

**WESTERN UNIVERSITY**  
**FACULTY OF ENGINEERING**  
**DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING**  
**SE 3352A – SOFTWARE REQUIREMENTS AND ANALYSIS**

**Assignment 2: Stock Market Application**

**Due Date: November 28<sup>th</sup> , 2014**

This assignment is to be done individually. Do not show your code to any other student and do not look at any other student's code. Do not put your code in any public domain. However, you may get all the help you need from the instructor or the TAs. Also you are encouraged to discuss your problems (if any) with other students.

## Objectives

In this assignment you are required to develop a stock market application, where the Observer design pattern is utilized. C# and .NET framework libraries will be your programming language and the tools to build your application. The stock Market application is responsible for displaying real-time stock information in multiple format (e.g., Stock data summary, Stock market depth) so that when the stock price change or Bid/Ask orders are issued, the application will need to update the stock summary display and market depth.

## Requirements

Figure 1 below represents the structure model for the required application. This model has been design to utilize the structure of the observer design pattern.

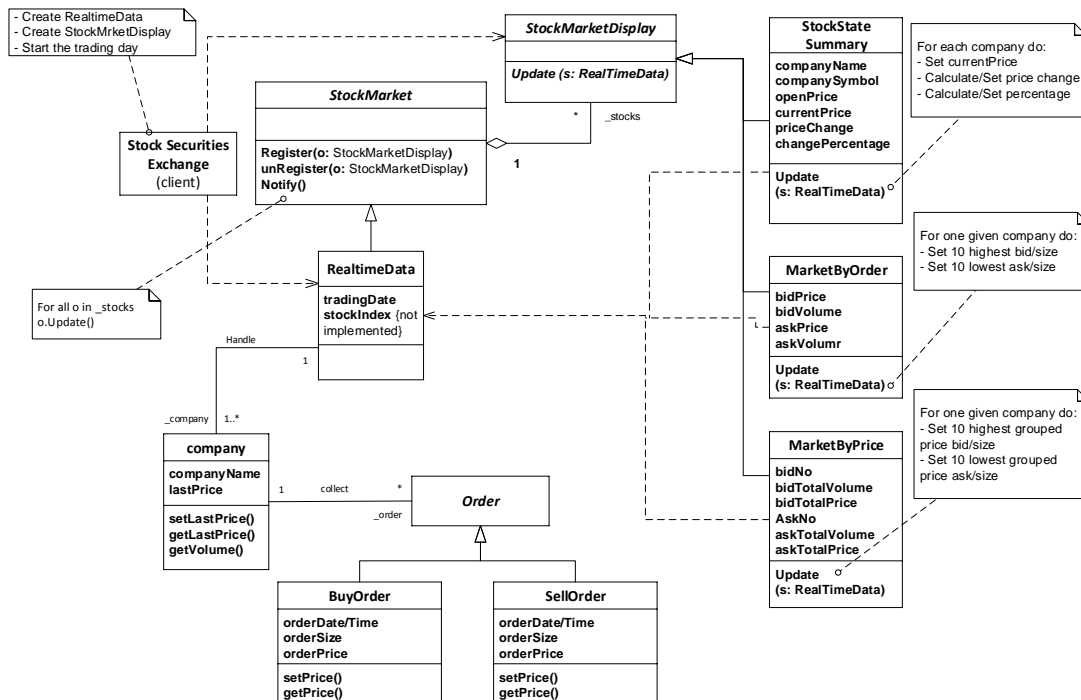


Figure 1

By considering the above class diagram, your application should include the following components:

### 1. Application Data Model:

In general, this component represents the subject of the observer pattern. This model is responsible to handle all the trader buy/sell orders related to all company registered in the market. Note that, All orders need to be archived (during the running time), and the deal (the transaction) will take place when the order price is agreed upon between the buyer and the seller. The current price of company stock is the price of the last successful transaction. When a trading day starts, all traders can issue their buy/sell orders, the application will collect all of these orders by which it finds deals and executes them.

In particular, we have a **StockMarket** abstract class that models various stocks. Within this application, an instance of the StockMarket class acts as the concrete subject and named **RealTimeData**. This class contains an instance variable which represents:

- List of companies represented in the market.
- For each company, the current (last) price of the company stock, which fluctuates throughout the day.
- For each company, list of pending buy orders and sell orders.

