ICT167 Principle of Computer Science

Table of Contents

[**1.** **Title** 3](#_Toc162530113)

[**2. Requirements/Specification** 4](#_Toc162530114)

[**3. User Guide** 5](#_Toc162530115)

[**4. Structure/Design/Algorithm** 5](#_Toc162530116)

[**5. Limitations** 5](#_Toc162530117)

[**6. Testing** 5](#_Toc162530118)

# **1. Title**

**Student Management System Documentation:** Authored by Keith Neo Kai Si on March 28, 2024, the Student Management System Documentation provides an overview of the functionalities of the Java-Based Student Management System. The system is designed to handle student data through CSV files and comprises several Java files, including **CsvHandler.java**, **Client.java**, **Student.java**, **Student\_Research.java**, **Student\_Course.java**, **Unit.java**, **Unit\_Course.java**, and **Research.java**. Its core objectives include efficiently loading student information from **student.csv**, adding marks from **student\_marks.csv**, student removal, information display, and exporting organized student data to a CSV file. Employing Java’s object-oriented programming principles, such as inheritance and polymorphism, the system effectively models various student types and their enrolment in different units. Furthermore, the documentation elaborates on Java’s file-handling capabilities, illustrating the system’s CSV file reading and writing processes for sustained data integrity. This comprehensive guide serves as a valuable reference for developers seeking to implement or enhance student management systems using Java.

# **2. Requirements/Specification**

The Student Management System is created to effectively handle and manage student information, such as importing student details from CSV files, adding marks to students, deleting students, displaying student information, and exporting organized student data back to a CSV file. The system assumes that the input CSV files are correctly formatted, with student.csv containing student type, name, and ID and student\_marks.csv containing student IDs followed by marks or course and marks. The desired output is a sorted list of students with their marks included, which can be saved to a new CSV file.

The system’s design utilizes fundamental object-oriented programming concepts:

**1. Inheritance**: The Student class acts as the base class, with Student\_Research and Student\_Course extending it to represent different types of students. This allows for reusing code and adding specific attributes and methods for research and course students.

**2. Polymorphism**: The reportGrade method in both Student\_Research and Student\_Course classes overrides the reportGrade method in the Student class, showcasing polymorphism. This enables different implementations of the method based on the student type.

**3. Dynamic Binding**: The system employs dynamic binding to determine the appropriate method to call at runtime. For instance, when calling reportGrade on a student object, the JVM decides whether to call the Student, Student\_Research, or Student\_Course implementation based on the actual type of the student object.

**4. Sorting Algorithm**: The system utilizes an insertion sort algorithm to arrange students by their IDs. This is done using the sortStudents method, which sorts the list of students in ascending order of their IDs.

**5. Handling CSV Files**: The system uses Java’s built-in Scanner class to read from CSV files. It reads each line of the student\_marks.csv file, separates the line into components based on the comma separator, and processes the data accordingly. For writing to a CSV file, the system uses PrintWriter to output the sorted list of students to a new file.

This documentation presents a comprehensive overview of the requirements, specifications, and implementation details of the Student Management System, ensuring that developers and users comprehend the system’s functionality and how it fulfills the specified needs.

# **3. User Guide**

**Compilation and Execution:**

1. **IDE:** [Eclipse IDE 2024-03](https://www.eclipse.org/downloads/packages/installer) is used for this program.

2. **Compilation:** To begin compiling the Java program in Eclipse, first open the Eclipse IDE on your computer. After launching Eclipse, navigate to the top left corner and click “File.” Select " New " from the dropdown menu and click “Java Project.” In the “New Java Project” menu, enter a name for the project, such as “Assignment\_2”. Make sure to deselect the “Use default location” option, and then navigate to the folder where the Java source codes are stored, “ICT167-Assignment-2-Keith-Neo-35107628”. Inside this folder, find the “java\_codes” folder and select it as the location for the new project. Ensure that the “Create module-info.java file” option is not checked. Once all the settings are configured, click “Finish” to finalize the process.

**A screenshot of a computer

Description automatically generated**

3. **Execution:** Navigate to the Package Explorer panel after successfully creating the Java project in Eclipse. Right-click on the Java project that was made. From the dropdown menu that appears, select “Run As” and then choose “Java Application.” This action will execute the Java program within the Eclipse environment.

**A screenshot of a computer

Description automatically generated**

**Using the Program:**

* When the program is launched, users will receive a welcome message containing student information, indicating the current student profile that developed the program. After the student information, users will be presented with the program's main menu, providing various options for interacting with student data and carrying out different tasks.

**A screen shot of a computer

Description automatically generated**

* If the user enters nothing, letters, or unknown options, the system will display “Invalid input. Try again.” Or “Unknown choice. Try again.” This will prompt the user to retry the input.

