CSY2030: Systems Design & Development

Date of Issue	**15/12/2022**	**Last Date for Submission:**	**29/01/2023 By 23:59:59**
Part 2 Weight 30%		Module Tutor:	Dr John Kanyaru

Gives practice in: Inheritance; GUI; UML - Use cases, class diagram.

Introduction

This assignment builds on your application, produced in **Part 1**, which holds a list of competitors.

You should develop your program **incrementally**. 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.

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. **This Part 2 contributes 30% of the overall module grade.**

UML Use case Diagram and Activity Diagram

You should draw these diagrams using a UML tool such as StarUML, or Visio.

The details in the paragraph below is ONLY for the use case diagram – the application that you are asked to write is a little different, and should be simpler. For example, in your application, the competitor details and scores are loaded from a text file (to save time!) but in this description, they are entered in a GUI.

Draw a Use Case Diagram for the requirements in the following paragraphs. There are probably several equally correct interpretations of these requirements so, different students will have non-identical diagrams.

A system is required to keep details of a competition, which has various categories for competitors. Before the competition, competitors register by filling in their details using an online form.

On the day of the competition, competitors are awarded scores for their performance. These scores are typed into the system by a member of staff, who first searches for the competitor using their number.

When all the people in a given category have competed, a staff member requests details of the results.

After the competition is over, competitors and staff can search for a particular competitor using their number, and view their details, including the basic and overall scores. They can also print out various summary reports.

Here is a bit more detail about the registration process. Competitors enter their name, email, date of birth, category and level, and are supplied with a unique customer number. If any fields are omitted, an *error message is displayed, and the competitor is asked to resubmit the form. If a competitor already exists with the same email address and the same category, the registration is refused. If a competitor already exists with the same email address and for a different category, their registration is accepted and they are allocated a different competitor number for this category. If a competitor's age is incompatible with the level, the competitor is offered the opportunity to resubmit the form for a different level. If everything is ok, the competitor is allocated a customer number and the registration is accepted.*

Using Inheritance

N.B. You are NOT being asked to develop a complete competition system as described for the Use Case section. Just do what is specified below:

In this section, you should refactor your program in Assignment one. You will use inheritance to include several categories of competitor.

Competitor classes

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 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.

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.

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

Competitor Report

Your program should create a competitor report to a text file consisting of :

- A table of all competitors consisting of short details only. Order is not important.
- Details of the competitor with the highest overall score (taken from ALL the competitors).

Using a GUI

Create one 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 within 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

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.

Submission guidelines

- E-Submission of document through Turnitin on NILE as one WORD document.
  [Document 1 = FullSourceCodeListing]
  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.
- Video Demonstration:
  You must submit a video demo (URL) of your assignment. The demo should be no longer than 10 minutes. Your face and voice need to be clear for the marker to see/hear. This should include a walkthrough of the functionalities. The module tutor reserves the right to invite you for a viva-voce. Poor demo or viva could negatively influence all the marking criteria and may result in a fail grade. When submitting your video demonstration, use of Kaltura (https://video.northampton.ac.uk/) is recommended. You must ensure that the video link is accessible to the marker (do not set it to private access).
- Failure to follow the above submission guidelines may result in a capped or fail grade.

Marking scheme

	Application	Diagrams
70 - 100	- Part 1 application is adequately refactored to use inheritance - GUI is functional and responsive	- UML diagrams are comprehensive and clear
60 - 69	Inheritance applied well, but more refactoring could have been done. GUI is working	Diagrams are detailed, no missing details.
50 - 59	Inheritance is used but there is still code duplication; GUI is good but some events not handled	Diagram is clear but some detail is missing or unclear
40 - 49	Inadequate use of inheritance and GUI is partially working	Diagrams are presented with minimal detail and some detail seeming incorrect
30 - 39	Inadequate use of inheritance. GUI not responsive	Diagrams do not match the scenario and have minimal detail
0 - 29	Hardly any work done	No diagrams or they are inadequate