Two forms need to be developed that help facilitate traders to make orders. Both should have a combo box field to select one individual company, one field to enter the price and other field for the number of shares. One form for entering bid orders and the other form for entering the selling orders. These information will effects the data model directly. If there is matching between the bidding and selling order, a deal will be made and the quantities of the share and the current price will be adjusted accordingly. Figure 2 shows a sample look-and-feel of these two forms.

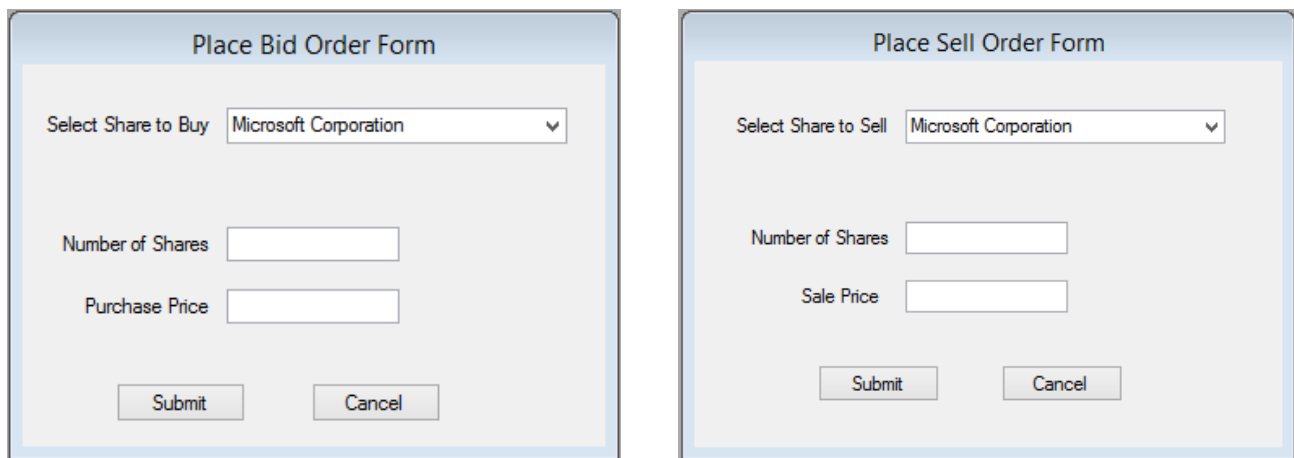


Figure 2

### 2. Application Views:

In order to display the real-time stock information to the traders, the application uses a **StockMarketDisplay** interface that displays to a form-based user interface (GUI). Within this application, the **StockMarketDisplay** interface acts as the observer and its implementations as the concrete observers. These concrete observers are, Stocks stats summary, Stock market depth arranged by orders, and Stock market depth arranged by price. Once each of these concert observer instantiated it **register itself** to the application data model. It also **unregister itself** when its corresponding form is closed. As the buy/sell price changes over the course of the trading day, the current buy/sell price of the StockMarket instance changes as well. Since the StockMarketDisplay is observing the StockMarket

instances, these state changes are displayed to the traders as they occur. The following describe each of the concrete observer.

### 2.1. Stocks state summary:

We are going to deal with only three companies, and this will be hard coded to the application. The company class should maintain the following:

1. Company name. (hard coded)
2. Open Price: the initial share price at the beginning of the treatment day (hard coded).
3. Last Sale (price): The price at which the most recent trade was completed. This field should be updated automatically once the value of the last price in the company class has changed.
4. Change Net: the net increase or decrease value from the last trade. This field will be calculated on the fly to display the net difference between the new deal price and the current price.
5. Change icon: display ▲ for increasing changes, ▼ for decreasing changes and ◆ for zero net changes. These three images are given.
6. Change Percentage: the percentage of the net change. This field will be calculated on the fly to display the percentage of the net increase or decrease of the new deal price.
7. Share Volume (Last size): The number of shares that were traded in the trading day for each individual company. This field will be updated by calling getVolume() method implemented in the company class.

Figure 3 shows a sample look-and-feel of the Stock State Summary form.

