Homework 14 - Stock Market

Authors: Shishir, Carl, Jack Kelly, Andrew Chafos, Davis Williams

Problem Background

Many times when programming, you will come across situations where you're trying to deal with data that doesn't come right away. Tasks like accessing a webpage over the internet or reading through a file system need some processing time in the background before they can be done. For tasks like these, we sometimes employ event-driven programming in order to make our programs fast and reactive when the data comes back.

In this homework, you will be creating an application that keeps track of changes in a stock market database and reports information out when relevant. In our sample stock market, data might be coming in and changing in real time. When values of stocks change, we want to build a system that can reevaluate the new values of these stocks and print out updated analysis.

Solution Description

For this assignment, create the following file: `StockTracker.java`.

You have also been provided with `Stock.java`, `Market.java`, and `Tester.java`. We have a few analysis questions that you must respond to as a comment!

Provided classes/interfaces

`Stock.java`:

An enum that holds instances of the Stocks that we might track. You should look at the JavaDoc comments for `getCompanyName` and `getInitialValue` inside this file.

`Market.java`:

An interface that specifies a contract for a stock markets with stocks that change values. You should look at the JavaDocs for `registerHandler` inside this file. You will NOT be implementing Market yourself but you will be using instances of classes that implement Market. These instances will be created for you and passed into the methods that you create inside of `StockTracker.java`.

`Tester.java`:

This class is provided for you so that you can test your code. It contains a main method which creates an instance of Market and calls the methods inside of `StockTracker`. It is setup to simulate asynchronously accessing information from the actual stock market. It may be useful to read through the main method but don't worry about the implementation details.

StockTracker Class:

Instance Variables:

- `List<Stock> trackedStocks` - A `List` of the stocks that this tracker is tracking
- `int cutoff` - The threshold for buying or selling a stock
- `Market market` - The Market that this StockTracker is tracking

All fields should be private and non-static.

Constructor:

- `StockTracker(Market market, Stock[] stocks, int cutoff)` - Takes in a an array of Stocks and adds them to `trackedStocks`. Afterwards, assign the instance variables `market` and `cutoff` to their respective fields. When this is done, register a buy, hold, and sell tracker on every stock passed in using the private helper methods in the class.

Constructor should be public.

Methods:

- getter for `trackedStocks`. Returns a deep copy of `trackedStocks`
- getters for `market` and `cutoff`
- `private void registerHoldTracker(Stock stock)` - Should make sure that a message is printed any time a change in the stock's value causes it to be valued both >= the stock's `initalValue` and <= the stock's `initialValue` + `cutoff`. When this happens print out `"{Company Name} is valued at ${value}. Hold on to what you have."`.
  - You should achieve this by registering an event handler on the market for the given stock.
  - **This method must use only an instance of a named inner class when creating the handler**
- `private void registerBuyTracker(Stock stock)` - Should make sure that a message is printed any time a change in the stock's value causes it to be valued < the stock's `initialValue` - `cutoff`. When this happens print out `"{Company Name} just dropped to ${value}. Buy now!"`.
  - You should achieve this by registering an event handler on the market for the given stock.
  - **This method must use only an anonymous inner class when creating the handler**
- `private void registerSellTracker(Stock stock)` - Should make sure that a message is printed any time a change in the stock's value causes it to be valued > the stock's `initialValue` + `cutoff`. When this happens print out `"{Company Name} just rose to ${value}. Sell now!"`.
  - You should achieve this by registering an event handler on the market for the given stock.
  - **This method must use only a lambda expression when creating the handler**

Getters should follow best practices taught in class

Analysis Questions

Answer these at the top of StockTracker.java in a comment. Put your collab statement first and then a multiline comment with your answers.

1. What are the pros and cons of using named inner classes, anonymous inner classes, and lambda expressions? (4-7 bullet points. < 200 words.)
2. Why syntactically can we use a lambda expression when we are registering an event handler for the market? (1-3 sentences. < 150 words.)
3. Why would we decide to use event driven programming to access the stock market? (2-5 sentences. < 220 words)

Testing your code:

If you want to test out your own code before submitting we recommend creating a separate java file and implementing a main method there.

