MURDOCH UNIVERSITY

ICT167 Principles of Computer Science

September Trimester 2021

Assignment 2 (worth 25% for the unit)

Due Date: Midnight, Friday on 24 November 2021

All Students: Submit the Assignment via LMS by the due date. **Make sure you click on Submit when you have uploaded your file to complete the submission.**

Late penalty: 10% per day penalty for delayed submissions unless prior extension of deadline is obtained from the unit coordinator.

You should keep a copy of your work. Your submission must include a completed assignment cover sheet. An electronic copy of the assignment cover sheet is available at the unit LMS site.

***Note:* You can only use ArrayList for this assignment.** You should start reading on how to use it early instead of waiting for lecture materials for the week.

You may be asked to demo the program to your lecturer and answer some questions. Make sure you understand everything you are submitting. If you are requested to demo, you will only receive the mark after you have attended the demo session.

Your examination may base on your assignment by asking you to modify the functionality of your program (details later in the term). You have to know your code well to be able to answer such question.

If you are unsure of anything about the assignment question, you need to get clarification early. Read this document very carefully and start to discuss with your lecturer.

Question

First, you need to design, code in Java, test and document a **base** class, Student. The Student class will have the following information, and all of these should be defined as Private:

A. A first name (given name)

B. A last name (family name/surname)

C. Student number (ID) – an integer number (of type **long**)

D. A date of birth (in day/month/year format – three integers) - **(Create your own Date class, the Date class should be a separate class from this. Do NOT use the Date class from JAVA.)**

The Student class will have **at least** the following constructors and methods:

(i) two constructors - one without any parameters (the **default constructor**), and one with parameters to give initial values to all the instance variables of Student.

(ii) only necessary set and get methods for a valid class design.

(iii) a reportGrade method, which you have nothing to report here, you can just print to the screen a message "There is no grade here.". This method will be overridden in the respective child classes.

(iv) an equals method which compares two student objects and returns true if they have the same student number (ID), otherwise it returns false.

You may add other methods in the Student class as you see appropriate, however you will need to justify why those methods are required.

Design, code in Java, test and document (at least) the following classes –

- a UndergraduateStudent class, a GraduateStudent class (both derived from the Student class)
- a UndergraduateUnit class, GraduateUnit class (both derived from the Unit class specified below)
- and a Client class.

Assuming in this program, you allow multiple student objects to be created (i.e. arraylist of student objects).

For undergraduate students (UndergraduateStudent class):

(a) Contain information of the UndergraduateUnit object (assuming that there is only one unit)

(b) Provide a reportGrade method such that it will output "U" (to identify as undergraduate student), the Name (first name and last name), Student number, the unit ID, the overall mark, and the final grade of the student.

For graduate students (GraduateStudent class):

(a) Contain information of the GraduateUnit object (assuming that there is only one unit)

(b) Provide a reportGrade method such that it will output "G" (to identify as graduate student), the Name (first name and last name), Student number, enrolment type and the final grade of the student.

Unit class:

(a) Enrolment type: "U" for undergraduate and "G" for graduate

(b) A final grade reporting method for reporting the "NA" for not available.

Undergraduate unit (UndergraduateUnit class):

(a) Will have the information of the unit ID (of type string; e.g. ICT167), and the level of the unit (of type integer; e.g. 1 for first year)

(b) There are two assignments, each marked out of a maximum of 100 marks and equally weighted. The marks for each assignment are recorded separately.

(c) There is weekly practical work. The marks for this component are recorded as a total mark obtained (marked out of a maximum of 20 marks) for all practical work demonstrated during the semester.

(d) There is one final examination that is marked out of a maximum of 100 marks and recorded separately.

(e) An overall mark (to be calculated within the class)

(f) A final grade, which is a string (avoid code duplication for calculating this)

The final grade, for undergraduate students, is to be awarded on the basis of an overall mark, which is a number in the range 0 to 100 and is obtained by calculating the weighted average of the student's performance in the assessment components. The criteria for calculating the weighted average is as defined below:

> The two assignments together count for a total of 50% (25% each) of the final grade, the practical work is worth 20%, and the final exam is worth 30% of the final grade.

A grade is to be awarded for undergraduate students as follows: An overall mark of 80 or higher is an HD, an overall mark of 70 or higher (but less than 80) is a D, an overall mark of 60 or higher (but less than 70) is a C, an overall mark of 50 or higher (but less than 60) is a P, and an overall mark below 50 is an N.

Graduate unit (GraduateUnit class):

(a) Enrolment type of the graduate student; whether the student is a Master or PhD student (of type string; e.g. Master or PhD)

(b) The number of years used to complete the thesis (of type integer, e.g. 3 for taking three years to complete).

(c) A final grade, which is a character (C for completed; N for not completed; S for suspended)

Client class:

The client program will allow entry of these data for several different student into an **ArrayList** and then perform some analysis and queries.

Your client class (program) will provide the user with a menu to perform the following operations. You will also need to load the information of the students from **a CSV file (student.csv)** before displaying the menu. You only need one ArrayList and one menu for this. For the csv file, your first item can be U or G to differentiate whether the entry is creating a UndergraduateStudent object, or a GraduateStudent object. You can then decide how you want other data to be listed in the csv file.

You should specify clearly in your documentation the data format of the CSV files used in this assignment.

