F21SF / F20SF Assignment 2
weight 60% individual assignment
Deadline: 7 December 2020 at 15:30 local time

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

1 Introduction

This assignment builds on your application, produced in Assignment 1, which holds a list of competitors.
You should work on this assignment individually; this is not a group assignment.

You should develop your program using **incremental development:**

- 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 use GitLab, make sure your project is set to "private", so that only you can view it.
- 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.

As always, your code must be well-designed, well-laid out and well-commented – and if it isn't you may lose marks.

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

2 Conditions for this coursework

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.

3 Timing Guidelines

From week 7 – Stage One	– draw UML diagrams
From week 8 – Stage Two	– use inheritance, list of competitors
From week 9 – Stage Three	– add GUI
From week 10 – Final stage	– add program reports. Write overall report.

4 Stage One - UML Use case Diagram, Text description and Activity Diagram

You can draw these diagrams using a UML tool, a drawing package, or by hand.

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 paragraph, and write a detailed description for the 'registration' use case – this is the numbered ideal scenario with alternatives. Also draw an activity diagram for this use case. There are probably several equally correct interpretations of these requirements so don't worry if you don't all have 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 competitor 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 competitor number and the registration is accepted.

5 Stage Two - Using Inheritance – after three inheritance lectures

Note: You are NOT required to develop a complete competition system as described in the Use Case section above. Just do what is specified below:

1 Competitor classes

Recall that you created a 'competitor' class in the Assignment 1.

Now create two additional 'competitor' classes (making three in total), making different design choices (you might like to consider different types of competitions, but this is not compulsory so long as there are significant differences in design). These classes should be different in some parts and be similar in others. Try to get some decent variation so that you can interestingly …

… look at your three classes 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.
    b) The competitor number should be a common attribute in the superclass. Renumber some of the competitors in each 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), but probably different method bodies in each one of the 'competitor' classes. This method will become an abstract method in the superclass, and the full details of the method will be provided in the subclass.
1) Which instance variables and methods are only in one specific 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).

Each 'competitor' class of the three ones that you have created now becomes a subclass which extends the `Competitor` super class. You'll need to remove all the instance variables and methods which are fully declared in the superclass.

2 CompetitorList class

Adapt your CompetitorList class 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).

3 Competitor Report

Your program should create a competitor report in 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).

6 Stage Three - Using a GUI - Do this after the lectures on GUIs.

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

Your GUI must alter the data that's held in the program - it's not necessary to update the text files. 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

Mostly you should use the components introduced in the lectures, but you might like to find out how to use one or two other components (e.g. radio buttons for the search). You won't get full marks for just copying lecture GUIs with minimal alterations.

Learn the basics of GUIs in java and design your GUI from first principles: 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. You can use one GUI if you like, but if you use a separate window for each GUI, make sure to adjust the location of each window so that when you open them together you can see all of them.

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

7 Report

Write a pdf report – try to be short but comprehensive: longer is not better, more *informative* is better, which is not quite the same thing – which should contain:

- A **technical section** which explains your program code for the features implemented in your GUI, aimed at a reader who has studied the material in this course but has not done the coursework.

  Start from a user action (e.g. click button or enter location id to request location details), and go through the method calls until the result is achieved (e.g. data displayed in text area). Use appropriate UML diagrams, code fragments and screenshots.

- **A status section:** Does your program meets 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 that sorting in descending order of overall score doesn't show all the competitors"

Include the following lists and tables; you are NOT required to submit a table of tests done for this assignment:

If necessary: a list of known bugs and a list of unimplemented (or partially implemented) requirements.

A table showing what user input errors your application can handle, and which it can't. In the table below, these examples show one user mistake that is handled and one that is not. They are just examples - ideally your program should not crash for non-numeric data.
e.g.

Competitor number entered in GUI which does not correspond to any in the list	Error message is displayed
Non-numeric data entered in GUI for the competitor number	Not handled, program will crash

- **Diagrams**
  - The use case diagram and textual description and activity diagram.
  - A class diagram (maybe showing overview and detail separately).

8 Submission

Please upload precisely one zip file containing:

- Your pdf report.
- An eclipse project, or just the source and input files in a format which can be imported into eclipse easily.

Do not use rar. **Do not** nest zip files. **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 are in the GitLab repo.