Part 2 - Implementation, GUI, using Inheritance and Design Patterns

Due 9 Jan by 10:00 **Points** 100 **Submitting** a file upload
Available until 15 Jan at 10:00

This assignment was locked 15 Jan at 10:00.

Weighting %:	50	**Submission deadline (for students):**	See Canvas
Authorship:	Individual	**Target date for returning marked coursework:**	February, 2024
Tutor setting the work:	Dr John Kanyaru	**Number of hours you are expected to work on this assignment:**	50

This Assignment assesses the following module Learning Outcomes:

1. Be able to apply common software architecture patterns to design systems and to describe a system architecture using common modelling notation.

2. Be able to implement a system using a three−tier architecture.

3. Be able to use the Model−View−Controller pattern to design a user interface.

Assignment Tasks:

The tasks outlined here follow those of Part 1 and do concern the same scenario.

Stage Three– Implementing your design

Now, turn your attention to implementing (using Java) your design so far. The Competitor class, which was suggested to you in stage 1, should have these methods:

- 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 g*etShortDetails()* 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*.
- You can enhance your data model with data about other classes you have identified

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.

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.

In your competitor class, 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.

The competitor class has been outlined in this manner to give you an indication of the level of detail you will need for other classes you will include in your design and implementation.

Stage Four - 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. You should carefully consider objects from other classes also, that may be needed in your application. Your program should do the following:

- Read in the details of each competitor from a text file (can be JSON, CSV). Initially don't do any validation on the text file, just use valid data. You can use this sample file or invent your own **RunCompetitor.csv (https://herts.instructure.com/courses/107703/files/7848366?wrap=1) (https://herts.instructure.com/courses/107703/files/7848366/download?download_frd=1)**
- Produce a final report to a text file, although you may want to start by sending this to System.out. The final report 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

Stage Five - Inheritance & Adding a GUI

Inheritance

Consider various competitor categories and their likely structure. Consider them and identify:

1) Which instance variables and methods are identical.

a) The level is considered identical in two classes if they are both Strings, or both integers, even if the data has different values (e.g. one person has String values Amateur and Professional and another person has String values Novice and Expert).

b) The competitor number should be a common attribute in the superclass. Everyone was asked to hold this data as an integer. Renumber some of the competitors in each person's original data if necessary, to get a set of unique numbers.

2) Which instance variables and methods have the same signature but a different body

a) E.g. the getOverallScore method will have identical signatures (method name, return type) in each person's original class, but probably different method bodies. This method will become an abstract method in the superclass, and the full details of the method will be provided in the subclass.

3) Which are only in one person's competitor class

Now create a general Competitor class which is a superclass containing common instance variables and methods (1 above), and abstract method signatures for methods with the same signature and different bodies (2 above).

The idea is to refactor your code to produce several competitor subclasses which extends the base Competitor class. You'll need to remove all the instance variables and methods which are fully declared in the superclass. It suffices to work with only 2 subclasses.

CompetitorList class

Use your CompetitorList class for this section. This class should now be adapted so that the ArrayList can contain all Competitor objects. You can still read in the data from separate text files, one for each subclass. For this assignment, you can assume that your text files contain correct data – you don't need to do any checking/validation.

Check that your methods still make sense (the getFullDetails probably won't look right, but leave that for now).

You need to refactor your class diagram to reflect what you have done in this stage.

The GUI

Create a GUI to enable a user to interact with the list of competitors. The GUI can be one window containing 3 separate panels, or 3 separate windows. You may also have to add methods to your other classes.

Your GUI should allow altering of the data that's held in the program - it's not necessary to update the text files. Each GUI section should provide useful functionality such as:

- View and alter the scores for one competitor and see immediately the overall score.
- View table of competitors sorted in different ways. Be careful with this one, if you change the original ArrayList order and leave it in a different order, it might affect other methods.
- View details for competitors, only for those with in a particular subclass or with certain attributes E.g. level, category, your chosen attribute…..
- View full or short details for a competitor, given a competitor number
- Edit competitor details
- Remove a competitor

Don't use a drag and drop IDE to help with the GUI. The drag and drop features usually introduce a lot of complex code, so although your GUI might look nice, it will be harder for you to alter and document.

First you could write the GUI layout only and just check that this looks good. Then combine the GUI with the list of competitors.

Your GUIs should be opened by the manager class.

There should be a Close button which writes the competitor report to a text file and then closes the program.

For your GUI, use the **MVC design pattern** to organise and structure your classes.

Submission Requirements:

Submit your source code as a project with only the source files (.java) and any data files needed to run the code. If you used packages, ensure to submit your project with the package structure you used to develop it. Your single submission zip file should have the name **your_student_number.surname_part2.zip**.

Please submit your final working version, and not intermediate iterations.

Marks awarded for:

- **Class setup as described in Stage three - 10**
  - this includes the provided Competitor class and others you provided in your class diagram produced in part 1
  - you should take care for Java classes to match your UML classes
- **Refactoring your class setup as indicated in Stage four - 15**
  - use of extensible collections and persistence i/o (files)
  - also handling of checked exceptions using suitable constructs
  - reading from a file, writing to a file, searching
- **The use of inheritance to obviate code duplication, and adding a GUI - 20**
  - inheritance used appropriately and does make sense
  - GUI handles expected input and out actions
- **Git log 5**
  - please submit your git log that should indicate that your work has been developed in iterations.
  - Git log should show your student names and email as held by the university

Type of Feedback to be given for this assignment:

Through grades to various assessed parts and comments from the marker

Additional information:

· Regulations governing assessment offences including Plagiarism and Collusion are available from https://www.herts.ac.uk/__data/assets/pdf_file/0007/237625/AS14-Apx3-Academic-Misconduct.pdf (https://www.herts.ac.uk/__data/assets/pdf_file/0007/237625/AS14-Apx3-Academic-Misconduct.pdf) (UPR AS14).

· Guidance on avoiding plagiarism can be found here: https://herts.instructure.com/courses/61421 (https://herts.instructure.com/courses/61421) (see the **Referencing** section)

· For **undergraduate modules**:

o a score of 40% or above represents a pass performance at honours level.

o late submission of any item of coursework for each day or part thereof (or for hard copy submission only, working day or part thereof) for up to five days after the published deadline, coursework relating to modules at Levels 0, 4, 5, 6 submitted late (including deferred coursework, but with the exception of referred coursework), will have the numeric grade reduced by 10 grade points until or unless the numeric grade reaches or is 40. Where the numeric grade awarded for the assessment is less than 40, no lateness penalty will be applied.

Graded Components

Criteria	Ratings		Pts
Classes match the scenario, the provided Competitor class and others are implemented as per the scenario. Attributes, methods are clear.	**10 Pts Full marks**	**0 Pts No marks**	10 pts
Collections, Files, Exceptions: There is use of extensible collections such as ArrayList, and data persistence using files; - there is some handling of checked exceptions;	**15 Pts Full marks**	**0 Pts No marks**	15 pts
Inheritance and GUI : - inheritance has been suitably used to avoid code duplication; - a functional GUI has been built and handles expected input and output actions;	**20 Pts Full marks**	**0 Pts No marks**	20 pts
Git log - a log of commits is provided indicating iterative development of the project; the commit log does show the user name and email of the student.	**5 Pts Full marks**	**0 Pts No marks**	5 pts
			Total points: 50