

**School of Engineering and Information Technology**

**ASSIGNMENT COVER SHEET**

* Please complete and insert this form as the first page of EACH electronic assignment.
* Submit the assignment with attached coversheet electronically as per the instructions in the Assignment Question sheet.
* Please make sure you keep a copy of the assignment.

**Student Details**

|  |  |  |  |
| --- | --- | --- | --- |
| Surname | Tee | Given name | Yee Kang |
| Student Number | 34315323 | Email | Yeekang88@gmail.com |

**Assignment details**

|  |  |  |  |
| --- | --- | --- | --- |
| Unit name | Principles of Computer Science | Unit Code | ICT167 |
| Unit Coordinator | Kevin Wong | Tutor/Tutorial time | Aaron Yeo Sze Wee |
| Due date/time | 31/Jul/2021 – 11:59PM | Submission date | 30/Jul/2021 |
| Assignment title | FT MUR T221 ICT167 C – Assignment 2 | | |
| Other information | - | | |

***All forms of plagiarism, cheating and unauthorized collusion are regarded seriously by the University and could result in penalties including failure in the unit and possible exclusion from the University. If in doubt, please contact the Unit Coordinator.***

**Student’s Declaration**

Please double click on all the check boxes.

✓

Except where I have indicated, the work I am submitting in this assignment is my own work and has not been submitted for assessment in another unit.

✓

This submission complies with Murdoch University policies regarding plagiarism, cheating and collusion.

✓

I have retained a copy of this assignment for my own records.

**External Documentation**

* **Title :** FT MUR T221 ICT167 C – Assignment 2
* **Author :** Tee Yee Kang
* **Date :** 31/7/2021
* **File Name :** FTC-34315323-Assignment2
* **Purpose :** This program allowed user to create either CourseWorkStudent or ResearchStudent object by reading the value from the .csv file. The program also contains several methods for the user to conduct some tasks.

1. **Requirements/Specification:**

\*Version: Java SE Development Kit 13.0.2

This program contains 4 different classes which is the Client class, Student class (base class) and two child class: CourseWorkStudent class and ResearchStudent class. The child class inherit the variables and methods in the base class. Therefore, the user required the values of base class variable in order to create CourseWorkStudent or ResearchStudent object. This program allowed user to create CourseWorkStudent or ResearchStudent object by reading all the values from the csv file instead of entering keyboard value. The program will then store all the object into an ArrayList. At the beginning of the program will ask user to choose between CourseWorkStudent or ResearchStudent, so that the program will only deal with one type of student object. After created the CourseWorkStudent or ResearchStudent object, the program will display a menu for user to conduct 10 different tasks. The tasks included:

* Task 1: Exit the program
* Task 2: Read all marks from csv file
* Task 3: Remove student from the list based on studentID
* Task 4: Output the details of student
* Task 5: Compute and output the overall mark and grade of student
* Task 6: Find the number of students obtained an overall mark equal to or above and below the average overall mark
* Task 7: Output the details of student based on the given studentID
* Task 8: Output the details of student based on the given student’s name
* Task 9: Sorted the ArrayList into ascending order of studentID
* Task 10: Output the sorted ArrayList from to a csv file

After each task is completed, the program will continue to execute the menu until the user selects task 1 to exit the program. The program also has the ability to handle some error situations such as invalid input, Exception, etc.

1. **User Guide:**

|  |  |  |
| --- | --- | --- |
| 1. | To run the program, users only need to insert the entire file into Java SE (JDK). |  |
| 2. | At the beginning, the program will ask the user to choose between coursework student and research student. (Enter 0 for coursework student, 1 for research student). If the user enters any number apart from 0 or 1, the program will display an error message and prompt the user to enter again. |  |
| 3. | After user has choose a valid option (0 or 1), program will display the type of the student followed by the menu. The menu includes 10 different tasks for users to conduct some queries. The menu will continue to execute after each task is completed.  \*The example output here I choose coursework student |  |
| 4. | To terminate the program, the user can select task 1. After selecting task 1, the program will display a message to notify the user before terminate. |  |

|  |  |  |
| --- | --- | --- |
| 5. | After selecting the student type, the detailed information of the student only includes the name, student ID and date of birth. The user can select task 2 to read all the marks of the student from the file according to user’s choice between coursework and research student. Then the program will display a success message to inform the user. |  |
| 6. | After reading the marks of students, the user able to display all the student’s information by selecting the task 4. Task 4 will display all the information of coursework or research student including the marks, overall marks and grade. |  |
| 7. | The program also provides a function that allows users to remove students from the student list by selecting task number 3. In order to complete the task, the program will prompt the user to enter the studentID of the student.  After the user enters the studentID, the program will send a confirmation message to confirm whether to delete the student. The user has to enter yes (case not sensitive) to complete the analysis and query. The program will display a success message to inform the user. After that, user can enter task 4 to check if the student is successfully removed.  \*Example here will remove student 1111 (studentID). |  |

|  |  |  |
| --- | --- | --- |
| 8. | In task number 3, if the user tries to remove a non-existing student, the program will display a message to tell the user.  \*Example here try to remove studentID 9999 which is a non-existing student. |  |
| 9. | In addition, if the user enters “no” (case not sensitive) in the confirmation remove stage, the program will display a message and ignore the query. |  |
| 10. | However, if the user enters any value apart from yes or no, the program will display an error message to notify the user and go back to the menu stage.  \*Example here I enter “gg” instead of yes or no |  |
| 11. | This program also provides a function (task 5) to help the user to calculate the **individual** overall mark and grade of all coursework or research student. After the user select task 5, the program will display the **individual** overall marks and grade of all coursework or research student. If the user previously removed any student, that particular student will not appear in the list. |  |

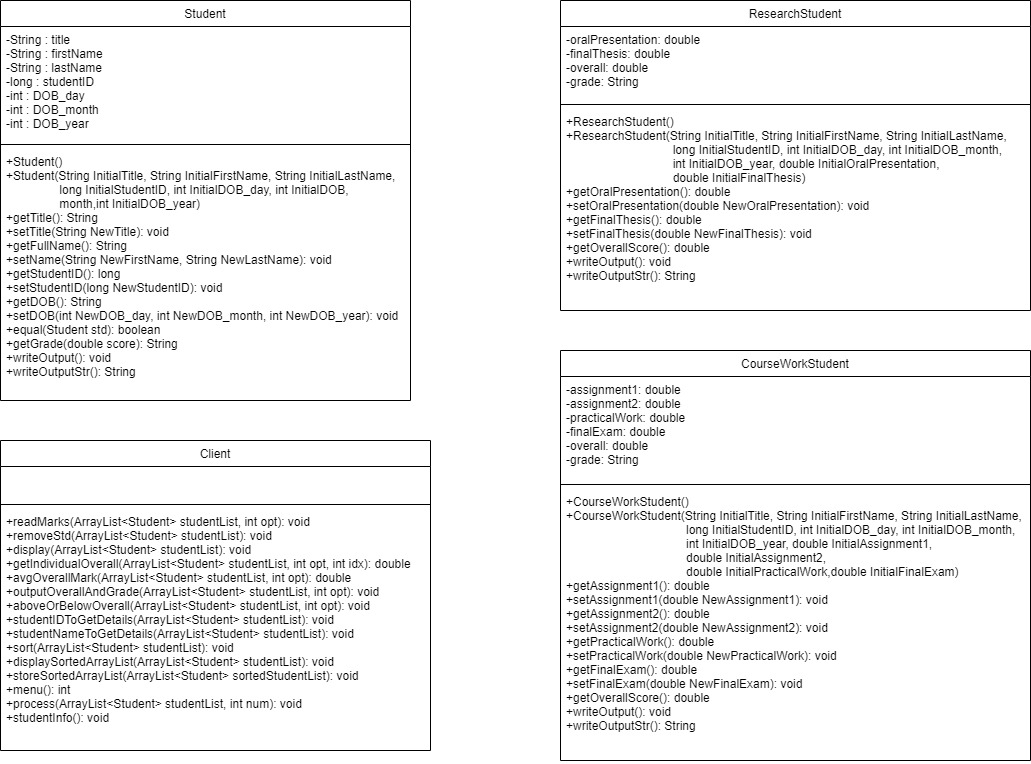
|  |  |  |
| --- | --- | --- |
| 12. | This program also provides a function to help user calculate the **average** overall mark and grade of all student. The user can select task 6 to calculate this and find the number of coursework or research students obtained an overall mark equal to or above the **average** overall mark and how many obtained an overall mark below the **average** overall mark. The program will then do the calculation and display the output to the screen. |  |
| 13. | Besides that, the user can also select task 7 to display the information of a specific student including all the marks and grade by entering the studentID.  \*Example here display student 1222 (studentID). |  |
| 14. | However, if the user tries to retrieve the information of a non-existing student, the program will display an error message to inform the user.  \*Example here the user tries to retrieve the information of a non-existing student (studentID: 9999) |  |
| 15. | The studentID is a combination of 4 digits. If the user enters a non-numerical value in task 7, the program will also display an invalid input message to inform the user. |  |

|  |  |  |
| --- | --- | --- |
| 16. | Apart from studentID, the user can also retrieve the student’s information based on student’s first name and last name (case not sensitive). If the user select task 8 and enters an existing student’s name, the program will display the details of that student.  \*Example here is an existing student (student name: A5\_FN A5\_LN) |  |
| 17. | Similar to task 7, if the same student’s name is not found in the student list, task 8 will also display a no found message to notify the user.  \*Example here the student’s first name and last name is “aa”. |  |
| 18. | The user can sort the student list into ascending order of the studentID and output the sorted array by selecting task 9. After the user enters the number 9, the program does not require any other input values ​​and displays the sorted student list in the output. |  |
| 19. | Task 9 allowed the user to sort the ArrayList of the student objects into ascending order of the studentID and output the sorted array. Task 10 allowed user to output the sorted student list to a csv file. However, task 10 can only be execute if the student list is sorted. This means that task 10 can only be executed after task 9 is executed. If the user tries to output the unsorted student list to the csv file, the program will ignore it and display an error message. |  |

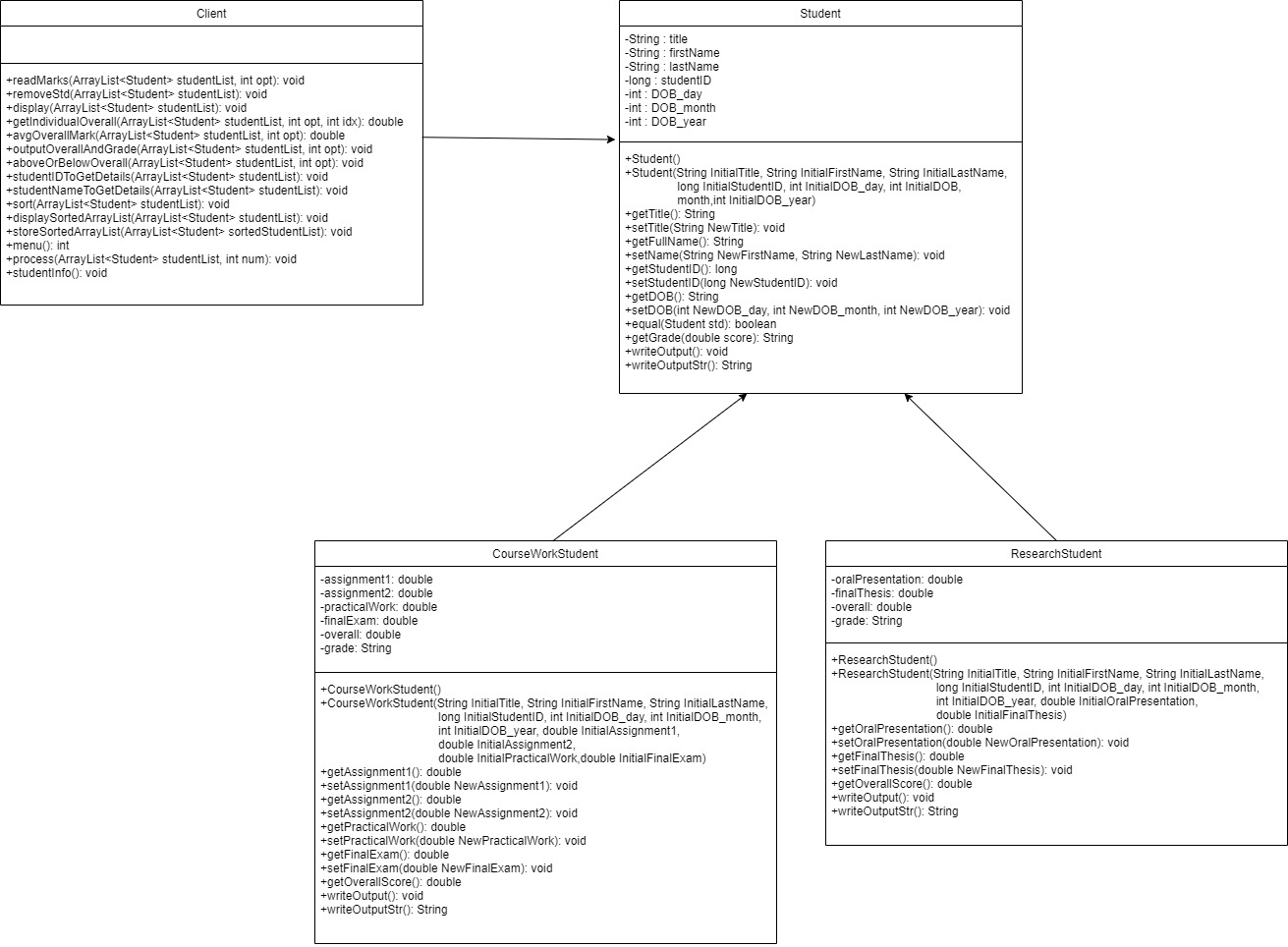
|  |  |  |
| --- | --- | --- |
| 20. | After completing task 10, the user is able to see the sorted student list in the SortedArrayListOutput.csv file. |  |
| 21. | Lastly, if the user enters any task apart from task 1 to 10 or input a non-numerical value in the menu stage, the program will display an error message to inform the user. |  |

1. **Structure/Design/Algorithm:**

**UML Class Diagram**



**Structure Chart**



**Design of the program** \*Pseudocode is in red colour

* This program contains 4 different classes which is the Client class, Student class (base class) and two child class: CourseWorkStudent class and ResearchStudent class. The entire file also contains 3 csv files for reading students' values ​​and scores. At the beginning of the program, the program will ask the user what type of student he/she is dealing with. In addition, the program also provides the user different tasks to perform some analysis and query. The program will continue to execute until the user terminates it.

**Student Class (base class)**

* The student class contain 7 instances variables such as title, firstName, lastName, studentID, DOB\_day, DOB\_month and DOB\_year. The student class also include 2 constructor (with & without parameters) and appropriate getter and setter method.
* **equal(Student std)**

This method is to compare the Student object with the pass in Student object. The method will return true is both Student object is same, else return false.

Equal

IF this.studentID == pass\_in\_studentID

Return true

ELSE

Return false

ENDIF

END

* **getOverallScore()**

This method is just a dummy method that use to calculate the overall mark of the CourseWorkStudent or ResearchStudent. The method containing the calculation will be in the child class. This method will only return 0 after being called.

Get\_Overall\_Score

Return 0

END

* **getGrade(double score)**

This method is to get the grade of the coursework or research student based on student’s overall mark. The overall mark will be calculated in the respective child class and pass into this method as an argument. The overall mark will go through the if-else statement to find the corresponding grade. Once the grade is found, this method will return the grade. I only implement this method in the Student class (base class). Because the CourseWorkStudent and ResearchStudent class (child class) inherit the methods of base class. Therefore, child class can get the grade of student by calling the base class method.

Get\_Grade

Method takes in a score with double value

Set score to double overall

Set grade to null

IF overall >= 80

grade = “HD”

ELSE IF overall >= 70 AND < 80

grade = “D”

ENDIF

ELSE IF overall >= 60 AND < 70

grade = “C”

ENDIF

ELSE IF overall >= 50 AND < 60

grade = “P”

ENDIF

ELSE

grade = “N”

ENDIF

Return grade

END

* **writeOutput()**

This method is to display the student’s information. Once the method is being called, the program will display all the information of coursework or research student.

Write\_Output

Print student information.

END

* **writeOutputStr()**

This method is used to return all the information of coursework or research students. The purpose for this method is for task 10 in client class – output the sorted ArrayList into a csv file. The client class can call this method to print the student’s information to the csv file.

