CS 245 Assign 3: Stock Observers

Objective:

- Apply the Strategy and Observer Patterns to a practical application.
- Work with Graphical User Interfaces.

Problem:

We have a bunch of investors who are trying to get rich on the stock market. These investors carefully watch the price of the stock and decide when to buy and when to sell. Each investor uses their own strategy to manage their money:

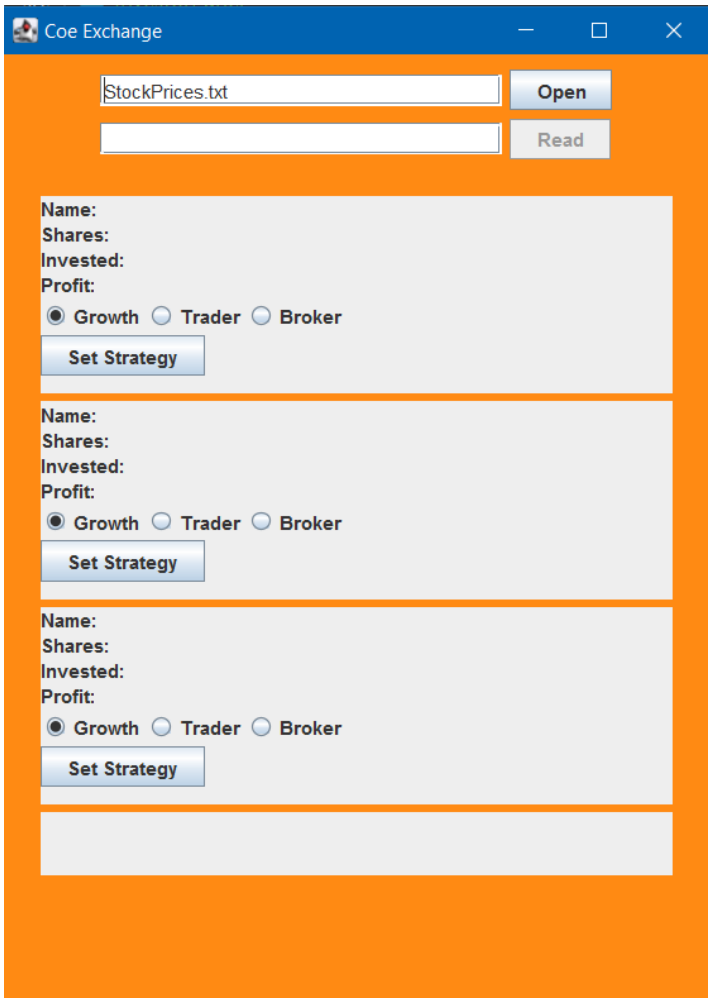
- Some favor steady investment
- Some follow the buy-low, sell high model
- Some listen to a broker.

Your job is to create a program that can allow you to watch their progress.

Starter Code

Make sure you study the code
– understand it –
Don't change anything yet.

1. There are classes that define Stocks, Investors and Brokers. These represent the data that we are going to work with. You probably need to add some getters/setters, and may need to add some additional behaviors (methods) but the core data is there.
2. There are panels to display each of these classes (StockPanel, InvestorPanel, and BrokerPanel).
3. The InvestorPanel has a sub-panel: StrategyPanel. This contains the Radio Buttons. Radio buttons have a 'selected' property, and are mutually exclusive because they are part of a ButtonGroup.
4. The StockPanel reads stocks from a file (From Task 7). The other panels have not been activated yet.
5. CoeExchange is the main class. It creates Investors and Brokers, and then sets up the GUI.



Approach

You are still not writing code, just trying to understand how to organize your work for the assignment. Read through this entire section and make sure you understand the tasks ahead of you. Here is an outline for the general sequence that you should work through.

1. **Stock Observer:** The program can read an updated stock price from a file – but nobody is paying attention. Your first job is implement the observer pattern with the Stock class as the source; the Investor and the Broker are observers.
2. **Start Investing:** Now that the Investors are getting some data, we want to have them do something. In this phase, you will add behaviors to the Investor to allow them to buy stocks and track their investments.
3. **Brokering 101:** All I have ever seen a broker do is yell "BUY!" or "SELL!". Now that the Broker is observing the stock, there probably is something in the data that tells them which to yell.
4. **GUI Updates:** Now that the data classes (Investor and Broker) are listening and reacting, you need to reflect those changes on the respective GUI Panels. That's right – the GUI panels need to observe the data objects for changes.
5. **Investment Strategies:** Give the Investor some options for how they want to react to changing stock prices.
6. **Listen to your Broker:** Instead of watching the price, let's listen to the broker to figure out what to do. That means that the Broker is both observing Stocks and sending messages to its Observers.

Note: you should commit and push your work frequently. However, at a minimum, you are *required* to have a commit at the completion of each phase. The commit message should be in all caps and indicate that the phase has been completed. For example: when you complete the first phase, you will commit with the message "STOCK OBSERVER COMPLETED".

Implementation

Finally, time to roll up your sleeves and start coding
(it really is amazing how much thought needs to take place before you touch the keyboard).

Phase 1: Stock Observer

The goal here is to allow others to observe the state of the stock class. To accomplish this, you will add three additional classes/interfaces to your project:

- StockObserver : the interface that describes what observers do.
- StockSource: the interface that describes how to add/remove observers as well as how to notify all the observers that the Source has changed.
- StockEvent: This is a class designed to pass information along to the observers. For now, we want to pass the price along to the observers. In fact, we could just pass the price along directly, but a) it is good practice to encapsulate this data in an "Event" object and b) You never know when you may want to pass along more data (foreshadowing). The StockEvent should have a single instance variable that contains price, which is set by the constructor. It also should have a getter for the price.

