F21SF / F20SF Assignment 1
weight 40% individual assignment
Deadline: October 24, 2020 (end week 6)

Gives practice in: Simple OOP classes; selection, repetition and Boolean expressions; numeric calculations, basic array handling including finding counts, totals and maximum values. File handling. Testing. Class diagram, sequence diagram, activity diagram.

Introduction

You will work individually for **Assignment 1**.
F21SF students must also do a **peer review** of Assignment 1.

The work is given in three 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 (e.g. using the MACS GitLab server).
- 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.

Remember: code is for humans, not machines. Therefore, all code should be well-designed, well-laid out, and well-commented. Code that is poorly-designed, hard to read, or scrappily commented may lose marks.

Some words on basic data security and hygiene:

- DON'T keep all your work e.g. on your laptop without backup. If laptop hard drive fails you'll be left with nothing.
  Oh, so you're backed up? Are you sure?? Pretend your main and secondary machines are toast (asteroid strike): now pretend to recover, and time yourself.
- DO keep privacy settings set to "private" in the cloud or MACS server. Exposing your work on an unsecured cloud drive is the same as plagiarism.

Finally: always read the question carefully. Read this document several times before starting work, to ensure that you understand what is involved.

Conditions for this coursework

This year the course is 100% coursework. This Assignment is worth 40% of your final grade.

The course applies the university's submission of coursework policy:

- No individual extension for coursework submissions.
- Deduction of 30% from the mark awarded for up to 5 days late submission.
- Submission more than 5 days late will not get a mark.
- If you have mitigating circumstances for an extension, talk to your programme director and submit an MC form with supporting documentation to the School's Office.

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

Timing Guidelines

From week 3 (or earlier!) – Stage One – develop your competitor class
From week 4 (or earlier!) – Stage Two – alter your competitor class
From week 5 (or earlier!) – Stage Three – expand and test
From week 6 (or earlier!) – Final stage – finalise your program and individual report

Overview

Various people take part in a competition. You can choose the type of competition – it could be anything, from ice skating to an 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 – after lecture 'More basics' – develop your competitor class

Don't use input or output files or any 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 Stage 2 (below).

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, but document these decisions clearly.

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 whose design decisions and code structures are identical to those of another student.

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 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 Two– after lecture 'Basic arrays' – alter your competitor class

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 Three - after ArrayLists and file I/O have been covered

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 – more than the code already provided in lecture notes.

Your Submission Report

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

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

2 Status report. Does your program meets the specification fully, or is it incomplete? This can be 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 "

  For the following lists and tables, there are examples in the Testing lecture.
  - If necessary, a list of known bugs and a list of requirements that you have not implemented.
  - Include a table of limitations – these are things that you decided not to handle, either through lack of time or because you think that they are unlikely to occur.
  - Include a table of tests done.

3 Diagrams
  - Class diagram showing all classes with instance variables, methods and associations
  - A sequence diagram showing a task of your choice, but using at least 2 classes. E.g. reading the input file or creating a report.

4 Write `javadoc` style comments for the `CompetitorList` class, and generate the html. Include in your submission the html generated for the `CompetitorList` class.

You can use any tool to draw the diagrams; remember that there are good tools online, if you don't have something on your computer. Or, draw by hand if you like.

How to submit

Submit your pdf report electronically on Vision.

Additionally, submit your application as a single file through the link in Vision. Please either submit
- a zipped eclipse project or
- a zip file containing just the source and input files in a format which can be imported into eclipse easily (**not** rar or nested zips; do not include bytecode `.class` or `.jar` files).

The lecturers will load the code into eclipse to view the source, and they will run your application.

The marking scheme and marking sheet for this assignment are in the GitLab repo.