Write\_Output\_Str

Return student’s information

END

**ResearchStudent Class (child class)**

* The Research class contain 4 variables which is the double type of oralPresentation, finalThesis, overall and a String type grade. In addition, ResearchStudent class also inherit all the variables and method of Student class because it the child class of the Student class. Therefore, the user needs value of parent class’s variables to create a ResearchStudent object. This class also contains some of its own methods and appropriate getter/setter methods. This method has 4 constructors (overload constructor): 1 default constructor (no parameter), 1 constructor with all Student class variables as parameters, 1 constructor with all marks only as parameters, and 1 constructor with all parameters including Student class variables and its own variables (marks). Therefore, the user can create ResearchStudent object according to the values he/she has. The super keyword in the constructors and some methods is to invoke the variables or methods in parent class.
* **getOverallScore()**

This method is used to calculate the overall mark of research student. There are no arguments required for this method. The maximum mark of oral presentation is 20 marks and it accounts for 20% of the overall score. Therefore, no required further calculation. The maximum mark of final thesis is 100 marks and it accounts for 80% of the overall score. Overall mark is the sum of oralPresentationMarks and finalThesisMarks. The method will return the overall mark after the calculation is completed.

Get\_Overall\_Score

Set oralPresentation to oralPresentationMarks

Set finalThesisMarks to finalThesis \* 80 / 100

overall = oralPresentation + finalThesisMarks

Return overall

END

* **writeOutput()**

This method is to overriding method. Similar to the one in parent class, this method is to display all the information of research student including mark and grade by using the super keyword. The super keyword will call the writeOutput method in parent class and display the student information followed by the marks and grade of research student.

Write\_Output

Call writeOutput in parent class

Print student information.

END

* **writeOutputStr()**

This method is to overriding method. Similar to the one in parent class, this method is used to return all the information of research students including marks and grade by using the super keyword. The purpose for this method is for task 10 in client class – output the sorted ArrayList into a csv file. The client class can call this method to print the student’s information to the csv file.

Write\_Output\_Str

Return student’s information including call writeOutput in parent class

END

**CourseWorkStudent Class (child class)**

* The Research class contain 6 variables which is the double type of assignment1, assignment2, practicalWork, finalExam, overall and a String type grade. In addition, CourseWorkStudent class also inherit all the variables and method of Student class because it the child class of the Student class. Therefore, the user needs value of parent class’s variables to create a CourseWorkStudent object. This class also contains some of its own methods and appropriate getter/setter methods. This method has 4 constructors: 1 default constructor (no parameter), 1 constructor with all Student class variables as parameters, 1 constructor with all marks only as parameters, and 1 constructor with all parameters including Student class variables and its own variables (marks). Therefore, the user can create ResearchStudent object according to the values he/she has. The super keyword in the constructors and some methods is to invoke the variables or methods in parent class.
* **getOverallScore()**

Similar to the one in ResearchStudent class, this method is used to calculate the overall mark of research student. There are no arguments required for this method.

However, this method contains more variables to calculate the overall mark.

Each mark accounts for a different percentage of the overall mark. The method will return the overall mark after the calculation is completed.

Get\_Overall\_Score

Set assignment1Score to assignment1 \*25/100

Set assignment2Score to assignment2 \*25/100

Set practicalWorkScore to practicalWork

Set finalExamScore to finalExam \* 30 / 100

overall = assignment1Score + assignment2Score + practicalWorkScore + finalExamScore

Return overall

END

* **writeOutput()**

This method is to overriding method. Similar to the one in ResearchStudent class, this method is to display all the information of coursework student including mark and grade by using the super keyword. The super keyword will call the writeOutput method in parent class and display the student information followed by the marks and grade of coursework student.

Write\_Output

Call writeOutput in parent class

Print student information.

END

* **writeOutputStr()**

This method is to overriding method. Similar to the one in ResearchStudent class, this method is used to return all the information of coursework students including marks and grade by using the super keyword. The purpose for this method is for task 10 in client class – output the sorted ArrayList into a csv file. The client class can call this method to print the student’s information to the csv file.

Write\_Output\_Str

Return student’s information including call writeOutput in parent class

END

**Client Class**

* The client class allowed the user to create either research student or coursework student. The client class also contains some methods for users to perform certain tasks. The menu will continue to execute until the user selects task 1 to exit the program.
* **readMarks(ArrayList<Student> studentList, int opt)**

This method is used to read all the marks of coursework or research student based on user’s choice. At the beginning of the program will create the CourseWorkStudent or ResearchStudent object and store it into an ArrayList. Then pass the ArrayList and the type of the student (0 = CourseWorkStudent or 1 = ResearchStudent) into this method. The whole method is split into 2 big parts: a part for dealing with CourseWorkStudent, and another for ResearchStudent. If the pass in type of student is 0, the method will read the marks of coursework student from the csv file, else read marks of research student. The whole reading process is inside a try catch block. If the file to read all the marks is not found, it will throw a FileNotFoundException for the catch block to catch. The while loop will check if the csv file has next line of value, it will split the value by a comma and store it into an array. The reason why doing this is because we can retrieve each mark from the array. Therefore, we can use the setter methods in the child class to set all the marks for CourseWorkStudent or ResearchStudent (because previously all the marks are 0). The process for both CourseWorkStudent and ResearchStudent is the same.

Read\_Marks

Method takes in an ArrayList of Student type and opt with int value

Set int counter = 0

IF opt == 0

Try

create instance of File and Scanner to read value from file coursework.csv

DOWHILE coursework file has next line

Set courseworkLine = courseworkFile.nextLine();

Split courseworkLine by comma and store into an array

Retrieve each value from the array

IF dealing with coursework student

Set all marks for coursework student

ENDIF

counter++

ENDDO

Catch if coursework file not found

ELSE

Try

create instance of File and Scanner to read value from file research.csv

DOWHILE research file has next line

Set researchLine = researchFile.nextLine();

Split researchLine by comma and store into an array

Retrieve each value from the array

IF dealing with research student

Set all marks for research student

ENDIF

counter++

ENDDO

Catch if research file not found

ENDIF

Print success message

END

* **removeStd (ArrayList<Student> studentList)**

Previously we create the CourseWorkStudent or ResearchStudent object and store it into an ArrayList. This method allows the user to remove the student/object from the ArrayList. To execute this method, we need to pass in the Student ArrayList as an argument. Then, the program will prompt the user to enter the studentID of the student to be removed. The program will have a confirmation stage after read the studentID. The user needs to enter yes to confirm the query or no to cancel it. If the user’s option is no, the method will terminate and go back to the menu stage, else the for loop will search for the same studentID. Once the if statement found the coursework or research object, the program will remove the particular student. After the whole process is completed, the program will display a successful or student not found message.

Remove\_Std

Method takes in an ArrayList of Student type

Set boolean isFound to false

Try

Get studentID

Get confirmation option

IF option == “yes”

REPEAT

IF studentID from ArrayList == studentID from user

Remove the student

Set isFound == true

ENDIF

UNTIL (reach ArrayList size or studentID is found)

IF isFound == true

Print successful message

ELSE

Print student not found message

ENDIF

ELSE IF option == “no”

Print leave message

ELDIF

ELSE

Throw InputMismatchException

ENDIF

Catch InputMismatchException

Display error message

END

* **display (ArrayList<Student> studentList)**

This method is use to display all the information of the pass in ArrayList by using the for-each loop. To display the information of student, we can call the writeOutput method create in the base class and child class.

Display

Method takes in an ArrayList of Student type

REPEAT

Print student’s information by calling writeOutput method

UNTIL (reach the end of ArrayList)

END

* **getIndividualOverall(ArrayList<Student> studentList, int opt, int idx)**

This method is to calculate and return the individual overall mark for coursework or research student. The arguments for this method required the Student ArrayList, opt with int value (the type of student) and an idx with int value (index of the student). This method will calculate and return the overall mark of the particular student based on the arguments by calling the getOverallScore method created in the child or base class.

Get\_Individual\_Overall

Method takes in an ArrayList of Student type, int opt and int idx

Set overallMarks to 0

IF opt == 0

overallMarks = call getOverallScore method to retrieve the overall mark of coursework student

ELSE

overallMarks = call getOverallScore method to retrieve the overall mark of research student

ENDIF

Return overallMarks

END

* **avgOverallMark (ArrayList<Student> studentList, int opt)**

This method is similar to the previous method, but this method is to calculate and return the average overall mark of all coursework of research student in the pass in ArrayList. The for loop will loop through the pass in ArrayList and call the getIndividualOverall method to get the individual overall mark of all student. Then sum the running total of all individual overall mark. Lastly, return the running total of overall mark divide by size of the ArrayList.

Avg\_overall\_Mark

Method takes in an ArrayList of Student type and int opt

Set total = 0

Set avgOverallMark = 0

REPEAT

Total += call getIndividualOverall and pass in the required arguments

UNTIL (reach the end of ArrayList)

Set avgOverallMark = total / size of ArrayList

Return avgOverallMark

END

* **outputOverallAndGrade (ArrayList<Student> studentList, int opt)**

This method is the controller for task 5. Previously we already implement a getIndividualOverall method to calculate and retrieve the individual overall mark of student. Here we call the getIndividualOverall method and getGrade in a for loop to calculate and display all the overall mark and grade of student including some output design. The arguments for this method required a Student type ArrayList and an opt with int type

Value (opt is the type of student).

Output\_Overall\_And\_Grade

Method takes in an ArrayList of Student type and an int opt

Set overall = 0

Set grade = null

REPEAT

Set overall = call getIndividualOverall method to get student’s overall

Set grade = call getGrade to get student’s grade

Retrieve the studentID of the particular student

Display studentID, overall mark and grade

UNTIL (reach the size of pass in ArrayList)

END

* **aboveOrBelowOverall (ArrayList<Student> studentList, int opt)**

Previously I have implemented a method called avgOverallMark() to calculate the average overall mark of all coursework or research student. This method is to determine and display the number of coursework or research students obtained an overall mark equal to or above the average overall mark and how many obtained an overall mark below the average overall mark. The method will first display the value of average overall mark and the corresponding grade. After that the for loop will loop through all the objects in the ArrayList and compare the object’s overall mark and the average overall mark. If the object’s overall mark is greater than or equal to the average overall mark, the variable equalOrAbove will increase by one, else below will increase by one (means less than the average overall mark). At the end, the method will display the result after the calculation is completed.

Above\_Or\_Below\_Overall

Method takes in an ArrayList of Student type and an int opt

Set avgOverall = call avgOverallMark to get the average overall mark

Set equalOrAbove = 0

Set below = 0

Set averageOverall = 0

Set avgOverallMarkGrade = null

Set averageOverall = call avgOverallMark to get the average overall mark

Set avgOverallMarkGrade = call getGrade to get grade for average overall mark

Display number of student in ArrayList

Display average overall mark of all student

Display grade of average overall mark of all student

REPEAT

IF opt == 0

IF coursework student’s overall mark > avgOverall

equalOrAbove++

ELSE

below++

ENDIF

ELSE

IF research student’s overall mark > avgOverall

equalOrAbove++

ELSE

below++

ENDIF

ENDIF

UNTIL (reach the size of the ArrayList)

Display the result of equalOrAbove and Below

END

* **studentIDToGetDetails (ArrayList<Student> studentList)**

This method allowed the user to retrieve the information of specific coursework or research student including the marks and grade based on the studentID. The beginning of the method will prompt and get the studentID from the user. After that, the program will search through the pass in ArrayList to find the object with the same studentID. Once the student is found, the program will display the information of the particular student by calling the writeOutput method and break the for loop. The program will display a not found message to inform the user if the studentID was not found in the student list.

StudentID\_To\_Get\_Details

Method takes in an ArrayList of Student type

Set isFound = false

Try

Get studentID

REPEAT

IF student\_number\_from\_the\_list == studentID

Display student’s information

Set isFound = true

ENDIF

UNTIL (reach the size of the ArrayList)

IF isFound == false

Display student not found message

ENDIF

Catch InputMismatchException

END

* **studentNameToGetDetails (ArrayList<Student> studentList)**

Similar to the studentIDToGetDetails method, this method also allowed user to retrieve the information of student including the marks and grade. However, this method will read the first and last name of the student to find a specific student and display the information of the corresponding student by calling the writeOutput method. In addition, if there are more than one student with the exact same name, the program will output the information of all students with the same name (*demo in test table*). The program will also display a not found message if the student’s name is not found int the pass in ArrayList.

Student\_Name\_To\_Get\_Details

Method takes in an ArrayList of Student type

Set isFound to false

Get student’s first name

Get student’s last name

Set fullName = concatenate first name and last name

REPEAT

IF student\_name\_from\_the\_list == fullName

Display the information of corresponding student

Set isFound to true

ENDIF

UNTIL (reach the size of the ArrayList)

IF isFound == false

Display student not found message

ENDIF

END

* **sort (ArrayList<Student> studentList)**

This method is used to sort the pass in ArrayList of the student objects into ascending order of the studentID. Here I use the bubble sort method to sort the ArrayList. The if statement inside the for loop will compare the studentID of adjacent object. If the object’s studentID is greater than the latter one, it will swap the position of the two objects and set boolean swap to true for next round comparing. The for loop will terminate until the boolean swap is false (means the sorting is done).

One of the reasons why I choose bubble sort method to sort the ArrayList is because bubble sort is easier to understand compare to insertion sort and selection sort. This is better for me and even other developer who looking at my code. In addition, the studentID given in the csv file is already almost sorted. Therefore, bubble sort is the most effective method to sort the ArrayList in ascending order.

Sort

Method takes in an ArrayList of Student type

Set swap to true

DOWHILE swap == true

Set swap to false

REPEAT

IF former object’s studentID > latter object’s studentID

Swap the position of two object

Set swap to true

ENDIF

UNTIL (swap == false)

ENDDO

END

* **displaySortedArrayList (ArrayList<Student> studentList)**

This method is the main controller of task 9. The method will first call the sort method to sort the pass in ArrayList of the student objects into ascending order of the studentID. After that output the sorted ArrayList by calling the display method. The argument for this method required a Student type ArrayList.

Display\_Sorted\_ArrayList

Method takes in an ArrayList of Student type

Sort the pass in ArrayList

Display sorted ArrayList

END

* **storeSortedArrayList (ArrayList<Student> studentList)**

This method is used to output the sorted ArrayList to sortedArrayListOutput.csv file. After executing this method, we can view all the information of coursework or research in ascending order of studentID. The method will first open outputFile and connect to the object outputStream (sortedArrayListOutput.csv). After that, the for loop will output the information of sorted ArrayList object by calling the writeOutputStr method. It will throw an FileNotFoundException if the outputFile is not found. Then the catch block will catch the FileNotFoundException and display some error message. The program will display a successful message if successfully output the result to the sortedArrayListOutput.csv file.

Store\_Sorted\_ArrayList

Method takes in an ArrayList of Student type

Set PrintWriter outputStream to null

Try

open outputFile and connect to object outputStream

REPEAT

Output the sorted ArrayList to the outputFile

UNTIL (reach the size of ArrayList)

Catch FileNotFoundException

Print error message

Close outputStream

Display successful message

END

* **menu ()**

This method is the output of menu. Inside this method will print out all the description for each menu task and read the user’s choice. Lastly return the user’s choice. This method will throw an Exception if the user enters an invalid option (not 1 to 10 or non-numerical value).

Menu

Print all menu description

Read user’s choice

Return user’s choice

END

* **process(MoneyChange[] array)**

This is the main controller for the whole program. The switch case will call different methods according to user’s choice on menu stage. Inside the main class just need to call this process() method and pass in appropriate argument to run all tasks. If the user enters a non-numerical value, the error exception will catch and print an error message. The menu will continue execute until user enter task 1 to exit the program. After executing the task 2, the program will set the variable validOrder to true (initial is false). The reason why I doing this is because is the user tries to remove student, display overall mark or get the number of coursework or research students obtained an overall mark equal to or above the average overall mark and how many obtained an overall mark below the average overall mark (task 3, 5 and 6) before read in all the marks of the student, the program will display a message to remind the user to execute task 2 to read the mark first. Therefore, there have this if else statement in task 3, 5 and 6. Similar to task 9 and 10, is the user tries to output the unsorted ArrayList into the csv file, the program will display a message to remind the student to sort the ArrayList in ascending order before execute task 10.