Once you have added these classes/interfaces to the project, you need to:

- Make Stock implement StockSource. Whenever the price changes, you should create a new StockEvent (containing the price) and send it to all of its observers.
- Make the Investors and the Broker observers. For right now, just have them use the console (System.out.println) to indicate their name (or "broker") and the price they heard.

Phase 2 Start Investing:

Now that the Investors are getting some data, we want to have them do something. Update your Investor class to add a method:

```
public void makeTransaction(int numShares, double price)
```

This method should handle both buying and selling of shares (if numShares is negative, it should be considered selling. This method should appropriately adjust the number of shares owned and the amount invested. (Note: you cannot own less than zero shares).

For now, whenever the Investor receives a StockEvent update, you should buy 10 shares.

Phase 3 Brokering 101:

Brokers are also carefully watching stocks with the following behavior:

a. The broker watches for streaks; a streak is established when a stock has either increased for five consecutive days or decreased for five consecutive days.
b. When an increasing streak ends (i.e. there is a decrease after a series of increases), the Broker shouts (prints), "SELL!!!!"
c. When a decreasing streak ends, the Broker shouts, "BUY!!!".

To implement this behavior, you will need to add some additional instance variables to the Broker class to track the streaks.

Phase 4 GUI Updates:

A. Update InvestorPanel

Typically, the GUI is reactive to user events; there are built-in listeners (observers) that we can add and override to achieve our goals. However, in this case, the InvestorPanel needs to be responsive to changes from an object that we created (Investor), so we need to establish the observers ourselves.

1. Implement the Investor Observer Pattern: Create two interfaces, InvestorSource.
2. Create an InvestorEvent that encapsulates the data that has changed: sharesOwned and investment Amount. These values should be set by the constructor and have the appropriate getters.
3. Implement the necessary methods to make Investor a source and InvestorPanel an observer. The investor panel should extract the information from the InvestorEvent to update the appropriate fields. (Note the Profit is calculated by comparing the current value of the shares to the amount that was invested.
4. Finally, we need to connect these two classes. Notice that the InvestorPanel has a setInvestor method. This would be a good place to tell the investor that ***this*** InvestorPanel would like to listen to it. Also to be complete, if the InvestorPanel was already listening to another Investor, it would be a good idea to stop.

B. Update BrokerPanel

We *could* repeat the same process for theBrokerPanel (i.e.creating a Broker Observer). However, looking ahead, a little bit, we realize that the Broker is going to be listened to by BrokerPanel – but it is also going to be listened to by other Investors. Moreover, when the Investor listens to the Broker, they are going to evoke a reaction similar to the response to a StockEvent --- namely they may Buy or Sell stock. Therefore, it makes sense to have the Broker's advice incorporated into the StockEvent. and make the BrokerPanel a StockObserver. Try this:

1. Update the StockEvent class:
   a) Add an instance variable to hold the advice.
   b) Overload the constructor to accept values for both instance variables (price, advice)
   c) Update the original constructor to set advice to ""
      (if you don't do this, you will likely have to contend with a NullPointerException.)
   d) Observe that the advice is always "BUY!", "SELL!" or "Hold". To advertise these values to users of this class, you should create constants hold these values. Recall that this is done using public static variables and by convention are named with All Caps e.g.

```
public static String BUY = "BUY!";
```

2. Update the Broker class to implement StockSource. When it hears a stock price change, it formulates its advice and then notifies its observers with a new StockEvent – including the price and its advice (Use the constant from the StockEvent class). Yes, the broker implements both StockObserver and StockSource!
3. Update the BrokerPanel to be a StockObserver. It should update its label whenever it receives a StockEvent.
4. The constructor for the BrokerPanel is passed a broker – this is a good place to tell *this* BrokerPanel should listen to the broker.

Phase 5 Investment Strategies:

Currently the Investor is always buying 10 shares of stock whenever it hears a StockEvent. Let's refactor our code to make it so that the Investor adjust their behavior dynamically. To accomplish this, we will create an Investment Strategy. Based on a StockEvent, the strategy should allow us to determine the number of shares to buy or sell:

```
public interface InvestmentStrategy {
    int buyOrSell(StockEvent e);
}
```

Remember: To sell stock, the return value should be negative.

For this phase, our job is to implement 2 strategies: a GrowthStrategy and TraderStrategy.

- For the GrowthStrategy, you should buy as many shares as you can (whole shares only) for $100.
- For the TraderStrategy, we want to sell-high, buy-low. Start with a $100 investment. When the stock price is 10% more than the average price paid, they will sell 10% of their holdings (note: partial stocks cannot be traded). When the stock price is 10% less than the average price they paid, they will buy an additional $500 worth of shares,
    - Note: this strategy should be created with information about the Investor's holdings and amount invested. Is should <u>not</u> have access to the actual investor.

After creating these strategies, update the StrategyPanel to update its investors strategy.

Phase 6 Listen to your Broker:

Investing on your own can be risky – even with a good strategy. Instead we are going to give our Investors the opportunity to listen to their brokers. Notice that way back in CoeExchange, each investor was provided with a broker; now we just need to pay attention to what they are advising us to do.

Create a new BrokerStrategy class – based on the advice contained in the StockEvent, your investor should:

- Buy 20 shares when the broker says "BUY!" (a StockEvent constant).
- Sell 20 shares when the broker says "SELL!" (a StockEvent constant).
- Do nothing otherwise. (Note – this accounts for StockEvents where the broker says "Hold" (a stock Event constant) as well as StockEvents where there is no advice (i.e. when heard that the stock price changed from the Stock class).