HW2 – Decoupling with Observers and Extending with Decorators

Estimated time: 12-16 hours

Objectives

- Become more familiar with UML modeling
- Become familiar with the Observer pattern
- Become familiar with the Decorator pattern
- Improve unit testing skills
- Become broaden and strengthen programming skills in related areas, including network communications and user interfaces.
- Become familiar with creating a virtual machine in a cloud and running programs on that machine.

Overview

In this assignment, you will build a simply real-time stock-price monitoring system that gives users multiple ways of looking at portfolios of stocks, their prices, volume, and current news. An external program (provided by the instructor) will send stock ticker information to your monitor program in real-time via a UDP socket. Your program will use this incoming data to update in-memory stock objects, which in turn will update various panels of information on a graphical user interface (GUI). See Figure 1-3 is for illustrations of how GUI may look.

The challenge is to build this system in a way that allows user to control what information panel to show and how they look. Application of the observer and decorator patterns will help you achieve these objectives so your design has low coupling, high cohesion, and is reasonably extensible.

Technical Skills

To build this system you will need know a little bit about

A. event-driven architectures, which are common for graphical user interface software (GUI's);
B. Intra-process concurrency using threading;
C. GUI development frameworks, tools, and techniques; and
D. UDP (Datagram) sockets.
E. Creating and operating a virtual machine in a cloud environment

CS2410 or CS2412 should have provided you with some background in A and B. Nevertheless, we will go through the essential skills together as a class. For C-E, we will do some tutorials and provide some code in C# and Java that you may adapt or port.

Background

A publically traded company allows its stock to be bought and sold on stock exchange, e.g., the New York Stock Exchange, NASDAQ, London Stock Exchange Group, and Japan Exchange Group. We'll limit ourselves to one factious exchange, similar to NASDAQ. A company, and therefore, its stock is uniquely identified on an exchange with a stock symbol. For example, Amazon's symbol on NASDAQ is AMZN.

The instructor will provide a CSV file, called Companies.csv, containing a list of companies and their symbols for our factious exchange. Each line in this file will represent one company and will be formatted as following:

```
<symbol>,<company name>
```

A message from the exchange (i.e., the simulator program) will contain following information about a company:

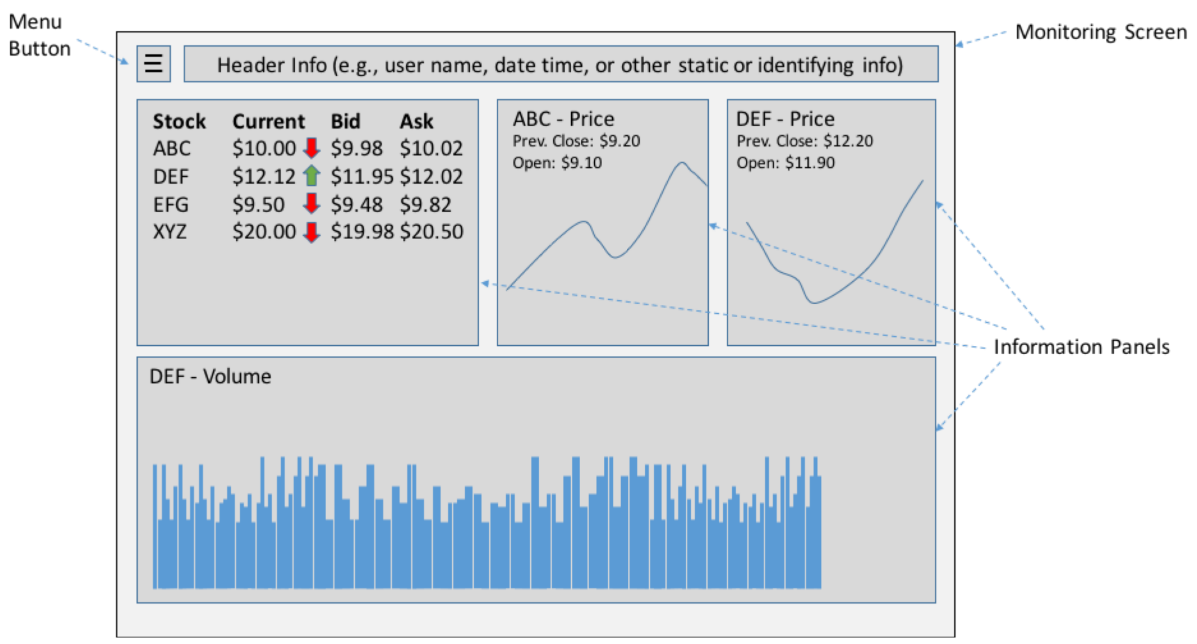
- The company's stock symbol
- Today's opening price
- Previous closing price
- Current price
- Bid price
- Ask price
- Volume sold today
- Rolling 10-day average volume

