INFORMATICS INSTITUTE OF TECHNOLOGY

Software Development II

Coursework Report 2023/2024

N. Joshua Fernando

w2083099

20221912

Table of Contents

1. Task 01 – Source Code

```
import java.io.File;    // Create file object
import java.io.FileNotFoundException;   // Exception thrown when the file doesn't exist
import java.io.FileWriter;  // Write to file
import java.io.IOException; // Superclass to handle errors
import java.util.Random;    // Generating random numbers. Used to create random ID
import java.util.Scanner;   // Read user input

public class Management {
   static String[][] students = new String[100][2];   // Making 2D array for student name and ID

   public static void main(String[] args) {
      initialize(students);

      Scanner sc = new Scanner(System.in);

      while (true) {
         try {
            System.out.println("1. Check Available Seats");
            System.out.println("2. Register Student (With ID)");
            System.out.println("3. Delete Student");
            System.out.println("4. Find student (with student ID)");
            System.out.println("5. Store Student Details");
            System.out.println("6. Load student details from the file to the system ");
            System.out.println("7. View the list of students based on their names");
            System.out.println("8. Exit");


            System.out.print("Enter the option you want: ");
            int choice = sc.nextInt();
            sc.nextLine();  // Get a new line

            switch (choice) {
               case 1:
                  checkAvailableSeats();
                  break;
               case 2:
                  registerStudent(sc);
                  break;
               case 3:
                  deleteStudent();
                  break;
```

```
                case 4:
                    findStudent();
                    break;
                case 5:
                    storeDetails("StudentRecords.txt");
                    break;
                case 6:
                    loadDetails();
                    break;
                case 7:
                    viewDetails();
                    break;
                case 8:
                    System.out.println("Thank you for using Student Management System...");
                    System.exit(0);
                default:
                    System.out.println("Invalid choice. Please Try Again");
            }
        }
        catch (Exception e) {
            System.out.println("Input miss match..");
            sc.next();
        }
    }
}


private static void initialize(String[][] students) {
    for (int x = 0; x < 100; x++) {
        for (int y = 0; y < 2; y++) {
            students[x][y] = "empty";
        }
    }
}


// Option 1: Method for check the availability of seat numbers and also show how many seats are
available
private static void checkAvailableSeats() {

    int availableSeats = 0; // Initialize variable to set as the counting seat number to 0
    for (int x = 0; x < students.length; x++) {
        if (students[x][0].equals("empty")) {
            System.out.println("Seat " + (x + 1) + " is empty");
            availableSeats++;   // Increasing the value of available seats by 1
        }
    }
```

```
    System.out.println("Number of Available Seats: " + availableSeats);   // Print the number of seat
available
  }


  private static String generateRandomID() {      // Generating random ID's for students
    Random ranID = new Random();
    String id = "w";

    for (int i = 0; i < 7; i++) {
      id += ranID.nextInt(10);
    }
    return id;
  }


  // Option 2: Registering student
  private static void registerStudent(Scanner sc) {
    String name;    // Variable for name

    // Validate the name to contain only characters
    while (true) {
      System.out.print("Enter Student Name: ");
      name = sc.nextLine().toLowerCase();

      if (name.matches("[a-zA-Z ]+")) {
        break;
      } else {
        System.out.println("Invalid name. Please enter a name containing only letters.");
      }
    }

    String id;
    boolean registered = false; // Start with false until registration succeeds

    while (!registered) {
      id = generateRandomID();    // Calling the method to generate random ID
      boolean idExist = false;

      // Check if the generated ID already exists in the students array
      for (int x = 0; x < students.length; x++) {
        if (students[x][1].equals(id)) {
          idExist = true;
          break;
        }

      }
```

```
        // If ID does not exist register the student
        if (!idExist) {
            for (int x = 0; x < students.length; x++) {
                if (students[x][0].equals("empty")) {
                    students[x][0] = name;  // Inserting name into the first column of 2D array
                    students[x][1] = id;    // Inserting id into the second column of 2D array
                    System.out.println("Registered " + name + " with ID " + id + " in seat number " + (x + 1));
                    registered = true;
                    break; // Exit loop once registered
                }
            }
        }
    }
}


// Helper method to check registered students in delete students
private static void reservedSeats() {
    System.out.println("Registered students are...");
    if (students[0][0].equals("empty")) {
        System.out.print("Students are not enrolled yet");
    } else {
        for (int x = 0; x < students.length; x++) {
            if (!students[x][0].equals("empty")) {
                System.out.println("Seat " + (x + 1) + " is " + students[x][1]);
            }
        }
    }
}

// Option 3: Delete students based on their ID
private static void deleteStudent() {
    Scanner sc = new Scanner(System.in);

    reservedSeats();

    System.out.print("Enter ID of the student you want to delete: ");
    String id = sc.nextLine().toLowerCase();

    boolean found = false;
    for (int x = 0; x < students.length; x++) {
        if (students[x][1].equals(id)) {
            students[x][0] = "empty";
            students[x][1] = "empty";
            break;
        }
    }
    if (found) {
```

```
            System.out.println("Student " + id + " has been deleted successfully");
        } else {
            System.out.println("Student " + id + " not found");
        }

    }


    // Option 4: Find student
    private static void findStudent() {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter ID of the student you want to find: ");
        String id = sc.nextLine().toLowerCase();

        boolean found = false;
        for (int x = 0; x < students.length; x++) {
            if (students[x][1].equals(id)) {
                System.out.println("Student " + id + " has been found");
                found = true;
                break;
            }
        }
        if (!found) {
            System.out.println("Student " + id + " not found");
        }
    }

    // Option 5: Store details to text file
    private static void storeDetails(String filename) {
        try (FileWriter writer = new FileWriter(filename, true)) {
            for (String[] student : students) {
                if (!student[0].equals("empty") && !idStored(filename, student[1])) {
                    writer.write(student[0] + "," + student[1] + "\n");
                }
            }
            System.out.println("Data has been saved to " + filename);
        } catch (IOException e) {
            System.out.println("An error occurred while saving data to file: " + e.getMessage());
        }
    }

    // Helper method to check if the ID is stored
    private static boolean idStored(String filename, String id) {
        try {
            Scanner scan = new Scanner(new File(filename));
            while (scan.hasNextLine()) {
                String line = scan.nextLine();
```

```
            String[] details = line.split(",");
            if (details.length == 2 && details[1].equals(id)) {
                return true;
            }
        }
    } catch (FileNotFoundException e) {
        System.out.println("File Not Found" + e.getMessage());
    }  // ID can't be in the file cuz File not found
    return false;
}


// Option 6: Load from text file to the system
private static void loadDetails() {
    try {
        File file = new File("StudentRecords.txt");
        Scanner scan = new Scanner(file);

        while (scan.hasNextLine()) {
            String line = scan.nextLine();
            String[] details = line.split(",");    // Split the line from the text file to get the name and ID
separately

            if (details.length == 2) {
                // Checking if the ID already exists
                boolean idExist = false;
                for (int x = 0; x < students.length; x++) {
                    if (students[x][1].equals(details[1])) {
                        idExist = true;
                        break;
                    }
                }
                if (!idExist) {
                    for (int x = 0; x < students.length; x++) {
                        if (students[x][0].equals("empty")) {
                            students[x][0] = details[0];
                            students[x][1] = details[1];
                            break;
                        }
                    }
                } else {    // Remove this else, can't solve the printing problem
                    System.out.println("Already loaded. Try again after registering a new student.");
                }
            }
        }
        scan.close();
        System.out.println("Data has been loaded to the System successfully");
    }
```

```
        catch (FileNotFoundException e) {
            System.out.println("An error occurred while loading data from file");
        }
    }

    // Option 7: View details of the students
    private static void viewDetails() {
        sortStudents();

        int seatNo = 1;
        if (students[0][0].equals("empty")) {
            System.out.println("There are no registered students to view");
        } else {
            for (int x = 0; x < students.length; x++) {
                if (!students[x][0].equals("empty")) {
                    System.out.println("Seat number '" + seatNo + " is reserved for  '" + students[x][0] + "'.  ID
Number: " + students[x][1]);
                    seatNo++;
                }
            }
        }
    }

    // Helper method to sort students in viewing
    private static void sortStudents() {
        for (int x = 0; x < students.length - 1; x++) {
            int minIndex = x;
            for (int y = x + 1; y < students.length; y++) {
                if (!students[y][0].equals("empty") &&
students[y][0].compareToIgnoreCase(students[minIndex][0]) < 0) {
                    minIndex = y;
                }
            }

            // Swap the elements
            String[] temp = students[minIndex];
            students[minIndex] = students[x];
            students[x] = temp;
        }
    }
}
```

