Java and Object Oriented Software Engineering
Semester 2 Coursework Laboratory 5
JAgora - An Electronic Stock Exchange

Friday 18th March, 2016

1 Coursework Overview and Working Arrangements

During the course of semester 2 you will work on the development of a single software project. Each laboratory will concentrate on a different aspect of the software development process:

Laboratory	Week	Beginning	Topic	Marks
1	19	25th Jan	Warm up	0
2	21	8th Feb	Problem domain analysis	25
3	23	22nd Feb	Unit and acceptance test harness	25
4	25	7th Mar	Design and implementation	25
5	27	21st Mar	Applying design patterns	25

During semester 2, we will be using *pair programming* to improve the quality of software produced during the laboratories. You should prepare for your scheduled laboratory as normal, by reading the laboratory sheet problem description and undertaking background research. At the start of the scheduled laboratory, your tutor will pair you with a different student to work with in each of the five JOOSE laboratories. You must work with your assigned partner throughout the scheduled laboratory. Your tutor will monitor how you work as a development team as part of your assessment. Following the laboratory each partner should take a copy of the solution developed thus far and make further independent improvements as desired before submission.
Note: As pair programming is part of the assessment for each laboratory, non-attendance without good cause will result in a loss of all 5 marks available in this category. You must note down the name and matriculation number of your pair programming partner on your submission.

2 Task

You have been provided with *three* related software projects in the laboratory handout.

- `jagora-api` contains a set of interfaces that define the application programming interface of the JAgora application. Each of the interfaces is accompanied by JavaDoc documentation describing the semantics of the operational signatures. You may need to modify some of these interface definitions, as well as create your own new interfaces.
- `jagora-test` contains a suite of JUnit test cases for the `jagora-impl` project. You should add supplementary test cases for the additional functionality you implement during this laboratory.
- `jagora-impl` contains a reference implementation of JAgora that realise the interfaces in the `jagora-api` project. You will need to modify the reference implementation to comply with modifications to the interfaces, as well as providing reference implementations for new interfaces.

Working with your lab partner, you are required to identify, tailor and implement a design pattern for each of the following design problems:

1. Some (but not all) types of `Trader` need to know about the execution of trades on a `StockExchange`. For example, a trader may want to calculate the historic average price for a stock trade so that they can place a buy (sell) order when the current best offer (bid) is below (above) the historic average. Other tools could also use this facility, to log trades, for example.

   Hint: what *behavioural* pattern separates the issuing of notifications of events in one class from the action to be taken in another class when the event actually occurs?

2. The implementation of `DefaultTrader` and `RandomTrader` have complex constructors for specifying a trader's initial configuration. In addition, it would be desirable to specify a wider range of inventory quantities for stocks: the current constructor implementation only allows a single stock to be specified.

   Hint: what *creational* design pattern improves the convenience of constructing objects with complex constructors?

3. The existing specification of the `Trader` interface provides implementations with unrestricted and unregulated access to a `StockExchange` when the `Trader.speak(StockExchange)` method is invoked. It is desirable that `Trader` implementations are only able to invoke methods for placing and cancelling orders and for determining the current best bid and offer. However, other classes, such as the `Engine` for driving the stock exchange should still be able to access the additional functionality (such as clearing) of the `StockExchange`.

   Hint: what *structural* design pattern limits access to a class for unprivileged clients?

Follow these steps to get started:

1. Start by creating *three* new projects in Eclipse for each of the three projects supplied in the laboratory archive. For example, for the project called `jagora-api` you should create a project called `jagora-api` with a root directory of `lab4-distrib/jagora-api`. You may find it convenient to change the Eclipse workspace to the `lab4-distrib` directory first.

2. The source code for the `jagora-api` and `jagora-impl` projects is contained in the directory `src/main/java`. Make sure that this directory is used as a source folder in both projects so that the code can be compiled in Eclipse. Eclipse may do this for you automatically, but if not you should: *Right click on the source directory* (e.g. `src/main/java`) the choose *Build Path* and *Use as Source Folder*.

3. The source code for tests in the `jagora-test` project is contained in a directory called `src/test/java`, so that we can separate test and application code. Make sure that this directory is also used as a source folder following a similar step to above.

4. You may also need to add the JUnit libraries in the `lib` directory to the `jagora-test` project build path. To do this, select the three libraries then *Right click*, *Build Path* and *Add to Build Path*.

5. You need to create the dependencies between projects. These are as follows:

   - `jagora-impl` depends on `jagora-api`
   - `jagora-test` depends on `jagora-api` and `jagora-impl`

To configure the dependencies for`jagora-impl` right click on the project, then *Build Path* and *Configure Build Path...*. Select the *Projects* tab and click on *Add*. Make sure that the `jagora-api` project is checked, then click *Ok* and *Ok*.

Repeat this process for the `jagora-test` project.

6. At this point your code should compile. Try running the test suite contained in the `jagora-test` project by right clicking on the project and choosing *Run As* then *JUnit Test*. Initially, all the tests should pass, because you haven't yet made any changes to the code.

Now proceed to implement your first design pattern:

1. Study the first problem description carefully. You may want to sketch out the changes that you propose to make as a UML class diagram to help you organise the solution. You may also want to discuss your proposed solution with your tutor before committing the solution to code.

2. Once you have selected a design pattern alter or create any necessary interfaces in the `jagora-api` project to apply the design pattern. This may well cause compilation problems in the `jagora-impl` and `jagora-test` projects.

3. Next, create or alter test cases in the `jagora-test` to accommodate changes to the `jagora-api` project and check for the intended behaviour of the proposed design pattern. You may also need to create more stubs for the test case.

4. Implement the changes needed to apply the design pattern in the `jagora-impl` project. Once you have successfully implemented the design pattern your test cases should pass.

5. Create a text file called `change1.txt` in the directory above the three project directories. Use this text file to create a short *source configuration management* system message explaining what changes you have made to the JAGora system. You do not need to list the specific files changed (this can be detected automatically). Instead, outline what change has been made, the problem it addresses and any trade-offs that have resulted in the system design.

Repeat these steps for design problems two and three.

3 Assessment

Submissions for this exercise are due by 10am **four** days after your lab during week 27 (week beginning 22nd Mar). You should submit your solution on Moodle in the appropriate upload slot for the laboratory. You must submit a zip compressed archive containing the three projects that you have changed. The root of this zip archive must contain the three project directories called `jagora-api`, `jagora-impl`, and `jagora-test`, with the source code and other files for the project below this node.

Call your submission file `<matriculation>.zip`, where `<matriculation>` should be replaced with your matriculation number, for example `2001234.zip`.

Your submission will be marked on a basis of 25 marks. Credit will be awarded for:

section	marks
Pair programming	5
Design problem 1	
API Change	1
Test cases	2
Implementation	2
Explanation	1
Design problem 2	
API Change	1
Test cases	2
Implementation	2
Explanation	1
Design problem 3	
API Change	1
Test cases	2
Implementation	3
Explanation	2

Note that attendance at the laboratory is mandatory: any student who does not attend the laboratory and work in a pair without good cause will suffer a 5 mark penalty.

As per the Code of Assessment policy regarding late submissions, submissions will be accepted for up to 5 working days beyond this due date. Any late submissions will be marked as if submitted on time, yielding a band value between 0 and 22; for each working day the submission is late, the band value will be reduced by 2. Submissions received more than 5 working days after the due date will receive an H (band value of 0).