CSY2030: Systems Design & Development			
Date of Issue	**15/12/2022**	**Last Date for Submission:**	**29/01/2023 By 23:59:59**
Part 1 Weight 20%		Module Tutor:	Dr John Kanyaru

Assesses knowledge in: OOD, OOP, selection, repetition and boolean expressions; numeric calculations, basic array handling including finding counts, totals and maximum values. File handling. Unit Testing. Class diagram.

Introduction

This is an individual piece of work.

The work is given in 3 stages, so that you can start as soon as possible. You only hand in your final version for marking.
Working in stages, you can develop your program using **incremental development** (bit by bit, not all at once). Develop your program a little bit at a time and keep testing it. Save a working version before making the next set of alterations. If you run out of time, submit a working program which provides some of the functionality. You will get very few marks for handing in lots of code that does not run correctly.

All code should be well-designed, well-laid out and commented.

Read this document several times before starting work, to ensure that you understand what is involved.

Conditions for this coursework

All students MUST complete coursework to a satisfactory standard. Your unit tests coverage should be >95%. This part of the course work contributes 20% of the overall module grade. Part 2 of the coursework contributes 30%.

The second assignment builds on the first one, so as a minimum, you will need a list of competitors containing the essential information.

Overview

Various people take part in a competition. You can choose the type of competition – it could be anything, from ice skating to some electronic game. Each competitor gets a fixed number of scores, and a calculation is done with these scores to determine their overall score.

In this assignment, you will create an application to hold details of the competitors, and produce a report including details of all competitors, the winner, and some summary statistics.

Example of input file of competitors (without own choice of extra attribute):

```
100, Keith John Talbot, Novice, 5,4,5,4,3
101, Jane Macdonald, Novice, 3,3,3,2,4
. . . .
106, Karen Harding, Expert,  5,5,5,4,4
107, Sarah Green, Expert, 4,4,5,4,3
```

Example report:

```
Competitor              Level       Scores           Overall
100  Keith John Talbot   Novice      5 4 5 4 3        4.2
101  Jane Macdonald      Novice      3 3 3 2 4        3.0
. . . .
106  Karen Harding       Expert      5 5 5 4 4        4.6
107  Sarah Green         Expert      4 4 5 4 3        4.0


Full details for 100:
Competitor number 100, name Keith John Talbot.
Keith is a Novice and received these scores : 5,4,5,4,3
This gives him an overall score of 4.2.

Short details for 106:
CN 106 (KH) has overall score 4.6.

STATISTICAL (you have some choice as to what to include here)
There are 9 competitors
The person with the highest score is Karen Harding with a score of 4.6.
The following individual scores were awarded:
Score :      1    2    3    4    5
Frequency:    2    7    10   12   8
```

Stage One – Class design and setup

Don't use input or output files or any sort of graphical user interface in this stage. Just develop your competitor class, without the list of scores. Using a main method, create several objects of the class, and test all your methods, which you will use in the next stage.

Details for each competitor should be held in a class named either `XXXCompetitor` (where XXX is your initials) or some more specific name e.g. `IceSkater`. Develop this class, according to the requirements that are listed below. Some of these requirements are described fully, and in other cases you have to make some of your own decisions. You can base your competition on a known sport or game, or just invent values.

When you have to make your own decisions, don't make the same decisions as your friends. You will lose marks if you hand in work in which all design decisions and code structures are identical to those of other students.

Your competitor class - Instance variables

For each competitor, you should hold the following essential information in instance variables:

- Competitor number – **must be an integer**
- Competitor name – use the Name class
- (Don't have any scores at all at the moment)
- Level of competitor – between 2 and 4 values which describe the standard of the competitor. Invent your own levels. E.g. Novice, Standard, Veteran or 1, 2, 3 ,4. These are fixed values for each person, they should not be derived from their overall score.
- An extra attribute of your choice (e.g. age, country…..)

Your competitor class - Methods

The Competitor class should have (use **exact** method names please) :

- A Constructor and get and set methods.
- A method which will be used later, to get the overall score. You will change it in the next stage, but for now it should look exactly like this:

```
public double getOverallScore() { return 5; }
```

- a `getFullDetails()` method which returns a String containing all the details of the competitor (including your extra attribute) as text. It should be clear what all the values are. It doesn't have to look exactly like the example below.

```
Competitor number 100, name Keith John Talbot, country UK.
 Keith is a Novice aged 21 and has an overall score of 5.
```

- a `getShortDetails()` method which returns a String containing a ONLY competitor number, initials and overall score. The format should look **exactly** like the line below, (with different data). 'CN' stands for 'competitor number'.
  e.g. `CN 100 (KJT) has overall score 5.`

Stage 2– adding simple collections using arrays

Alter your competitor class to include the list of individual scores, as follows:

- Add an instance variable - an array of integer scores with values between 0 and 5. Choose how many scores there will be (a number between 4 and 6).
- Provide a method `getScoreArray()` to return the array of integer scores.
- Alter your `getOverallScore()` method to calculate the overall score from the array of integer scores. You must make your own decision on how to calculate the overall score, which should be a decimal number in the range 0 – 5. It should ideally be a little more complicated than a simple average of the values, and could involve a decision based on the level or other attribute. Some ideas are:
  - Average of top `n` scores where `n` is the level number
  - Weighted average of all scores (choose different weights for different scores and levels)
  - Average of scores disregarding the highest and lowest score.
- Edit the `getFullDetails` method to include all the scores

  e.g.
  ```
  Competitor number 100, name Keith John Talbot, country UK.
  Keith is a Novice aged 21 and received these scores : 5,4,5,4,3
  This gives him an overall score of 4.2.
  ```

Using a main method, create several objects of the class, and test all your methods.