2. Task 02 – Source Code

```
import java.io.File;    // Create file object
import java.io.FileNotFoundException;   // Exception thrown when the file doesn't exist
import java.io.FileWriter;  // Write to file
import java.io.IOException; // Superclass to handle errors
import java.util.Random;    // Generating random numbers. Used to create random ID
import java.util.Scanner;   // Read user input


public class Management2 {
   private static String[][] studentsArray = new String[100][5]; // 2D array for tasks 1-7
   private static Student[] students = new Student[100]; // Array of Student objects for task 8

   public static void main(String[] args) {
      initialize(studentsArray);

      Scanner sc = new Scanner(System.in);

         while (true) {         // Loop will continue and keep display the menu until the user exit
            try {
            // Displaying 8 options to user as menu
            System.out.println("1. Check Number of Available Seats");
            System.out.println("2. Register Students (With ID)");
            System.out.println("3. Delete Student Based on ID");
            System.out.println("4. Find student with student ID");
            System.out.println("5. Store Student Details to Text file");
            System.out.println("6. Load student details from the file to the system");
            System.out.println("7. View the list of students based on their names");
            System.out.println("8. Manage students Results");
            System.out.println("9. Exit");

            System.out.print("Enter the option you want: ");
            int choice = sc.nextInt();
            sc.nextLine();  // Get a new line

            switch (choice) {
               case 1:
                  checkAvailableSeats();
                  break;
               case 2:
                  registerStudent(sc);
                  break;
               case 3:
                  deleteStudent(sc);
                  break;
```

```
            case 4:
               findStudent(sc);
               break;
            case 5:
               storeDetails("StudentRecordsTask2.txt");
               break;
            case 6:
               loadDetails();
               break;
            case 7:
               viewDetails();
               break;
            case 8:
               manageResults(sc);
               break;
            case 9:
               System.out.println("Thank you for using Student Management System...");
               System.exit(0);
            default:
               System.out.println("Invalid choice. Please Try Again");
         }
      }
      catch (Exception e) {
         System.out.println("Input miss match..");
         sc.next();
      }

   }
}

// Initialize the 2D array to "empty"
private static void initialize(String[][] studentsArray) {
   for (int x = 0; x < 100; x++) {
      for (int y = 0; y < 2; y++) {
         studentsArray[x][y] = "empty";
      }
   }
}

// Option 1: Check and display available seats
private static void checkAvailableSeats() {
   int availableSeats = 0;
   for (int x = 0; x < studentsArray.length; x++) {
      if (studentsArray[x][0].equals("empty")) {
         System.out.println("Seat " + (x + 1) + " is empty");
         availableSeats++;
      }
   }
```

```
    System.out.println("Number of Available Seats: " + availableSeats);
    System.out.println("==============================================");
}

// Generate a random ID for student
private static String generateRandomID() {
    Random ranID = new Random();
    String id = "w";    // write "w" character in front of each ID
    // Loop till the ID get 7 digits
    for (int i = 0; i < 7; i++) {
        id += ranID.nextInt(10);
    }
    return id;  // Return ID for access in other methods
}

// Option 2: Registering new student
private static void registerStudent(Scanner sc) {
    String name;    // Variable for name

    // Validate the name to contain only characters
    while (true) {
        System.out.print("Enter Student Name: ");
        name = sc.nextLine().toLowerCase();

        if (name.matches("[a-zA-Z ]+")) {
            break;
        } else {
            System.out.println("Invalid name. Please enter a name containing only letters.");
        }
    }

    String id;
    boolean registered = false;

    while (!registered) {
        id = generateRandomID();
        boolean idExist = false;

        for (String[] student : studentsArray) {
            if (student[1].equals(id)) {
                idExist = true;
                break;
            }
        }

        if (!idExist) {
            // Find the first empty slot and register the student
            for (int x = 0; x < studentsArray.length; x++) {
```

```
            if (studentsArray[x][0].equals("empty")) {
                studentsArray[x][0] = name;
                studentsArray[x][1] = id;
                studentsArray[x][2] = null;
                studentsArray[x][3] = null;
                studentsArray[x][4] = null;
                System.out.println("Registered " + name + " with ID " + id + " in seat number " + (x + 1));
                registered = true;
                break;
            }
        }
    }
}

// Option 3: Delete student by their ID
private static void deleteStudent(Scanner sc) {
    System.out.print("Enter ID of the student you want to delete: ");
    String id = sc.nextLine().toLowerCase();

    boolean found = false;

    // Find the student based on the entered ID and delete their record
    for (int x = 0; x < studentsArray.length; x++) {
        if (studentsArray[x][1].equals(id)) {
            studentsArray[x][0] = "empty";
            studentsArray[x][1] = "empty";
            studentsArray[x][2] = "empty";
            studentsArray[x][3] = "empty";
            studentsArray[x][4] = "empty";
            found = true;
            break;
        }
    }
    if (found) {
        System.out.println("Student " + id + " has been deleted successfully");
        updateFile("StudentRecordsTask2.txt");
    }
    else {
        System.out.println("Student " + id + " not found");
    }
}

// Helper method to update text file after deleting method
private static void updateFile(String fileName) {

    // Opening Filewriter wihin the try block ensures it closes properly
    try (FileWriter writer = new FileWriter(fileName)) {
```

```
            // Iterate through each element in studentsArray
            for (String[] student : studentsArray) {
                if (!student[0].equals("empty")) {
                    // Write data to text file(name, ID, mark1, mark2, mark3)
                    writer.write(student[0] + "," + student[1] + "," + student[2] + "," + student[3] + "," +
student[4] + "\n");
                }
            }
            System.out.println("Data has been updated in " + fileName);
        }
        catch (IOException e) {
            System.out.println("An error occurred while updating data in file");
        }
    }

    // Option 4: Find a student by ID
    private static void findStudent(Scanner sc) {
        System.out.print("Enter ID of the student you want to find: ");
        String id = sc.nextLine().toLowerCase();

        boolean found = false;
        for (int x = 0; x < studentsArray.length; x++) {
            if (studentsArray[x][1].equals(id)) {
                System.out.println("Student " + id + " has been found");
                found = true;
                break;
            }
        }
        if (!found) {
            System.out.println("Student " + id + " not found");
        }
    }

    // Option 5: Storing details to text file(StudentRecords.txt)
    private static void storeDetails(String filename) {
        try (FileWriter writer = new FileWriter(filename, true)) {
            for (String[] student : studentsArray) {
                if (!student[0].equals("empty") && !idStored(filename, student[1])) {
                    writer.write(student[0] + "," + student[1] + "," + student[2] + "," + student[3] + "," +
student[4] + "\n");
                }
            }
            System.out.println("Data has been saved to " + filename);
        } catch (IOException e) {
            System.out.println("An error occurred while saving data to file");
        }
    }
```

```
// Helper method to check if ID is already stored
