CPT206 Computer Programming for Financial Mathematics: Coursework 3 Task Specification

Thomas Selig

Set: Monday, 6 May, 2024
Due date: Sunday, 26 May, 2024, 23:59

This is the specification task sheet for the Coursework 3 assessment component of your CPT206 module. The task covers all Learning Outcomes, and has a weighting of **70%** towards the final grade for this module. This assignment has two parts: a coding part described in Section 1, and a report described in Section 2. The submission deadline for this assignment is **Sunday, 26 May, 2024, at 23:59** (China-time). Detailed submission instructions are provided in Section 3.

1 Program description (70 marks)

The aim of this coursework is to build a system which manages investments in companies' shares. All the work should be coded into a single Java NetBeans project, with the class structure and different functionalities of the program described below. All classes should be properly encapsulated, as seen in the Lectures and Labs throughout the semester. Your project should also contain a Controller class for testing. You may leave some or all of your test code in the Controller class if you wish when submitting, but no marks are allocated for this class's contents in your submission. Instead you will be asked to describe your testing in the report (see Section 2.3), and marked on that.

1.1 `Market` class (7 marks)

The `Market` class represents the current companies that are on the market, that is whose shares can be traded (bought or sold). It therefore stores a collection of `Company` objects (see Section 1.2). Any company can only appear on the market at most once. The class should also have methods `add()` and `remove()` to add or remove a given company from the market, and `contains()` to check if a given company is on the market or not. There will only be one market in the program, so all members of the class should be *static*.

1.2 `Company` class (25 marks)

A `Company` object should store its (legal) name, a *stability* coefficient (a percentage between 0 and 100) indicating how stable its share prices are, and the *history* of its share prices. When a `Company` object is created, its initial share price is specified (as well as the name and stability coefficient). The history then consists of this initial share price. The `Company` class should have a method `getCurrentSharePrice()` which returns the current share price of the company (this will be the last share price in its history) and a method `getInitialSharePrice()` which returns the

initial share price of the company (this will be the first share price in its history). There should be methods to add or remove the company from the market, and to check if it is currently on the market. There should be a method `updateSharePrice()` to update the valuation of a company's share price, according to the following formula:

$$\text{Share}_{\text{new}} = \text{Share}_{\text{last}} \left(1 + (1 - \text{stab}) * U\right),$$

where:

- $\text{Share}_{\text{new}}$ is the new current share price of the property (after update);
- $\text{Share}_{\text{last}}$ is its old current share price (before update);
- stab is the company's stability coefficient;
- U is a uniform random variable on the interval $[-1, 1]$.

The new share price should then be added to the company's history (as its last element).

Finally, there should be a method `getTrend(int movementNumber, int significance)` to determine the `trend` of a company's share value based on a number of recent movements. Given two share prices in a company's history s_1 and s_2, the *movement* is simply the difference $s_2 - s_1$. A movement is `consecutive` if it is between two consecutive share prices in the company's history. A movement is considered *significant* if the ratio $\frac{|s_2 - s_1|}{s_1}$ is greater than or equal to the `significance` threshold. Otherwise it is *insignificant*. The trend of the company is then calculated by looking at the last `movementNumber` consecutive movements as follows:

- If strictly more than half of these are significant increases, and the share price has overall increased significantly over the whole period, then the trend is *increasing*.
- If strictly more than half of these are significant decreases, and the share price has overall decreased significantly over the whole period, then the trend is *decreasing*.
- If strictly more than half of these are insignificant, and the share price movement over the whole period is also insignificant, then the trend is *stable*.
- Otherwise, the trend is *uncertain*. This includes the case where there is insufficient data in the history.

For example, suppose `movementNumber = 3`, and the last four values in a company's share history are $[100, 105, 97, 103]$, so that the last three consecutive movements are $+5, -8, +6$. The overall movement over this period is $103 - 100 = 3$.

- If the significance threshold is equal to 3 (or less), then the trend is *increasing*: the consecutive movements $+5$ and $+6$ both meet the significance threshold (this is a strict majority of them), as does the overall movement.
- If the significance threshold is equal to 7 (or more), then the trend is *stable*: the consecutive movements $+5$ and $+6$ are both within the stability threshold, as is the overall movement.
- If the significance threshold is equal to 5, then the trend is *uncertain*: while there are two significant increases in the consecutive movements, the overall movement is too small to be significant.
- If the significance threshold is equal to 6, then the trend is also *uncertain*: the consecutive movement $+5$ is insignificant, -8 is a significant decrease, and $+6$ is a significant increase, so there is no majority of consecutive movement type.

1.3 `(Basic)Strategy` class (12 marks)

The investor will need to devise strategies determining when they should invest in a given company, and how many shares they should buy when doing so. The building block for this will be a `Strategy`

interface, with a method `invest(company)` that takes as parameter a `Company` object, and returns the number of shares to invest in based on the company's current performance in the market. The number returned is a signed integer, with a negative value indicating that the investor should sell shares in that company.

The strategy should then be implemented through a `BasicStrategy` class. This class should store a `maxTransaction` variable, indicating how much they are willing to invest or divest in shares in a given transaction. It should also store variables `movementNumber` and `significance`. The implementation of the `invest(company)` method should first calculate the company's trend based on these two variables, and then decide how much to invest/divest as follows.

- If the company's trend is stable or uncertain, there is no investment to be made (the method should return 0).
- If the company's trend is increasing, the investor should buy shares. The number they buy is equal to the maximal value such that they spend no more than `maxTransaction` on the transaction.
- If the company's trend is decreasing, the investor should sell shares. As above, the number they sell is equal to the maximal value such that they receive no more than `maxTransaction` from the transaction.