Process

Set option to 0

Set valid to false

Set validOverall to false

DOWHILE option != 1

Get option

CASE OF (option)

1: Exit the program

2: Call task 2

Set validOrder to true

3: IF validOrder == true

Call tasks 3

ELSE

Display error message

4: Call task 4

5: IF validOrder == true

Call task 5

ELSE

Display error message

6: IF validOrder == true

Call task 6

ELSE

Display error message

7: Call task 7

8: Call task 8

9: Call task 9

Set valid = true

10: IF valid == true

Call task 10

ELSE

Display error message

Default: Print error message

ENDDO

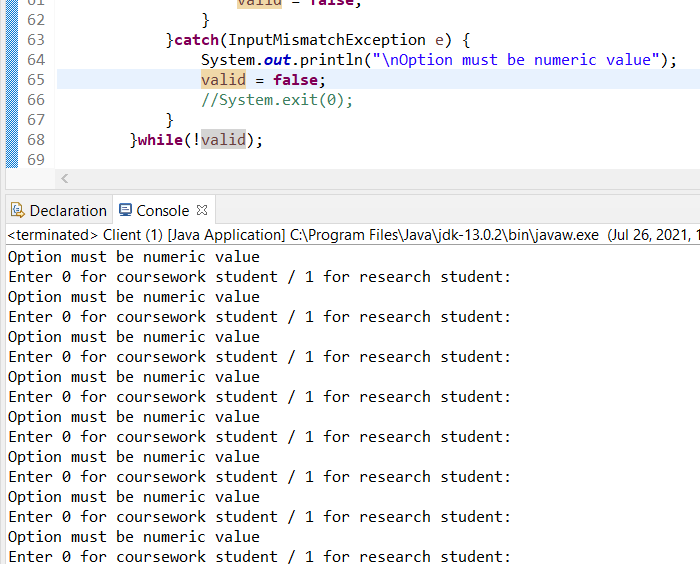
END

1. **Limitation**
2. **Non-numerical value for type of student**

At the beginning, the program will prompt the user to enter the type of student to be processed (0 = coursework, 1 = research). However, if the user has entered an invalid value, the program should display an error message and ask the user to re-enter the value. Therefore, I set the boolean valid to false first. Then, I put the whole process inside a do-while loop. After enter the try block, program set valid to true for terminate the loop. If the user enters a valid option, the program will do specific work based on the user’s input and end the loop. If the user enters an invalid option, the program will display error message and set valid back to false, so that the loop can continue execute.

Initially, my idea is to use the catch block to catch the non-numerical option (InputMismatchException). If the user enters an invalid option, the program will display an error message and set the valid back to false so that the loop can continue. However, the program will execute the code indefinitely instead of my expected result.

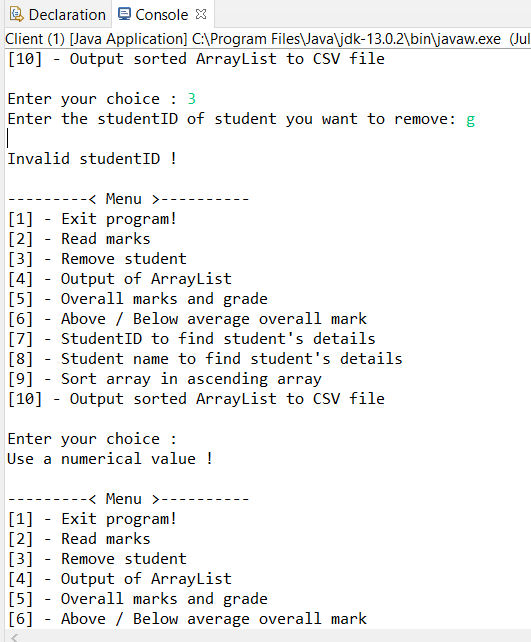
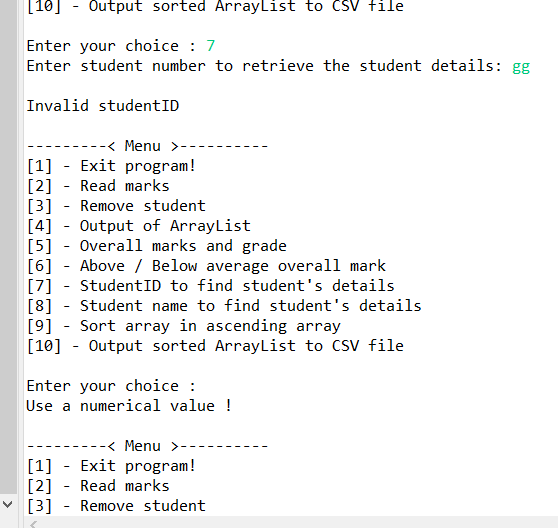
To solve this issue, I use a System.exit(0) method to terminate the program if the user enters a non-numerical option for the type of student.



1. **Non-numerical value for studentID**

In task 3 and 7, the program will ask the user to enter the studentID and perform some analysis or query for the corresponding student object. The studentID is consist of 4 digits. However, if the user enters a studentID with non-numerical value, the program should display and error message.

My idea of ​​dealing with this invalid studentID is to use a InputMismatchException. So that, the InputMismatchException can catch the error and display some error message when the user enters a non-numerical value. Eventually, the program successfully displayed the expected result. However, the menu will be executed again with an error message. (I think is because the invalid studentID input goes to the menu stage.)



**Task 3**

**Task 7**

1. **Testing**

**Test Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test id** | **Test description/justification – what is the test for and why this particular test** | **Actual data for this test** | **Expected output** | **Actual desk check result when desk check is carried out** | **Desk check outcome – Pass/Fail** |
| 1. | Test for the beginning of the program. The program should only deal with either coursework or research student. Therefore, the program should ask the user what type of students to deal with. User should enter 0 for coursework student 1 for research student. After the user enters his/her options, the program should display the student type followed by the menu. | Input 0 - (0 is the valid input for dealing with coursework student. The program should be able to display the expected output and continue running.) | ---< Coursework student >---  -----< Menu >------  [1] - Exit program!  [2] - Read marks  [3] - Remove student  [4] - Output of ArrayList  [5] - Overall marks and grade  [6] - Above / Below average overall mark  [7] - StudentID to find student's details  [8] - Student name to find student's details  [9] - Sort array in ascending array  [10] - Output sorted ArrayList to CSV file  Enter your choice : | ---< Coursework student >---  -----< Menu >------  [1] - Exit program!  [2] - Read marks  [3] - Remove student  [4] - Output of ArrayList  [5] - Overall marks and grade  [6] - Above / Below average overall mark  [7] - StudentID to find student's details  [8] - Student name to find student's details  [9] - Sort array in ascending array  [10] - Output sorted ArrayList to CSV file  Enter your choice : | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2. | Test for the beginning of the program. The program should only deal with either coursework or research student. Therefore, the program should ask the user what type of students to deal with. This test is for research student and to test if the program can successfully display the expected output when user enter 1 for dealing with research student. | Input 1 – (1 is the valid input for dealing with research student. The program should be able to display the expected output and continue running.) | ---< Research student >---  -----< Menu >------  [1] - Exit program!  [2] - Read marks  [3] - Remove student  [4] - Output of ArrayList  [5] - Overall marks and grade  [6] - Above / Below average overall mark  [7] - StudentID to find student's details  [8] - Student name to find student's details  [9] - Sort array in ascending array  [10] - Output sorted ArrayList to CSV file  Enter your choice : | ---< Research student >---  -----< Menu >------  [1] - Exit program!  [2] - Read marks  [3] - Remove student  [4] - Output of ArrayList  [5] - Overall marks and grade  [6] - Above / Below average overall mark  [7] - StudentID to find student's details  [8] - Student name to find student's details  [9] - Sort array in ascending array  [10] - Output sorted ArrayList to CSV file  Enter your choice : | Pass |
| 3. | This test is to test invalid options of the student type. When the user input any option apart from 0 and 1, the program should be able to handle this error by displaying an error message and prompt the user again. The program should continue asking until the user has entered a valid option. | Input 1: 5  Input 2: 99  (5 and 99 are the example of invalid option. There are only two types of students: 0 for coursework student 1 for research student.) | Invalid option  Enter 0 for coursework student / 1 for research student: 99 (enter invalid option again)  Invalid option  Enter 0 for coursework student / 1 for research student: | Invalid option  Enter 0 for coursework student / 1 for research student: 99 (enter invalid option again)  Invalid option  Enter 0 for coursework student / 1 for research student: | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4. | This test will test if the user enters a non-numeric value for the type of student. The input for coursework and research student must be 0 and 1 respectively. If user enter a non-numeric value, the program will display and error message and terminate the program. | Input g –  (g is an example of non-numeric value. The program should display and terminate the program because the input for type of student can only be 0 or 1) | Option must be numeric value | Option must be numeric value | Pass |
| 5. | After successfully get the type of student, the program should create the particular coursework or research student object and display the menu to allow the user to perform some tasks. The menu will continue displaying the menu after each task is completed. This test is to check whether the program will terminate when the user enters task 1 to exit the program. | Input 1 –  (Task 1 allowed the user to exit the program. The program should display a goodbye message and terminate automatically. No other input value required) | Thank you Bye!! | Thank you Bye!! | Pass |
| 6. | Task 5 and 6 is used to conduct some analysis with the overall mark of student. However, if the user tries to execute task 5 & 6 before execute task 2 to read all the mark of the students, the program should ignore the execution and display some error message. This test will test if user execute task 5 before task 2 in the menu stage. | Input 5 –  (In this scenario, task 2 not execute yet. But the user tries to execute task 5 to output the overall mark of all student. However, there is not any mark for student yet.) | Marks not read yet, select task 2 to read the marks first  (Followed by the menu stage) | Marks not read yet, select task 2 to read the marks first  (Followed by the menu stage) | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 7. | Similar to the test no.6. The program should not allow the user to execute task 6 (task 6 is to find the average overall mark of all student) before reading the mark of the student and calculate the overall mark (task 2). If that happen, the program should display an error message to inform the user to execute task 2 first. | Input 6 – (Similar to test 6 scenario, task 2 not execute yet. But the user tries to execute task 6 to find the average overall mark and grade of all student. However, there is not any mark for student yet.) | Marks not read yet, select task 2 to read the marks first  (Followed by the menu stage) | Marks not read yet, select task 2 to read the marks first  (Followed by the menu stage) | Pass |
| 8. | To avoid the error, occur in tests 6 and 7. The user should read all the mark of all individual students. Task 2 is the task that provides this function. After the user enter task 2, the program should read all the mark from the corresponding csv file and display a successful message after the reading process is completed. After that, the mark of student also can be seen in the student information output. | Input 2 – (task 2 not require any other user input. After select task 2, the program will read the mark for all student a from corresponding csv file. After completed, the program will display a successful message. | Success - read all marks  (Followed by the menu stage) | Success - read all marks  (Followed by the menu stage) | Pass |
| 9. | As mentioned earlier, the marks and grades will be reflected in the student information output after successfully read the mark from the csv file (task 2). The program provides task 4 to allow users to display coursework or research student information. After select task 4, the program should display all student information including student details, all marks and grade. | Input 4 – (task 4 not required any other input value, the program will automatically display all information of coursework or research student)    \*Scenario deal with coursework student | Student : Mr.A1\_FN A1\_LN  Student Number : 1111  Date Of Birth : 1/11/2000  ----< Marks >-----  Assignment 1: 50.0  Assignment 2: 60.0  Practical work: 15.0  Final exam : 60.0  Overall mark: 60.5  Grade : C  … until student 1555(studentID) | Student : Mr.A1\_FN A1\_LN  Student Number : 1111  Date Of Birth : 1/11/2000  ----< Marks >----  Assignment 1 : 50.0  Assignment 2 : 60.0  Practical work : 15.0  Final exam : 60.0  Overall mark : 60.5  Grade : C  … until student 1555(studentID) | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10. | Task 3 allow the user to remove student from the ArrayList. This test is to check whether the student is successfully removed from the list after completing task 3. This program will display a successful message to inform user after the execution is completed. In order to remove a student, the user needs to enter the studentID of the student he/she wants to remove and input yes (case not sensitive) in the next confirmation stage. | Input 1 = 3  Input 2 = 1111  Input 3 = yes  (Scenario here we try to remove student with studentID 1111 and input yes to confirm the analysis and query.  After that, we can see that student 1111 is no longer in the list by performing task 4) | Student 1111 successfully removed  (Followed by the menu stage)  \*Execute task 4: student 1111 (studentID is no longer in the list) | Student 1111 successfully removed  (Followed by the menu stage)  \*Execute task 4: There is no student 1111(studentID) in the list) | Pass |
| 11. | Task 3 allow the user to remove student from the ArrayList. This test will check if the user tries to remove a non-existing student. The program should display a student not found message to inform the user if the program cannot find the same studentID in the list. | Input 1 = 3  Input 2 = 9999  Input 3 = yes  (Scenario here we try to remove student with studentID 9999 which is a non-existing student.) | Student 9999 not found  (Followed by the menu stage) | Student 9999 not found  (Followed by the menu stage) | Pass |
| 12. | In order to remove student, the user need to enter “yes” in the confirmation stage. If the user enters “no” during the confirmation stage, the program will not remove the student from the list and display a goodbye message. The particular student will still be in the student list and can be seen when executing task 4. | Input 1 = 3  Input 2 = 1222  Input 3 = no  (Scenario here we try to remove student with studentID 1222. However, we enter “no” in the confirmation stage. The program ignore transaction.) | See you then ~  (Followed by the menu stage)  \*Execute task 4: student 1222(studentID is in the list, no deletion is completed) | See you then ~  (Followed by the menu stage)  \*Execute task 4: student 1222(studentID is in the list, no deletion is completed) | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 13. | In task 3, the program will ask the user to enter studentID, confirmation option to complete the transaction. The confirmation stage only allow user to enter yes/no. If the user enters any other input, the program will ignore the analysis and query and display an error message. | Input 1 = 3  Input 2 = 1222  Input 3 = gg  (Scenario here we try to enter an invalid confirmation option (gg) to test the program.) | Invalid option !  (Followed by the menu stage)  \*Execute task 4: There is no students removed from the list | Invalid option !  (Followed by the menu stage)  \*Execute task 4: There is no students removed from the list | Pass |
| 14. | In task 3, the program will ask the user to enter studentID and remove the particular student. If the user enters the studentID with non-numerical value, the program displays an error message to inform the user. It is because studentID is comprise of 4 digits. | Input 1 = 3  Input 2 = gg  (Scenario here we try to enter a studentID with non-numerical value – “gg”.) | Invalid studentID !  (Followed by the menu stage) | Invalid studentID !  (Followed by the menu stage) | Pass |
| 15. | After successfully reading the student's mark by performing task 2. The user can execute task 5 to display the individual overall mark and grade of coursework or research student. (The scenario where task 2 was not performed in advance has been shown in test no. 6) The is no any other input required. | Input 5 –  (Input 5 to execute the task. No other input value is required) | StudentID|Overall\_Marks|Grade  ---------------------------  1111 60.5 C  1222 68.0 C  1333 75.5 D  1444 83.0 HD  1555 90.5 HD | Successfully display the expected output  \*Not showing here because of the space | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 16. | After successfully reading the student's mark by performing task 2. The user can execute task 6 to calculate and display the average overall mark and grade of all coursework and research student and display the number of coursework or research students obtained an overall mark equal to or above the average overall mark and how many obtained an overall mark below the average overall mark. (The scenario where task 2 was not performed in advance has been shown in test no. 7) The is no any other input required. The program will first display the value of average overall mark and grade of all student followed by the expected result. | Input 6 –  (Input 6 to execute the task. No other input value is required) | -> Total number of student : 5  -> Average Overall Mark : 75.5  -> Grade : D  3 students obtained an overall mark equal to or above the average overall mark of 75.5  2 students obtained an overall mark below the average overall mark of 75.5 | -> Total number of student : 5  -> Average Overall Mark : 75.5  -> Grade : D  3 students obtained an overall mark equal to or above the average overall mark of 75.5  2 students obtained an overall mark below the average overall mark of 75.5 | Pass |
| 17. | Task 7 allowed user to retrieve the information of coursework or research student based on the studentID. The program will ask the user to enter the studentID and display the information of the particular student. This test is to check whether this task can execute successfully and display the correct result. | Input 1: 7  Input 2: 1222  (Example here we try to retrieve the information of student 1222(studentID).  StudentID 1222 is an existing student.) | Student details :  ------------------------  Student : Mr.A2\_FN A2\_LN  Student Number : 1222  Date Of Birth : 2/12/2000  -------< Marks >--------  Assignment 1 : 60.0  Assignment 2 : 70.0  Practical work : 16.0  Final exam : 65.0  Overall mark : 68.0  Grade : C | Student details :  ------------------------  Student : Mr.A2\_FN A2\_LN  Student Number : 1222  Date Of Birth : 2/12/2000  -------< Marks >--------  Assignment 1 : 60.0  Assignment 2 : 70.0  Practical work : 16.0  Final exam : 65.0  Overall mark : 68.0  Grade : C | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 18. | Task 7 allowed user to retrieve the information of coursework or research student based on the studentID. This test will test if the user tries to retrieve the information of non-existing student. The program should display a not found message to inform the user. | Input 1: 7  Input 2: 9999  (Scenario here we try to retrieve the information of student 9999. However, there is no student 9999(studentID) in the ArrayList) | Student not found  (Followed by the menu stage) | Student not found  (Followed by the menu stage) | Pass |
| 19. | Task 7 allowed user to retrieve the information of coursework or research student based on the studentID. This test will test if the user enters a non-numeric value (because studentID is a composite of 4 numerical values). The program should display an error message and continue execute the menu. | Input 1: 7  Input 2: gg  (Scenario here the user enters a studentID with non-numerical value – “gg”, which is an invalid input.) | Use a numerical value !  (Followed by the menu stage) | Use a numerical value !  (Followed by the menu stage) | Pass |
| 20. | Similar to task 7, task 8 also allowed the user to retrieve the student’s information. But this task takes in the first name and last name of the student (case not sensitive) and display the information of the particular student. | Input 1:8  Input 2: A5\_FN  Input 3: a5\_ln  (Scenario here the user tries to retrieve the information of the student with name A5\_FN A5\_LN. We also use upper and lower case together to the system. There should be no issue when retrieving. | Student details :  ------------------------  Student : Mr.A5\_FN A5\_LN  Student Number : 1555  Date Of Birth : 5/15/2000  -------< Marks >--------  Assignment 1 : 90.0  Assignment 2 : 100.0  Practical work : 19.0  Final exam : 80.0  Overall mark : 90.5  Grade : HD | Student details :  ------------------------  Student : Mr.A5\_FN A5\_LN  Student Number : 1555  Date Of Birth : 5/15/2000  -------< Marks >--------  Assignment 1 : 90.0  Assignment 2 : 100.0  Practical work : 19.0  Final exam : 80.0  Overall mark : 90.5  Grade : HD | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 21. | This test is to test for retrieving the information of a non-existing student in task 8. The program should display an error message if the system cannot find a same name student. | Input 1:8  Input 2: aa  Input 3: bb  (In the scenario here, the user tries to retrieve the information of the student named aa bb, which does not exist in the list.) | Student not found  (Followed by the menu stage) | Student not found  (Followed by the menu stage) | Pass |
| 22. | Task 10 allowed the user to output the sorted ArrayList into a csv file. However, if the user tries to execute task 10 before execute task 9 to sort the ArrayList in ascending order of the studentID, the program will ignore the query and display an error message to remind the user to execute task 9 first. | Input 1: 10  (Scenario here task 9 is not executed yet, means that the ArrayList is not sorted. Task 10 should not be allowed to execute) | ArrayList not sorted yet. Select task 9 to sort first  (Followed by the menu stage) | ArrayList not sorted yet. Select task 9 to sort first  (Followed by the menu stage) | Pass |
| 23. | In order to execute task 10, the user need to execute task 9 first. Task 9 allowed the user to sort the student list in ascending order of the studentID. No other input is required, the program will automatically sort the student list and display the sorted student list on the screen after the user select task 9. | Input 1: 9  (No other input is required, and program will display the sorted student list including the mark and grade.) | ----Sorted student list----  Student : Mr.A1\_FN A1\_LN  Student Number : 1111  Date Of Birth : 1/11/2000  -------< Marks >--------  Assignment 1 : 50.0  Assignment 2 : 60.0  Practical work : 15.0  Final exam : 60.0  Overall mark : 60.5  Grade : C  \*until student 1555 | ----Sorted student list----  Student : Mr.A1\_FN A1\_LN  Student Number : 1111  Date Of Birth : 1/11/2000  -------< Marks >--------  Assignment 1 : 50.0  Assignment 2 : 60.0  Practical work : 15.0  Final exam : 60.0  Overall mark : 60.5  Grade : C  \*until student 1555 | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 24. | Once the student list is sorted in ascending order. The user can execute task 10 to output the sorted student list to the csv file and the program will display a success message to inform the user. After that, the user can see the sorted list of students in the csv file. | Input 1: 10  (No other input is required. The program will display a success message after the query is successfully complete) | Successfully stored in csv file  \*The sorted student list is appearing in the sortedArrayListOutput.csv file. | Successfully stored in csv file  \*The sorted student list is appearing in the sortedArrayListOutput.csv file. | Pass |
| 25. | In the menu stage, the program provide task 1 – 10 for the user to perform different analysis and queries. This test will check if the user enters any task apart form 1- 10, the program should not crash and display an error message. | Input 1: 99  (Scenario here the user tries to execute task 99, which is an invalid option.) | Invalid option!  (Followed by the menu stage) | Invalid option!  (Followed by the menu stage) | Pass |
| 26. | In the menu stage, the program provide task 1 – 10 for the user to perform different analysis and query. This test will check if the user enters a non-numerical value, the program should not crash and display an error message. | Input 1: test  (Scenario here the user enters a non-numerical value in the menu stage, which is an invalid input.) | Use a numerical value !  (Followed by the menu stage) | Use a numerical value !  (Followed by the menu stage) | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 27. | This test is to test whether the program can display the information of all students with the same name when performing task 8. In order to perform this test, I will temporarily change the name of the course student with student ID 1222. I changed this student’s name to be the same as student 1111 (student ID). So that, the program should display the information of both students when the user enters name A1\_FN and A1\_LN (case not sensitive). | Input 1: 8  (The scenario here student 1111 and student 1222 have the same name.) | The program should display student 1111 and student 1222 information. | The program successfully displays the information of student 1111 and student 1222. | Pass |