Functional Requirements

Below is a list of functional requirements for your stock monitoring system (SMS)

1. **Portfolio setup**: SMS needs to allow the user to create and edit a set of symbols to monitor, e.g., {AMZN, AAPL, MSFT, GOOGL}. We'll call a set of stock symbols, a *portfolio*.
   1.1. The portfolio may only contain symbols listed in the Companies.csv file
   1.2. The user should be able to save a portfolio to a file
   1.3. The user should be able to load a portfolio from a file

2. **Display Panel Management**: SMS must allow the user choice up to six different display panels to show on a monitoring screen at any given moment. See Figure 1 for an examples of how the monitoring screen may look.
   2.1. The user must be able to create an instance of any information panel type. Requirements 2.6-2.8 describe several types of panels that SMS must include.
   2.2. After dynamically create an information panel, SMS must add it to the monitoring screen.
   2.3. SMS should allow the user to create multiple instances of the same kind of panel.
   2.4. When creating an instance of some panel types, it may be necessary to capture customization parameters from the user that control what information the panel with contain, how the panel



Stock	Current		Bid	Ask
ABC	$10.00	⬇	$9.98	$10.02
DEF	$12.12	⬆	$11.95	$12.02
EFG	$9.50	⬇	$9.48	$9.82
XYZ	$20.00	⬇	$19.98	$20.50

will look, or how it will function. See Requirements 2.6-2.9 for details about each of the required information panel types.

2.4.1. The user must be able to remove an information panel from the monitor screening at any time.

2.5. For simplicity, SMS may have all of the panels to be the same size. However, if desired, you may allow the panels to vary in size and fill available space on the monitoring screen, as shown in Figure 1.

2.6. Portfolio Stock Prices Panel

2.6.1. A Portfolio Stock Prices Panel should show a table contain one row of per symbols. This information must include: today's opening price: Current price, direction (which is whether the current price is greater or less then the last price), bid price, ask price, and ask size. The developer may add other columns, as desired

2.6.2. When creating a new portfolio panel, the user will need to specify which symbols from the portfolio it will display.

2.6.3. The user should be able to determine if the panel displays the stock's open price or previous close price

2.6.4. The actual style and appear of the graph is at the developer's discretion

2.7. Individual Stock Price Graph

2.7.1. An Individual Stock Price Graph is a panel that contains a rolling graph that shows the current stock price in 1-minutes interval over that last hour.

2.7.2. When creating an Individual Stock Price Graph, the user will need to specify which symbol from the portfolio to display.

2.7.3. The user should be able to determine if the panel displays the stock's open price or previous close price

2.7.4. The actual style and appear of the graph is at the developer's discretion

2.8. Individual Stock Volume Graph

2.8.1. An Individual Stock Volume Graph is a panel that contains a rolling graph that shows the current stock volume for the day in 1-minutes interval over that last hour.

2.8.2. When creating an Individual Stock Price Graph, the user will need to specify which symbol from the portfolio to display.

2.8.3. The user should be able to determine if the panel displays the stock's open price or previous close price

2.8.4. The actual style and appear of the graph is at the developer's discretion

2.9. Other panel types

2.9.1. You may develop other types of information panels, as desired. Be creative, but keep them meaningful in the context of the stock monitoring.

3. **Communications and Message Processing:** SMS must be able to communicate with the exchange simulator using UDP datagrams.

