1 Program description (65 marks)

The aim of this coursework is to build a program to store, manipulate, and retrieve information from the evolution of stock prices over time. All the work should be coded into a single Java NetBeans project, with the class structure and different functionalities of the program described as follows. 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.

1.1 `PricePoint` class (10 marks)

The basic building block of the program will be a simple `PricePoint` data class. A `PricePoint` object comprises of a pair $(t, p(t))$, where t is a time coordinate, and $p(t)$ the price at time t. `PricePoint` objects should be compared according to their time coordinate.

1.2 `PriceData` class (25 marks)

A `PriceData` object stores a collection of `PricePoint` objects. You should choose a suitable data structure in the Java collection framework for this, according to the following conditions.

- There should be no duplicate time coordinates among the `PricePoint` objects in the collection.
- The collection should be maintained in increasing order of time coordinates.

Leave a comment in your code explaining your choice of data structure for this.

In addition, your `PriceData` class should have the following functionalities.

1. It should be possible to create either an empty `PriceData` object (containing no `PricePoint` objects), or a customised object according to a specified (Java) `Collection` of `PricePoint` objects.

2. There should be a method `add(PricePoint newPp)` to add a new `PricePoint` to a `PriceData` object.

3. We can view a `PriceData` object as a function $p : t \mapsto p(t)$, where we take the linear interpolation between any consecutive time coordinates of PricePoints. Let t_i and t_f be the first and last time coordinates in a given `PriceData` object. In addition to being able to get the value $p(t)$ of this linear interpolation for any time coordinate $t \in [t_i, t_f]$, it should be possible to calculate the following statistics:

   - the *spread* of the `PriceData`, given by $\max_{t\in[t_i,t_f]}(p(t)) - \min_{t\in[t_i,t_f]}(p(t))$;
   - the *average* of the `PriceData`, given by $\frac{1}{t_f-t_i}\int_{t_i}^{t_f} p(t)\,\mathrm{d}t$;
   - the *maximal differential* of the `PriceData`, given by $\max_{t_1,t_2\in[t_i,t_f]}\left|\frac{p(t_1)-p(t_2)}{t_1-t_2}\right|$.

   Each of these should be implemented through a method with the same name as the statistic in question (e.g. `spread()` for the spread statistic). There should also be a general `display()` method in the `PriceData` class which displays the above statistics over the given time period in a sensible and informative manner. Figure 1 below illustrates these three statistics (and the linear interpolation) on an example of a `PriceData` object with five price points.

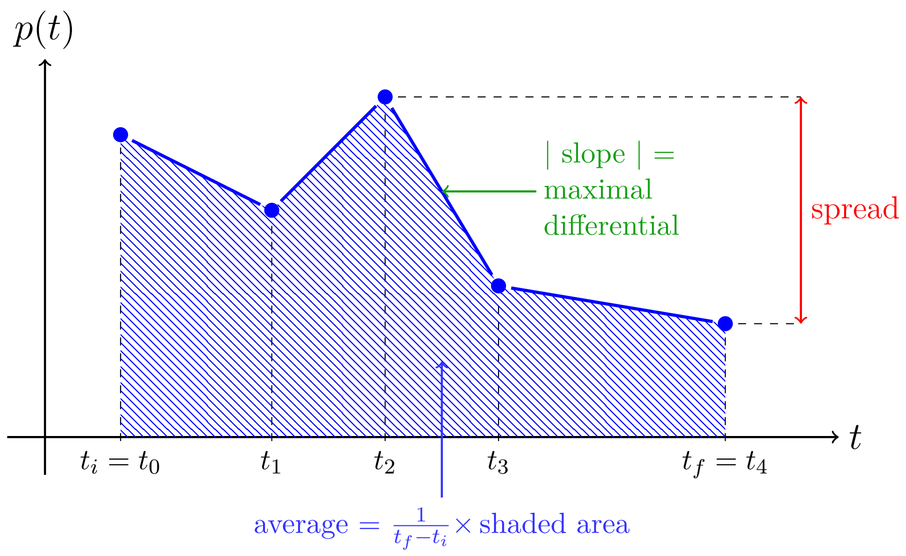
$p(t)$
t
| slope | = maximal differential
spread
$t_i = t_0$
t_1
t_2
t_3
$t_f = t_4$
average = $\frac{1}{t_f-t_i}\times$ shaded area

Figure 1: Illustration of the `PriceData` statistics.

1.3 `StockPriceData` class (15 marks)

Finally, a `StockPriceData` object should be a `PriceData` object with the additional information of the name of the stock, and two coefficients:

- a positive volatility coefficient v;
- a probability $\pi \in [0,1]$ measuring how likely price trends are to continue.

The `StockPriceData` class should have the same functionalities as the `PriceData` class. In addition, there should be a method that, given a time differential Δt, estimates a future `PricePoint` for the stock, as follows.

- Draw a random variable $r \in [0, v]$.
- Let $\varepsilon \in \{-1, +1\}$ be the monotinicity of $p(t)$ at t_f^-, that is $\varepsilon = +1$ if the function $p(t)$ is increasing in the final time interval, and $\varepsilon = -1$ otherwise[1]. Then the future `PricePoint` is given by the pair $\left(t_f + \Delta t,\, p(t_f + \Delta t)\right)$, where

$$p(t_f + \Delta t) = \begin{cases} p(t_f)(1 + \varepsilon r \Delta t) & \text{with probability } \pi \\ p(t_f)(1 - \varepsilon r \Delta t) & \text{with probability } (1 - \pi) \end{cases} .$$

1.4 Code quality (15 marks)

The remaining marks **(15)** 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 (35 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).

2.1 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 `PricePoint` class[2]...

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

[1]On Figure 1, we have $\varepsilon = -1$, since $p(t)$ is decreasing in the final time interval $[t_3, t_4]$.
[2]Give a brief description of the purpose of the `PricePoint` class here.

2.2 Testing description (15 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. This section should be one-and-a-half to two pages in length (screenshots excluded).

2.3 Improvements (10 marks)

Finally, this program is, by necessity, a simplified model. In your critical evaluation document, you will list **two (2)** possible improvements to the system. These could be for instance additional features to be implemented, changes to existing features so that the system is a more accurate reflection of a real-world system, etc. Give a brief justification for why these would improve the system. This part should be no longer than one page in length.