID, and marks.
     * Creates a new module instance.
     *
     * @param name the student's name.
     * @param id the student's id.
     * @param mark1 marks of the first module
     * @param mark2 marks of the second module
     * @param mark3 marks of the third module
     */
    public Student(String name, String id, double mark1, double mark2, double
mark3) {
        this.name = name;
        this.id = id;
        // creates a new module instance for the module variable
        this.module = new Module(mark1, mark2, mark3);
    }

    /**
     * Gets the student's name.
     *
     * @return the student's name.
     */
    public String getName() {
        return name;
    }

    /**
     * Gets the student's id.
     *
     * @return the student's id.
     */
    public String getId() {
        return id;
    }

    /**
     * Gets the student's module marks.
     * Returning the instance of the module created for the student.
     *
     * @return the student's module marks.
     */
    public Module getModule(){
```

```
        return module;
    }
}
```

Module Class

```
/**
 * Represents a module in the Student Management System which contains marks
for three permanent modules.
 */
class Module {
    private double mark1;
    private double mark2;
    private double mark3;

    /**
     * Creates a parameterised constructor
     *
     * @param mark1 marks of the first module
     * @param mark2 marks of the second module
     * @param mark3 marks of the third module
     */
    public Module(double mark1, double mark2, double mark3) {
        this.mark1 = mark1;
        this.mark2 = mark2;
        this.mark3 = mark3;
    }


    /**
     * Gets the mark for the first module.
     *
     * @return the mark of the first module.
     */
    public double getMark1() {
        return mark1;
    }

    /**
     * Gets the mark for the second module.
     *
     * @return the mark of the second module.
     */
    public double getMark2() {
        return mark2;
    }

    /**
     * Gets the mark for the third module.
     *
     * @return the mark of the third module.
     */
    public double getMark3() {
        return mark3;
```

```
    }

    /**
     * Calculate the Grade based on the average of each three marks for three
modules.
     *
     * @param average the average mark calculated based on the user prompted
marks.
     * @return the grade (Average >= 80 - Distinction, >=70 - Merit, >= 40 -
Pass Else Fail)
     */
    public String Grade(double average){
        String grade;
        if(average >= 80 && average <= 100){
            grade = "Distinction";
        } else if (average >= 70 && average <= 80){
            grade = "Merit";
        } else if (average >= 40 && average <= 70){
            grade = "Pass";
        } else {
            grade = "Fail";
        } return grade;
    }
}
```

Task 03 – Source Code

```
/**
 * Generates a report.
 * option 1 - summary report of student registrations and total number of
students who scored more than 40 for all modules
 * option 2 - detailed report including student details, module marks, total
marks, average and grade based on average.
 * option 0 - back to main menu
 */
public static void generateReport() {
    int option;
    Scanner scan = new Scanner(System.in);
    if (seats <= studentCount) {
        System.out.println("All Seats are Full. ");
    } else {
        while (true) {
            System.out.println("\n---------Generate Report---------");
            System.out.println("""
                    1. GENERATE SUMMARY.
                    2. GENERATE COMPLETE REPORT.
                    0. RETURN TO MAIN MENU.""");

            while (true) {
                System.out.println("\n Choose your option : ");
                try {
                    option = scan.nextInt();
                    break;
                } catch (InputMismatchException e) {
                    System.err.println("Invalid Option");
                    scan.next();
                }
            }
            switch (option) {
                case 1:     // summary report
                    int gradeCount1 = 0;
                    int gradeCount2 = 0;
                    int gradeCount3 = 0;
                    System.out.println("-------------------------------------
-Summary-----------------------------------\n");
                    System.out.println("Total Student Registrations : " +
studentCount + " registrations.");
                    for (Student student : students) {
                        if (student != null) {
                            if (student.getModule().getMark1() >= 40) {
                                gradeCount1++;
                            }
                            if (student.getModule().getMark2() >= 40) {
                                gradeCount2++;
                            }
                            if (student.getModule().getMark3() >= 40) {
                                gradeCount3++;
                            }

