CIS605 HW05: Practice 03

VB Classes - II

Total Points: 20

Practice Assignments

Practice assignments are given with the intention that you will use them for self-learning. The expected approach is that you will create / write / solve them on your own, and then after you have completed this (and not before) you will look at the solution to see possibly other good (or better) ways to solve the assignment. You must complete and submit running practice problems in order to get credit for them, but you will not be given feedback on what you did right, wrong, or could have done better. You may derive this information yourself by studying the provided solution. Practice assignments are generally due before midnight on the day that the test that the material covered by the practice assignment is opened/released. In general this means you will have finished the practice assignment(s) before starting on the corresponding test.

VB Classes – Business Logic Classes, Constructors.

The purpose of this assignment is to continue working with the distinction between User Interface and Business Logic Classes, to focus on Constructors, and to build more skills and confidence working with business logic classes, their property procedures, and behavioral methods. In this assignment you will create one User Interface Class (as you have been doing so far from the start of the semester) and one or two Business Logic Classes for each project.

Do the following programming projects, using Visual Basic .NET, using the version described in the course syllabus.

As always, create Form_Load() and Exit_Click() event procedures, and use _initializeBusinessLogic() and _initializeUserInterface() methods on the main form. Again, as always, include tool-tips, and default and cancel buttons for the form, and validate any user input that might be able to cause the program to crash.

Also, as always, create standard get/set property procedures and a ToString() method for each Business Logic Class, and, as we have discussed in lecture, create and use Business Logic behavioral methods to encapsulate “transactions” that need to be carried out by the programs. Use pass-by-value and return liberally to communicate between methods and objects.

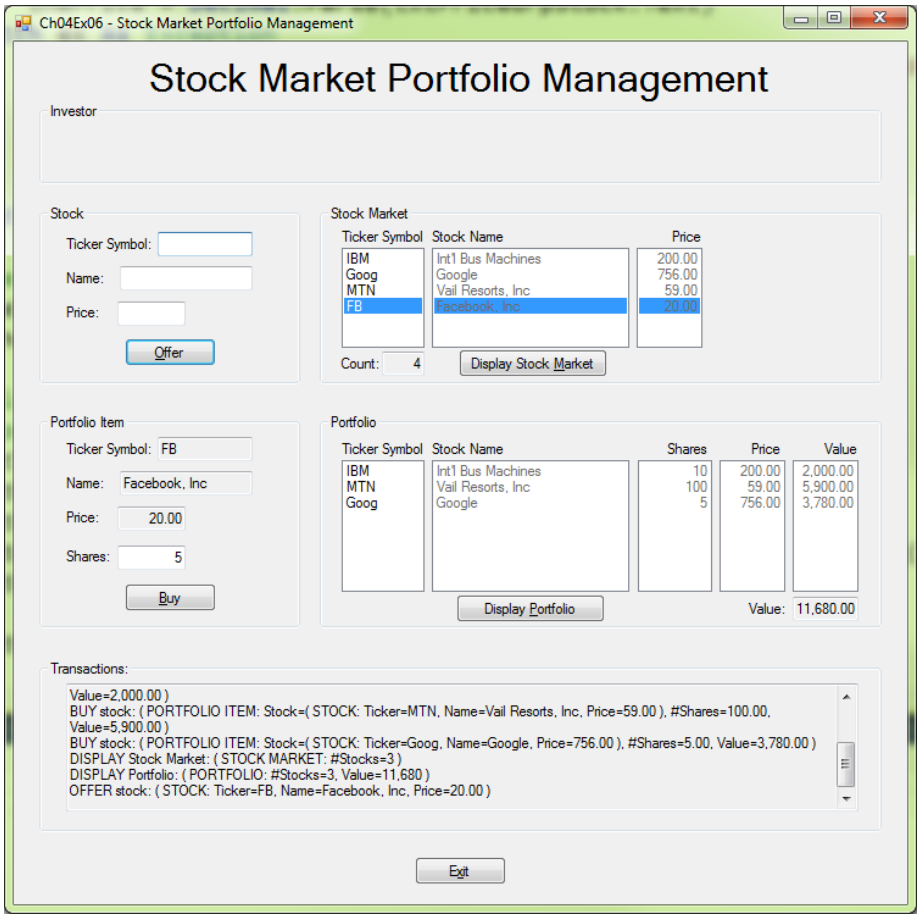
Do not use user-interface controls (TextBoxes, etc.) to update or keep track of running totals. Use module-level variables or object attributes to do this, and simply display the information stored in them on the user interface when needed.