A screenshot of a computer program

Description automatically generated

* If option 1 is selected, the program will exit and display a farewell message

A screen shot of a computer

Description automatically generated

* If option 2 is selected, the program will dynamically aggregate the marks and units (for course students). The program will display “Student <count> marks loaded” for each student, dynamically loading the marks based on the student type from student\_marks.csv. It’s important to note that marks for course students contain a unit, while marks for research students do not include a unit. Upon completion, the program will display “Student marks loaded,” indicating that all the student marks have been successfully loaded.

**A screenshot of a computer program

Description automatically generated**

* If option 3 is chosen, the program will ask the user to input a student ID for deletion. If the user enters nothing, the program will exit this option and return to the main menu. If the user enters letters, the program will display “Invalid ID. Try again.” The program will attempt to locate the student if a valid student ID is entered. If the student is found, the program will display “Student found.” And present a confirmation option to the user. If the student is not found, the program will display “Student not found. Try again.” And prompt the user for another student ID.

Once the student is found and the user is at the confirmation menu, selecting “yes” or “y” (not case-sensitive) will result in the program removing everything related to the student and displaying “Student removed.” Choosing “no” or “n” will display “Student not removed.” After this, the program will display the main menu.

A screenshot of a computer program

Description automatically generated

**A computer screen shot of a computer error

Description automatically generated**

**A computer screen shot of a computer error

Description automatically generated**

* If option 4 is chosen, the program will display each student's type, full name, and student number. If option 2 (Add marks from file) is previously selected, resulting in the student marks being loaded, the program will display the Unit ID for course students and the string “Has Unit” for research students.

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated A computer screen shot of a student type

Description automatically generated

* If option 5 is chosen, the program will show the number, of course, students who scored equal to or higher than the average, as well as the number of course students with scores lower than the average. The program will only display these numbers if option 2(Add marks from file) was previously selected. If option 2 was not previously selected, the program will display 0 for each count.

A screen shot of a computer

Description automatically generated A screen shot of a computer

Description automatically generated

* If option 6 is chosen, the user will be asked to enter a student ID to view the student report. If the user enters a letter, an error message will be displayed, prompting the user to enter the student ID again. Upon entering a valid student ID, the program will attempt to find the student with the given ID. If the student is not found, the program will prompt “Student not found. Try again” and ask the user for the student ID again. If the student is found, the program will display the student's information using the reportGrade() function. However, if option 2 (Add marks from file) was not previously selected, the program will display “Student marks not loaded” and return to the main menu.

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated A screen shot of a computer

Description automatically generated

* If option 7 is chosen, the program will use the insertion sort algorithm to sort the student array list based on the student’s IDs. Upon completion of the sorting process, the program will display “Student sorted.”

A screen shot of a computer

Description automatically generated

* If option 8 is chosen, the student array list will only be saved if option 7 was previously selected to sort the students in ascending order. If the student array list is not sorted, the program will display “Student not sorted” and return to the main menu. If the student array list is sorted, the program will indicate that the sorted student array list has been saved to “csv\_files/sorted\_students.csv.”