We have also provided sample tests in `Tester.java` **These tests are not comprehensive!**

This should be the output of `Tester.java`:

```
Apple is valued at $402. Hold on to what you have.
Pfizer is valued at $609. Hold on to what you have.
Cisco is valued at $411. Hold on to what you have.
Apple is valued at $403. Hold on to what you have.
AT&T just dropped to $117. Buy now!
Pfizer just rose to $614. Sell now!
Facebook is valued at $709. Hold on to what you have.
Intel is valued at $258. Hold on to what you have.
Pfizer just rose to $622. Sell now!
Facebook just rose to $715. Sell now!
Apple is valued at $407. Hold on to what you have.
General Electric just dropped to $488. Buy now!
Cisco is valued at $418. Hold on to what you have.
AT&T just dropped to $121. Buy now!
Facebook just rose to $719. Sell now!
Apple just rose to $412. Sell now!
Microsoft is valued at $354. Hold on to what you have.
Intel is valued at $254. Hold on to what you have.
Cisco is valued at $420. Hold on to what you have.
Facebook just rose to $720. Sell now!
Apple just rose to $411. Sell now!
General Electric just dropped to $482. Buy now!
Intel is valued at $258. Hold on to what you have.
Intel is valued at $254. Hold on to what you have.
Ford is valued at $303. Hold on to what you have.
Apple just rose to $420. Sell now!
Cisco is valued at $411. Hold on to what you have.
AT&T just dropped to $114. Buy now!
Ford is valued at $305. Hold on to what you have.
Facebook just rose to $728. Sell now!
Bank of America is valued at $300. Hold on to what you have.
General Electric just dropped to $477. Buy now!
Intel is valued at $255. Hold on to what you have.
Pfizer just rose to $630. Sell now!
Facebook just rose to $724. Sell now!
```

Note that Tester will not finishing running immediately because it is (fake) accessing data from the Stock Market.

Feel free to modify or create your own tests in the `main` method!

Allowed Imports

- You are allowed to import the following classes:
  - `java.util.Arrays`
  - `java.util.ArrayList`
  - `java.util.List`
  - `java.util.function.Consumer`

Rubric

- [100] StockTracker
  - [3] Instance variables

- [6] Getters
- [10] Constructor
  - [1] constructor assigns instance variables correctly
  - [9] constructor registers all the trackers
- [20] `registerBuyTracker`
  - [10] Correctly registers the handler
  - [10] Uses an anonymous inner class
- [20] `registerSellTracker`
  - [10] Correctly registers the handler
  - [10] Uses a lambda expression
- [20] `registerHoldTracker`
  - [10] Correctly registers the handler
  - [10] Uses an instance of a named inner class
- [21] Analysis Questions

Javadocs

For this assignment, you will be commenting your code with Javadocs. Javadocs are a clean and useful way to document your code's functionality. For more information on what Javadocs are and why they are awesome, the online overview for them is extremely detailed and helpful.

You can generate the javadocs for your code using the command below, which will put all the files into a folder called javadoc:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to include are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```
import java.util.Scanner;

/**
 * This class represents a Dog object.
* @author George P. Burdell
* @version 1.0
*/

public class Dog {

    /**
     * Creates an awesome dog (NOT a dawg!)
     */

    public Dog() {
    ...
    }

    /**
     * This method takes in two ints and returns their sum
     * @param a first number
     * @param b second number
     * @return sum of a and b
     */
```

```
    public int add(int a, int b) {
    ...
    }
}
```

A more thorough tutorial for Javadocs can be found here. Take note of a few things:

1. Javadocs are begun with `/**` and ended with `*/`.
2. Every class you write must be Javadoc'd and the `@author` and `@verion` tag included. The comments for a class should start with a brief description of the role of the class in your program.
3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an `@param` tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method has a non-void return type, include the `@return` tag which should have a simple description of what the method returns, semantically.

Checkstyle can check for Javadocs using the -a flag, as described in the next section.

Checkstyle

For this assignment, we will be enforcing style guidelines with Checkstyle, a program we use to check Java style. Checkstyle can be downloaded here. To run Checkstyle, put the `jar` file in the same folder as your homework files and run

```
java -jar checkstyle-6.2.2.jar -a *.java
```

The Checkstyle cap for this assignment is **100 points**. This means that up to 100 points can be lost from Checkstyle errors.

Collaboration Statement

To ensure that you acknowledge collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment AT THE TOP of at least one java file that you submit**. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials.

or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Recall that comments are special lines in Java that begin with `//`.

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `StockTracker.java`

Make sure you see the message stating "HW14 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your last submission: be sure to **submit every file each time you resubmit**.

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

1. Prevent upload mistakes (e.g. forgetting checkstyle, non-compiling code)
2. Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.