private static boolean idStored(String filename, String id) {
   try {
      Scanner scan = new Scanner(new File(filename));
      while (scan.hasNextLine()) {
         String line = scan.nextLine();
         String[] details = line.split(",");
         if (details.length >= 2 && details[1].equals(id)) {
            return true;
         }
      }
   } catch (FileNotFoundException e) {
      System.out.println("File Not Found" + e.getMessage());
   }
   return false;
}

// Option 6: Load student details from the file to the system
private static void loadDetails() {
   try {
      File file = new File("StudentRecordsTask2.txt");
      Scanner scan = new Scanner(file);

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

         if (details.length >= 5) {
            boolean idExist = false;
            for (int x = 0; x < studentsArray.length; x++) {
               if (studentsArray[x][1] != null && studentsArray[x][1].equals(details[1])) {
                  idExist = true;
                  break;
               }
            }
            if (!idExist) {
               for (int x = 0; x < studentsArray.length; x++) {
                  if (studentsArray[x][0].equals("empty")) {
                     studentsArray[x][0] = details[0];
                     studentsArray[x][1] = details[1];
                     studentsArray[x][2] = details[2];
                     studentsArray[x][3] = details[3];
                     studentsArray[x][4] = details[4];
                     break;
                  }
               }
            } else {
```

```
                System.out.println("Already loaded. Try again after registering a new student.");
            }
        } else {
            System.out.println("Invalid data format: " + line);
        }
      }
      scan.close();
      System.out.println("Data has been loaded to the System successfully");
    } catch (FileNotFoundException e) {
      System.out.println("An error occurred while loading data from the file: " + e.getMessage());
    }
  }

  // Option 7: View the lit of students based on their name(Sorted)
  private static void viewDetails() {
    for (int i = 0; i < studentsArray.length; i++) {
      for (int j = i + 1; j < studentsArray.length; j++) {
        if (studentsArray[i][0].compareTo(studentsArray[j][0]) > 0) {
          String[] temp = studentsArray[i];
          studentsArray[i] = studentsArray[j];
          studentsArray[j] = temp;
        }
      }
    }

    System.out.println("Sorted list of students based on their names:");
    System.out.println("=====================================================");
    for (int x = 0; x < studentsArray.length; x++) {
      if (!studentsArray[x][0].equals("empty")) {
        System.out.println("Name: " + studentsArray[x][0] + ", ID: " + studentsArray[x][1]);
      }
    }
  }

  // Option 8: Manage student results
  private static void manageResults(Scanner sc) {
    loadStudentsFromFile("StudentRecordsTask2.txt");    // Call the load helper method to load data
from text file

    while(true){
      try {
        // Display menu option for sub options in option 8
        System.out.println("1. Add student name");
        System.out.println("2. Module marks 1, 2 and 3");
        System.out.println("3. Back to main menu");

        System.out.print("Enter your choice: ");
        int option = sc.nextInt();
```

```
            sc.nextLine();  // Get a new line

            // Handle each method through menu
            switch (option) {
               case 1:
                  addStudentName(sc);
                  break;
               case 2:
                  addModuleMarks(sc);
                  break;
               case 3:
                  return;    // Return from the option 8 and going back to main menu
               default:
                  System.out.println("Invalid choice. Please try again.");
            }
         }
         catch (Exception e) {
            System.out.println("Input miss match..");
            sc.next();
         }
      }
   }

   // Sub Option 1: Check if the entered name is available
   private static void addStudentName(Scanner sc) {
      System.out.print("Enter student name: ");
      String addName = sc.nextLine();

      boolean nameExists = false;

      for(int x = 0; x < students.length; x++) {
         if(students[x] != null && addName.equalsIgnoreCase(students[x].getName())){
            System.out.println("Student " + addName + " already exists with ID: " + students[x].getStID());
            nameExists = true;
            break;
         }
      }
      // Print error message when the entered student doesn't exist
      if (!nameExists) {
         System.out.println("Student " + addName + " doesn't exist..");
      }
   }

   // Sub Option 2: Add marks for each module based on entered student ID
   private static void addModuleMarks(Scanner sc) {
      System.out.print("Enter Student ID: ");
      String id = sc.nextLine().toLowerCase();
```

```
    Student student = null;
    for (Student s : students) {
        if (s != null && s.getStID().equals(id)) {
            student = s;
            break;
        }
    }

    if (student == null) {
        System.out.println("Student not found. Please register the student first.");
        return;
    }

    Module[] modules = student.getModules();
    for (int i = 0; i < 3; i++) {
        System.out.print("Enter marks for Module " + (i + 1) + ": ");
        double marks = sc.nextDouble();
        modules[i].setMarks(marks);
    }

    // Update the file with the new data
    updateStudentFile("StudentRecordsTask2.txt");
    System.out.println("Marks for student " + student.getName() + " have been updated.");
}

// Helper method to update the text file
private static void updateStudentFile(String filename) {
    try (FileWriter writer = new FileWriter(filename)) {
        for (Student student : students) {
            if (student != null) {
                writer.write(student.getName() + "," + student.getStID());

                Module[] modules = student.getModules();
                if (modules != null) {
                    for (Module module : modules) {
                        writer.write("," + module.getMarks());
                    }
                }

                writer.write("\n");
            }
        }
        System.out.println("Data has been updated in " + filename);
    } catch (IOException e) {
        System.out.println("An error occurred while updating data in file");
    }
}
```

```
    // Helper method to load data from text file into objects
    private static void loadStudentsFromFile(String filename) {
        try {
            File file = new File(filename);
            Scanner scan = new Scanner(file);

            int index = 0;
            while (scan.hasNextLine()) {
                String line = scan.nextLine();
                String[] details = line.split(",");

                if (details.length >= 5) {
                    String name = details[0];
                    String id = details[1];

                    // Initialize marks array with default values (0.0)
                    double[] marks = new double[3];
                    for (int i = 0; i < 3; i++) {
                        marks[i] = details[i + 2].equals("null") ? 0.0 : Double.parseDouble(details[i + 2]);
                    }

                    Module[] modules = new Module[3];
                    for (int i = 0; i < 3; i++){
                        modules[i] = new Module();
                        modules[i].setMarks(marks[i]);
                    }

                    students[index] = new Student(name, id);
                    students[index].setModules(modules);
                    index++;
                }
                else {
                    System.out.println("Invalid data format: " + line);
                }
            }
            System.out.println("Data loaded from " + filename + " to Student objects successfully.");
        }
        catch (FileNotFoundException e) {
            System.out.println("File Not Found");
        }
        catch (NumberFormatException e) {
            System.out.println("Error parsing module marks");
            e.printStackTrace(); // Print stack trace for debugging
        }
    }
}
```