\* All the tests we have done above use data of course students. However, all tasks are also work for research students. In addition, the input and the method to execute the task for the research student is exactly the same as the coursework student.

**Results of Program Testing** \*Output screen capture can be found at the bottom

* All the tasks are also work for both coursework and research students. In addition, the input and the method to execute the task for both type of student is exactly the same. Therefore, some of the test here will only use either coursework or research student to perform the 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. | At the beginning, the program will ask the user what type of student he/she wants to deal with (0 means coursework, 1 means research). The user should enter the desire option to create the student object by reading all the value in the csv file and display a successful message. | Input 1: 1  (Scenario her the user enter 1 to create research student object. 1 is the input for creating research student.) | ---< Research student >---  ---< Menu >----  [1] - Exit program!  [2] - Read marks  [3] - Remove student  [4] - Output of ArrayList  [5] - Overall marks and grade  [6] - Above / Below average overall mark  [7] - StudentID to find student's details  [8] - Student name to find student's details  [9] - Sort array in ascending array  [10] - Output sorted ArrayList to CSV file | The program successfully created the research student object and display the success message followed by the menu. | Pass |
| 2. | At the beginning, the program will ask the user what type of student he/she wants to deal with (0 means coursework, 1 means research). The user should enter the desire option to create the student object by reading all the value in the csv file and display a successful message. In this test we will deal with coursework student. | Input 1: 0  (Scenario her the user enter 0 to create coursework student object. 0 is the input for creating coursework student.) | ---< Coursework student >---  ---< Menu >-----  [1] - Exit program!  [2] - Read marks  [3] - Remove student  [4] - Output of ArrayList  [5] - Overall marks and grade  [6] - Above / Below average overall mark  [7] - StudentID to find student's details  [8] - Student name to find student's details  [9] - Sort array in ascending array  [10] - Output sorted ArrayList to CSV file  Enter your choice : | The program successfully created the coursework student object and display the success message followed by the menu. | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3. | This test is to test invalid options of the student type. When the user input any option apart from 0 and 1, the program should be should displaying an error message and continue prompting the user again. The program should continue asking until the user has entered a valid option. | Input 1: 5  Input 2: 100  (5 and 100 are the example of invalid option. There are only two types of students: 0 for coursework student 1 for research student.) | Invalid option  Enter 0 for coursework student / 1 for research student: | The program successfully displays the expected result and continue asking for user’s option. | Pass |
| 4. | This test will test if the user enters a non-numeric value for the type of student. The program only accepts value of 0 or 1. If user enter a non-numeric value, the program will display and error message and terminate the program. | Input 1: g  (g is an example of non-numeric value. The program should display and terminate the program because the input for type of student can only be 0 or 1) | Option must be numeric value | The program successfully displays the error message and terminate the program | Pass |
| 5. | After successfully get the type of student, the program should display the menu to allow the user to perform different tasks. The menu will continue displaying the menu after each task is completed. This test is to check whether the program will display a goodbye message and terminate when the user enters task 1 to exit the program. | Input 1: 1  (Task 1 allowed the user to exit the program. The program should display a goodbye message and terminate. No other input value is required) | Thank you Bye!!  \*program stop executing the menu. | The program successful display the goodbye message before terminate the program. | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 6. | Task 5 and 6 is used to conduct some analysis and query with the overall mark of student. Therefore, the user should execute task 2 to read all the student’s mark from the csv file before they can proceed to execute task 5 and 6.  However, if the user tries to execute task 5 & 6 before execute task 2, the program should ignore the execution and display some error message. This test will test whether the system can handle this situation. | Input 1: 5  (In this scenario, task 2 not execute yet. But the user tries to execute task 5 to output the overall mark of all student. However, there is not any mark for student yet.) | Marks not read yet, select task 2 to read the marks first  (Followed by the menu stage) | The program ignores the analysis and query and display an error message to remind the user to execute task 2 first. | Pass |
| 7. | Similar to the test no.6. The program should not allow the user to execute task 6 (task 6 is to find the average overall mark and grade of all student) before execute task 2. If that happen, the program should ignore the analysis and query and display an error message to inform the user to execute task 2 first. | Input 1:6  (Similar to test 6 scenario, task 2 not execute yet. But the user tries to execute task 6 to find the average overall mark and grade of all student. However, there is not any mark for student yet.) | Marks not read yet, select task 2 to read the marks first  (Followed by the menu stage) | The program ignores the analysis and query and display an error message to remind the user to execute task 2 first. | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 8. | In order to execute task 5 and 6 in the menu stage, the user need to execute task 2 to read all the mark of coursework and research from the corresponding csv file. There is no other input value required for task 2. The program should automatically read the mark from the corresponding file based on the type of student. Once the reading process is completed, the program will display a message to inform the user. | Input 1: 2  (User just have to select task 2 to read the mark of student from the file. There is no other input value required) | Success - read all marks  (Followed by the menu stage) | The program successfully read the mark from the csv file and display the success message. The user now is able to perform task 5 and 6. | Pass |
| 9. | The program provides task 4 to allow users to display coursework or research student information. After select task 4, the program should display all student information including student details, all marks and grade. If, the user execute task 4 before execute task 2, the program will display the student’s information without the value of marks and grade (mark will be 0 and grade will be N). | Input 1: 4  (task 4 not required any other input value, the program will automatically display all information of coursework or research student)    \*Scenario deal with research student | Student : Mr.B1\_FN B1\_LN  Student Number : 2111  Date Of Birth : 6/16/2000  -------< Marks >--------  Oral presentation : 15.0  Final thesis : 50.0  Overall score : 55.0  Grade : P  … until student 2555(studentID) | The program successfully display the information of research student. | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10. | Task 3 allow the user to remove student from the ArrayList. This test is to check whether the student is successfully removed from the list after selecting task 3. The program will display a successful message to inform user after the execution is completed. In order to remove a student, the user needs to enter the studentID of the student he/she wants to remove and input yes (case not sensitive) in the next confirmation stage. | Input 1 = 3  Input 2 = 2111  Input 3 = yes  (Scenario here we try to remove student with studentID 2111 and input yes to confirm the analysis and query.  After that, we can see that student 2111 is no longer in the list by performing task 4) | Student 2111 successfully removed  (Followed by the menu stage)  \*Execute task 4: student 2111 (studentID is no longer in the list) | The program successfully removes the student from the list and display successful message. The student no longer in the list after execute task 4. | Pass |
| 11. | This test is to check if the user tries to remove a non-existing student from the list in task 3. The program should display a student not found message to inform the user if the program cannot find the same studentID in the list. | Input 1 = 3  Input 2 = 9999  Input 3 = yes  (Scenario here we try to remove student with studentID 9999 which is a non-existing student.) | Student 9999 not found  (Followed by the menu stage) | The program successfully displays the expected output when the program cannot find the particular student (9999). | Pass |
| 12. | In order to remove student, the user need to enter “yes” in the confirmation stage. If the user enters “no” during the confirmation stage, the program will ignore the analysis and query and display a goodbye message. Execute task 4 and there will no changes. | Input 1 = 3  Input 2 = 2222  Input 3 = no  (Scenario here we try to remove student with studentID 2222. However, the user enters “no” in the confirmation stage.) | See you then ~  (Followed by the menu stage)  \*Execute task 4: student 2222(studentID is in the list, no deletion is completed) | No student has been removed from the list and the program successfully display the expected output. | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 13. | In task 3, the program will ask the user to enter studentID, confirmation option to completed the analysis and query. The confirmation stage only allow user to enter yes/no. If the user enters any other input, the program will ignore the analysis and query and display an error message. | Input 1 = 3  Input 2 = 2222  Input 3 = gg  (Scenario here we try to enter an invalid confirmation option (gg) to test the program.) | Invalid option  (Followed by the menu stage)  \*Execute task 4: There is no students removed from the list | The program successfully displays the expected output. | Pass |
| 14. | After successfully reading the student's mark by executing task 2. The user can execute task 5 to display the individual overall mark and grade of coursework or research student. (The scenario where task 2 was not performed in advance has been shown in test no. 6) The is no any other input required. | Input 1: 5  (Input 5 to execute the task. No other input value is required)  \*Scenario here is dealing with research student | StudentID|Overall\_Marks|Grade  ----------------------------  2111 55.0 P  2222 64.0 C  2333 73.0 D  2444 82.0 HD  2555 91.0 HD | The program successfully displays the information of research student | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 15. | After successfully reading the student's mark by performing task 2. The user can execute task 6 to calculate and display the average overall mark and grade of all coursework and research student and display the number of coursework or research students obtained an overall mark equal to or above the average overall mark and how many obtained an overall mark below the average overall mark. (The scenario where task 2 was not performed in advance has been shown in test no. 7) | Input 1:6  (Input 6 to execute the task. No other input value is required. The program should display the expected output of research student)  \*Scenario here is dealing with research student | -> Total number of student : 5  -> Average Overall Mark : 73.0  -> Grade : D  3 students obtained an overall mark equal to or above the average overall mark of 73.0  2 students obtained an overall mark below the average overall mark of 73.0 | The program successfully display the expected output of research student. | Pass |
| 16. | This test also for task 6. The test here is to test is the user has removed a student before executing task 6. Therefore, the average overall mark and grade of all student might be changes and the total number of students in the list has also been deducted. | Input 1:6  (This scenario has 4 students in the student list because student 2111 (studentID) has been removed from the list. The average overall mark also different.  \*Scenario here is dealing with research student | -> Total number of student : 4  -> Average Overall Mark : 77.5  -> Grade : D  2 students obtained an overall mark equal to or above the average overall mark of 77.5  2 students obtained an overall mark below the average overall mark of 77.5 | Successful – because the total number of students is 4 and the average overall mark is changes. | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 17. | Task 7 allowed user to retrieve the information of coursework or research student based on the studentID. The program will ask the user to enter the studentID and display the information of the particular student. This test is to check whether this task can be executed successfully and show the correct result. | Input 1: 7  Input 2: 2222  (Example here we try to retrieve the information of student 2222(studentID))  \*Scenario here is dealing with research student | Student details :  ------------------------  Student : Mr.B2\_FN B2\_LN  Student Number : 2222  Date Of Birth : 7/17/2000  -------< Marks >--------  Oral presentation : 16.0  Final thesis : 60.0  Overall score : 64.0  Grade : C | The program successfully displays the correct student's information. | Pass |
| 18. | Task 7 allowed user to retrieve the information of coursework or research student based on the studentID. This test will test if the user tries to retrieve the information of non-existing student. The program should display a not found message to inform the user. | Input 1: 7  Input 2: 9999  (Scenario here we try to retrieve the information of student 9999. However, there is no student 9999(studentID) in the ArrayList) | Student not found  (Followed by the menu stage) | The program successfully displays the expected output. | Pass |
| 19. | Task 7 allowed user to retrieve the information of coursework or research student based on the studentID. This test will test if the user enters a non-numeric value (because studentID is a composite of 4 numerical values). The program should display an error message when the user enters a studentID with non-numeric value | Input 1: 7  Input 2: invalid  (Scenario here the user enters a studentID with non-numerical value, which is an invalid input.) | Use a numerical value !  (Followed by the menu stage) | The program successfully displays the expected output. | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 20. | Similar to task 7, task 8 also allowed the user to retrieve the student’s information. But this task takes in the first name and last name of the student (case not sensitive) and display the information of the particular student. | Input 1:8  Input 2: B5\_FN  Input 3: b5\_ln  (Scenario here the user tries to retrieve the information of the research student with name B5\_FN B5\_LN. We also use upper and lower case together to the system. There should be no issue when retrieving. | Student details :  ------------------------  Student : Mr.B5\_FN B5\_LN  Student Number : 2555  Date Of Birth : 10/20/2000  -------< Marks >--------  Oral presentation : 19.0  Final thesis : 90.0  Overall score : 91.0  Grade : HD | The program successfully displays the information of research student with name B5\_FN B5\_LN. | Pass |
| 21. | This test is to test for retrieving the information of a non-existing student in task 8. The program should display an error message when the system cannot find a same name student. | Input 1:8  Input 2: no  Input 3: exist  (In the scenario here, the user tries to retrieve the information of the student named “no exist”, which does not exist in the list.) | Student not found  (Followed by the menu stage) | The program successfully displays the expected result. | Pass |
| 22. | Task 10 allowed the user to output the sorted ArrayList into a csv file. However, if the user tries to execute task 10 before execute task 9 (sort the ArrayList in ascending order of the studentID), the program will ignore the analysis and query and display an error message to remind the user to execute task 9 first. | Input 1: 10  (Scenario here task 9 is not executed yet, means that the ArrayList is not sorted. The program should ignore this query) | ArrayList not sorted yet. Select task 9 to sort first  (Followed by the menu stage) | The program successfully displays the expected result because task is not executed yet. | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 23. | The user needs to execute task 9 first in order to execute task 10. Task 9 is used to sort the student list in ascending order of the studentID. No other input is required, the program will automatically sort the student list and display the sorted student list on the screen after the user select task 9. **In this scenario, the student list only contains 4 students. It is because the user has removed the student 2111 previously.** | Input 1: 9  (No other input is required, and program will display the sorted student list including the mark and grade.)  \*Scenario here is dealing with research student | ----Sorted student list----  Student : Mr.B2\_FN B2\_LN  Student Number : 2222  Date Of Birth : 7/17/2000  -------< Marks >--------  Oral presentation : 16.0  Final thesis : 60.0  Overall score : 64.0  Grade : C  \*Until student 1555 | The program successfully displays the information of 4 research students. | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 24. | Once the student list is sorted in ascending order. The user can execute task 10 to output the sorted student list to the csv file. The program will display a success message to inform the user after successfully output the ArrayList. After that, the user can see the sorted list of students in the csv file. **In this scenario, the total number of students is 4. Because the user has removed the student 2111 (studentID) from the ArrayList. Therefore, the csv file will contain only 4 student’s information.** | Input 1: 10  (No other input is required. The program will display a success message after the analysis and query is successfully complete) | Successfully stored in csv file  \*The sorted student list is appearing in the sortedArrayListOutput.csv file. | The sorted ArrayList successfully appear in the sortedArrayListOutput.csv  File. In addition, there is only information for 4 research students. | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 25. | In the menu stage, the program only contain task 1 – 10 for the user to perform different analysis and querys. This test will check if the user enters any task apart form 1- 10, the program should not crash and display an error message. | Input 1: -5  (Scenario here the user tries to execute task 99, which is an invalid option.) | Invalid option!  (Followed by the menu stage) | The program successfully displays the expected result and execute the menu. | Pass |
| 26. | In the menu stage, the program provide task 1 – 10 for the user to perform different analysis and querys. This test will check if the user enters a non-numerical value, the program should not crash and display an error message. | Input 1: invalidTest  (Scenario here the user enters a non-numerical value – “invalidTest”) | Use a numerical value !  (Followed by the menu stage) | The program successfully displays the expected result and execute the menu. | Pass |
| 27. | In task 3, the program will ask the user to enter studentID and remove the particular student. If the user enters the studentID with non-numerical value, the program displays an error message to inform the user. It is because studentID is comprise of 4 digits. | Input 1 = 3  Input 2 = invalid  (Scenario here we try to enter a studentID with non-numerical value.) | Invalid studentID !  (Followed by the menu stage) | The program successfully displays the expected result and execute the menu. | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 28. | This test is to test if the program can display the information of all students with the same name when performing task 8. In order to perform this test, I will temporarily change the name of the course student with student ID 1222. I changed this student’s name to be the same as student 1111 (student ID). So that, the program should display the information of both students. | Input 1: 8  (The scenario here student 1111 and student 1222 have the same name – FirstName=A1\_FN  LastName= A1\_LN) | The program should display student 1111 and student 1222 information when user enters first name A1\_FN and last name A1\_LN. | The program successfully displays the information of student 1111 and student 1222. | Pass |