Stage 3 - enhancing data handling with ArrayLists, persistence with file I/O

Introduce a `Manager` class and a `CompetitorList` class to contain an `ArrayList` of your 'competitor' objects containing between 10 and 15 competitors, with data carefully chosen to test all your methods. Your program should do the following:

- Read in the details of each competitor from a text file. Initially don't do any validation on the text file, just use valid data.
- Produce a final report to a text file, although you may want to start by sending this to `System.out`. The final report should be similar to that on page 2. It should contain:
  - A table of competitors with full details (no special ordering is required in this assignment)
  - Details of the competitor with the highest overall score
  - Four other summary statistics of your choice. E.g.
    - Totals, averages, max, min (fairly straightforward)
    - Frequency report, for example showing how many times each individual score was awarded (more challenging).

  The method which produces this report should not do all this work itself but should call methods in the `CompetitorList` class (e.g. to get details for the table, to return a frequency report, to find the average overall mark etc).
- Allow the user to enter a competitor number, and check that this is valid. If it is valid, the short details about the competitor should be displayed.
- Finally, write some code to check a few errors in the input file

Your Submission Report

You should provide a report on your program, consisting of the following information (nothing else):

1) Your decisions
    a. Which extra competitor attributes you chose (level, number of scores etc)
    b. How the overall score is calculated.

2) Status report. Does your program meet the specification fully, or is it incomplete? This can be very brief (don't provide a list of everything that you were asked to do – the marker knows this). E.g.
    - "This application meets the specification fully"
    - "The application does everything except allowing the user to enter a competitor number"
    - "This application meets the specification almost completely but I know the overall score calculation isn't always correct"

3) Diagrams
    - Class diagram showing all classes with instance variables, methods and associations

4) Screenshot showing code coverage of unit tests.

You should use any tool e.g., StarUML or Visio to draw the diagrams. To submit them, convert them to pdf or png.

How to submit

Any diagrams should be converted to pdf or png formats. Do not use proprietary tools such as Database Management Systems.

Submission procedure

- E-Submission of documents through Turnitin on NILE as TWO separate WORD documents. [Document 1 = **Report** & Document 2 = **FullSourceCodeListing including unit tests**]
  To do this, go to the NILE site for this module and use the link labelled 'Submit your work'.
- E-Submission of a single ZIP archive that contains all the source code files (.java), unit tests, data files and png/pdf. The archive must be named with your student ID, e.g., *12345678.zip* where 12345678 is your student ID. To do this, go to the NILE site for this module and use the link labelled 'Submit your work'. Clicking on the link (*SourceCodeEsubmission*), will take you into the submission form, where you can upload your ZIP archive using the 'Attach File' button (*Browse for Local File*). Finally, click the *Submit* button.
- Failure to follow the above submission guidelines may result in a capped or fail grade.

Marking scheme

	Stage 1	Stage 2	Stage 3
70 - 100	- Class setup is done according to specification provided. - appropriate names used for classname, fields and methods	- arrays used well, and exceptions handled well. - comprehensive summary statistics well done	- code is adequately refactored to use array lists - storage is done into csv/text files with exceptions handled adequately - Reading and writing files is excellent - Class diagram is comprehensive - Unit tests and coverage >95%
60 - 69	requirements for 50-59 met	Overall, well done; requirements for 50-59 met.	- Well done but code comments could be added in some places. - use of extensible collections is good; - Comprehensive class diagram - data persistence is fine - some further exception handling could be done - Unit tests done - >95%
50 - 59	requirements for 40-49 met	requirements for 40-49 met	exceptions not caught for files that have inadequate data. code style and comments could be improved class diagram relationships done but some information missing some unit tests done though coverage is less than 90%
40 - 49	Class setup is fine	arrays used, though there is potential for problems with indexing; summary statistics are incomplete	- some refactoring for data persistence, but reading and writing does raise exceptions that should be caught; array lists used

			- class diagram missing some methods, attributes or relationships and multiplicity - unit tests done but minimal
30 - 39	Class setup inadequate, and methods return types and parameter lists, including naming inadequate	initialisation of array lists but no realistic use summary statistics not done	Code does not run
0 - 29	Hardly any work done or the work does not meet the specification provided. No appreciation of class design evident in the submission	hardly any work done, use of arrays inadequate and no evidence of summary counts	compilation errors and code does not run