                        }
                    }
```

```
                    System.out.println("No of students who scored more than
40 marks in Module 1        : " + gradeCount1 + " students.");
                    System.out.println("No of students who scored more than
40 marks in Module 2        : " + gradeCount2 + " students.");
                    System.out.println("No of students who scored more than
40 marks in Module 3        : " + gradeCount3 + " students.");
                    System.out.println("-----------------------------------
-----------------------------------------------------");
                    System.out.println("Total No of students who scored more
than 40 marks in 3 Modules : " + (gradeCount1 + gradeCount2 + gradeCount3) +
" students.");
                    break;

                case 2:     // detailed report
                    double total;
                    double average;
                    sortStudentsByAverage(students);
                    System.out.println("---------------Complete Student
Report---------------\n");
                    for (Student student : students) {
                        if (student != null) {
                            System.out.println("Student ID     : " +
student.getId());
                            System.out.println("Student Name   : " +
student.getName());
                            System.out.println("Module 1 marks : " +
student.getModule().getMark1());
                            System.out.println("Module 2 marks : " +
student.getModule().getMark2());
                            System.out.println("Module 3 marks : " +
student.getModule().getMark3());
                            total = student.getModule().getMark1() +
student.getModule().getMark2() + student.getModule().getMark3();
                            System.out.println("Total          : " + total);
                            average = total / 3;
                            System.out.println("Average        : " +
average);
                            System.out.println("Grade          : " +
student.getModule().Grade(average));

System.out.println("_____________________________________________");
                        }
                    }
                    break;
                case 0:
                    return;
                default:
                    System.err.println("Input out of range.\n");
            }