1.4 `Investor` class (16 marks)

Finally, your project should have an `Investor` class responsible for managing investment in the market (this could be an individual investor, or a company's own investments in the market, for example). Each investor should have a collection of companies (on the market) they are interested in investing in. The investor should then associate to each of these a `Strategy` dictating how they will invest, and the number of shares in the company they currently own. By default, `Investor` objects are created with no companies of interest. They can also be created with a specified collection of such companies together with their corresponding strategies, but no shares in any of the companies. The various functionalities of the class are as follows.

- Investors can select a new company they are interested in, along with a specified strategy for investing in that company (as above, initially they will have no shares in the company).
- Investors can decide they are no longer interested in a particular company. In that case, they sell all the shares they currently own in that company, and it is removed from their collection of interested companies.
- Otherwise, investment or divestment is entirely automatic, via an `updateInvestment()` method. This updates the share prices of all companies the investor is interested in. For each of these, the investor then runs the `invest()` method from the corresponding strategy, and chooses to buy or sell shares according to this method's output. This will potentially modify the number of shares the investor currently holds in that company. Note that if the output of `invest()` would cause the investor to sell more shares in a company than they currently own, they should simply sell all of the owned shares.

1.5 Code quality (10 marks)

The remaining marks **(10)** will be awarded for the quality of your code, as covered throughout the semester in the Lectures and Labs.

- Keep your code neat and tidy; make sure it is properly indented throughout.

- Choose suitable names for variables and methods, respecting standard Java naming conventions.
- Comment your code as needed.
- Split your code into separate methods as appropriate; methods should not be too long.

2 Report (30 marks)

For this part of the assignment, you will write a report detailing how you designed, implemented, and tested the program described in Section 1. The report should be typed into e.g. a Word document, and submitted as a PDF (see Section 3 for more details). Where suitable in the report, you should refer to specific lecture slides (or parts of Lab worksheets), e.g. "as seen in Lecture 10, slides 32-34".

2.1 Collection choices (5 marks)

In the project, there are a number of classes which will use elements from the Java collection framework. For example, the `Market` stores a collection of `Company` objects, while a `Company` stores its history of share prices. For each use of a Java collection in your program, you should state and justify here your choice of collection object. This section should be under one page in length.

2.2 OOP features (10 marks)

Over the course of the semester, you have learned a number of OOP features (e.g. encapsulation) and principles (e.g. single responsibility principle). In your report, you will explain where you have incorporated these in your design and how you have done so; include a brief definition of the features/principles in question. Be as precise as possible, illustrating with small portions of code if necessary. Note that not all the features and principles we saw in the lectures need to be incorporated into your design: your report should only discuss those that are. This section should be one-and-a-half to two pages in length.

Good example: The Single Responsibility Principle states that every class in the program should have responsibility over a single functionality of the program; a class should do one thing. This principle is incorporated into our class design: all the classes have their own, separate, purpose. For instance, the `Company` class[1]...

Bad example: Encapsulation and inheritance are two core features of OOP; they are used in many parts in my program.

2.3 Testing description (10 marks)

As covered throughout the Lectures and Lab sessions in this module, testing is an essential part of writing computer programs. In your report, you will include a description of how you tested the various parts of the program described in Section 1. You will state clearly what functionalities you tested, and describe how you tested them, thinking carefully about possible corner cases. You may include some sample code if you wish. You should test in the `Controller` class of your project, using only tools and techniques that we covered in the Lectures and Labs throughout the semester.

[1] Give a brief description of the purpose of the `Company` class here.

You must **NOT** use any new or more advanced tools such as `JUnit` that weren't taught. This section should be one-and-a-half to two pages in length (screenshots excluded).

2.4 Model discussion (5 marks)

In this part, you should critically discuss the chosen investment strategy in `BasicStrategy`. Is this a realistic model? What are some of its limitations? What would a more realistic strategy look like? You may include possible implementations of alternate strategies if you wish. This part should be no more than one page in length.

3 Submission instructions

In the dedicated "Coursework 3 submission" Assignment activity on the Learning Mall Online, you will need to submit the following **two (2)** documents.

- A single ZIP archive of your **entire NetBeans project**. Include all the resources your project needs to run. This file will be named "CPT206_CW3_Project_studentId.zip".
- Your report from Section 2, typed into e.g. a Word document, and **converted into a PDF file**. This file will be named "CPT206_CW3_Report_studentId.pdf".

The submission deadline is: **Sunday, 26 May, 2024, at 23:59** (China-time).

This assignment is individual work. Plagiarism (e.g. copying materials from other sources without proper acknowledgement) is a serious academic offence. Plagiarism and collusion will not be tolerated and will be dealt with in accordance with the University Code of Practice on Academic Integrity. Submitting work created by others, whether paid for or not, is a serious offence, and will be prosecuted vigorously. The use of generative AI for content generation is not permitted on this assignment. Such a use would be considered in breach of the University Code of Practice on Academic Integrity, and dealt with accordingly. Individual students may be invited to explain parts of their code in person during a dedicated interview session, and if they fail to demonstrate an understanding of the code, no credit will be given for that part of the code.

Late submissions. The standard University policy on late submissions will apply: 5% of the total marks available for the component shall be deducted from the assessment mark for each working day after the submission date, up to a maximum of five working days, so long as this does not reduce the mark below the pass mark (40%); submissions more than five working days late will not be accepted.

This is intended to be a challenging task, and quite a step up from what you have been doing so far, so think about things carefully. We can - and will - discuss some aspects in the Lab sessions, and of course, as usual, you can ask me anything by email, during Office Hours, or in the LMO Forums. Good luck!