**Screen capture of program’s output**

|  |  |
| --- | --- |
| Test id | Program Output |
| 1 |  |
| 2 |  |

|  |  |
| --- | --- |
| 3. |  |
| 4. |  |
| 5. |  |

|  |  |
| --- | --- |
| 6. |  |
| 7. |  |
| 8. |  |

|  |  |
| --- | --- |
| 9. | \*Until student 2555 |
| 10. | Execute task 4 and student 2111 no in the list |
| 11. |  |

|  |  |
| --- | --- |
| 12. |  |
| 13. |  |
| 14. |  |
| 15. |  |

|  |  |
| --- | --- |
| 16. |  |
| 17. |  |
| 18. |  |
| 19. |  |

|  |  |
| --- | --- |
| 20. |  |
| 21. |  |
| 22. |  |

|  |  |
| --- | --- |
| 23. | \*Until student 2555 |
| 24. | **csv file output** |

|  |  |
| --- | --- |
| 25. |  |
| 26. |  |
| 27. |  |

|  |  |
| --- | --- |
| 28. | Text  Description automatically generated |

1. **Java source code**

* **Student Class**

//Title : FT MUR T221 ICT167 C – Assignment 2 (Student class)

//Author : Tee Yee Kang

//Date : 30/Jul/2021

//File Name : FTC-34315323-Assignment 02

//Purpose : Student class is the base class. This program allowed user to create object

// The constructor required 7 different variables to create a student object. User can also create

// student class object without any arguments (default value) by using the constructor with no parameter.

// There are appropriate getter & setter method for user to retrieve or set student object value.

// Child class can also use the getGrade method to find the grade of student by pasing in the overall mark.

/\*\*

\* **@author** TeeYeeKang yeekang88 @ gmail.com

\* **@version** 13.0.2

\*/

**public** **class** Student {

//instance variables

/\*\*

\* The String title (e.g. Mr, Mrs)

\*/

**private** String title;

/\*\*

\* The String firstName

\*/

**private** String firstName;

/\*\*

\* The String lastName

\*/

**private** String lastName;

/\*\*

\* The long student number

\*/

**private** **long** studentID;

/\*\*

\* The int day of birth

\*/

**private** **int** DOB\_day;

/\*\*

\* The int month of birth

\*/

**private** **int** DOB\_month;

/\*\*

\* The int year of birth

\*/

**private** **int** DOB\_year;

//default constructor - no parameter

/\*\*

\* Default Constructor of Student(base) class

\* <p>

\* This method is used to represent the default value of String title, String firstName, String lastName, <br>

\* int studentID, int day of birth, int month of birth, int year of birth

\* </p>

\* Precondition - Nil <br>

\* Postcondition - A Student object is created with the default value of title, first name, last name <br>

\* studentID, date of birth(divided in 3 different int component e.g day/month/year).

\*/

**public** Student() {

title = "None";

firstName = "None";

lastName = "None";

studentID = 0; //

DOB\_day = 0;

DOB\_month = 0;

DOB\_year = 0;

}

//constructor with parameters

/\*\*

\* Constructor of Student(base) class

\* <p>

\* This constructor is used to represent the value of String title, String firstName, String lastName, <br>

\* int studentID, int day of birth, int month of birth, int year of birth

\* </p>

\* Precondition - The value of title, firstName and lastName must be a String type, data type of studentID is long and<br>

\* date of birth need to be int type(divided into 3 components e.g. day/month/year)<br>

\* Postcondition - A Student object is created with the value of title, firstName, lastName, studentID,<br>

\* day of birth, month of birth and year of birth.

\* **@param** InitialTitle Title of the student (e.g. Mr, Mrs, etc)

\* **@param** InitialFirstName First name of the student

\* **@param** InitialLastName Last name of the student

\* **@param** InitialStudentID Student's student number

\* **@param** InitialDOB\_day Day of birth of the student

\* **@param** InitialDOB\_month Month of birth of the student

\* **@param** InitialDOB\_year Year of birth of the student

\*/

**public** Student(String InitialTitle, String InitialFirstName, String InitialLastName,

**long** InitialStudentID, **int** InitialDOB\_day, **int** InitialDOB\_month,**int** InitialDOB\_year){

title = InitialTitle;

firstName = InitialFirstName;

lastName = InitialLastName;

studentID = InitialStudentID;

DOB\_day = InitialDOB\_day;

DOB\_month = InitialDOB\_month;

DOB\_year = InitialDOB\_year;

}

//getter and setter methods

/\*\*

\* This method is used to get the title of the student (Student object)

\* <p>

\* Precondition: Title is initialised at the Constructor <br>

\* Postcondition: The title of the Student object is returned.

\* </p>

\* **@return** title Return the Student object's title

\*/

**public** String getTitle() {

**return** title;

}

/\*\*

\* This method is used to set the new title of the student (Student object)

\* <p>

\* Precondition: Take in a new title of Student object with String value<br>

\* Postcondition: The title of the Student object is set.

\* </p>

\* **@param** NewTitle Title of Student object

\*/

**public** **void** setTitle(String NewTitle) {

title = NewTitle;

}

/\*\*

\* This method is used to get the full name(firstName + lastName) of the student (Student object)

\* <p>

\* Precondition: Both first name and last name are initialised at the Constructor <br>

\* Postcondition: The Student object's full name is returned.

\* </p>

\* **@return** lastName + " " + firstName Return the Student object's full name (firstName + lastName)

\*/

**public** String getFullName() {

**return** firstName + " " + lastName;

}

/\*\*

\* This method is used to set the student's (Student object) new full name (first name + last name)

\* <p>

\* Precondition: Take in a both new first name and last name of Student object with String values<br>

\* Postcondition: The firstName and lastName of the Student object is set.

\* </p>

\* **@param** NewFirstName First name of Student object

\* **@param** NewLastName Last name of Student object

\*/

**public** **void** setName(String NewFirstName, String NewLastName) {

firstName = NewFirstName;

lastName = NewLastName;

}

/\*\*

\* This method is used to get the student number(studentID) of the student (Student object)

\* <p>

\* Precondition: studentID is initialised at the Constructor <br>

\* Postcondition: The Student object's studentID is returned.

\* </p>

\* **@return** studentID Return the Student object's student number (studentID)

\*/

**public** **long** getStudentID() {

**return** studentID;

}

/\*\*

\* This method is used to set the student's (Student object) studentID

\* <p>

\* Precondition: Take in a new studentID of Student object with long value<br>

\* Postcondition: The studentID of the Student object is set.

\* </p>

\* **@param** NewStudentID StudentID of Student object

\*/

**public** **void** setStudentID(**long** NewStudentID) {

studentID = NewStudentID;

}

//combine day, month, and year together

/\*\*

\* This method is used to get the date of birth (day/month/year) of the student (Student object)

\* <p>

\* Precondition: DOB\_day, DOB\_month, DOB\_year are initialised at the Constructor <br>

\* Postcondition: The Student object's date of birth including day, month and year are returned.

\* </p>

\* **@return** DOB\_day + "/"+ DOB\_month + "/" + DOB\_year Return the Student object's date of birth (day/month/year)

\*/

**public** String getDOB() {

**return** DOB\_day + "/"+ DOB\_month + "/" + DOB\_year;

}

//to set a new DOB, the user have to input day, month, and year

/\*\*

\* This method is used to set the student's (Student object) date of birth including day, month and year

\* <p>

\* Precondition: Take in the including day of birth, month of birth and year of birth of Student object with int values<br>

\* Postcondition: The day of birth, month of birth and year of birth of the Student object is set.

\* </p>

\* **@param** NewDOB\_day Day of birth of Student object

\* **@param** NewDOB\_month Month of birth of Student object

\* **@param** NewDOB\_year Year of birth of Student object

\*/

**public** **void** setDOB(**int** NewDOB\_day, **int** NewDOB\_month, **int** NewDOB\_year){

DOB\_day = NewDOB\_day;

DOB\_month = NewDOB\_month;

DOB\_year = NewDOB\_year;

}

//check for same Student object by studentID

//return true is same, else return false

//argument required a Student type object

/\*\*

\* This method is used to check for same Student object by comparing 2 Student objects' studentID

\* <p>

\* Precondition: Take in a Student object (type Student)<br>

\* Postcondition: Return true is this studentID is same as the passed in Student object's studentID, else return false.

\* </p>

\* **@param** std Student type object

\* **@return** this.studentID == std.studentID True is both objects are same

\*/

**public** **boolean** equals(Student std) {

**return** (**this**.studentID == std.studentID);

}

//dummy method for calculate overall mark

/\*\*

\* This method is just a dummy calculate overall mark method

\* <p>

\* Precondition: Nil<br>

\* Postcondition: Return a dummy value 0.

\* </p>

\* **@return** 0 A dummy value

\*/

**public** **double** getOverallScore() {

**return** 0;

}

//find grade based on student's overall score

//pass in overall score and find grade

//the if-else statement will find the corresponding grade, then return the grade

/\*\*

\* This method is used to find the student's grade by passing in the overall score

\* <p>

\* Precondition: Take in a socre with double type value<br>

\* Postcondition: Return student's grade based on student's score.

\* </p>

\* **@param** score Student's overall score

\* **@return** grade Return student's grade

\*/

**public** String getGrade(**double** score) {

//get overall to find grade

**double** overall = score;

String grade = "null";

//find grade

**if**(overall >= 80) {

grade = "HD";

}**else** **if**(overall >= 70 && overall <80) {

grade = "D";

}**else** **if**(overall >=60 && overall <70) {

grade = "C";

}**else** **if**(overall >=50 && overall <60) {

grade = "P";

}**else** {

grade = "N";

}

**return** grade;

}

//display output method

//display the details of student including title, full name, date of birth and studentID

/\*\*

\* This method is used to display the details of student including title, full name, date of birth and studentID

\* <p>

\* Precondition: Nil<br>

\* Postcondition: Display the detials of student object.

\* </p>

\*/

**public** **void** writeOutput() {

System.***out***.println("Student : " + title + "." + firstName + " " + lastName);

System.***out***.println("Student Number : " + studentID);

System.***out***.println("Date Of Birth : " + DOB\_day + "/"+ DOB\_month + "/" + DOB\_year);

System.***out***.println("-------< Marks >--------");

}

//display output method in String type

/\*\*

\* This method is used to return the details of student including title, full name, date of birth and studentID in String type

\* <p>

\* Precondition: Nil<br>

\* Postcondition: Return the detials of student object.

\* </p>

\* **@return** "Student : " + title + "." + lastName + " " + firstName + "\n"

+"Student Number : " + studentID + "\n" + "Date Of Birth : " + DOB\_day + "/"+ DOB\_month + "/" + DOB\_year

+"\n" + "-------< Marks >--------" Return student's details

\*/

**public** String writeOutputStr() {

**return** ("Student : " + title + "." + firstName + " " + lastName + "\n"

+"Student Number : " + studentID + "\n" + "Date Of Birth : " + DOB\_day + "/"+ DOB\_month + "/" + DOB\_year

+"\n" + "-------< Marks >--------");

}

}

* **ResearchStudent Class**

//Title : FT MUR T221 ICT167 C – Assignment 2 (Student class)

//Author : Tee Yee Kang

//Date : 30/Jul/2021

//File Name : FTC-34315323-Assignment 02

//Purpose : There are two different type of student which is CourseWorkStudent and ResearchStudent.

// ResearchStudent class is the child class of Student class. Therefore, it required variables in base class

// and its own variables to create object. User can also create student class object without

// any arguments (default value) by using the constructor with no parameter.

// There are appropriate getter & setter method for user to retrieve or set student object value.

// It can also use the methods in the base class.