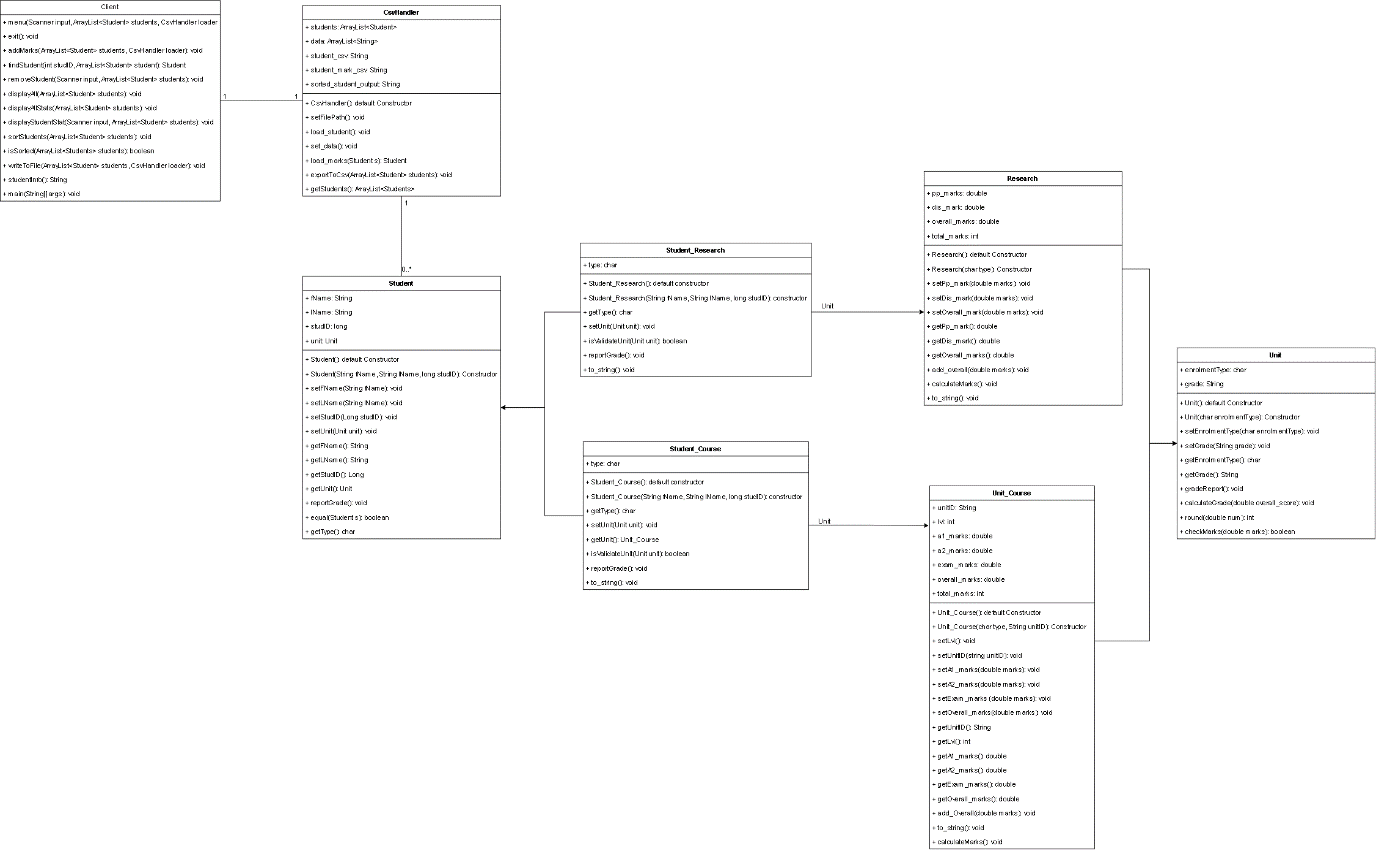
A screen shot of a computer

Description automatically generated A screen shot of a computer

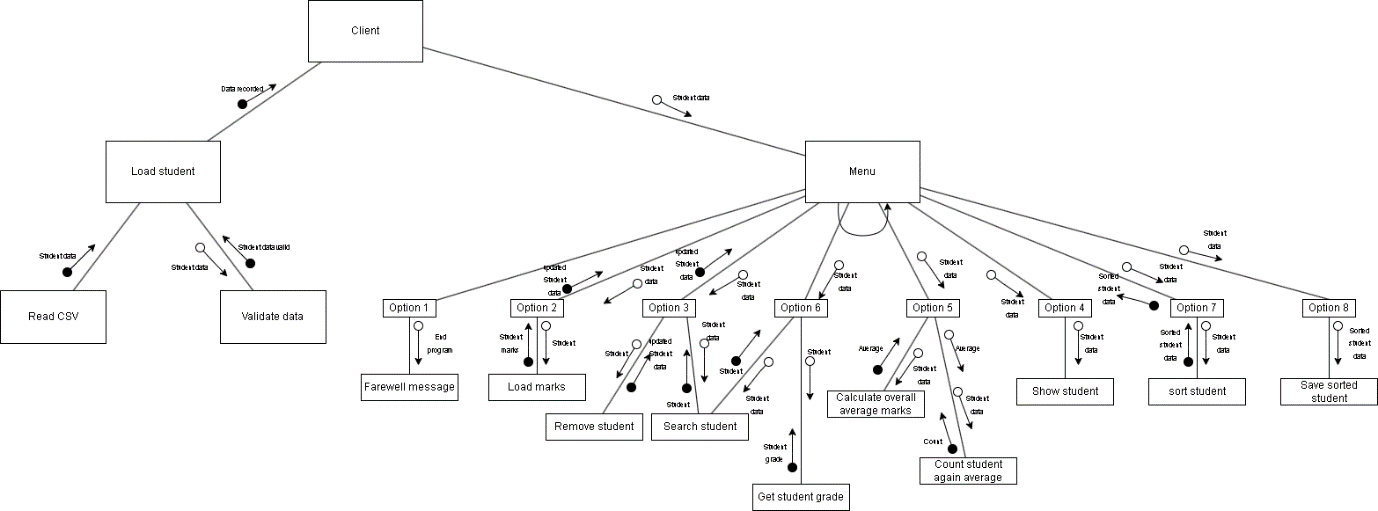
Description automatically generated

**4. Structure/Design/Algorithm**

**UML for the Client program**

****

**Structure chart for the Client program**



**Student Class**

The Student class serves as the base class for all students in the system. It encapsulates the basic attributes of a student, such as first name, last name, student ID, and a reference to a Unit object. The class provides constructors for creating a student with default values or specific values, getter and setter methods for accessing and modifying these attributes, and a reoportGrade() method that prints a default message indicating no grade is available. The equal method checks if two Student objects have the same student ID, and the getType() method returns a default character “N,” indicating a generic student type.