Module Class

```
public class Module {
    private double marks;

    // Default constructor
    public Module() {
        this.marks = 0.0; // Initialize marks to 0.0
    }

    // Getter and setter for marks
    public double getMarks() {
        return marks;
    }

    public void setMarks(double marks) {
        this.marks = marks;
    }
}
```

Student Class

```
public class Student {
    private String name;
    private String stID;
    private Module[] modules;

    public Student(String name, String stID) {
        this.name = name;
        this.stID = stID;
        this.modules = new Module[3]; // Assuming each student has 3 modules by default
    }

    public String getName() {
        return name;
    }

    public String getStID() {
        return stID;
    }

    public Module[] getModules() {
        return modules;
    }
```

```
    public void setModules(Module[] modules) {
        this.modules = modules;
    }

    public double calculateTotalMarks() {
        double total = 0;
        for (Module module : modules) {
            total += module.getMarks();
        }
        return total;
    }

    public double calculateAverageMarks() {
        double total = calculateTotalMarks();
        double average = total / modules.length;
        return average;
    }

    public String calculateGrade() {
        double average = calculateAverageMarks();

        if (average >= 80) {
            return "Distinction";
        } else if (average >= 70) {
            return "Merit";
        } else if (average >= 40) {
            return "Pass";
        } else {
            return "Fail";
        }
    }
}
```