Company	Open Price	Last Sale	Change Net - %	Share Volume
Microsoft Corporation (MSFT)	\$46.13	\$47.24	1.11 ▲ 2.47%	57,268,074
Apple Inc. (AAPL)	\$105.22	\$104.83	0.39 ▼ 0.37%	43,432,023
Facebook, Inc. (FB)	\$80.67	\$80.67	0 ◆ 0%	31,548,302

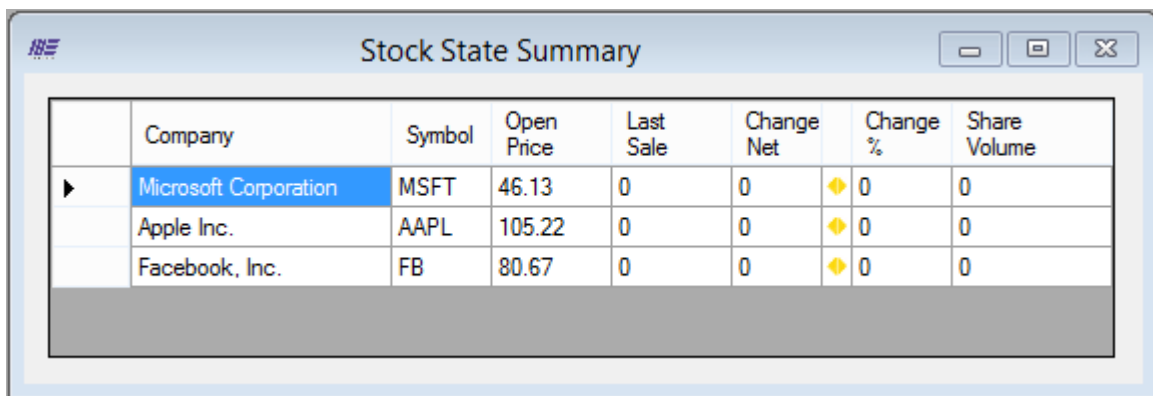


Figure 3

### 2.2. Stock Market Depth (by order):

The depth of market arranged by order (for individual stock) should display, at minimum, the following:

1. Highest Buy (bid) prices: The highest 10 available prices that traders are willing to pay to buy a share.
2. Bid sizes: The number of shares that are available at each of the highest bid prices.
3. Lowest Sell (ask) prices: The lowest 10 available prices that traders are willing to accept to sell a contract.
4. Ask sizes: The number of contracts that are available at each of the lowest ask prices.

All the above fields should be updated automatically once an order (buy/sell) has took place. Figure 4 shows a sample look-and-feel of the Stock Market Depth (by order) form.

Market By Order - Apple Inc. (AAPL)			
Buy		Sell	
Volume	Price	Price	Volume
48,577	104.80	105.40	29,478
125,000	104.80	105.40	50,000
250,000	104.80	105.40	48,000
223,779	104.80	105.40	70,439
8,506	104.70	105.40	10,000
28,497	104.70	105.40	50,000
10,000	104.60	105.50	50,000
10,000	103.90	105.50	25,000
100,000	103.90	106.00	102,524
90,000	103.80	106.00	50,000

Market Depth By Order (Microsoft Cor...			
Buy		Sell	
Volume	Price	Price	Volume

Figure 4

### 2.3. Stock Market Depth (by price):

The depth of market (grouped by price) for individual stock should display, at minimum, the following:

1. The number of the highest Buy (bid) prices: lists up to 10 different groups of the highest prices.
2. Accumulated bid sizes: the total number of contracts per each highest group
3. Accumulated bid prices: the total value (prices) per each highest group.
4. The number of the lowest Sell (ask) prices: lists up to 10 different groups of the lowest prices.
5. Accumulated Sell sizes: the total number of contracts per each lowest group
6. Accumulated Sell prices: the total value (prices) per each lowest group.

All the above fields should be updated automatically once an order (buy/sell) has took place. Figure 5 shows a sample look-and-feel of the Stock Market Depth (by price) form.

Market By Price - Apple Inc. (AAPL)					
Buy			Sell		
#	Volume	Price	Price	Volume	#
4	647,356	104.8	105.4	257,917	6
1	10,000	104.6	105.5	75,000	2
2	110,000	103.9	106	152,524	2
1	90,000	103.8			

Market Depth By Price (Apple Inc.)						
Buy			Sell			
#	Volume	Price	Price	Volume	#	

Figure 5

### 3. Application Client, Stock Securities Exchange:

Stock Securities Exchange application acts as the client in the structure model in Figure 1. In reality, Stock Exchange applications are hosted by dedicated servers and offer their services over highly secured systems. Though in this assignment we will ignore these details and we will develop the entire system locally so that Stock Securities Exchange application (the client) will be responsible to create the application data model and controls the creation of the three observers.

The client will be a multiple-document interface (MDI) with a menu bar that includes the top level items Market <<closed/Open>>, Watch, Orders, and Windows. Please see the layout in Figure 6 below.