**Student\_Course Class**

The Student\_Course class extends the Student class, indicating that it is a specific type of student. It introduces a new attribute type with a default value of ‘C,’ representing a course student. The class overrides the getType() method to return this specific type. It also overrides the setUnit() method to validate that the unit being set is an instance of Unit\_Course, ensuring that only course units can be associated with course students. The reportGrade() method in this class prints detailed information about the student, including their type, name, student number, unit ID, overall marks, and final grade if a course unit is associated. The to\_string() method provides a string representation of the student’s information, including whether they have an associated course unit.

**Student\_Research Class**

Similarly, the Student\_Research class extends the Student class, representing a research student. It introduces a type attribute with a default value of ‘R,’ indicating a research student. The getType() method is overridden to return this specific type. The setUnit() method is overridden to validate that the unit being set is an instance of Research, ensuring that only research units can be associated with research students. The reportGrade() method prints detailed information about the student, including their type, name, student number, overall marks, and final grade, if a research unit is associated. The to\_string() method provides a string representation of the student's information, including whether they have a research unit associated with it.

**Unit Class**

The Unit class represents a generic unit of study or course in which a student can enroll. It encapsulates the basic attributes of a unit, such as the enrolment type and grade. The enrolment type is a character that indicates the type of enrolment (e.g., ‘N’ for not enrolled, ‘C’ for course, ‘R’ for research). The grade is a string that holds the unit's grade, with a default value of “NA,” indicating that the grade is unavailable. The class provides constructors for creating a unit with default values or a specific enrollment type. It also includes getter and setter methods for the enrolment type and grade. The gradeReport() method prints the unit's grade with a special message if the grade is unavailable. The calculateGrade() method calculates the grade based on an overall score, assigning a letter grade based on the score’s range. The checkMarks() method validates that the marks are within the acceptable range of 0 to 100.

|  |  |  |  |
| --- | --- | --- | --- |
| Pseudocode Explanation: calculateGrade() | | | |
| 1. | **Function** calculateGrade(overall\_score) | | |
| 2. |  | **IF** overall\_score is not a valid number (e.g., not within 0 to 100) | |
| 3. |  |  | Return “Invalid score” |
| 4. |  | **END IF** | |
| 5. |  |  | |
| 6. |  | **IF** overall\_score >= 80 | |
| 7. |  |  | **SET** grade to “HD” (High Distinction) |
| 8. |  | **ELSE IF** overall\_score >= 70 | |
| 9. |  |  | **SET** grade to “D” (Distinction) |
| 10. |  | **ELSE IF** overall\_score >= 60 | |
| 11. |  |  | **SET** grade to “C” (Credit) |
| 12. |  | **ELSE IF** overall\_score >= 50 | |
| 13. |  |  | **SET** grade to “P” (Pass) |
| 14. |  | **ELSE** | |
| 15. |  |  | **SET** grade to “N” (No grade) |
| 16. |  | **END IF** | |
| 17. |  | **RETURN grade** | |
| 18. | **END FUNCTION** | | |

**Unit\_Course Class**

The Unit\_Course class extends the Unit class, indicating that it represents a course unit. It introduces additional attributes specific to course units, such as the unit ID, level, and marks for assignments and exams. The class provides constructors for creating a course unit with default values or specific values for the type and unit ID. It also includes methods for setting and getting these additional attributes. The setLvl() method extracts the level from the unit ID, which is assumed to be the fourth character of the ID. The setUnitID() method the unit ID and recalculates the level. The setA1\_marks(), setA2\_marks(), and setExam\_marks() methods allow setting marks for assignments and exams, updating the overall marks, and recalculating the grade. The calculateMarks() method calculates the grade based on the overall marks and updates the grade accordingly. The to\_string() method provides a string representation of the course unit’s information.

|  |  |  |  |
| --- | --- | --- | --- |
| Pseudocode Explanation: calculateMarks() for Unit\_Course class | | | |
| 1. | **Function** calculateMarks() | | |
| 2. |  | **IF** a1\_marks, a2\_marks, and exam\_marks are all valid numbers (0 to 100) | |
| 3. |  |  | calculate overall\_marks by summing a1\_marks, a2\_marks, and exam\_marks |
| 4. |  |  | Calculate grade based on overall\_marks |
| 5. |  |  | Updates grade attributes with the calculated grade |
| 6. |  | **End If** | |
| 7. | **End Function** | | |