3. Task 03 – Source Code

```
import java.io.File;    // Create file object
import java.io.FileNotFoundException;   // Exception thrown when the file doesn't exist
import java.io.FileWriter;  // Write to file
import java.io.IOException; // Superclass to handle errors
import java.util.Random;    // Generating random numbers. Used to create random ID
import java.util.Scanner;   // Read user input


public class Management3 {
   private static String[][] studentsArray = new String[100][5]; // 2D array for tasks 1-7
   private static Student[] students = new Student[100]; // Array of Student objects for task 8

   public static void main(String[] args) {
      initialize(studentsArray);

      Scanner sc = new Scanner(System.in);

         while (true) {         // Loop will continue and keep display the menu until the user exit
            try{
            // Displaying 8 options to user as menu
            System.out.println("1. Check Available Seats");
            System.out.println("2. Register Student (With ID)");
            System.out.println("3. Delete Student");
            System.out.println("4. Find student (with student ID)");
            System.out.println("5. Store Student Details");
            System.out.println("6. Load student details from the file to the system");
            System.out.println("7. View the list of students based on their names");
            System.out.println("8. Manage students Results");
            System.out.println("9. Exit");

            System.out.print("Enter the option you want: ");
            int choice = sc.nextInt();
            sc.nextLine();  // Get a new line

            switch (choice) {
               case 1:
                  checkAvailableSeats();
                  break;
               case 2:
                  registerStudent(sc);
                  break;
               case 3:
                  deleteStudent(sc);
                  break;
```

```
            case 4:
               findStudent(sc);
               break;
            case 5:
               storeDetails("StudentRecordsTask3.txt");
               break;
            case 6:
               loadDetails();
               break;
            case 7:
               viewDetails();
               break;
            case 8:
               manageResults(sc);
               break;
            case 9:
               System.out.println("Thank you for using Student Management System...");
               System.exit(0);
            default:
               System.out.println("Invalid choice. Please Try Again");
         }
      }
         catch (Exception e) {
            System.out.println("Input miss match..");
            sc.next();
         }
   }

}

// Initialize the 2D array to "empty"
private static void initialize(String[][] studentsArray) {
   for (int x = 0; x < 100; x++) {
      for (int y = 0; y < 2; y++) {
         studentsArray[x][y] = "empty";
      }
   }
}

// Option 1: Check and display available seats
private static void checkAvailableSeats() {
   int availableSeats = 0;
   for (int x = 0; x < studentsArray.length; x++) {
      if (studentsArray[x][0].equals("empty")) {
         System.out.println("Seat " + (x + 1) + " is empty");
         availableSeats++;
      }
   }
```

```
    System.out.println("Number of Available Seats: " + availableSeats);
    System.out.println("==========================================");
}

// Generate a random ID for student
private static String generateRandomID() {
    Random ranID = new Random();
    String id = "w";    // write "w" character in front of each ID
    // Loop till the ID get 7 digits
    for (int i = 0; i < 7; i++) {
        id += ranID.nextInt(10);
    }
    return id;  // Return ID for access in other methods
}

// Option 2: Registering new student
private static void registerStudent(Scanner sc) {
    String name;    // Variable for name

    // Validate the name to contain only characters
    while (true) {
        System.out.print("Enter Student Name: ");
        name = sc.nextLine().toLowerCase();

        if (name.matches("[a-zA-Z ]+")) {
            break;
        } else {
            System.out.println("Invalid name. Please enter a name containing only letters.");
        }
    }

    String id;
    boolean registered = false;

    while (!registered) {
        id = generateRandomID();
        boolean idExist = false;

        for (String[] student : studentsArray) {
            if (student[1].equals(id)) {
                idExist = true;
                break;
            }
        }

        if (!idExist) {
            // Find the first empty slot and register the student
            for (int x = 0; x < studentsArray.length; x++) {
```

```
            if (studentsArray[x][0].equals("empty")) {
                studentsArray[x][0] = name;
                studentsArray[x][1] = id;
                studentsArray[x][2] = null;
                studentsArray[x][3] = null;
                studentsArray[x][4] = null;
                System.out.println("Registered " + name + " with ID " + id + " in seat number " + (x + 1));
                registered = true;
                break;
            }
        }
    }
}

// Option 3: Delete student by their ID
private static void deleteStudent(Scanner sc) {
    System.out.print("Enter ID of the student you want to delete: ");
    String id = sc.nextLine().toLowerCase();

    boolean found = false;

    // Find the student based on the entered ID and delete their record
    for (int x = 0; x < studentsArray.length; x++) {
        if (studentsArray[x][1].equals(id)) {
            studentsArray[x][0] = "empty";
            studentsArray[x][1] = "empty";
            studentsArray[x][2] = "empty";
            studentsArray[x][3] = "empty";
            studentsArray[x][4] = "empty";
            found = true;
            break;
        }
    }
    if (found) {
        System.out.println("Student " + id + " has been deleted successfully");
        updateFile("StudentRecordsTask3.txt");
    }
    else {
        System.out.println("Student " + id + " not found");
    }
}

// Helper method to update text file after deleting method
private static void updateFile(String fileName) {

    // Opening Filewriter wihin the try block ensures it closes properly
    try (FileWriter writer = new FileWriter(fileName)) {
```

```
        // Iterate through each element in studentsArray
        for (String[] student : studentsArray) {
            if (!student[0].equals("empty")) {
                // Write data to text file(name, ID, mark1, mark2, mark3)
                writer.write(student[0] + "," + student[1] + "," + student[2] + "," + student[3] + "," +
student[4] + "\n");
            }
        }
        System.out.println("Data has been updated in " + fileName);
    }
    catch (IOException e) {
        System.out.println("An error occurred while updating data in file: " + e.getMessage());
    }
  }

  // Option 4: Find a student by ID
  private static void findStudent(Scanner sc) {
    System.out.print("Enter ID of the student you want to find: ");
    String id = sc.nextLine().toLowerCase();

    boolean found = false;
    for (int x = 0; x < studentsArray.length; x++) {
        if (studentsArray[x][1].equals(id)) {
            System.out.println("Student " + id + " has been found");
            found = true;
            break;
        }
    }
    if (!found) {
        System.out.println("Student " + id + " not found");
    }
  }

  // Option 5: Storing details to text file(StudentRecords.txt)
  private static void storeDetails(String filename) {
    try (FileWriter writer = new FileWriter(filename, true)) {
        for (String[] student : studentsArray) {
            if (!student[0].equals("empty") && !idStored(filename, student[1])) {
                writer.write(student[0] + "," + student[1] + "," + student[2] + "," + student[3] + "," +
student[4] + "\n");
            }
        }
        System.out.println("Data has been saved to " + filename);
    } catch (IOException e) {
        System.out.println("An error occurred while saving data to file: " + e.getMessage());
    }
  }
```

```
// Helper method to check if ID is already stored
private static boolean idStored(String filename, String id) {
   try {
      Scanner scan = new Scanner(new File(filename));
      while (scan.hasNextLine()) {
         String line = scan.nextLine();
         String[] details = line.split(",");
         if (details.length >= 2 && details[1].equals(id)) {
            return true;
         }
      }
   } catch (FileNotFoundException e) {
      System.out.println("File Not Found" + e.getMessage());
   }
   return false;
}

// Option 6: Load student details from the file to the system
private static void loadDetails() {
   try {
      File file = new File("StudentRecordsTask3.txt");
      Scanner scan = new Scanner(file);

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

         if (details.length >= 5) {
            boolean idExist = false;
            for (int x = 0; x < studentsArray.length; x++) {