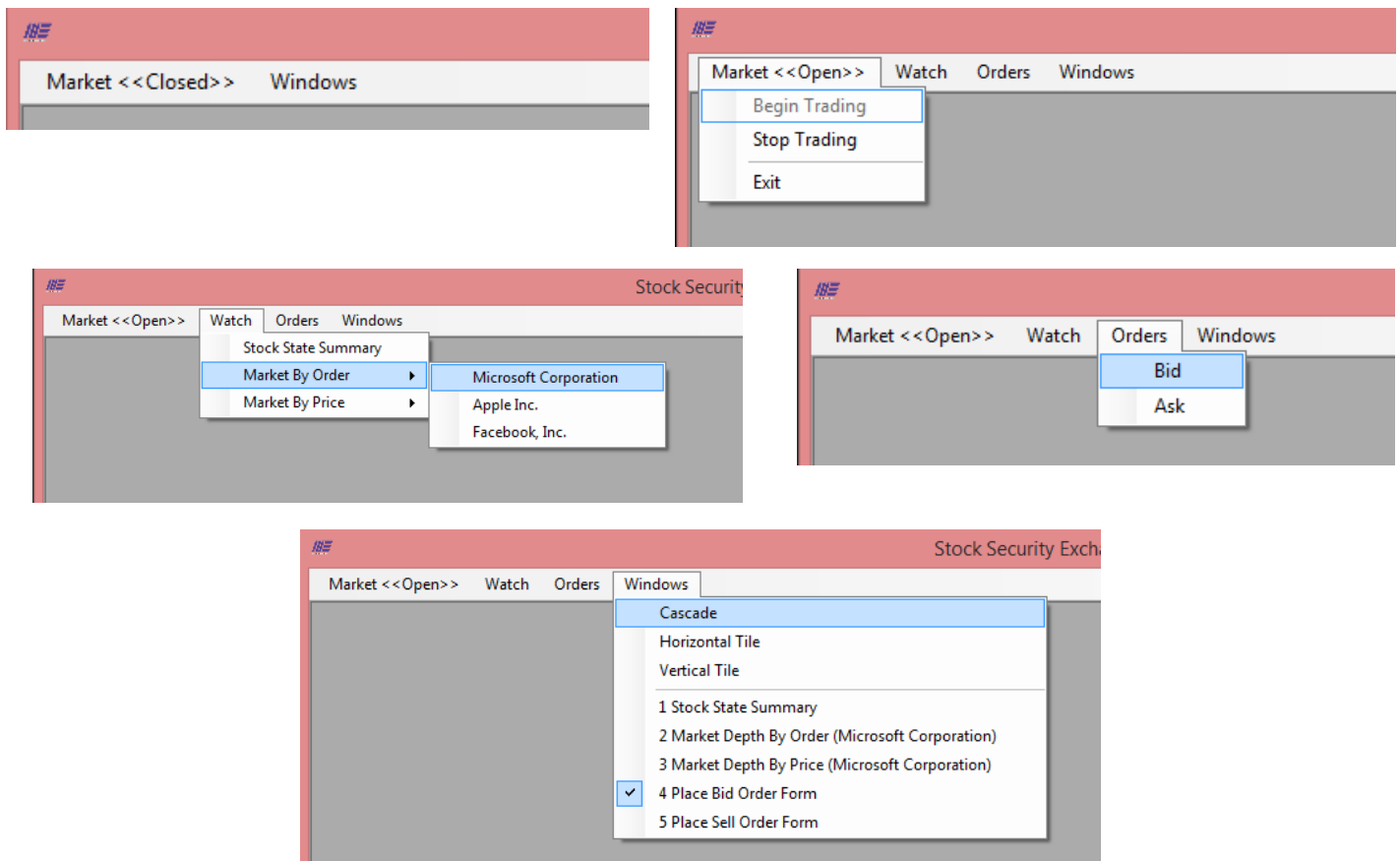


Figure 6

This client application should be able to facilitate the following functionalities:

1. When the application start, Market menu item will be shown with a stereotype "<<Closed>>", and the Watch and Orders menu items will be hidden.
2. The company names in the Watch menu item need to be dynamic and collected from the application data model (not hard coded).
3. When Begin Trading menu option is clicked the Application data model will be created and the hidden menu items will be appeared. In addition, the stereotype "<<Closed>>" will be change to "<<Open>>".
4. Any of the observers can appears at any time through the Watch menu item, and disappear by closing the observer form.
5. A trader can make a bid order or Ask order by selecting the corresponding option in the Order menu item.