**Research Class**

The Research class also extends the Unit class, representing a research unit. It introduces attributes specific to research units, such as proposal marks, dissertation marks, and overall marks. The class provides constructors for creating a research unit with default or particular values for the type. It also includes methods for setting and getting these additional attributes. The setPp\_mark(), setDis\_mark(), and setOverall\_mark() methods allow setting marks for the proposal, dissertation, and overall research project, updating the overall marks and recalculating the grade. The calculateMarks method calculates the grade based on the overall marks and updates the grader accordingly. The to\_string() method provides a string representation of the research unit’s information.

|  |  |  |  |
| --- | --- | --- | --- |
| Pseudocode Explanation: calculateMarks() for Research class | | | |
| 1. | **Function** calculateMarks() | | |
| 2. |  | **IF** pp\_mark, dis\_mark are valid numbers (0 to 100) | |
| 3. |  |  | calculate overall\_marks by summing pp\_mark and dis\_mark |
| 4. |  |  | Calculate grade based on overall\_marks |
| 5. |  |  | Updates grade attributes with the calculated grade |
| 6. |  | **End If** | |
| 7. | **End Function** | | |

**CsvHandler Class**

The CsvHandler class in Java is designed to manage student data from CSV files, including loading student information, processing marks, and exporting sorted student data to a CSV file. It supports different types of students, such as Student\_Course and Student\_Research, by distinguishing them based on a prefix in the CSV file. Upon initialization, the class set file paths for student data, marks, and the output file for sorted students. It then loads student data, where each line from the student CSV file is read, and either a Student\_Course or Student\_Research object is created based on the prefix, adding them to an ArrayList of students.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pseudocode Explanation: load\_student() | | | | | |
| 1. | **Function** load\_student() | | | | |
| 2. |  | **Try** | | | |
| 3. |  |  | **Open** fileReader for reading from student\_csv file | | |
| 4. |  |  | **While** there are more lines in the file | | |
| 5. |  |  |  | **Read** the next line from the file | |
| 6. |  |  |  | **Split** the line into an array of strings using comma as the delimiter | |
| 7. |  |  |  | **If** the first element of the array is “C” | |
| 8. |  |  |  |  | **Create** a new Student\_Course object with the second, third, and fourth element of the array |
| 9. |  |  |  |  | **Add** the new Student\_Course object to the student's ArrayList |
| 10. |  |  |  | **Else** If the first element of the array is “R” | |
| 11. |  |  |  |  | **Create** a new Student\_Research object with the second, third, and fourth elements of the array |
| 12. |  |  |  |  | **Add** the new Student\_Research object to the student's ArrayList |
| 13. |  |  | **End** **While** | | |
| 14. |  |  | **Close** fileReader | | |
| 15. |  | **Catch** any **exception** | | | |
| 16. |  |  | **Print** an error message | | |
| 17. |  | **End** **Try** | | | |
| 18. | **End Function** | | | | |

The set\_data() method loads the student marks from another CSV file into an ArrayList of data strings. The load\_marks() method iterates through the marks data to find and set marks for each student based on their ID, supporting different types of students by checking the instance of the student object and setting the marks accordingly.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pseudocode Explanation: load\_marks(Student s) | | | | | |
| 1. | **Function** load\_marks(Student s) | | | | |
| 2. |  | **For** **each** line in the data ArrayList | | | |
| 3. |  |  | **If** the line contains the student’s ID | | |
| 4. |  |  |  | **If** s in an instance of Student\_Course | |
| 5. |  |  |  |  | **Split** the line into an array of strings using comma as the delimiter |
| 6. |  |  |  |  | **Create** a new Unit\_Course object with the second element of the array |
| 7. |  |  |  |  | **Set** the A1, A2, and Exam marks of the Unit\_Course object using the third, fourth, and fifth elements of the array |
| 8. |  |  |  |  | **Set** the Unit of the Student\_Course object to the new Unit\_Course object |
| 9. |  |  |  |  | **Return** the Student\_Course object |
| 10. |  |  |  | **Else IF** s is an instance of Student\_Research | |
| 11. |  |  |  |  | **Split** the line into an array of strings using comma as the delimiter |
| 12. |  |  |  |  | **Create** a new Research object |
| 13. |  |  |  |  | **Set** the PP and Dis marks of the Research object using the second and third elements of the array |
| 14. |  |  |  |  | **Set** the Unit of the Student\_Research object to the new Research object |
| 15. |  |  |  |  | **Return** the Student\_Research object |
| 16. |  |  | **End If** | | |
| 17. |  | **End For** | | | |
| 18. |  | **Return** the updated student object s | | | |
| 19. | **End Function** | | | | |