               if (studentsArray[x][1] != null && studentsArray[x][1].equals(details[1])) {
                  idExist = true;
                  break;
               }
            }
            if (!idExist) {
               for (int x = 0; x < studentsArray.length; x++) {
                  if (studentsArray[x][0].equals("empty")) {
                     studentsArray[x][0] = details[0];
                     studentsArray[x][1] = details[1];
                     studentsArray[x][2] = details[2];
                     studentsArray[x][3] = details[3];
                     studentsArray[x][4] = details[4];
                     break;
                  }
               }
            } else {
```

```
                System.out.println("Already loaded. Try again after registering a new student.");
            }
        } else {
            System.out.println("Invalid data format: " + line);
        }
    }
    scan.close();
    System.out.println("Data has been loaded to the System successfully");
  } catch (FileNotFoundException e) {
    System.out.println("An error occurred while loading data from the file: " + e.getMessage());
  }
}

// Option 7: View the lit of students based on their name(Sorted)
private static void viewDetails() {
  for (int i = 0; i < studentsArray.length; i++) {
    for (int j = i + 1; j < studentsArray.length; j++) {
      if (studentsArray[i][0].compareTo(studentsArray[j][0]) > 0) {
        String[] temp = studentsArray[i];
        studentsArray[i] = studentsArray[j];
        studentsArray[j] = temp;
      }
    }
  }

  System.out.println("Sorted list of students based on their names:");
  System.out.println("=====================================================");
  for (int x = 0; x < studentsArray.length; x++) {
    if (!studentsArray[x][0].equals("empty")) {
      System.out.println("Name: " + studentsArray[x][0] + ", ID: " + studentsArray[x][1]);
    }
  }
}

// Option 8: Manage student results
private static void manageResults(Scanner sc) {
  loadStudentsFromFile("StudentRecordsTask3.txt");    // Call the load helper method to load data
from text file

  while(true){
    try {
      // Display menu option for sub options in option 8
      System.out.println("1. Add student name");
      System.out.println("2. Module marks 1, 2 and 3");
      System.out.println("3. Generate a summary of the system");
      System.out.println("4. Generate complete report");
      System.out.println("5. Back to main menu");
```

```
            System.out.print("Enter your choice: ");
            int option = sc.nextInt();
            sc.nextLine();  // Get a new line

            // Handle each method through menu
            switch (option) {
                case 1:
                    addStudentName(sc);
                    break;
                case 2:
                    addModuleMarks(sc);
                    break;
                case 3:
                    generateSummary();
                    break;
                case 4:
                    generateCompleteReport(sc);
                    break;
                case 5:
                    return;     // Return from the option 8 and going back to main menu
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        }
        catch (Exception e) {
            System.out.println("Input miss match..");
            sc.next();
        }
    }
}

// Sub Option 1: Check if the entered name is available
private static void addStudentName(Scanner sc) {
    System.out.print("Enter student name: ");
    String addName = sc.nextLine();

    boolean nameExists = false;

    for(int x = 0; x < students.length; x++) {
        if(students[x] != null && addName.equalsIgnoreCase(students[x].getName())){
            System.out.println("Student " + addName + " already exists with ID: " + students[x].getStID());
            nameExists = true;
            break;
        }
    }
    // Print error message when the entered student doesn't exist
    if (!nameExists) {
        System.out.println("Student " + addName + " doesn't exist..");
```

```
    }
}

// Sub Option 2: Add marks for each module based on entered student ID
private static void addModuleMarks(Scanner sc) {
    System.out.print("Enter Student ID: ");
    String id = sc.nextLine().toLowerCase();

    Student student = null;
    for (Student s : students) {
        if (s != null && s.getStID().equals(id)) {
            student = s;
            break;
        }
    }

    if (student == null) {
        System.out.println("Student not found. Please register the student first.");
        return;
    }

    Module[] modules = student.getModules();
    for (int i = 0; i < 3; i++) {
        System.out.print("Enter marks for Module " + (i + 1) + ": ");
        double marks = sc.nextDouble();
        modules[i].setMarks(marks);
    }

    // Update the file with the new data
    updateStudentFile("StudentRecordsTask3.txt");
    System.out.println("Marks for student " + student.getName() + " have been updated.");
}

// Helper method to update the text file when the
private static void updateStudentFile(String filename) {
    try (FileWriter writer = new FileWriter(filename)) {
        for (Student student : students) {
            if (student != null) {
                writer.write(student.getName() + "," + student.getStID());

                Module[] modules = student.getModules();
                if (modules != null) {
                    for (Module module : modules) {
                        writer.write("," + module.getMarks());
                    }
                }

                writer.write("\n");
```

```
            }
        }
        System.out.println("Data has been updated in " + filename);
    } catch (IOException e) {
        System.out.println("An error occurred while updating data in file: " + e.getMessage());
    }
}


// Sub Option 3: Generate a summary about each module
private static void generateSummary() {
    int totalRegistrations = 0;
    int[] passedModuleCount = new int[3];

    for (Student student : students) {
        if (student != null) {
            totalRegistrations++;
            Module[] modules = student.getModules();
            if (modules != null) {
                for (int i = 0; i < 3; i++) {
                    if (modules[i] != null && modules[i].getMarks() >= 40) {
                        passedModuleCount[i]++;
                    }
                }
            }
        }
    }

    System.out.println("Total student registrations: " + totalRegistrations);
    for (int i = 0; i < 3; i++) {
        System.out.println("Total no of students who scored more than 40 marks in Module " + (i + 1) + ":
" + passedModuleCount[i]);
    }
}



// Helper method to sort. Sort by average highest to lowest
private static void sortByAverage() {
    // Bubble sort based on average marks in descending order
    for (int i = 0; i < students.length - 1; i++) {
        for (int j = 0; j < students.length - i - 1; j++) {
            if (students[j] == null || students[j + 1] == null) continue;
            if (students[j].calculateAverageMarks() < students[j + 1].calculateAverageMarks()) {
                // Swap students[j] and students[j + 1]
                Student temp = students[j];
                students[j] = students[j + 1];
                students[j + 1] = temp;
```

```
        }
      }
    }
}

// Sub Option 4:  Generate complete report with list of students includes
private static void generateCompleteReport(Scanner sc) {
   sortByAverage();       // Sort all the students based on average in the start

   System.out.println("Complete Report");
   System.out.println("=================================================");

   for(Student student : students) {
      if (student == null) continue;

      // Assign calculated total, average and Grade
      double total = student.calculateTotalMarks();
      double average = student.calculateAverageMarks();
      String grade = student.calculateGrade();


      // Display total and average
      System.out.println("Student ID: " + student.getStID());
      System.out.println("Student Name: " + student.getName());
      // Get each module's marks
      Module[] modules = student.getModules();
      for (int i = 0; i < modules.length; i++) {
         System.out.println("Module " + (i + 1) + " mark: " + modules[i].getMarks());
      }
      System.out.println("Total: " + total);
      System.out.println("Average: " + average);
      System.out.println("Grade: " + grade);
      System.out.println(".....................................................................");
   }

   // Update the file with the new data
   updateStudentFile("StudentRecordsTask3.txt");
   System.out.println("Marks for all students have been updated...");
}


// Helper method to load data from text file into objects
private static void loadStudentsFromFile(String filename) {
   try {
      File file = new File(filename);
      Scanner scan = new Scanner(file);

      int index = 0;
```

