CSC 130
Programming Project #3

Due Date: ***by 11:55pm*** **on Monday, November 25, 2013**

Objectives:

- To design and implement multiple classes
- To practice using the array-based implementation of an indexed list
- To practice reading data from a file

Problem:

A friend has hired you to keep track of their stock portfolio. You will need to keep track of the stocks purchased and their purchase price as well as the worth of the portfolio over time as stocks are bought and sold. Stock purchases and sales will be input from a text file and processed in a first-in, first-out manner.

Specifications:

Your project (*project3*) should contain all classes, including exception classes (i.e. you will not need to link to other projects and include import statements.) The following is excerpted from our textbook, *Data Structures and Algorithms in Java*, and can be found on page 235. You should note that the description in the textbook refers to an input sequence that is different than the one you are required to use. I've included below the portion of the specification that applies to this project.

> When a share of common stock of some company is sold, the capital gain (or, sometimes, loss) is the difference between the shares's selling price and the price originally paid to buy it. This rule is easy to understand for a single share, but if we sell multiple shares of stock bought over a long period of time, then we must identify the shares actually being sold. A standard accounting principle for identifying which shares of a stock were sold in such a case is to use a FIFO protocol—the shares sold are the ones that have been held the longest (indeed, this is the default method built into several personal finance software packages). For example, suppose we buy 100 shares at \$20 each on day 1, 20 shares at \$24 on day 2, 200 shares at \$36 on day 3, and then sell 150 shares on day 4 at \$30 each. Then applying the FIFO protocol means that of the 150 shares sold, 100 were bought on day 1, 20 were bought on day 2, and 30 were bought on day 3. The capital gain in this case would therefore be $100\cdot10+20\cdot6+30\cdot(-6)$, or \$940.

Input Files:

There are two input files (available on Blackboard). The structure for each is outlined below.

symboldata.txt – each line of the file will have the following format:
CSCO Cisco Systems, Inc.
BAC Bank of America Corporation
…

Note that the ticker symbol, which appears first, does not contain spaces, but the company name does.

stockdata.txt – each line of the file will have the following format:

```
b 100 42.47 MSFT
b 50 38.56 GM
…
b 100 21.06 CSCO
s 150 40.15 MSFT
b 150 3.12 SIRI
```

The first character indicates whether the stock is being bought or sold.
The integer that follows indicates the number of shares being bought/sold.
The double indicates the price at which the stock was bought/sold.
The string is the ticker symbol for the stock.

UML Class Diagrams:

You will be utilizing the ArrayIndexList and IndexOutOfBoundsException classes developed in class/lab. You may find it helpful to include an *add* method which will store the item passed to it in the next available position. The other classes that need to be developed are outlined below.

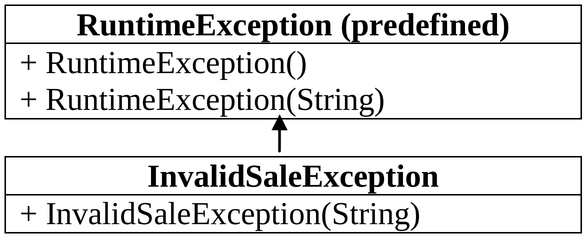
SymbolPair
- tickerSym : String
- companyName : String
+ SymbolPair(String, String)
+ getTickerSymbol() : String
+ getCompanyName() : String

SymbolTable
- symbolPairs : ArrayIndexList<SymbolPair>
+ SymbolTable(File)
+ findCompany(String) : String

Stock
- tickerSym : String
- sharesOwned : int
- purchasePrice : double
+ Stock(int, double, String)
+ getTickerSymbol() : String
+ getSharesOwned() : int
+ getPurchasePrice() : double
+ setSharesOwned(int) : void
+ toString() : String

Portfolio
- stocks : ArrayIndexList<Stock>
- symbols : SymbolTable
- worth : double
+ Portfolio(SymbolTable)
+ processTransaction(char, int, double, String) : void

+ toString() : String



RuntimeException (predefined)
+ RuntimeException()
+ RuntimeException(String)
InvalidSaleException
+ InvalidSaleException(String)

The Application Class:

The application class is responsible for creating a SymbolTable object, creating a Portfolio object, and then processing the transactions in *stockdata.txt*.

Output:

For each transaction processed you should display the portfolio before the transaction, followed by information describing the transaction and finally the portfolio after the transaction. All output should be displayed in the console and should be properly labeled. You should display the full company name, not the ticker symbol.

Invalid Input:

If there is an attempt to sell more stock than is currently contained in the portfolio, an error message should be displayed instead of the transaction description. Processing should continue with the next transaction.

Plagiarism Policy:

This assignment is to be completed individually (not with your pair programming partner). If you receive substantial help from another person you must give them credit for their contribution in a comment. It is not considered acceptable for you to share your code with another student in any way. You may discuss an approach from a conceptual perspective only.

General Requirements:

- You should spend some time making your output as neat as possible. **The overall appearance of the output will be graded.**
- Neatness Counts!!! Proper indentation is required.
- Be sure to select meaningful variable names.
- Include plenty of comments. The comments must be formatted according to **JavaDoc** standards. **Documentation will be considered when grading this assignment!**

Submit:

- **Create a .zip file to upload to Blackboard:**
    - **Right-click the project folder, choose Send To | Compressed (zipped) Folder**
- **Upload the .zip file to the Project #3 assignment in Blackboard by 11:55pm on Monday, November 25th.**

Late Assignments:
Assignments received by 11:55pm on Tuesday, November 26th will receive a maximum of 90 points.

Assignments received by 11:55pm on Wednesday, November 27th will receive a maximum of 80 points.
Assignments will not be accepted after this date/time for any reason.