Finally, the exportToCsv() method writes the sorted list of student objects back to a CSV file, formatting each student’s information into a string and writing it to the output file.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Pseudocode Explanation: exportToCsv(ArrayList<Student> students) | | | | |
| 1. | **Function** exportToCsv(ArrayList<Student> students) | | | |
| 2. |  | **Try** | | |
| 3. |  |  | **Create** a new PrintWriter object for writing to the sorted\_student\_output file | |
| 4. |  |  | **For each** Student object in the students ArrayList | |
| 5. |  |  |  | **Format** the student’s information into a string with the format: type, full name, student ID |
| 6. |  |  |  | **Write** the formatted string to the sorted\_student\_output file |
| 7. |  |  | **End For** | |
| 8. |  | **Catch** any exception | | |
| 9. |  |  | **Print** an error message | |
| 10. |  | **Finally** | | |
| 11. |  |  | **If** the PrintWriter object is not null | |
| 12. |  |  |  | **Close** the PrintWriter object |
| 13. |  |  | **End If** | |
| 14. |  | **End Try** | | |
| 15. | **End Function** | | | |

**Client Class**

The Client class in Java is a comprehensive tool designed to manage student data through a user-friendly interface. It incorporates several functionalities to facilitate the efficient handling of student information, including adding marks from a file, removing students, displaying student information, and saving sorted student data to a CSV file. The class leverages Java's object-oriented programming features, file-handling capabilities, and user input processing to ensure a seamless experience for users. The main interface for user interaction is the menu method, which presents a menu with various options for managing student data, such as adding marks, removing students, and displaying information. This method operates within a while loop, continuously displaying options until the user chooses to exit, effectively managing user input and calling the appropriate methods based on the user's choice. The exit method is responsible for terminating the program, printing a farewell message to the user before exiting, effectively ending the session. The addMarks() method iterates through the list of students, loading marks for each student from a CSV file using the load\_marks() method of the CsvHandler, and then updates the student's marks in the ArrayList, ensuring that each student's marks are accurately loaded and updated.

|  |  |  |  |
| --- | --- | --- | --- |
| Pseudocode Explanation: addMarks(ArrayList<Student> students, CsvHandler loader | | | |
|  | **Function** addMarks(ArrayList<Student> students, CsvHandler loader) | | |
|  |  | **For each** student in students | |
|  |  |  | **Set** student to the result of loader.load\_marks(student) |
|  |  |  | **Print** “student marks loaded.” |
|  |  | **End For** | |
|  | **End Function** | | |

This method is crucial for maintaining up-to-date student records. The findStudent() method searches for a student by their ID within the ArrayList of students, returning the student object if found; otherwise, it returns null. This method is essential for operations that require identifying specific students. The removeStudent() method prompts the user for a student ID. If found, it removes the corresponding student from the ArrayList, including confirmation before removal, ensuring that students are only removed upon user confirmation.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Pseudocode Explanation: removeStudent(Scanner input, Arraylist<Student> students) | | | | | | |
| 1. | **Function** removeStudent(Scanner input, Arraylist<Student> students) | | | | | |
| 2. |  | **String** userInput | | | | |
| 3. |  | **Student** studentToRemove | | | | |
| 4. |  | **Char** confirmation | | | | |
| 5. |  | **While True** | | | | |
| 6. |  |  | **Print** “Enter nothing to exit,” | | | |
| 7. |  |  | **Print** “Enter student ID to delete” | | | |
| 8. |  |  | **Print** “>> “ | | | |
| 9. |  |  | **Set** userInput to input.nextLine() | | | |
| 10. |  |  | **Print** “\n” | | | |
| 11. |  |  | **If** userInput **is empty** | | | |
| 12. |  |  |  | **Break** | | |
| 13. |  |  | **End If** | | | |
| 14. |  |  | **Try** | | | |
| 15. |  |  |  | **Set** studentToRemove to findStudent(userINput, student) | | |
| 16. |  |  |  | **If** studentToRemove is **not null** | | |
| 17. |  |  |  |  | **Break** | |
| 18. |  |  |  | **Else** | | |
| 19. |  |  |  |  | **Print** “ Student not found. Try again.” | |
| 20. |  |  |  | **End IF** | | |
| 21. |  |  | **Catch Exception** | | | |
| 22. |  |  |  | **Print** “Invalid ID. Try again.” | | |
| 23. |  |  | **End Try** | | | |
| 24. |  | **End While** | | | | |
| 25. |  | **If** studentToRemove is **not null** | | | | |
| 26. |  |  | **While True** | | | |
| 27. |  |  |  | **Print** “Student found.” | | |
| 28. |  |  |  | **Print** “Do you want to continue? [y/n]” | | |
| 29. |  |  |  | **Print** “>> “ | | |
| 30. |  |  |  | **Set** confirmation to input.nextLine().charAt(0) | | |
| 31. |  |  |  | **If** confirmation is **‘y’** or confirmation is **‘n’** | | |
| 32. |  |  |  |  | **If** confirmation is **‘y’** | |
| 33. |  |  |  |  |  | **Remove** studentToRemove from students |
| 34. |  |  |  |  |  | **Print** “student removed.” |
| 35. |  |  |  |  |  | **Break** |
| 36. |  |  |  |  | **Else** | |
| 37. |  |  |  |  |  | **Print** “student not removed.” |
| 38. |  |  |  |  |  | **Break** |
| 39. |  |  |  |  | **End IF** | |
| 40 |  |  |  | **Else** | | |
| 41. |  |  |  |  | **Print** “Invalid choice. Try again.” | |
| 42. |  |  |  | **End If** | | |
| 43. |  |  | **End While** | | | |
| 44. |  | **End If** | | | | |
| 45 | **End Function** | | | | | |