6. The Market menu item has Stop Trading and Exit as a submenus, by clicking any of them the entire application will be shutdown.
7. The Windows menu item has Cascade, Horizontal Tile, and Vertical Tile as a submenu to arrange the current working child form.

Figure 7 shows one working scenario of the required application.

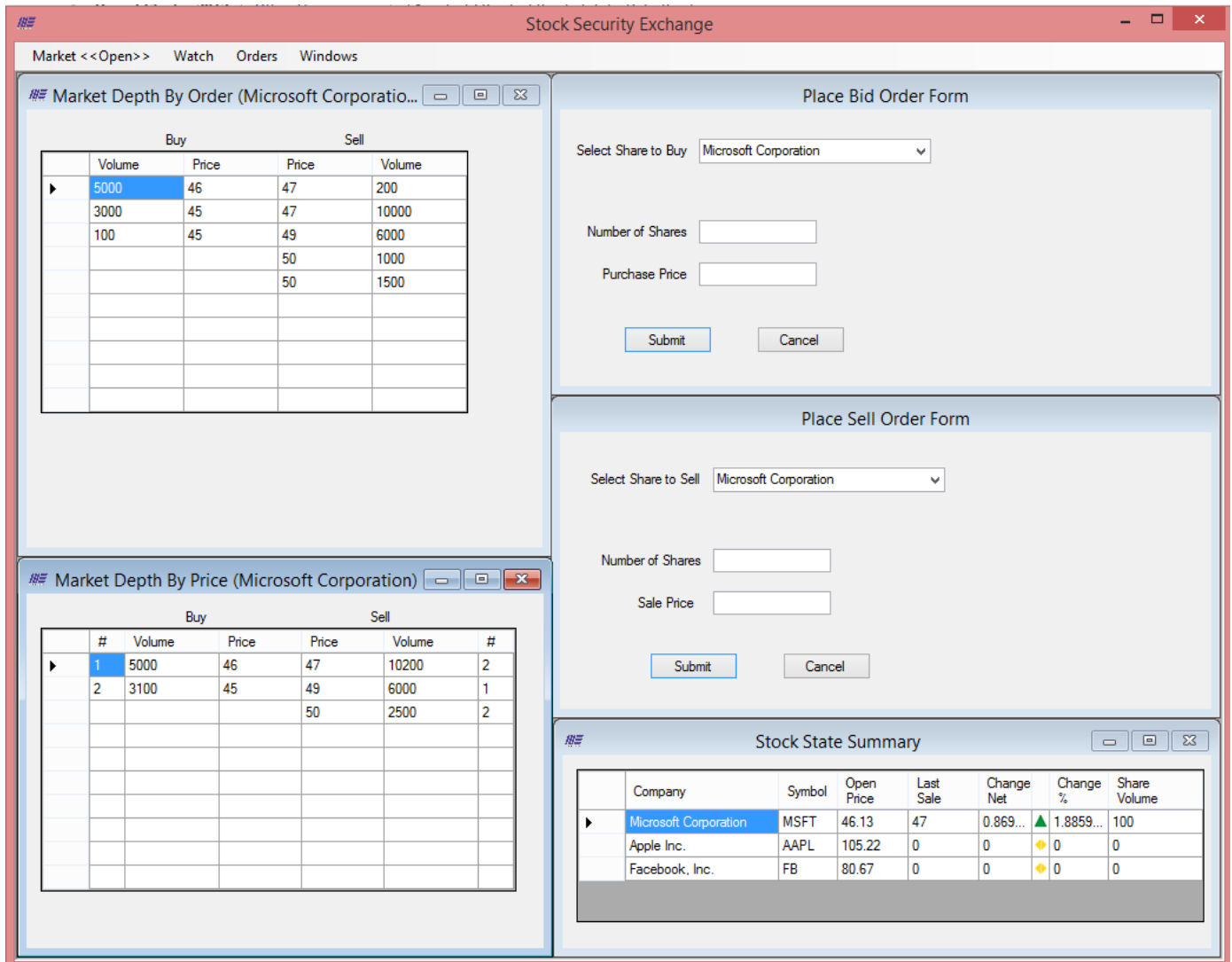


Figure 7

## Hand In

Submit the final version of your source code (Fully commented and formatted) through Sakai by the due date mentioned above. This will include a compressed achieve file (zip file) for the entire working directory of your application. Name the Zip file as assignment2\_yourUWoid.