U08186 Coursework 2013

This coursework has three parts, worth respectively 10%, 10% and 30% of the overall module mark. Part 3 is for self-chosen pairs. Parts 1 and 2 must be your own work.

Submission Details

Part 1 Upload a `.pdf` or a `.doc` to Moodle by 5pm on Thursday of Week 4

Part 2 Upload a `.pdf` to a `.doc` to Moodle by 5pm on Thursday of Week 7.

Part 3 Upload two `.zip` files, one containing your Java code and another containing your C# code, to Moodle by 5pm on Thursday of Week 10. Only one member of the pair needs to do this but make sure it is clear in each `.zip` file which member of the pair has done which part.

Part 1

Essay on Barbara Liskov's Lecture

Write 1000 words describing how Barbara Liskov's work on abstraction, as related in her Karen Spärk Jones lecture and the following Q & A session, has influenced the Java programming language and the practices associated with object-orientated software development. Barbara Liskov's lecture can be viewed at `http://academy.bcs.org/ksj`. You will be marked according to how well you link the concepts introduced in the 1970s that she explains with the 2010s programming language Java and associated practices as related in Lectures 1 and 2. Extra research on all of these may help explain the ideas and organise your thoughts but the primary focus should be the linkage.

Part 2

Examples of Design Patterns

For this part, you need to find five examples of five different design patterns and find an example, either online or in a textbook. It must not be an example from the lecture notes or practical sheets. You then need to do two things, each for 1% of the module marks.

- check that the example conforms to the pattern, as demonstated in the slide "Checking that the AttackDemo Conforms to the Strategy pattern"
- extend the example, in keeping with the pattern, eg by adding another subclass

So you will need to submit a `.doc` or `.pdf` file containing, for each of the five patterns

- the class diagram before the extension
- working to show that it conforms to the pattern, which will also identify all the elements of the pattern within the example
- the code before the extension
- the class diagram after the extension
- the code after the extension

You can produce the class diagrams by any means you wish. You can even hand-draw them on paper, so long as you can scan the result into the document submitted. The tool I use for the lectures is umlet, free to download from `www.umlet.com`. Alternatives include Microsoft Visio, installed in the labs.

You do not need to get this code to compile and run (although you are welcome to do so) since this exercise is about learning the patterns rather than learning Java; minor syntactic errors will therefore be tolerated.

Part 3

GUI with MVC, DBC, Unit Testing

Finally, for this part, you have to write a program in Java using Model View Controller (MVC), supply a second view, and then rewrite and unit test the Model in C# and Spec#. Be warned that merely getting a Java GUI to work is worth only a small fraction of the available marks, as that would demonstrate your understanding of only a small proportion of the taught material on the module. This work is to be done in pairs. Note that there are some exact requirements in the marking scheme concerning what each member of the pair should do with his partner and what he or she should do on his own. As well as observing this, please ensure that each method is annotated with the name of the member of the pair chiefly responsible for writing it. You should split the joint work equally between you.

Problem Specification

You are required to write a results visualiser for an election in a UK parliamentary constituency. You should keep track of the following quantities:

- the total number of voters,
- the total number of votes cast,
- the total number of votes cast for each of five candidates

The names of the candidates and the total number of voters can be hardcoded. However, the votes cast for each candidate, and hence the total number of votes must **not** be hard-coded. You must have two views of the model, one textual and one graphical; it is probably simplest to use a Canvas for both. You should also have within the GUI, buttons to give a vote to and take a vote from each candidate. The textual view will show the values of the quantities above. The graphical view will show the relative proportions of the candidates eg with a bar chart or a pie chart.

Marking Scheme

- one view for each member of the pair (10% each), each completed separately, but working on the buttons together, following good code quality practices as described in Lecture 1. (5% ie half of the marks available will be a judgement of code quality)
- a model and a controller, marked according to adherence to the Model View Controller (MVC) design pattern (10%). Note in particular that the model should not change either view directly and should not contain any information related to how it should be displayed.
- the model reimplemented in C# (5%)
- unit testing for the model (5%) by one member of the pair and a Design By Contract specification of the model (5%) by the other member of the pair; the specification must include object invariants and pre/postconditions for the methods that change the model. Partial credit will be given for describing these in English. Some statements may be unprovable due to the limitations of Spec#, even if properly specified in a formal manner. In such cases, there will be no negative impact on your marks. The unit testing will be marked according to how many of the different test cases you have covered (without simply doing dozens of randomly-chosen tests). The specification will be marked according to how complete it is (all appropriate invariants and pre/postconditions expressed as precisely as possible and no incorrect ones).

If you are not in a pair, you can choose which view to do and whether to unit test or to produce the Design By Contract specification.