Do this as an "individual" assignment.

Classes – Constructing a Second Time Around.

Project *Ch04Ex02*: Stock Market.

In this project you will write a program that simulates a Stock Market Portfolio Management system – a program that might let an investor keep track of stocks in the market, and buy shares of these stocks for their portfolio of stocks. Each stock for which the investor owns shares will be tracked and referred to as a portfolio item. Below is one possible main form that supports this:



This program should allow the user to do the following things (we'll call each of these a "transaction"):

a. Offer stocks.
    1) Enter the Ticker Symbol, Name, and price.
    2) Track and display these on the main form in parallel ListBoxes in a Stock Market section of the form.
b. Buy shares in any of the stocks offered.
    1) Select the stock to buy from those displayed in the parallel ListBoxes.
    2) Indicate how many shares to buy.
    3) Record the buying of the selected stock and display this portfolio item information in parallel ListBoxes in a Portfolio section of the form.
    4) We will assume that once the investor buys stock s/he will never sell it, and will never try to buy more of that same stock. Clearly this is not the way it works in the real world, but it will greatly simplify our program for now!
c. Display information about the stock market.
d. Display information about the investor's portfolio.

For each of these things, a scrolling TextBox "log" on the form should display the pertinent information about each "transaction".

Because we do not yet have all the tools necessary, this program will not fully track all information in "business logic" classes, though it will make a good start. Some of it will only be tracked by what we see on the main form. You should have a single main form, and four business logic classes – Stock, StockMarket, PortfolioItem, and Portfolio. The following gives you some guidance for these classes.

Stock class

Attributes

Ticker Symbol (e.g. "IBM", "Goog", "FB", etc.)

Name (e.g. "International Business Machines", "Google, Inc", "Facebook, Inc", etc.)

Price (e.g. 200.00, 756.00, 19.75, etc.)

Constructors

A special constructor with 3 parameters: Ticker Symbol, Name, and Price.

Property Procedures

One regular private property procedure for each attribute. Use these to actually interact with the underlying attributes, and use these anywhere within this class that the attributes must be accessed/updated.

One read-only public property procedure for each attribute. Use the private property procedures to actually interact with the underlying attributes.

Behavioral Methods

Standard ToString() methods.

StockMarket class

Attributes

Number of different Stocks listed in the Market. (e.g. 5, 1200, 1372, etc.)

Constructors

No constructors will be created for this class. The "for free" default constructor will be used.

Property Procedures

One regular private property procedure for each attribute. Use these to actually interact with the underlying attributes, and use these anywhere within this class that the attributes must be accessed/updated.

One read-only public property procedure for each attribute. Use the private property procedures to actually interact with the underlying attributes.

Behavioral Methods

Public Sub offerStock(ByVal pStock As Stock) and corresponding private method.

This method will simulate the offering of a new Stock on the StockMarket. Since we don't have enough tools to do more at this point, it will simply keep track of how many

stocks are in the market. (e.g. Each time a new stock is offered, it will add one to the count.)

Standard ToString() methods.

PortfolioItem class

Attributes

A reference to the Stock that this portfolio item is for. (You will store a reference, not a Ticker Symbol, not a Name, etc.)

The number of shares the investor owns of this Stock.

Constructors

A special constructor with 1 parameter: a reference to a Stock. Using this constructor will allow an investor to set up a portfolio item for the specified Stock, but without yet having purchased any shares. They will actually purchase shares with the buy() behavioral method described below.

A special constructor with 2 parameters: the number of shares to purchase, and a reference to the Stock to purchase. Using this constructor will allow an investor to set up a PortfolioItem for the specified Stock, and at the same time to indicate the number of shares they want of this Stock. They will not have to subsequently buy() the shares they want. They will do it all in a single step, when this constructor is used.