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

                if (details.length >= 5) {
                    String name = details[0];
                    String id = details[1];

                    // Initialize marks array with default values (0.0)
                    double[] marks = new double[3];
                    for (int i = 0; i < 3; i++) {
                        marks[i] = details[i + 2].equals("null") ? 0.0 : Double.parseDouble(details[i + 2]);
                    }

                    Module[] modules = new Module[3];
                    for (int i = 0; i < 3; i++){
                        modules[i] = new Module();
                        modules[i].setMarks(marks[i]);
                    }

                    students[index] = new Student(name, id);
                    students[index].setModules(modules);
                    index++;
                }
                else {
                    System.out.println("Invalid data format: " + line);
                }
            }
            System.out.println("Data loaded from " + filename + " to Student objects successfully.");
        }
        catch (FileNotFoundException e) {
            System.out.println("File Not Found: " + e.getMessage());
        }
        catch (NumberFormatException e) {
            System.out.println("Error parsing module marks: " + e.getMessage());
            e.printStackTrace(); // Print stack trace for debugging
        }
    }
}
```

Module Class

```
public class Module {
    private double marks;
```

```
    // Default constructor
    public Module() {
        this.marks = 0.0; // Initialize marks to 0.0
    }

    // Getter and setter for marks
    public double getMarks() {
        return marks;
    }

    public void setMarks(double marks) {
        this.marks = marks;
    }
}
```

Student Class

```
public class Student {
    private String name;
    private String stID;
    private Module[] modules;

    public Student(String name, String stID) {
        this.name = name;
        this.stID = stID;
        this.modules = new Module[3]; // Assuming each student has 3 modules by default
    }

    public String getName() {
        return name;
    }

    public String getStID() {
        return stID;
    }

    public Module[] getModules() {
        return modules;
    }

    public void setModules(Module[] modules) {
        this.modules = modules;
    }
```

```
    public double calculateTotalMarks() {
        double total = 0;
        for (Module module : modules) {
            total += module.getMarks();
        }
        return total;
    }

    public double calculateAverageMarks() {
        double total = calculateTotalMarks();
        double average = total / modules.length;
        return average;
    }

    public String calculateGrade() {
        double average = calculateAverageMarks();