1. Quit (exit the program)
2. Add (to the ArrayList) all the marks information about an undergraduate or graduate student by reading it from **another CSV file**. Your program will ask for the file name.
3. Given student number (ID), remove the specified student and relevant information from the ArrayList. It is always good to ask the user to confirm again before removing the record. For confirmation, output the student number (ID) and the name to the user.
4. Output all details currently held in the ArrayList.
5. Determine and display how many undergraduate 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. Note: You can only perform this task on undergraduate students only.
6. Given a student number (ID), report the grade information (using reportGrade) of the student with that number. If the student is not found in the ArrayList, an appropriate error message is to be displayed
7. Sort the ArrayList of the student objects into ascending order of the students' numbers (IDs), and output the sorted array - implement an appropriate sorting algorithm for this, and explain why such algorithm is selected (in internal and external documentation).
8. Output the sorted ArrayList from (7) to **a CSV file**. If the ArrayList is not sorted, this option cannot be selected. Remember to include the identification of whether the student is undergraduate (U) or graduate (G) student.

Note that the program will loop around until the user selects the first option (Quit).

Set up a student ArrayList of N student objects, and test it with N = 10 (at least). You have to store your test data in a file so that your program can read them. You should use 5 undergraduate and 5 graduate students in your test. Consider all possible enrolment types in your test.

The client class should be well-structured and should have the essential methods in addition to the main method.

The interaction with the user can be via the command line (i.e., no graphical user interface is expected).

Devise suitable test data to test all sections of program code. You will need to provide all the test data used.

Your client program should also include a method (e.g., StudentInfo( )) to output your student details (name, student number, mode of enrolment, tutor name, tutorial attendance day and time) at the start of program results.

Note: The question requires you to use an ArrayList. Also, the sorting algorithm used must be coded within your program and not called from any Java libraries. You should not use any Java libraries for sorting algorithm, date and output to CSV file (e.g. FileWriter). You are required to use only materials covered in the lecture notes and the textbook to complete this assignment.

Distribution of marks for assessment

An approximate distribution of marks for assessment is given below. The question will be marked out of 100 as follows:

Correct solution design and implementation: 75 marks
(which includes the design and implementation of classes as specified in the question above; programming style, use of comments, use of methods, parameters, input validation, readability, presentation of output etc.)

External Documentation (problem specification, pseudocode, program limitations, program testing and test results, program listings, etc.): 25 marks

Required internal documentation (i.e. in the source code):

- a beginning comment clearly stating title, author, date, file name, purpose and any assumptions or conditions on the form of input and expected output;
- other comments giving useful low-level documentation and describing each component;
- well-formatted readable code with meaningful identifier names and blank lines between components (like methods and classes).
- See https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html#examples of using the Javadoc comments.
- Other comments giving useful low-level documentation and describing each component on how you use the following in your code:
    - Inheritance
    - Polymorphism
    - Dynamic binding
    - Sorting algorithm
    - Handling CSV files

Required External Documentation:

1. **Title:** a paragraph clearly stating title, author, date, file names, and one-line statement of purpose.
2. **Requirements/Specification**: a paragraph giving a brief account of what the program is supposed to do. State any assumptions or conditions on the form of input and expected output.
3. **User Guide**: include clear instructions on how to access, compile and run the program.
4. **Structure/Design/Algorithm**: Outline the design of your program. Give a written description, provide the diagrams (**you have to provide the Structure Diagram, the UML Inheritance Class Diagram**) and use pseudocode for algorithms.
5. **Limitations:** Describe program shortfalls (if any), eg, the features asked for but not implemented, the situations it cannot handle etc.
6. **Testing**: Provide a thorough test plan (the more systematic, the better) and any errors noticed in your tests. You will need to provide reasons of the test plan and strategy you have adopted. Document exactly what are working and what are not working with explanations.

   Provide the set of test data used (in tabular form) with expected results. Each test data must be justified – reason for selecting that data. No marks will be awarded unless justification for each test data is provided.

Note that a copy of the sample test data and results from a program run is required (copy from the program output window and paste to a Word file). Your submitted test results should demonstrate a thorough testing of the program.

Use the following example tables.

Test Table

A set of test data in tabular form with expected results and desk check results from your algorithm. Each test data must be justified – reason for selecting that data. No mark will be awarded unless justification for each test data is provided.

Add rows to the following table as needed. Table can span more than one page. Each test id tests only one condition for the desk check.

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					
2					
3					
4					

Results of Program Testing

Results of applying your test data to your final program (tabular form), including a sample printout of your program in operation.

Add rows to the following table as needed. Table can span more than one page.

Each test id tests only one situation for the test run of the program. Table is copy/paste of the desk check with actual output column showing results of the program output. There should be no duplicated reasons listed in the second column.

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					
2					
3					
4					

After the above test table, copy/paste sample printouts of your program in operation. You can screen capture and paste here. Make sure you label each printout with the correct *Test id*.

7. **Source program listings**: save all your Java source code in a document in MS Word format.

All of the above external documentation must be included in that order in a file saved in MS Word format.

It is also necessary to submit all Java source code of your programs (i.e., all classes that you have designed and implemented). You should develop the programs using the NetBeans IDE. It will make it easy to collect sample output. The whole NetBeans project should be submitted.

The external documentation together with the NetBeans project folder containing all classes for the program must be compressed in a .zip file before submitting. Make sure that all necessary files are submitted so that the programs can be viewed, compiled and run successfully.

The final version of the program should compile and run under Java SE 8 and NetBeans IDE 8.0 (or later version). You will need to specify exactly which version you use in the documentation.