/\*\*

\* **@author** TeeYeeKang yeekang88 @ gmail.com

\* **@version** 13.0.2

\*/

**public** **class** ResearchStudent **extends** Student {

//instance variables

/\*\*

\* The double oral presentation marks

\*/

**private** **double** oralPresentation;

/\*\*

\* The double final thesis marks

\*/

**private** **double** finalThesis;

/\*\*

\* The double overall marks

\*/

**private** **double** overall;

/\*\*

\* The String student's grade

\*/

**private** String grade;

//default constructor

//super key word will inherit the default construtor in base class

/\*\*

\* Default Constructor of ResearchStudent(child) class

\* <p>

\* This method is used to represent the default value of double oralPresentation, finalThesis, overall and String grade<br>

\* </p>

\* Precondition - Nil <br>

\* Postcondition - A ResearchStudent object is created with all the default value including the value of base class<br>

\* and its own variables.

\*/

**public** ResearchStudent() {

**super**(); //call default constructor of super class Student

oralPresentation = 0;

finalThesis = 0;

overall = 0;

grade = "No grade";

}

//constructor overload - different arguments

//takes in the variables of Student class only

/\*\*

\* Overloading Constructor of ResearchStudent(child) class

\* <p>

\* This method is used to create ResearchStudent object without all the marks<br>

\* </p>

\* Precondition - Takes in Student object's variables. The value of title, firstName and lastName must be a String type,<br>

\* data type of studentID is long.<br>

\* date of birth need to be int type(divided into 3 components e.g. day/month/year)<br>

\* Postcondition - A ResearchStudent object is created with Student class variables only.

\* **@param** InitialTitle Title of the student (e.g. Mr, Mrs, etc)

\* **@param** InitialFirstName First name of the student

\* **@param** InitialLastName Last name of the student

\* **@param** InitialStudentID Student's student number

\* **@param** InitialDOB\_day Day of birth of the student

\* **@param** InitialDOB\_month Month of birth of the student

\* **@param** InitialDOB\_year Year of birth of the student

\*/

**public** ResearchStudent(String InitialTitle, String InitialFirstName, String InitialLastName,

**long** InitialStudentID, **int** InitialDOB\_day, **int** InitialDOB\_month,**int** InitialDOB\_year) {

**super**(InitialTitle,InitialFirstName, InitialLastName,InitialStudentID,InitialDOB\_day,InitialDOB\_month,InitialDOB\_year);

}

//constructor overload - different arguments

//takes in the value of ResearchStudent class only

/\*\*

\* Overloading Constructor of ResearchStudent(child) class

\* <p>

\* This method is used to create ResearchStudent object with marks only and without the variables of Student class<br>

\* </p>

\* Precondition - Takes in only double type of oral presentation marks and final thesis mark.<br>

\* Postcondition - A ResearchStudent object is created with all the marks only.

\* **@param** InitialOralPresentation Oral presentation marks

\* **@param** InitialFinalThesis Final thesis marks

\*/

**public** ResearchStudent(**double** InitialOralPresentation, **double** InitialFinalThesis) {

**super**(); //call default constructor of super class Student

oralPresentation = InitialOralPresentation;

finalThesis = InitialFinalThesis;

}

//constructor with all parameters

//super key word will inherit the construtor with parameters in base class

/\*\*

\* Constructor of ResearchStudent(child) class

\* <p>

\* This constructor is used to represent the value of String title, String firstName, String lastName, <br>

\* int studentID, int day of birth, int month of birth, int year of birth, double InitialOralPresentation marks,

\* and double InitialFinalThesis marks. Overall marks and grade not included because it has to be calculated.

\* </p>

\* Precondition - The value of title, firstName and lastName must be a String type, data type of studentID is long,<br>

\* date of birth need to be int type(divided into 3 components e.g. day/month/year)<br>

\* and double type of oral presentation marks and final thesis marks<br>

\* Postcondition - A ResearchStudent object is created with the value of title, firstName, lastName, studentID,<br>

\* day of birth, month of birth, year of birth, oralPresentation and finalThesis.

\* **@param** InitialTitle Title of the student (e.g. Mr, Mrs, etc)

\* **@param** InitialFirstName First name of the student

\* **@param** InitialLastName Last name of the student

\* **@param** InitialStudentID Student's student number

\* **@param** InitialDOB\_day Day of birth of the student

\* **@param** InitialDOB\_month Month of birth of the student

\* **@param** InitialDOB\_year Year of birth of the student

\* **@param** InitialOralPresentation Oral presentation marks

\* **@param** InitialFinalThesis Final thesis marks

\*/

**public** ResearchStudent(String InitialTitle, String InitialFirstName, String InitialLastName,

**long** InitialStudentID, **int** InitialDOB\_day, **int** InitialDOB\_month,**int** InitialDOB\_year,

**double** InitialOralPresentation, **double** InitialFinalThesis) {

//call default constructor of super class Student and pass in the arguments

**super**(InitialTitle,InitialFirstName, InitialLastName,InitialStudentID,InitialDOB\_day,InitialDOB\_month,InitialDOB\_year);

oralPresentation = InitialOralPresentation;

finalThesis = InitialFinalThesis;

}

//getter and setter methods

/\*\*

\* This method is used to get the oral presentation marks of research student(ResearchStudent object)

\* <p>

\* Precondition: oralPresentation is initialised at the Constructor <br>

\* Postcondition: The oral presentation mark of the Student object is returned.

\* </p>

\* **@return** oralPresentation Return the ResearchStudent object's oral presentation mark

\*/

**public** **double** getOralPresentation() {

**return** oralPresentation;

}

/\*\*

\* This method is used to set the new oral presentation mark of research student(ResearchStudent object)

\* <p>

\* Precondition: Take in a new oral presentation mark of double value<br>

\* Postcondition: The oral presentation mark of the ResearchStudent object is set.

\* </p>

\* **@param** NewOralPresentation Oral presentation mark of ResearchStudent object

\*/

**public** **void** setOralPresentation(**double** NewOralPresentation) {

oralPresentation = NewOralPresentation;

}

/\*\*

\* This method is used to get the final thesis marks of research student(ResearchStudent object)

\* <p>

\* Precondition: finalThesis is initialised at the Constructor <br>

\* Postcondition: The final thesis marks of research student(ResearchStudent object) is returned.

\* </p>

\* **@return** finalThesis Return the ResearchStudent object's final thesis mark

\*/

**public** **double** getFinalThesis() {

**return** finalThesis;

}

/\*\*

\* This method is used to set the new final thesis mark of research student(ResearchStudent object)

\* <p>

\* Precondition: Take in a new final thesis mark of double value<br>

\* Postcondition: The final thesis mark of the ResearchStudent object is set.

\* </p>

\* **@param** NewFinalThesis Final thesis mark of ResearchStudent object

\*/

**public** **void** setFinalThesis(**double** NewFinalThesis) {

finalThesis = NewFinalThesis;

}

//calculate the overall score of research student

//return sum of oralPresentationMarks and finalThesisMarks but

//oralPresentationMarks worth 20 marks so required futher calculation

//finalThesisMarks worth 80% of the total overall marks

/\*\*

\* This method is used to calculate the overall mark of research student(ResearchStudent object)

\* <p>

\* Precondition: Nil<br>

\* Postcondition: Return the overall mark of student after the calcualtion is completed.

\* </p>

\* **@return** overall Return the overall mark of research student (ResearchStudent object)

\*/

**public** **double** getOverallScore() {

//calculate each mark

**double** oralPresentationMarks = **this**.oralPresentation; //worth 20% and max score is 20 so no required futhur calculation

**double** finalThesisMarks = **this**.finalThesis \* 80 / 100;//worth 80%

//calculate overall score

overall = oralPresentationMarks + finalThesisMarks;

**return** overall;

}

//display output method

//display the details of research student including base class varaibles value and itw own marks and grade

//super key word inherit the writeOutput method in base class

/\*\*

\* This method is used to display the details of research student including base class variables

\* (title, full name, date of birth and studentID), its own variables oralPresenation, finalThesis and

\* call getOverallScore and getGrade method to get the overall mark and grade respectively

\* <p>

\* Precondition: Nil<br>

\* Postcondition: Display the detials of ResearchStudent object including base class variables.

\* </p>

\*/

**public** **void** writeOutput() {

**super**.writeOutput();//call base class writeOutput method

System.***out***.println("Oral presentation : " + oralPresentation);

System.***out***.println("Final thesis : " + finalThesis);

System.***out***.println("Overall score : " + getOverallScore());

System.***out***.println("Grade : " + getGrade(overall)); //call getGrade method in base class

System.***out***.println("");

}

//display output method in String type

//super key word inherit the writeOutputStr method in base class

/\*\*

\* This method is used to return the details of research student including base class varaibles

\* (title, full name, date of birth and studentID in String type) and its own variables oralPresenation, finalThesis and

\* call getOverallScore and getGrade method to get the overall mark and grade respectively

\* <p>

\* Precondition: Nil<br>

\* Postcondition: Return the detials of ResearchStudent object including base class variables.

\* </p>

\* **@return** super.writeOutputStr() + "\n" + "Oral presentation : " + oralPresentation

+ "\n" + "Final thesis : " + finalThesis + "\n" + "Overall score : " + getOverallScore()

+ "\n" + "Grade : " + getGrade(overall) + "\n" Return student's details

\*/

**public** String writeOutputStr() {

**return** (**super**.writeOutputStr() + "\n" + "Oral presentation : " + oralPresentation

+ "\n" + "Final thesis : " + finalThesis + "\n" + "Overall score : " + getOverallScore()

+ "\n" + "Grade : " + getGrade(overall) + "\n");

}

}

* **CourseWorkStudent Class**

//Title : FT MUR T221 ICT167 C – Assignment 2 (Student class)

//Author : Tee Yee Kang

//Date : 30/Jul/2021

//File Name : FTC-34315323-Assignment 02

//Purpose : There are two different type of student which is CourseWorkStudent and ResearchStudent.

// CourseWorkStudent class is the child class of Student class. Therefore, it required variables in base class

// and its own variables to create object. User can also create student class object without

// any arguments (default value) by using the constructor with no parameter.

// There are appropriate getter & setter method for user to retrieve or set student object value.

// It can also use the methods in the base class.

/\*\*

\* **@author** TeeYeeKang yeekang88 @ gmail.com

\* **@version** 13.0.2

\*/

**public** **class** CourseWorkStudent **extends** Student{

//instance variables

/\*\*

\* The double assignment1 marks

\*/

**private** **double** assignment1;

/\*\*

\* The double assignment2 marks

\*/

**private** **double** assignment2;

/\*\*

\* The double practical work marks

\*/

**private** **double** practicalWork;

/\*\*

\* The double final exam marks

\*/

**private** **double** finalExam;

/\*\*

\* The double overall marks

\*/

**private** **double** overall;

/\*\*

\* The String student's grade

\*/

**private** String grade;

//default constructor

//super key word will inherit the default construtor in base class

/\*\*

\* Default Constructor of CourseWorkStudent(child) class

\* <p>

\* This method is used to represent the default value of double assignment1, assignment2, practicalWork, finalExam<br>

\* overall and String grade

\* </p>

\* Precondition - Nil <br>

\* Postcondition - A CourseWorkStudent object is created with all the default value including the value of base class<br>

\* and its own variables.

\*/

**public** CourseWorkStudent() {

**super**(); //call default constructor of super class Student

assignment1 = 0;

assignment2 = 0;

practicalWork = 0;

finalExam = 0;

overall = 0;

grade = "No grade";

}

//constructor overload - different arguments

//takes in the variables of Student class only

/\*\*

\* Overloading Constructor of CourseWorkStudent(child) class

\* <p>

\* This method is used to create CourseWorkStudent object without all the marks<br>

\* </p>

\* Precondition - Takes in Student object's variables. The value of title, firstName and lastName must be a String type,<br>

\* data type of studentID is long.<br>

\* date of birth need to be int type(divided into 3 components e.g. day/month/year)<br>

\* Postcondition - A CourseWorkStudent object is created with Student class variables only.

\* **@param** InitialTitle Title of the student (e.g. Mr, Mrs, etc)

\* **@param** InitialFirstName First name of the student

\* **@param** InitialLastName Last name of the student

\* **@param** InitialStudentID Student's student number

\* **@param** InitialDOB\_day Day of birth of the student

\* **@param** InitialDOB\_month Month of birth of the student

\* **@param** InitialDOB\_year Year of birth of the student

\*/

**public** CourseWorkStudent(String InitialTitle, String InitialFirstName, String InitialLastName,

**long** InitialStudentID, **int** InitialDOB\_day, **int** InitialDOB\_month,**int** InitialDOB\_year) {

**super**(InitialTitle,InitialFirstName, InitialLastName,InitialStudentID,InitialDOB\_day,InitialDOB\_month,InitialDOB\_year);

}

//constructor overload - different arguments

//takes in the value of ResearchStudent class only

/\*\*

\* Overloading Constructor of CourseWorkStudent(child) class

\* <p>

\* This method is used to create CourseWorkStudent object with marks only and without the variables of Student class<br>

\* </p>

\* Precondition - Takes in only double type of assignment1 marks, assignment2 mark, practical work mark and final exam marks.<br>

\* Postcondition - A CourseWorkStudent object is created with all the marks only.

\* **@param** InitialAssignment1 Assignment 1 marks

\* **@param** InitialAssignment2 Assignment 2 marks

\* **@param** InitialPracticalWork Practical work marks

\* **@param** InitialFinalExam Final exam mark

\*/

**public** CourseWorkStudent(**double** InitialAssignment1, **double** InitialAssignment2, **double** InitialPracticalWork,**double** InitialFinalExam) {

**super**(); //call default constructor of super class Student

assignment1 = InitialAssignment1;

assignment2 = InitialAssignment2;

practicalWork = InitialPracticalWork;

finalExam = InitialFinalExam;

}

//constructor with parameters

//super key word will inherit the construtor with parameters in base class

/\*\*

\* Constructor of CourseWorkStudent(child) class

\* <p>

\* This constructor is used to represent the value of String title, String firstName, String lastName, <br>

\* int studentID, int day of birth, int month of birth, int year of birth, double InitialAssignment1 marks,

\* double InitialAssignment2 marks, double InitialPracticalWork mark and double InitialFinalExam mark.

\* Overall marks and grade not included because it has to be calculated.

\* </p>

\* Precondition - The value of title, firstName and lastName must be a String type, data type of studentID is long,<br>

\* date of birth need to be int type(divided into 3 components e.g. day/month/year)<br>

\* and double type of assignment1 marks, assignment2 mark, practical work mark and final exam marks<br>

\* Postcondition - A CourseWorkStudent object is created with the value of title, firstName, lastName, studentID,<br>

\* day of birth, month of birth, year of birth, assignment1, assignment2, practicalWork and finalExam.

\* **@param** InitialTitle Title of the student (e.g. Mr, Mrs, etc)

\* **@param** InitialFirstName First name of the student

\* **@param** InitialLastName Last name of the student

\* **@param** InitialStudentID Student's student number

\* **@param** InitialDOB\_day Day of birth of the student

\* **@param** InitialDOB\_month Month of birth of the student

\* **@param** InitialDOB\_year Year of birth of the student

\* **@param** InitialAssignment1 Assignment 1 marks

\* **@param** InitialAssignment2 Assignment 2 marks

\* **@param** InitialPracticalWork Practical work marks

\* **@param** InitialFinalExam Final exam mark

\*/

**public** CourseWorkStudent(String InitialTitle, String InitialFirstName, String InitialLastName,

**long** InitialStudentID, **int** InitialDOB\_day, **int** InitialDOB\_month,**int** InitialDOB\_year,

**double** InitialAssignment1, **double** InitialAssignment2, **double** InitialPracticalWork,**double** InitialFinalExam){

//call default constructor of super class Student and pass in the arguments

**super**(InitialTitle,InitialFirstName,InitialLastName,InitialStudentID,InitialDOB\_day,InitialDOB\_month,InitialDOB\_year);

assignment1 = InitialAssignment1;

assignment2 = InitialAssignment2;

practicalWork = InitialPracticalWork;

finalExam = InitialFinalExam;

}

//getter and setter methods

/\*\*

\* This method is used to get the assignment 1 marks of coursework student(CourseWorkStudent object)

\* <p>

\* Precondition: assignment1 is initialised at the Constructor <br>

\* Postcondition: The assignment 1 mark of the CourseWorkStudent object is returned.

\* </p>

\* **@return** assignment1 Return the CourseWorkStudent object's assignmetn 1 mark

\*/

**public** **double** getAssigment1() {

**return** assignment1;

}

/\*\*

\* This method is used to set the new assignment 1 mark of coursework student(CourseWorkStudent object)

\* <p>

\* Precondition: Take in a new assignment 1 mark of double value<br>

\* Postcondition: The assignment 1 mark of the CourseWorkStudent object is set.

