CECS 277 HOMEWORK OBSERVER PATTERN

OBJECTIVE: Apply what we learned in lecture and lab about the Observer Pattern to a toy stock market application.

INTRODUCTION: Please remember the coding standards here.

Imagine that you are an agent for a brokerage firm. There are some stocks that you trade in consistently, so you want to know the minute that anyone makes a buy or sell order in any of those stocks. So you subscribe as an Observer to all of those stocks of interest. Meanwhile, you have colleagues who also trade in various stocks. Some of those stocks you are interested in, some of them you are not. Each of your colleagues subscribe as an Observer to a set of stocks.

When you or some other agent places a buy or sell order on a given stock, that stock alerts all of the agents who have subscribed to that stock so that they know immediately about the trade. The information content of each trade is: the stock being traded, the number of shares in the trade, the dollar amount for the whole trade (not price/share but the whole amount) and the transaction type.

The Stock that you pass to the Bid constructor should be an instance of the Stock class, not just the stock symbol. That helps with data integrity in your stock market so that users do not put in just any old string for the stock symbol. Also, you will want to use an enumeration for the transaction type. The two types of transactions that you will want to work with are simply `BUY` and `SELL`.

PROCEDURE:

1. Either implement your own Subject and Observer interface, or use the `java.util.Observable` (AKA Subject) and `java.util.Observer` interfaces.
2. Build the `Bid` object with the necessary member variables and a `toString()` method. You will not need getters and setters for this since our little application will just rely on `toString` to return data from a given `Bid` instance.
3. Build the `Stock` class that implements the Subject/Observable interface.
    a. Give `Stock` a member method called `trade`. The `trade` method is how we update the `Stock` instance. The `trade` method accepts an instance of `Bid` as its only parameter, and stores that `Bid` instance locally.
    b. Give `Stock` a member method called `getBid()` that will return a **copy** of the `Bid` that happened most recently to that instance of `Stock`.
        i. Your `update()` method in your Agent class (more on that in a moment) will need to call `getBid()` to find out just what changed about the `Stock` that the `Agent` subscribes to.
4. Build the `Agent` class that implements the `Observer` interface.
    a. You will want to store the name of the agent as a member variable.
    b. The `update()` method needs to accept an instance of Subject so that the agent knows which Stock it was that had the bid placed.

5. Write a tester, called `ObserverRunner`, that will create several `Stock` instances, several `Agent` instances, have the agents subscribe to a few `Stocks`, and then place some `Bid`s on the `Stocks`.

WHAT TO TURN IN:

- Agent.java
- Bid.java
- Observer.java (if you roll your own)
- ObserverRunner.java (the test routine)
- Stock.java
- Subject.java (if you roll your own)
- TransactionType.java (for the Buy/Sell enumeration)
- Sample console output named console.txt.
- The UML class diagram.

SAMPLE OUTPUT:

```
Agent - name: Tom Clancey alerted to Bid- Symbol:ORCL Buying 20 shares for the amount: $3421.0
Agent - name: Robert Mitchner alerted to Bid- Symbol:ORCL Buying 20 shares for the amount: $3421.0
Agent - name: Noah ben Shea alerted to Bid- Symbol:ORCL Buying 20 shares for the amount: $3421.0
Agent - name: Richard Rohr alerted to Bid- Symbol:ORCL Buying 20 shares for the amount: $3421.0
Agent - name: Tom Clancey alerted to Bid- Symbol:BA Selling 10 shares for the amount: $50.0
Agent - name: Robert Mitchner alerted to Bid- Symbol:BA Selling 10 shares for the amount: $50.0
Agent - name: Noah ben Shea alerted to Bid- Symbol:TRKX Buying 30 shares for the amount: $204.36
Agent - name: Richard Rohr alerted to Bid- Symbol:TRKX Buying 30 shares for the amount: $204.36
```