        if (average >= 80) {
            return "Distinction";
        } else if (average >= 70) {
            return "Merit";
        } else if (average >= 40) {
            return "Pass";
        } else {
            return "Fail";
        }
    }
}
```

4. Task 04 – Testing

Task 1

Test Case	Expected Result	Actual Result	Pass/Fail
Menu Option	Display all the options possible and take the choice from user and leads to the specific method. If the choice number is wrong it will display "Invalid choice. Please Try Again". If user enters anything than number it will display “Input mismatched”	Display all the options possible and take the choice from user and leads to the specific method. If the choice number is wrong it will display "Invalid choice. Please Try Again". If user enters anything than number it will display “Input mismatched”	Pass
Check available seats	When the option 1 selected it will check for each empty seat and finally prints all the empty seats and the number of empty seats.	When the option 1 selected it will check for each empty seat and finally prints all the empty seats and the number of empty seats.	Pass
Generate Random ID	Generating random ID’s starting with ‘w’ and contains 7 characters. This will return the ID to register method.	Generating random ID’s starting with ‘w’ and contains 7 characters. This will return the ID to register method.	Pass
Register Student	Take name as uSer input. Generate Random ID for each student. Validate name and ID by validate name by to contain only characters. If not it will print "Invalid name. Please enter a name containing only letters.". Validate ID by checking if the ID is already stored. If the ID is stored then again	Take name as uSer input. Generate Random ID for each student. Validate name and ID by validate name by to contain only characters. If not it will print "Invalid name. Please enter a name containing only letters.". Validate ID by checking if the ID is already stored. If the ID is stored then again	Pass

	another ID will be generated.	another ID will be generated.	
Check reserved seats	Helper method to check reserved seats and display in the beginning of deletion.	Helper method to check reserved seats and display in the beginning of deletion.	Pass
Delete Student	Take the ID of student from user and then delete the student. After successfully deleting it will print “Student has been deleted succefully”. If the ID is not found it will print “Student not found”.	Take the ID of student from user and then delete the student. After successfully deleting it will print “Student has been deleted succefully”. If the ID is not found it will print “Student not found”.	Pass
Find Student	Take ID from user. If the student found it will print “Student has been found”. If not it print “Student not found”	Take ID from user. If the student found it will print “Student has been found”. If not it print “Student not found”	Pass
Store Details	When the option is selected it will store the registered students and their ID to the text file.	When the option is selected it will store the registered students and their ID to the text file.	Pass
Load Details	If there are data in the text file, it will store into the 2D array. Catch the exception and print "An error occurred while loading data from file".	If there are data in the text file, it will store into the 2D array. Catch the exception and print "An error occurred while loading data from file".	Pass
View Details	Display students sorted. If there are no students it will print "There are no registered students to view".	Display students sorted. If there are no students it will print "There are no registered students to view".	Pass
Sort Students	Helper method to sort students in view method.	Helper method to sort students in view method.	Pass

Task 2

The first 7 Options work as the task 1. Changed parts are from 8th option. There are two sub options to this. It will work with student and module classes.

Test Cases	Expected Result	Actual Result	Pass / Fail
Manage Results	Display sub options Add student, Add module marks and back to main menu. This will lead to their specific method. If the choice number is wrong it will display "Invalid choice. Please Try Again". If user enters anything than number it will display “Input mismatched”	Display sub options Add student, Add module marks and back to main menu. This will lead to their specific method.	Pass
Add Student Name	Get name from user and check if the student is registered. If not print "Student doesn't exist.."	Get name from user and check if the student is registered. If not print "Student doesn't exist.."	Pass
Add Module Marks	Take student ID and Take 3 marks for each module. If the student ID is not found it will Print “Student not found. Please register student first”.	Take student ID and Take 3 marks for each module. If the student ID is not found it will Print “Student not found. Please register student first”.	Pass
Load Details	Select option to load students from "StudentRecords.txt". It will print File Not Found if the file is missing. Print ”Invalid data format” if the data format is wrong	Select option to load students from "StudentRecords.txt". It will print File Not Found if the file is missing. Print ”Invalid data format” if the data format is wrong	Pass

Task 3

This will continue from the task 2 source code. This will contain the sub option 3 and 4 from main option8.

Test Case	Expected Result	Actual Result	Pass / Fail
Manage Results	This will display all the developed sub options for Option 8 including summary and report	This will display all the developed sub options for Option 8 including summary and report	Pass
Generate Summary	Display total students registered and how many students got above 40.	Display total students registered and how many students got above 40.	Pass
Generate Report	Generate a report including name, ID, three marks for each module, Total marks, Average marks, and Grade. This will generate for each and every student that has registered and stored their marks.	Generate a report including name, ID, three marks for each module, Total marks, Average marks, and Grade. This will generate for each and every student that has registered and stored their marks.	Pass
Sort by average	This is a helper method for sort students based on their average marks (Highest to lowest). This is used in the generating report method.	This is a helper method for sort students based on their average marks (Highest to lowest). This is used in the generating report method.	Pass

5. Task 04 – Testing – Discussion

Test Case description

Each menu option from one to eight in the student management system was tested to verify correct functionality. Each test case ensures each and every methods and every helper method is working properly. Each test case validate the accurate register of new student with new unique ID while stopping duplicates and handle of deletion, search and eliminate existing ID's, correct storage and loading without data loss or corruption. These test cases verify the system ability to generate accurate summary and complete report for every student.

By testing each method it ensures user friendly interactions and maintains data integrity, offering reliable way for managing student records. These test cases not only validate individual functionalities but also helps the overall quality assurance of the system.

Array & Classes. Which option is better and Why?

In the process of implementation of student management system, when comparing the use of 2D arrays and classes each way has its own advantages and disadvantages.
Using Arrays is simpler and more efficient in terms of memory and processing speed. This is less readable and less maintainable than the classes implementation. Arrays lack the ability to encapsulate related data. Operations such as searching, sorting and modifying records become more complex.

Making classes as 'student' and 'module' provides a more organized and maintainable structure. Classes encapsulate related data. This allows for easy addition of new attributes or methods. Furthermore object-oriented programming principles like inheritance and polymorphism helps to make the codebase more flexible and easier to understand.
In summary, classes offer significant benefits such as code organization, readability, and maintainability, making them the better choice for complex systems.

6. Self-Evaluation form

Criteria	Allocated marks	Expected marks	Student Feedback	Total
Task 1 Three marks for each option (1,2,3,4,5,6,7,8)	24	**22**	Fully implemented and working	**(30)**
Menu works correctly	6	**6**	Fully implemented and working	
Task 2 Student class works correctly	14	**10**	Fully implemented and working	**(30)**
Module class works correctly	10	**8**	Fully implemented and working	
Sub menu (A and B works well)	6	**5**	Fully implemented and working	
Task 3 Report – Generate a summary	7	**6**	Fully implemented and working	**(20)**
Report – Generate the complete report	10	**8**	Fully implemented and working	
Implementation of Bubble sort	3	**3**	Fully implemented and working	
Task 4 Test case coverage and reasons	6	**6**	Described on each test case and reasons	**(10)**
Write up on which version is better and why.	4	**2**	Described properly	
Coding Style (Comments, indentation, style)	7	**5**	Commented and kept indentation properly	**(10)**
Complete the self-evaluation form indicating what you have accomplished to ensure appropriate feedback.	3	**2**	Done as described	
Totals	100	**83**		**(100)**

7. References

- W3Schools:

W3Schools, 2024. Java Tutorial. Available at: https://www.w3schools.com/java/default.asp .

- Programiz:

Programiz, 2024. Java Classes and Objects. Available at: https://www.programiz.com/java-programming/class-objects.

- Lecture Materials:

Lecture slides, notes, tutorials from week one to 9. Available at: https://learning.westminster.ac.uk/ultra/courses/_97354_1/outline