\* </p>

\* **@param** NewAssignment1 Assignment 1 mark of CourseWorkStudent object

\*/

**public** **void** setAssignment1(**double** NewAssignment1) {

assignment1 = NewAssignment1;

}

/\*\*

\* This method is used to get the assignment 2 marks of coursework student(CourseWorkStudent object)

\* <p>

\* Precondition: assignment2 is initialised at the Constructor <br>

\* Postcondition: The assignment 2 mark of the CourseWorkStudent object is returned.

\* </p>

\* **@return** assignment2 Return the CourseWorkStudent object's assignment 2 mark

\*/

**public** **double** getAssignment2() {

**return** assignment2;

}

/\*\*

\* This method is used to set the new assignment 2 mark of coursework student(CourseWorkStudent object)

\* <p>

\* Precondition: Take in a new assignment 2 mark of double value<br>

\* Postcondition: The assignment 1 mark of the CourseWorkStudent object is set.

\* </p>

\* **@param** NewAssignment2 Assignment 2 mark of CourseWorkStudent object

\*/

**public** **void** setAssignment2(**double** NewAssignment2) {

assignment2 = NewAssignment2;

}

/\*\*

\* This method is used to get the practical work marks of coursework student(CourseWorkStudent object)

\* <p>

\* Precondition: practicalWork is initialised at the Constructor <br>

\* Postcondition: The practical work mark of the CourseWorkStudent object is returned.

\* </p>

\* **@return** practicalWork Return the CourseWorkStudent object's practical work mark

\*/

**public** **double** getPracticalWork() {

**return** practicalWork;

}

/\*\*

\* This method is used to set the new practical work mark of coursework student(CourseWorkStudent object)

\* <p>

\* Precondition: Take in a new practical work mark of double value<br>

\* Postcondition: The practical work mark of the CourseWorkStudent object is set.

\* </p>

\* **@param** NewPracticalWork Practical work mark of CourseWorkStudent object

\*/

**public** **void** setPracticalWork(**double** NewPracticalWork) {

practicalWork = NewPracticalWork;

}

/\*\*

\* This method is used to get the final exam marks of coursework student(CourseWorkStudent object)

\* <p>

\* Precondition: finalExam is initialised at the Constructor <br>

\* Postcondition: The final exam mark of the CourseWorkStudent object is returned.

\* </p>

\* **@return** finalExam Return the CourseWorkStudent object's final exam mark

\*/

**public** **double** getFinalExam() {

**return** finalExam;

}

/\*\*

\* This method is used to set the new final exam mark of coursework student(CourseWorkStudent object)

\* <p>

\* Precondition: Take in a new final exam mark of double value<br>

\* Postcondition: The final exam mark of the CourseWorkStudent object is set.

\* </p>

\* **@param** NewFinalExam Final exam mark of CourseWorkStudent object

\*/

**public** **void** setFinalExam(**double** NewFinalExam) {

finalExam = NewFinalExam;

}

//calculate the overall score of research student

//return sum of assignment1Score, assignment2Score, practicalWorkScore and finalExamScore

//assignment 1 and 2 worth 25%, finalExam worth 30% and practicalWork no required further calculation

/\*\*

\* This method is used to calculate the overall mark of coursework student(CourseWorkStudent object)

\* <p>

\* Precondition: Nil<br>

\* Postcondition: Return the overall mark of student after the calcualtion is completed.

\* </p>

\* **@return** overall Return the overall mark of coursework student(CourseWorkStudent object)

\*/

**public** **double** getOverallScore() {

//calculate each mark

**double** assignment1Score= **this**.assignment1 \* 25 / 100;

**double** assignment2Score= **this**.assignment2 \* 25 / 100;

**double** practicalWorkScore = **this**.practicalWork; //marked out of a maximum of 20 marks, no further calculation

**double** finalExamScore = **this**.finalExam \* 30 / 100 ;

//calculate overall score

overall = assignment1Score + assignment2Score + practicalWorkScore + finalExamScore;

**return** overall;

}

//display output method

//display the details of coursework student including base class varaibles value and itw own marks and grade

//super key word inherit the writeOutput method in base class

/\*\*

\* This method is used to display the details of coursework student including base class variables

\* (title, full name, date of birth and studentID), its own variables assignment1, assignment2,

\* practicalWork, finalExam and call getOverallScore and getGrade method to get the overall mark and grade respectively

\* <p>

\* Precondition: Nil<br>

\* Postcondition: Display the detials of CourseWorkStudent object including base class variables.

\* </p>

\*/

**public** **void** writeOutput() {

**super**.writeOutput();//call base class writeOutput method

System.***out***.println("Assignment 1 : " + assignment1);

System.***out***.println("Assignment 2 : " + assignment2);

System.***out***.println("Practical work : " + practicalWork);

System.***out***.println("Final exam : " + finalExam);

System.***out***.println("Overall mark : " + getOverallScore());

System.***out***.println("Grade : " + getGrade(overall)); //call getGrade method in base class

System.***out***.println("");

}

//display output method in String type

//super key word inherit the writeOutputStr method in base class

/\*\*

\* This method is used to return the details of coursework student including base class varaibles

\* (title, full name, date of birth and studentID in String type) and its own variables assignment1, assignment2,

\* practicalWork, finalExam and call getOverallScore and getGrade method to get the overall mark and grade respectively

\* <p>

\* Precondition: Nil<br>

\* Postcondition: Return the detials of CourseWorkStudent object including base class variables.

\* </p>

\* **@return** super.writeOutputStr() + "\n" + "Assignment 1 : " + assignment1 + "\n"

+ "Assignment 2 : " + assignment2 + "\n" + "Practical work : " + practicalWork

+ "\n" + "Final exam : " + finalExam + "\n" + "Overall mark : " + getOverallScore()

+ "\n" + "Grade : " + getGrade(overall) + "\n" Return student's details

\*/

**public** String writeOutputStr() {

**return**(**super**.writeOutputStr() + "\n" + "Assignment 1 : " + assignment1 + "\n"

+ "Assignment 2 : " + assignment2 + "\n" + "Practical work : " + practicalWork

+ "\n" + "Final exam : " + finalExam + "\n" + "Overall mark : " + getOverallScore()

+ "\n" + "Grade : " + getGrade(overall) + "\n");

}

}

* **Client Class**

//Title : FT MUR T221 ICT167 C – Assignment 2 (Student class)

//Author : Tee Yee Kang

//Date : 30/Jul/2021

//File Name : FTC-34315323-Assignment 02

//Purpose : The client class allowed user to create either CourseWorkStudent or ResearchStudent object. User here

// read the value in the csv.file to create the research or course work student instead of keyboard value.

// The client class also contains several methods for user to conduct some tasks. The program will keep executing

// until user choose to exit the program. The program can also handle invalid input value by displaying some error

// message instead of crash the program.

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.io.PrintWriter;

**import** java.util.ArrayList;

**import** java.util.InputMismatchException;

**import** java.util.Scanner;

/\*\*

\* **@author** TeeYeeKang yeekang88 @ gmail.com

\* **@version** 13.0.2

\*/

**public** **class** Client {

//declare static Scanner to read keyboard input

**static** Scanner *input* = **new** Scanner(System.***in***);

**public** **static** **void** main(String[] args) {

//display student information

*studentInfo*();

//declare variables

String fileName = "";

**boolean** valid = **false**;

**int** opt = -1;

**do**{

**try** {

//set valid to true

valid = **true**;

//ask for user's choice between courseWorkStd / researchStd

//keep asking until user entered a valid choice

System.***out***.print("Enter 0 for coursework student / 1 for research student: ");

opt = *input*.nextInt();

System.***out***.print("\n");

//declare file name based on user's option

**if**(opt == 0) {

fileName = "coursework.csv";

System.***out***.println("---< Coursework student >---"); //just some simple output after user input

//break;

}**else** **if**(opt == 1) {

fileName = "research.csv";

System.***out***.println("---< Research student >---");

//break;

}**else** {

System.***out***.println("Invalid option"); //if user entered a valid input

valid = **false**;

}

//catch InputMismatchException when user enterd non-numerical option for type of student

//display error message and terminate the program

}**catch**(InputMismatchException e) {

System.***out***.println("\nOption must be numeric value");

//valid = false;

System.*exit*(0);

}

}**while**(!valid);

//declare ArrayList to store objects

ArrayList<Student> studentList = **new** ArrayList<Student>();

**try** {

//create instance of File and Scanner to read value from file

Scanner file = **new** Scanner(**new** File("students.csv"));

//check if there is anymore line in file

//read and print line from file

**while**(file.hasNextLine()) {

//retrieve data from student.csv file

String line = file.nextLine();

//split each values by comma and store into an array

String[] data = line.split(",");

//retreive all value

String type = data[0];

String title = data[1];

String firstName = data[2];

String lastName = data[3];

**long** studentID = Integer.*parseInt*(data[4]); //convert String value into long

**int** day = Integer.*parseInt*(data[5]); //convert String value into int

**int** month = Integer.*parseInt*(data[6]);

**int** year = Integer.*parseInt*(data[7]);

//create correspond student object based on user's option

**if**(opt == 0) {

**if**(type.equals("coursework")){

//create CourseWorkStudent object by using the overload constructor with all parameters

//all requied marks set as 0 first

studentList.add(**new** CourseWorkStudent(title,firstName,lastName,studentID,day,month,year,0,0,0,0));

}

}**else** {

//create ResearchStudent object by using the overload constructor with Student class variables only

//The reason for using different constructors is just to test that all constructors are work

**if**(type.equals("research")){

studentList.add(**new** ResearchStudent(title,firstName,lastName,studentID,day,month,year));

}

}

}//end of while loop

//catch FileNotFoundException if file not found

}**catch**(FileNotFoundException e) {

System.***out***.println("File not found");

System.*exit*(0);

}

//menu - main controller for all tasks

*process*(studentList,opt);

}//end of main class

//methods

//task 2 - The object we created previously have no marks yet (set as 0 for all marks). Here read value

// from particular file to set all marks for CourseWorkStudent or ResearchStudent object

// Arguments required a Student type ArrayList and a numeric value with int type

/\*\*

\* This method is used to read the value from the student.csv file to create CourseWorkStudent or ResearchStudent object<br>

\* based on user's option

\*\* <p>

\* Precondition - Pass in a Student type ArrayList and an option with int value.<br>

\* Postcondition - Create either CourseWorkStudent or ResearchStudent object and store it into the ArrayList by reading<br>

\* all values from the student.csv file.

\* </p>

\* **@param** studentList An ArrayList of type Student

\* **@param** opt User's option on CourseWorkStudent or ResearchStudent

\*/

**public** **static** **void** readMarks(ArrayList<Student> studentList, **int** opt) {

//declare counter for the used of ArrayList index

**int** counter = 0;

//deal with course work student

**if**(opt == 0){

**try** {

//create instance of File and Scanner to read value from file coursework.csv

Scanner courseworkFile = **new** Scanner(**new** File("coursework.csv"));

//check if there is anymore line in file

//read and print line from file

**while**(courseworkFile.hasNextLine()) {

//retrieve data from coursework.csv file

String courseworkLine = courseworkFile.nextLine();

//split each values by comma and store into an array

String[] courseworkMarks = courseworkLine.split(",");

//retreive all value

String courseworkStdID = courseworkMarks[0];

**double** assignment1 = Integer.*parseInt*(courseworkMarks[1]);

**double** assignment2 = Integer.*parseInt*(courseworkMarks[2]);

**double** practical = Integer.*parseInt*(courseworkMarks[3]);

**double** exam = Integer.*parseInt*(courseworkMarks[4]);

//set marks for coursework students

**if**(studentList.get(counter) **instanceof** CourseWorkStudent){

((CourseWorkStudent)studentList.get(counter)).setAssignment1(assignment1);

((CourseWorkStudent)studentList.get(counter)).setAssignment2(assignment2);

((CourseWorkStudent)studentList.get(counter)).setPracticalWork(practical);

((CourseWorkStudent)studentList.get(counter)).setFinalExam(exam);

}

//incease counter by 1 for next ArrayList object

counter++;

}//end of while loop

}**catch**(FileNotFoundException e) { //catch FileNotFoundException if file not found

System.***out***.println("File not found");

System.*exit*(0);

}

//deal with research student

}**else** {

**try** {

//create instance of File and Scanner to read value from file research.csv

Scanner researchFile = **new** Scanner(**new** File("research.csv"));

//check if there is anymore line in file

//read and print line from file

**while**(researchFile.hasNextLine()) {

//retrieve data from research.csv file

String researchLine = researchFile.nextLine();

String[] researchMarks = researchLine.split(",");

String researchStdID = researchMarks[0];

**double** oral = Double.*parseDouble*(researchMarks[1]);

**double** thesis = Double.*parseDouble*(researchMarks[2]);

//set marks for research students

((ResearchStudent)studentList.get(counter)).setOralPresentation(oral);

((ResearchStudent)studentList.get(counter)).setFinalThesis(thesis);

//incease counter by 1 for next ArrayList object

counter++;

}//end of while loop

}**catch**(FileNotFoundException e) {

System.***out***.println("File not found");

System.*exit*(0);

}

}

//print successful message

System.***out***.println("");

System.***out***.println("Success - read all marks");

}

//task 3 - Remove student from ArrayList by entered the studentID.

// To complete this task, user have to enter yes in the confirmation query, else program will

// not remove any student from the ArrayList. The program can also handle error such as invalid input

// and non existing student.

// Argument required a Student type ArrayList

/\*\*

\* This method is used to remove object from the ArrayList by enter the particular studentID

\*\* <p>

\* Precondition - Pass in a Student type ArrayList .<br>

\* Postcondition - The student/object successfully removed from the ArrayList if the studentID is found and <br>

\* user entered yes in the confirmation query. Else, program will ignore this transaction.

\* </p>

\* **@param** studentList An ArrayList of type Student

\*/

**public** **static** **void** removeStd(ArrayList<Student> studentList) {

//declare boolean for display output purpose

**boolean** isFound = **false**;

**try** {

//prompt for studentID to be remove

System.***out***.print("Enter the studentID of student you want to remove: ");

**long** stdToRemove = *input*.nextInt();

//double confirm with the user

System.***out***.print("Confirm remove student " + stdToRemove + " ? Yes/No: ");

String opt = *input*.next();

//remove student/object

**if**(opt.equalsIgnoreCase("yes")) {

//loop through to find the student

**for**(**int** idx = 0; idx<studentList.size();idx++) {

//find same studentID

**if**(studentList.get(idx).getStudentID() == stdToRemove) {

studentList.remove(idx); //use the pre-define method -remove to remove std if found

isFound = **true**;

**break**; //break the loop once the studentID is found

}

}

//display successful or not found message

System.***out***.println("");

**if**(isFound == **true**) {

System.***out***.println("Student " + stdToRemove + " successfully removed");

}**else** {

System.***out***.println("Student " + stdToRemove + " not found");

}

}**else** **if** (opt.equalsIgnoreCase("no")) {

System.***out***.println("\nSee you then ~");

}**else** {

**throw** **new** InputMismatchException(); //any option apart from yes or no will throw InputMismatchException

}

//catch the InputMismatchException if the user enters a non-numerical studentID

}**catch**(InputMismatchException ex) {

System.***out***.println("\nInvalid option !");

}

}

//task 4 - Display all the details of CourseworkStudent or ResearchStudent.

// Pass in the Student type ArrayList, inside the for-each loop will call the writeOutput method

// in order to print the details of CourseworkStudent or ResearchStudent.

/\*\*

\* This method is used to output all the detials of CourseworkStudent or ResearchStudent

\*\* <p>

\* Precondition - Pass in a Student type ArrayList .<br>

\* Postcondition - The program successfully display all the detials of CourseworkStudent or ResearchStudent <br>

\* including student's identification, marks, grade, etc.

\* </p>

\* **@param** studentList An ArrayList of type Student

\*/

**public** **static** **void** display(ArrayList<Student> studentList) {

//print output

System.***out***.println("");

//for-each loop

**for**(Student std : studentList) {

std.writeOutput(); //call the writeOutput method to display all output

}

}

//task 5 - Calculate the overall marks of all CourseworkStudent or ResearchStudent in the ArrayList.

// This method is to find the individual overall mark of particular student by passing in the

// Student type ArrayList, option of which type of student with int value and the index of particular student.

/\*\*

\* This method is used to to find the individual overall mark of particular student for furthur task 5 calculation

\*\* <p>

\* Precondition - Pass in a Student type ArrayList, option of which type of student with int value and the index of particular student.<br>

\* Postcondition - The program will calculate the student's overall mark by calling the getOverallScore() method<br>

\* and return the overall mark.