        }
    }
}
```

Task 04 – Testing

Test Case		Expected Result	Actual Result	Pass/ Fail
Load Student Details Test Cases		Student details are expected to be loaded into the system after selecting option 6 from the main menu and notify the user about the successful data load	Student details loaded successfully and user was notified.	Pass
01.	Load Student Details from the file initially to perform any other operations.			
Main Menu Test Cases		The system is expected to display an error message to user to load the student details initially.	Displayed an error message indicating to load student details initially	Pass
01.	Selecting other options before loading student details initially.			
02.	Check an Invalid Input for the menu.	The system is expected to display an error message and prompt the user to enter a valid option.	Error message displayed and user was prompted to enter a valid option.	Pass
Find Available Seats Test Cases		The system is expected to return the total number of seats, as student details have not been loaded.	System displayed the available seats.	Pass
01.	Find available seats before loading student details.			
02.	Find available seats after data load.	The system is expected to return the number of available seats, as student details have been loaded.	System displayed the updated available seats as expected.	Pass
Register Student with ID Test Cases		The system is expected to register a student accurately.	Student registered successfully and user was notified.	Pass
01.	Register a student			
02.	Provide Invalid Input for Student Name.	The system is expected to display an error message indicating the name input is	Error message displayed for invalid student name as expected and	Pass

		invalid and prompt the user to enter a valid name.	prompted user to enter a valid name.	
03.	Provide Invalid Input for Student ID.	The system is expected to display an error message indicating the ID input is invalid and prompt the user to enter a valid student ID.	Error message displayed for invalid student ID and prompted user to enter a valid student ID.	Pass
04.	Provide Student ID which has the length more than the specified length.	The system is expected to display an error message indicating the ID input exceed the specified length and prompt the user to enter a valid ID.	Error message was displayed indicating the user about Student ID length exceeding limit.	Pass
05.	Provide Student ID which has the length less than the specified length.	The system is expected to display an error message indicating the ID input is shorter than the specified length and prompt the user to enter a valid ID.	Error message displayed indicating Student ID length is below the limit.	Pass
06.	Provide Student ID with an incorrect format.	The system is expected to display an error message indicating the ID input format is incorrect and prompt the user to enter a valid ID.	User was notified with an error message indicating that the entered Student ID is in incorrect format.	Pass
07.	Provide Invalid Input for Marks.	The system is expected to display an error message indicating the entered marks are invalid and prompt the user to enter a valid marks.	Error message was displayed for invalid marks input as expected.	Pass

08.	Provide numeral input for marks that are beyond the marks range from zero to hundred.	The system is expected to display an error message indicating the entered marks are out of range and prompt the user to enter marks within the range.	Error message was displayed for marks out of range and prompted user to enter marks within range.	Pass
09.	Find available seats after registration.	After a student is registered, displaying the updated number of available seats is expected.	Available seats updated and displayed to user correctly.	Pass
Delete Student Test Cases		Successful removal of student details based on the user entered Student ID is expected from the system.	Student details removed correctly and user was notified.	Pass
01.	Remove Student Details Correctly.			
02.	Finds Student based on user entered Student ID.	System is expected to search and display the student details based on the entered Student ID.	Student details were found and displayed by the system.	Pass
Store Student Details into file Test Cases		System is expected to save the student details into a text (.txt) file.	Student details saved to the text file successfully and user was notified.	Pass
01.	Save student Details with a specified text file name available.			
02.	Save student Details with unspecified text file name available.	Displaying an error message to the user indicating that the text file does not exist is expected from the system.	Error message was displayed as expected.	Pass
View List of Students Test Cases		System is expected to display the list of students sorted alphabetically by their name.	Student details displayed in an alphabetical order.	Pass
01.	Display list of students in alphabetical based on their name.			
Generate Report Test Cases				

01.	Generate summary report including total student registrations and total no of students who scored more than 40 marks in Module 1, 2, and 3.	The System is expected to generate and display a summary of the required details mentioned in the test case.	System generated and displayed the summary as expected.	Pass
02.	Generate Detailed Report of student details including total marks, average, and calculated grade based on average mark.	The system is expected to generate and display a detailed report with the required details for each student.	System displayed the detailed report for each Student.	Pass

Task 04 – Testing – Discussion

Creating and selecting test cases were done by analyzing the data types used for the program for validating the user inputs. Identifying the purpose of each option in the menu supported to test the user navigation from the menu to the selected option. File handling purposes including loading from a file and saving to the file were tested by analyzing the file exceptions.

For Student Management System, class based approach is the better choice. In a class-based approach, Separate classes can be created for different operations enhancing the efficiency in the code. In the student management system, two separate classes were created. The ‘Student ’class holds the student name, Student ID, and Module class instance which connects the Module class and returns the module marks. The ‘Module’ class holds the three marks for the three modules.

The class based approach increases code readability. Creating methods inside classes to perform operations gives developers an idea about the purpose of created method.

Using Classes increases code maintainability. By encapsulation, developers can easily maintain the system code. This involves making changes only on specific parts of the program thereby decreasing the need to change the entire code system-wide.

In a Class-based approach, common operations can be added as a method and can be reused across different parts of the program.

In a class based approach, adding a new feature for the system can be implemented by adding a new attribute to the class. This allows developers to modify the code easily.

Self-Evaluation Form

Criteria	Allocated marks	Expected marks	Total	Student Comment
Task 1 Three marks for each option (1,2,3,4,5,6,7,8)	24	24	**(30)**	• Each options have been fully Implemented and working
Menu works correctly	6	5		• Fully implemented and working. Menu directs user to the selected option.
Task 2 Student class works correctly	14	14	**(30)**	• Fully implemented and working. • Student Class creates student object, returns Student name and id. • Student class creates an instance of Module class to get the module marks.
Module class works correctly	10	10		• Fully implemented and working. • Module class returns module marks and calculate grade based on average.
Sub menu (A and B works well)	6	6		• Fully implemented and working. Adding Student name, ID and Adding Module marks are working as expected.
Task 3 Report – Generate a summary	7	7	**(20)**	• Fully implemented and working. • Summary is generated covering the required criteria as described in specification.
Report – Generate the complete report	10	10		• Fully implemented and working.

				• Detailed Report for each student gets generated covering the required aspects of the specification.
Implementation of Bubble sort	3	2		• Fully implemented and working. • Bubble sort algorithms have been used to sort the student name alphabetically and to sort the Grade from highest to lowest
Task 4 Test case coverage and reasons	6	3	**(10)**	• Executing test cases on the system supported in debugging process. Test cases are described simply for better understanding.
Write up on which version is better and why.	4	2		• The better version and the reasons were added.
Coding Style (Comments, indentation, style)	7	5	**(10)**	• Fully implemented and working. • Comments were added for developers to understand the purpose of each method and classes. • Program was coded with an indentation for code clarity.
Complete the self-evaluation form indicating what you have accomplished to ensure appropriate feedback.	3	2		• Self-evaluation form completed indicating the accomplishments.
Totals	100	91	**(100)**	-

References

Javatpoint. (no date). Bubble Sort in Java. *Javatpoint*. Available from https://www.javatpoint.com/bubble-sort-in-java [Accessed 08 July 2024].

Javatpoint. (no date). File Operations in Java. *Javatpoint*. Available from https://www.javatpoint.com/file-operations-in-java [Accessed 06 July 2024].

Javatpoint. (no date). How to Create Array of Objects in Java. *Javatpoint*. Available from https://www.javatpoint.com/how-to-create-array-of-objects-in-java [Accessed 04 July 2024].

Javatpoint. (no date). Java Regex | Regular Expression. *Javatpoint*. Available from https://www.javatpoint.com/java-regex [Accessed 09 July 2024].

W3schools. (2019). Java Constructors. *W3schools.* Available from https://www.w3schools.com/java/java_constructors.asp [Accessed 04 July 2024].

W3schools. (2019). Java Files. *W3schools*. Available from https://www.w3schools.com/java/java_files.asp [Accessed 06 July 2024].

W3schools. (2019). Java String Reference. *W3schools*. Available from https://www.w3schools.com/java/java_ref_string.asp [Accessed 04 July 2024].