Property Procedures

One regular private property procedure for each attribute. Use these to actually interact with the underlying attributes, and use these anywhere within this class that the attributes must be accessed/updated.

One private property procedure that calculates and "returns" the value of the PortfolioItem. This is calculated by taking the price of the Stock and multiplying it by the number of shares owned. This information can be accessed through the standard private property procedures.

One read-only public property procedure for each attribute. Use the private property procedures to actually interact with the underlying attributes.

One read-only public property procedure for the value of the PortfolioItem. Use the corresponding private property procedure to actually perform the calculation.

Behavioral Methods

Public Sub buy(ByVal pShares as Integer) and corresponding private method.

This method will simulate the buying of the specified number of shares of the Stock that this PortfolioItem references. It will simply add the number of shares specified to the number already owned.

Standard ToString() methods.

Portfolio class

Attributes

Number of different holdings (PortfolioItem's) the investor owns. (e.g. 1, 12, 72, etc.)

Total value of the stocks the investor owns. (e.g. 10,293.00, 100,000.75, 2,357,976.00, etc.)

Constructors

No constructors will be created for this class. The "for free" default constructor will be used.

Property Procedures

One regular private property procedure for each attribute. Use these to actually interact with the underlying attributes, and use these anywhere within this class that the attributes must be accessed/updated.

One read-only public property procedure for each attribute. Use the private property procedures to actually interact with the underlying attributes.

Behavioral Methods

Public Sub buy(ByVal pShares as Integer, ByVal pStock As Stock) and corresponding private method.

This method will simulate the buying of the specified number of shares of the Stock that the investor is adding to their Portfolio. It will simply add the value of the PortfolioItem (price of the Stock multiplied by number of shares) to the total value already accumulated in the Portfolio.

Standard ToString() methods.

For this program you will need to use a number of "parallel" ListBoxes. The following explains this concept and gives some sample code to help when working with them.

To add a String (description) to a ListBox (lstDescription):

```
lstDescription.Items.Add(description)
```

To add a Decimal (cost) to a ListBox (lstCost):

```
lstCost.Items.Add(cost.ToString("N2"))
```

To make these "parallel" you need to make sure that you add information to all "parallel" lists at the same time. For instance, if you want to add a Description of an item and its Cost, then you would have two lst.Items.Add() method calls to add to both lists at the same time. See commands for doing this shown above.

In order to make these lists easier for the user to interact with, it is sometimes helpful to make all but the "key" list disabled (Enabled = False), and only let the user make selections from the "key" list. For instance, if the Description list is "key", then it would be enabled but the Cost list would be disabled. But when this is done then the Cost cannot be selected by the user, so you want to create code that will automatically select the "parallel" list(s) item(s) when the "key" list item is selected. The following code illustrates how to do this:

```
Private Sub _lstDescription_SelectedIndexChanged( _
        sender As System.Object, _
        e As System.EventArgs) _
    Handles lstDescription.SelectedIndexChanged

  Dim selectedIndex As Integer =
      lstDescription.SelectedIndex

  lstNameGrpStockMarket.SelectedIndex = selectedIndex
  lstPriceGrpStockMarket.SelectedIndex = selectedIndex

End Sub '_lstTickerGrpStockMarket_SelectedIndexChanged(sender,e)
```

Finally, sometimes you want to retrieve information from the list(s), based on which item is selected. The following code illustrates how to do this:

```
txtDescription.Text = _
   lstDescription.SelectedItem.ToString
```

When buying a Stock, allow the user to select from the StockMarket parallel lists to indicate which Stock they wish to purchase. Display the ticker symbol, name, and price from this selection in the PortfolioItem area, "buy" the number of shares of this Stock that the user specifies in the PorfolioItem area.

Make sure the user-interface is nicely and cleanly designed, following practices we have been discussing this semester, and make sure the business logic classes are cleanly designed as described in this assignment and using the "best practices" we have been discussing in lectures, examples, help sessions, slides, and the textbook.

Creation and Submission of Programs

Creation and submission are as have been described earlier in the semester, and grading will be similar to how it was done on earlier assignments.