\* </p>

\* **@param** studentList An ArrayList of type Student

\* **@param** opt Option of which type of student

\* **@param** idx Index of the student/object to be dealing with

\* **@return** overallMarks Return the overall mark of the particular student/object

\*/

**public** **static** **double** getIndividualOverall(ArrayList<Student> studentList, **int** opt, **int** idx){

//declare variables to be return

**double** overallMarks = 0;

//opt 0 is CourseWorkStudent, else/1 is ResearchStudent. Validation of opt check are done in the main class

**if**(opt == 0) {

//call the getOverallMarks in child class to calculate and return the overall marks

overallMarks = ((CourseWorkStudent)studentList.get(idx)).getOverallScore();

}**else** {

overallMarks = ((ResearchStudent)studentList.get(idx)).getOverallScore();

}

**return** overallMarks;

}

//task 6 - Get average overall mark of all students/objects in the ArrayList

// Arguments required Student type ArrayList and option of which type of student with int value.

/\*\*

\* This method is used to to find the average overall mark of all students in the ArrayList

\*\* <p>

\* Precondition - Pass in a Student type ArrayList and the option of which type of student with int value.<br>

\* Postcondition - The for loop will keep calling the getIndividualOverall() method to get each individual<br>

\* overall mark and store in the running total until reach the ArrayList size. Lastly,<br>

\* return the running total.

\* </p>

\* **@param** studentList An ArrayList of type Student

\* **@param** opt Option of which type of student

\* **@return** avgOverallMark Return the overall mark of all CourseworkStudent or ResearchStudent.

\*/

**public** **static** **double** avgOverallMark(ArrayList<Student> studentList, **int** opt) {

//declare variables

**double** total = 0;

**double** avgOverallMark = 0;

//calculation of total

**for**(**int** idx=0; idx<studentList.size(); idx++) {

total += *getIndividualOverall*(studentList, opt, idx); //call getIndividualOverall() to calculate the total

}

//calculate average overall mark

avgOverallMark = total / studentList.size();

**return** avgOverallMark;

}

//task 5 controller - Calculate + display individual overall mark and grade

// Arguments required Student type ArrayList and option of which type of student with int value.

/\*\*

\* This method is the controller of task 5.

\*\* <p>

\* Precondition - Pass in a Student type ArrayList and the option of which type of student with int value.<br>

\* Postcondition - The method will find and output the individual overall mark and grade of of all CourseworkStudent or ResearchStudent <br>

\* </p>

\* **@param** studentList An ArrayList of type Student

\* **@param** opt Option of which type of student

\*/

**public** **static** **void** outputOverallAndGrade(ArrayList<Student> studentList, **int** opt) {

//declare variables for working storage

**double** overall = 0; //this overall is to displaying all individual student's overall marks

String grade = **null**; //Individual student grade

//display output for individual overall mark

System.***out***.println("");

System.***out***.println("StudentID | Overall\_Marks | Grade");

System.***out***.println("--------------------------------------");

**for**(**int** idx=0; idx<studentList.size(); idx++) {

//get individual overall score and grade first

overall = *getIndividualOverall*(studentList, opt, idx);

grade = studentList.get(idx).getGrade(overall);

//find for the particular studentID for displaying output

**long** studentID = studentList.get(idx).getStudentID();

//display individual overall marks and grade

System.***out***.println(" "+studentID + "\t\t" + overall + "\t\t" + grade);

}

}

//task 6 - Determine and display how many coursework or research students obtained an overall mark

// equal to or above the average overall mark and how many obtained an overall mark below the

// average overall mark

// Arguments required Student type ArrayList and option of which type of student with int value.

/\*\*

\* This method is to find number of student obtained an overall mark equal/above the average overall <br>

\* mark and the number of student obtained an overall mark below the average overall mark

\*\* <p>

\* Precondition - Pass in a Student type ArrayList and the option of which type of student with int value.<br>

\* Postcondition - The first part of the method will get and output the total number of student,<br>

\* average overall mark and grade of all students <br>

\* Then the for loop will calculate the number of student obtained an overall mark <br>

\* equal to or above the average overall mark and how many obtained an overall mark below the<br>

\* average overall mark.

\* </p>

\* **@param** studentList An ArrayList of type Student

\* **@param** opt Option of which type of student

\*/

**public** **static** **void** aboveOrBelowOverall(ArrayList<Student> studentList, **int** opt) {

//declare variables for working storage

**double** avgOverall = *avgOverallMark*(studentList, opt); //get average overall mark of all student

**int** equalOrAbove = 0;

**int** below = 0;

**double** averageOverall = 0; //average overall mark of all students

String avgOverallMarkGrade = **null**; //average grade of all students

//calculate and display average overall mark of all students

averageOverall = *avgOverallMark*(studentList, opt);

//find grade for average overall mark of all students

avgOverallMarkGrade = studentList.get(0).getGrade(averageOverall);

//display output

System.***out***.println("\n-> Total number of student : " + studentList.size());

System.***out***.println("-> Average Overall Mark : " + averageOverall);

System.***out***.println("-> Grade : " + avgOverallMarkGrade);

//calculate the number of student equal/above or below the average overall mark

**for**(**int** idx=0; idx<studentList.size(); idx++) {

//deal with coursework student

**if**(opt == 0) {

**if**(((CourseWorkStudent)studentList.get(idx)).getOverallScore() >= avgOverall) {

equalOrAbove ++;

}**else** {

below ++;

}

//deal with research student

}**else** {

**if**(((ResearchStudent)studentList.get(idx)).getOverallScore() >= avgOverall) { //deal with reasearch student

equalOrAbove ++;

}**else** {

below ++;

}

}

}

//display output

System.***out***.println("");

System.***out***.println(equalOrAbove + " students obtained an overall mark equal to or above the average overall mark of " + avgOverall);

System.***out***.println(below + " students obtained an overall mark below the average overall mark of " + avgOverall);

}

//task 7 - Get studentID from user and display the student's details. The program is able to handle some error

// such as non existing student/object.

// Arguments required Student type ArrayList

/\*\*

\* This method is used to display the details of CourseworkStudent or ResearchStudent based on the studentID

\*\* <p>

\* Precondition - Pass in a Student type ArrayList.<br>

\* Postcondition - The program will prompt user to enter the studentID. <br>

\* Then the for loop will find for the same studentID object and display the details of <br>

\* the particular student by calling the writeOutput method.

\* </p>

\* **@param** studentList An ArrayList of type Student

\*/

**public** **static** **void** studentIDToGetDetails(ArrayList<Student> studentList) {

//declare boolean for display output purpose

**boolean** isFound = **false**;

**try** {

//ask user for studentID

System.***out***.print("Enter student number to retrieve the student details: ");

**long** stdToRetrieve = *input*.nextInt();

//loop through to find the student/object

**for**(**int** idx = 0; idx<studentList.size();idx++) {

//find same studentID

**if**(studentList.get(idx).getStudentID() == stdToRetrieve) {

System.***out***.println("\nStudent details : ");

System.***out***.println("------------------------");

//call writeOutput() method to display student's details

studentList.get(idx).writeOutput();

isFound = **true**;

**break**;

}

}

//if no same studentID object is found

**if**(!isFound) {

System.***out***.println("\nStudent not found");

}

//catch the InputMismatchException if the user enters a non-numerical studentID

}**catch**(InputMismatchException exc) { //

System.***out***.println("\nInvalid studentID");

}

}

//task 8 - Get name to display student's details. The program is able to handle some error

// such as non existing student/object.

// Arguments required Student type ArrayList

/\*\*

\* This method is used to display the details of CourseworkStudent or ResearchStudent based on student's first and last name

\*\* <p>

\* Precondition - Pass in a Student type ArrayList.<br>

\* Postcondition - The program will prompt user to enter the student's first name and last name. <br>

\* Then the for loop will find for the same student full name object and display the details of <br>

\* the particular student by calling the writeOutput method.

\* </p>

\* **@param** studentList An ArrayList of type Student

\*/

**public** **static** **void** studentNameToGetDetails(ArrayList<Student> studentList) {

//declare boolean for display output purpose

**boolean** isFound = **false**;

//prompt user to entered student's first name and last name

System.***out***.println("Enter student's name to retrieve the student details ");

System.***out***.print("Enter student's first name: ");

String firstName = *input*.next();

System.***out***.print("Enter student's last name: ");

String lastName = *input*.next();

//concatenate firstName and lastName

String fullName = firstName + " " + lastName;

//search for the object

**for**(**int** idx = 0; idx<studentList.size();idx++) {

//find and display the object with same student name, case not sensative

**if**(studentList.get(idx).getFullName().equalsIgnoreCase(fullName)) {

System.***out***.println("\nStudent details : ");

System.***out***.println("------------------------");

studentList.get(idx).writeOutput();

isFound = **true**;

}

}

//if no same name object found

**if**(!isFound) {

System.***out***.println("\nStudent not found");

}

}

//task 9 - Sort ArrayList in ascending order of the studentID by using the Bubble sort

// One of the reasons why I choose bubble sort method to sort the ArrayList is because

// bubble sort is easier to understand compare to insertion sort and selection sort.

// This is better for me and even other developer who looking at my code.

// In addition, the studentID given in the csv file is already almost sorted.

// Therefore, bubble sort is the most effective method to sort the ArrayList in ascending order.

/\*\*

\* This method is used to sort ArrayList in ascending order of the studentID by using the Bubble sort

\*\* <p>

\* Precondition - Pass in a Student type ArrayList.<br>

\* Postcondition - This method will sort the passed in ArrayList in ascending order of the studentID by using the Bubble sort. <br>

\* The for loop will swap the positon of the objects if the previous studentID is greater than the latter.

\* </p>

\* **@param** studentList An ArrayList of type Student

\*/

**public** **static** **void** sort(ArrayList<Student> studentList){

**boolean** swap = **true**;

**while** (swap) {

swap = **false**;

**for** (**int** idx = 1; idx < studentList.size(); idx++) {

//Check if the previous studentID is greater than the latter

**if** (studentList.get(idx-1).getStudentID() > (studentList.get(idx).getStudentID())){

// swap the student

Student temp1 = studentList.get(idx-1);

Student temp2 = studentList.get(idx);

studentList.set(idx, temp1);

studentList.set(idx-1, temp2);

swap = **true**;

}

}

}

}

//task 9 - main controller of task 9

// Display sorted ArrayList

// This method will first sort the ArrayList by calling the sort() method, then display the sorted ArrayList.

/\*\*

\* This method is used to display the sorted ArrayList

\*\* <p>

\* Precondition - Pass in a Student type ArrayList.<br>

\* Postcondition - This method will sort the passed in ArrayList in ascending order of the studentID by <br>

\* calling the sort() method. Then display the sorted ArrayList.

\* </p>

\* **@param** studentList An ArrayList of type Student

\*/

**public** **static** **void** displaySortedArrayList(ArrayList<Student> studentList) {

//sort ArrayList first

*sort*(studentList);

//display sorted ArrayList

System.***out***.println("\n----Sorted student list----");

*display*(studentList);

}

//task 10 - Output the sorted ArrayList into a .csv file

/\*\*

\* This method is used to output the sorted ArrayList into the sortedArrayListOutput.csv file

\*\* <p>

\* Precondition - Pass in a Student type sorted ArrayList.<br>

\* Postcondition - The sorted ArrayList will be wrote into the .csv file and can be seen in the .csv file.

\* </p>

\* **@param** sortedStudentList An sorted ArrayList of type Student

\*/

**public** **static** **void** storeSortedArrayList(ArrayList<Student> sortedStudentList) {

//declare outputStream instance of PrintWriter

PrintWriter outputStream = **null**;

**try** {

//open outputFile and connect to object outputStream

//create the outputStream object here

outputStream = **new** PrintWriter("sortedArrayListOutput.csv");

//for-each loop to retrieve the elements in ArrayList

//and print into the csv file

// for(Student content : sortedStudentList) {

// //Print the content to outputStream

// outputStream.println(content);

// }

//print the sorted ArrayList into the sortedArrayListOutput.csv file

**for**(**int** idx=0; idx<sortedStudentList.size(); idx++) {

outputStream.println(sortedStudentList.get(idx).writeOutputStr());

}

}**catch**(FileNotFoundException ex) {

System.***out***.println("File not exist");

System.*exit*(0);

}

//close outputStream

outputStream.close();

//display success msg

System.***out***.println("\nSuccessfully stored in csv file");

}

//display menu and prompt for user choice

//the menu will keep executing after each task is completed until user entered task 1 to exit the program

/\*\*

\* This method is to display the menu and ask for user's choice.

\*\* <p>

\* Precondition - Nil<br>

\* Postcondition - Prompt user to enter a choice and return user's choice.

\* </p>

\* **@return** choice Return user's choice

\* **@throws** Exception Choice must be numeric value and between 1-10

\*/

**public** **static** **int** menu() **throws** Exception{

System.***out***.println("\n---------< Menu >----------");

System.***out***.println("[1] - Exit program! ");

System.***out***.println("[2] - Read marks");

System.***out***.println("[3] - Remove student ");

System.***out***.println("[4] - Output of ArrayList");

System.***out***.println("[5] - Overall marks and grade");

System.***out***.println("[6] - Above / Below average overall mark");

System.***out***.println("[7] - StudentID to find student's details");

System.***out***.println("[8] - Student name to find student's details");

System.***out***.println("[9] - Sort array in ascending array");

System.***out***.println("[10] - Output sorted ArrayList to CSV file");

System.***out***.print("\nEnter your choice : ");

**int** choice = Integer.*parseInt*(*input*.next());

**return** choice;

}//end of method

//Main controller that call all 10 different tasks based on user's choice

//the menu will keep executing after each task is completed until user entered task 1 to exit the program

//This method is able to handle some error input such as non-numerical value and invalid input.

//boolean valid and validOrder is to controll the order of execution

//the user need to execute task 2 to read student's mark first so that he/she

//can perform task 3, 5 and 6. Similar to task 9 and 10.

/\*\*

\* This method is the main controller which call and execute task according to user's choice

\*\* <p>

\* Precondition - Pass in a Student type ArrayList and the option of which type of student with int value.<br>

\* Postcondition - Perform the task selected by user.

\* </p>

\* **@param** studentList An ArrayList of type Student

\* **@param** num Option of which type of student

\*/

**public** **static** **void** process(ArrayList<Student> studentList, **int** num){

//declare variables

**int** option = 0;

**boolean** valid = **false**; //use for the order of execution

**boolean** validOrder = **false**; //use for the order of execution

**do**

{

**try**{

option = *menu*();

**switch**(option)

{

**case** 1 : System.***out***.println("\nThank you Bye!!");

**break**;

**case** 2 : *readMarks*(studentList,num); //once task 2 is executed, boolean validOverall will become true

validOrder = **true**;

**break**;

**case** 3 : **if**(validOrder) {

*removeStd*(studentList); //task 3 can only be executed when validOrder is true

}**else** {

System.***out***.println("\nPlease execute task 2 to read the marks first");

}

**break**;

**case** 4 : *display*(studentList);

**break**;

**case** 5 : **if**(validOrder) {

*outputOverallAndGrade*(studentList, num); //task 5 & 6 can only be executed when validOrder is true

}**else** { //else return the error message

System.***out***.println("\nSelect task 2 to read the marks first");

}

**break**;

**case** 6 : **if**(validOrder) {

*aboveOrBelowOverall*(studentList, num);

}**else** {

System.***out***.println("\nMarks not read yet, select task 2 to read the marks first");

}

**break**;

**case** 7 : *studentIDToGetDetails*(studentList);

**break**;

**case** 8 : *studentNameToGetDetails*(studentList);

**break**;

**case** 9 : *displaySortedArrayList*(studentList); //once task 9 is executed, boolean valid will become true

valid = **true**;

**break**;

**case** 10 : **if**(valid) { //task 10 can only be executed when valid is true

*storeSortedArrayList*(studentList); //means task 9 has been executed

}**else** { //else return the error message

System.***out***.println("\nArrayList not sorted yet. Select task 9 to sort first");

}

**break**;

**default** : System.***out***.println("\nInvalid option!");

}

}**catch** (Exception e) //if user enter non-numerical value

{

System.***out***.println("\nUse a numerical value !");

}

}**while** (option != 1);

}

//student information

/\*\*

\* This method is used display the student information

\*\* <p>

\* Precondition - Nil.<br>

\* Postcondition - Display student information.

\* </p>

\*/

**public** **static** **void** studentInfo() {

System.***out***.println("Name : Tee Yee kang" );

System.***out***.println("StudentNo : 34315323" );

System.***out***.println("Mode Of Enrolment : Full Time" );

System.***out***.println("Tutor : Aaron Yeo" );

System.***out***.println("Tutorial Attendence Day : Friday" );

System.***out***.println("Tutorial Attendence Time : 1030-1230" );

System.***out***.println("----------------------------------------------\n" );

}//end of method

}//end of class