3.1. To begin receiving Ticker Messages, SMS must send a Stream Stocks Message to the exchange simulator.

3.1.1. When you run the exchange simulator, it will display the communication end points with which SMS can communicate. SMS must send a Stream Stock Message to one of this end points.

3.1.2. The Stream Stocks Message must contain a list of stock symbols for which SMS wants to receive Ticker Messages.

3.1.3. The Stream Stock Message must be encoded as follows:

Number of symbols – 2 bytes in Network Standard Byte order
Symbol 1 – 6 bytes that represent ASCII string, space padded to the right
Symbol 2 – 6 bytes that represent ASCII string, space padded to the right
Symbol 3 – 6 bytes that represent ASCII string, space padded to the right
…

Note: See sample code for an implement of Stream Stock Message with an encoding method

3.1.4. To change which stocks the simulator streams Ticker Messages for, SMS simply sends a new Stream Stock Message.

3.1.5. To stop the streaming of Ticker Message, SMS can send a Stream Stock Message with a Number of Symbol count of 0 and no symbols.

3.2. Your SMS must be able to receive Ticker Messages from the exchange simulator at a rate of about 50 per second. The exchange simulator will send one ticker message per second per symbol.

3.2.1. The Ticker Message will contain the following information:

3.2.1.1. Stock Symbol – An ASCII string up to 6 character, padded to the right to 6 characters with spaces. The ASCII string will be encoded into 6 bytes.

3.2.1.2. Message Timestamp – A long integer that represents ticks, encoded in network standard byte order

3.2.1.3. Opening Price – An integer that represents today's opening price for the stock in pennies, encoded in network standard byte order

3.2.1.4. Previous Closing Price – An integer that represents the previous day's closing price for the stock in pennies, encoded in network standard byte order

3.2.1.5. Current Price – An integer that represents the current price for the stock in pennies, encoded in network standard byte order

3.2.1.6. Bid Price – An integer that represents the bid price for the stock in pennies, encoded in network standard byte order

3.2.1.7. Ask Price – An integer that represents the ask price for the stock in pennies, encoded in network standard byte order

3.2.1.8. Current Volume – An integer that represents the volume since the last tick, encoded in network standard byte order

3.2.1.9. Average Ten-day Volume – An integer that represents the previous average volume for the previous ten days, encoded in network standard byte order

Note: See sample code for an implement of Ticker Message with a decoding method

Design and Implementation Suggestions

- Study the Cruise Control and Bouncing Ball examples
- Study the sample Threading code; feel free to adapt it
- Study the sample message and socket code; feel free to adapt it
- Think about the data structures that you need to manage the list of possible companies, portfolios, and stocks
- Consider how you can use the observer and decorator patterns so your design has low coupling, high cohesion, and is extensible with respect to adding new types of information panels
- Consider how if there are opportunities to improve the design and implementation by applying the strategy pattern

Instructions

To build this system, you will need to do the following:

1. Design your SMS
   1.1. Capture your design with appropriate UML class, interaction, and state-chart diagrams.
2. Look for opportunities to apply the observer and decorator patterns
3. Implement your SMS sufficiently to satisfy the about functional requirements. Embellish and make it look nice, as you see fit, but stay within the estimated time.
4. Create meaningful and relatively thorough executable unit test cases for the non-GUI components of your program. Do not worry about testing the GUI components of your system in this homework assignment.

5. Create a virtual machine in a cloud environment and run the exchange simulator on that virtual machine
6. Test your whole program at the system level using the exchange simulator on the virtual machine and ad hoc testing methods

Submission Instructions

Zip up your entire solution, including test cases and sample input files, in an archive file called CS5700_hw2_<*fullname*>.zip, where fullname is your first and last names. Then, submit the zip file to the Canvas system.

Grading Criteria

Criteria	Max Points
A clear and concise design captured in UML class, interaction, and state diagrams	20
A working implement, with good encapsulation, abstractions, inheritance, and aggregation, and with appropriate use of the observer and decorator patterns. Also, appropriate use of the strategy pattern, if needed	40
Meaningful and thorough executable unit test cases for non-GUI components	40
Reasonable systems testing using the simulator	20