# MURDOCH UNIVERSITY

# ICT167 Principles of Computer Science

# Semester 2, 2020

## Assignment 2 (worth 25% for the unit)

**Due Date: Midnight, Friday on Teaching Week 12 (30 October 2020)**

**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 may be asked by your tutor to explain your submission. Make sure you understand everything you are submitting.

**Read and understand the information at: http://our.murdoch.edu.au/Educational-technologies/What-you-need-to-know/ - pay extra attention on Academic Misconduct section**

## 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. Title of the student (eg Mr, Miss, Ms, Mrs etc)

B. A first name (given name)

C. A last name (family name/surname)

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

E. A date of birth (in day/month/year format – three ints) - **(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.

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

(iii) method to write the output of all the instance variables in the class to the screen (which will be overridden in the respective child classes, as you have more instance variables that required to be output).

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

**You may add other methods in the Student class as you see appropriate.**

Design, code in Java, test and document (at least) three classes – a CourseWorkStudent class, a ResearchStudent class (which both derive from the Student class) and a Client program.

For coursework students:

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

(b) 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.

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

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

(e) A final grade, which is a string (think of a way to avoid code duplication for calculating this)

The final grade, for coursework 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.

For research students:

(a) There is a proposal component (marked out of a maximum of 100 marks).

(b) One final oral presentation (marked out of a maximum of 20 marks).

(c) One final thesis (marked out of a maximum of 100 marks).

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

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

The final grade, for research 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 proposal component is worth 30% of the final grade, the final oral presentations are worth a total of 10%, and the final thesis is worth 60% of the final grade

A grade is to be awarded for coursework and research 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.

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 need to think of a way to ask the user whether they are dealing with the coursework student or research student. You will also need to load the information of the students from **a text file (student.txt)** before displaying the menu.

1. Quit (exit the program)
2. Add (to the ArrayList) all the marks information about a coursework or research student by reading it from **another text file** and determine the student's overall mark and grade.

   You will need to consider how to deal with the different assessment components for the coursework and research students. You may use two different text files, each for coursework students and another for research students. The program should read the correct text file for this purpose.

   Use Exception Handling if the student cannot be found in the ArrayList.
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 from the ArrayList the details (all information including the overall mark and the grade) of all students currently held in the ArrayList.
5. Compute and output the overall mark and grade for coursework or research students
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
7. Given a coursework or research student number (ID), view all details of the student with that number. If the student is not found in the ArrayList, an appropriate error message is to be displayed
8. Given a coursework or research student's name (both surname and given name – ignoring case), view all details of that student. If the student is not found in the ArrayList, an appropriate error message is to be displayed. If more than one student having the same name, display all of them.
9. 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).
10. Output the sorted ArrayList from (9) to **a CSV file**.

**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.**

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 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

**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, use diagrams (eg, **UML 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.