This method provides a secure way to manage student data. The displayAll() method iterates through the ArrayList of students and prints their information, differentiating between Student\_Course and Student\_Research objects and calling their respective to\_string() methods, providing a comprehensive view of all students.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Pseudocode Explanation: displayAll() | | | | |
| 1. | **Function** displayAll(ArrayList<Student> students) | | | |
| 2. |  | **For each** student in students | | |
| 3. |  |  | **If** student is an **instance** of **Student\_Course** | |
| 4. |  |  |  | **Cast** student to **Student\_Course** |
| 5. |  |  |  | **Call** student.to\_String() |
| 6. |  |  | **Else If** student is an **instance** of **Student\_Research** | |
| 7. |  |  |  | **Cas**t student to **Student\_Research** |
| 8. |  |  |  | **Call** student.to\_String() |
| 9. |  |  | **End IF** | |
| 10. |  | **End For** | | |
| 11. | **End Function** | | | |

This method is helpful for quickly reviewing all student data. The displayAllStats() method calculates and displays statistics for course students, such as the number of students scoring higher or equal to the average and those scoring lower than the average, providing insights into the performance of course students.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pseudocode Explanation: displayAllStats(ArrayList<Student> students) | | | | | |
| 1. | **Function** displayAllStats(ArrayList<Student> students) | | | | |
| 2. |  | **ArrayList<Student\_Course>** tempSC | | | |
| 3. |  | **Double** totalOverall | | | |
| 4. |  | **Int** higherThanAvg | | | |
| 5. |  | **Int** lowerThanAvg | | | |
| 6. |  | **For each** student in students | | | |
| 7. |  |  | **If** student is an **instance** of **Student\_Course** | | |
| 8. |  |  |  | **Try** | |
| 9. |  |  |  |  | **Cast** student to **Student\_Course** |
| 10. |  |  |  |  | **Add** student to **tempSC** |
| 11. |  |  |  |  | **Add** student’s overall marks to **totalOverall** |
| 12. |  |  |  | **Catch Exception** | |
| 13. |  |  |  |  | **Continue** |
| 14. |  |  |  | **End Try** | |
| 15. |  |  | **End If** | | |
| 16. |  | **End For** | | | |
| 17. |  | **Calculate** average as totalOverall divided by the size of **tempSC** | | | |
| 18. |  | **For each** student in tempSC | | | |
| 19. |  |  | **Try** | | |
| 20. |  |  |  | **If** student’s overall marks is **greater than or equal** to average | |
| 21. |  |  |  |  | **Increment** higherThanAvg |
| 22. |  |  |  | **Else** | |
| 23. |  |  |  |  | **Increment** lowerThanAvg |
| 24. |  |  |  | **End IF** | |
| 25. |  |  | **Catch Exception** | | |
| 26. |  |  |  | **Continue** | |
| 27. |  |  | **End Try** | | |
| 28. |  | **End For** | | | |
| 29. |  | **Print** the number of students scoring high or equal to the average | | | |
| 30. |  | **Print** the number of students scoring lower than the average | | | |
| 31. | **End Function** | | | | |

The writeToFile() method checks if the students are sorted and, if so, save the sorted students to a CSV file using the exportToCsv() method of the CsvHandler, ensuring that only sorted student data is saved, maintaining the integrity of the data file.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Pseudocode Explanation: sortStudents() | | | | |
| 1. | **Function** sortStudents(ArrayList<Student> students) | | | |
| 2. |  | **For** I from 1 to the size of **students - 1** | | |
| 3. |  |  | **Set** key to the student object at **index I in students** | |
| 4. |  |  | **Set** j to i -1 | |
| 5. |  |  | **While** j is **greater than or equal** to 0 and the student object at index j in students has a **greater student ID** than key | |
| 6. |  |  |  | **Move** the student object at index j+1 in student to index j in students |
| 7. |  |  |  | **Decrement** j by 1 |
| 8. |  |  | **End While** | |
| 9. |  |  | **Place** key at **index j+1** in students | |
| 10. |  | **End For** | | |
| 11. |  | **Print** “Student sorted.” | | |
| 12. | **End Function** | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Pseudocode Explanation: writeToFile() | | | |
| 1. | **Function** writeToFile(ArrayList<Student> students, CsvHandler loader | | |
| 2. |  | **If** isSorted(students) is **true** | |
| 3. |  |  | **Call** loader.exportToCsv(students) |
| 4. |  |  | **Print** “Sorted students have been saved to csv\_files/sorted\_students.csv” |
| 5. |  | **Else** | |
| 6. |  |  | **Print** “Students not sorted.” |
| 7. |  | **End If** | |
| 8. | **End Function** | | |

# **5. Limitations**

One significant limitation is handling user input for option 3 in the confirmation menu within the client class. The current implementation trims the input and selects the first character to determine whether to confirm or stop the operation. This approach is problematic because it does not account for user inputs that start with either ‘y’ or ‘n’ but contain additional characters. For example, if a user enters ‘yes’ or ‘no,’ the program will still interpret it as a confirmation or cancellation due to how it processes the input. This limitation can lead to confusion and incorrect program behavior, as the user’s intention might not be accurately captured.

A screen shot of a computer

Description automatically generated

Another limitation is that the CsvHandler class does not support dynamic file paths or the ability to handle multiple CSV files at once due to project limitations. This limitation restricts the program’s flexibility and scalability, as it can only work with a predefined set of fields.

# 

# **6. Testing**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Test description/justification – what is the test for and why this particular test.** | **Actual data for**  **this test** | **Expected output** | **Actual program output when test is carried out** | **Test run outcome –**  **Pass/Fail** |
| 1 | The program should only allow selections that are displayed on the menu; any additional selections should raise an error. | 0 / ‘Foobar’ | The program raised an error and should request a new input | A screenshot of a computer program  Description automatically generated  A screen shot of a computer  Description automatically generated | Pass |
| 2 | The program should provide information that students' marks have been added when the user input 2 | 2 | The program displayed a message indicating that student marks had been added. | A screenshot of a computer program  Description automatically generated | Pass |
| 3 | The program should prompt for the student ID to be deleted when the user inputs 3 to show its functionality | 3 | The program displayed an input prompt for user to enter the student ID | A screen shot of a computer program  Description automatically generated | Pass |
| 4 | The program should delete the student ID inputted and display a confirmation message | 10669106 | The program displays a confirmation message to delete a student | A screen shot of a computer error  Description automatically generated | Pass |
| 5 | After confirmation, the program should delete the inputted student ID and display a message. | “Yes” / “y” | The program should display a message that confirms the student has been removed. | A computer screen shot of a computer error  Description automatically generated  A computer error message  Description automatically generated | Pass |
| 6 | The program should prompt for another input when user inputs an incorrect option. | “ni” | The program should ask user to confirm the deletion of the student again | A computer screen shot of a computer error  Description automatically generated | Fail |
| 7 | After confirmation, the program should not delete the inputted student ID and display a message. | “No” / “n” | The program should display a message stating that the student was not removed. | A screenshot of a computer program  Description automatically generated  A screen shot of a computer  Description automatically generated | Pass |
| 8 | The program should display all student information when user input 4 | 4 | All student information is displayed | A screenshot of a computer  Description automatically generated | Pass |
| 9 | The program should display course statistics when user inputs 5 | 5 | Overall statistics of course students were displayed | A screen shot of a computer  Description automatically generated | Pass |
| 10 | The program should request student ID to view respective students' statistics | 6 | The program requested for student ID. | A screen shot of a computer  Description automatically generated | Pass |
| 11 | The program should display respective student statistics when given the correct student ID | 36626085 / 78188612 | Student’s statistics to be displayed. | A screenshot of a computer  Description automatically generated  A screenshot of a computer program  Description automatically generated | Pass |
| 12 | The program should return to the main menu if user wants to avoid checking student statistics. | “ ” | Program to return to the main menu | A screenshot of a computer program  Description automatically generated | Pass |
| 13 | The program should exit and display a thank you message after the user inputs 1 | 1 |  | A screen shot of a computer  Description automatically generated | Pass |
| 14 | The program should sort the student ArrayList when user input 7 | 7 | The program should display a message to indicate that the program has sorted the student list. | A screenshot of a computer program  Description automatically generated | Pass |
| 15 | The program should save the sorted student ArrayList when user inputs 8 | 8 | The program should display a message indicating that the file has been saved. | A screenshot of a computer program  Description automatically generated | Pass |
| 16 | The program should not allow the user to save a sorted student list if the user has not sorted the students | 8 | The program should display a message to indicate that the student still needs to be sorted. | A screenshot of a computer program  Description automatically generated | Pass |
| 17 | The program should show an error message for an invalid student ID inputted by the user when they want to delete a student. | 12345678 | The program should display an error message stating that the student is not found | A screenshot of a computer program  Description automatically generated | Pass |