Assignment 2

Playing the Market

Objective:
Practice using stacks and queues.

Program Description:
When a share of common stock of some company is sold, the capital gain (or, sometimes, loss) is the difference between the share’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 indentifying 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 each on day 2
- 200 shares at $36 each on day 3

then sell:

- 150 shares at $30 each on day 4.
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

\[ \text{capital gain} = 100 \times 10 + 20 \times 6 + 30 \times (-6) = 940. \]

Or

\[ 100 \times (30-20) + 20 \times (30-24) + 30 \times (30-36) \]

Write a program with the following four main menu options:

1. Buy
2. Sell
3. Total Capital Gain So Far
4. Quit

If option 1 is chosen, the user should be prompted to enter how many shares they wish to buy and at what price. These shares should be added to the queue of shares currently held, and the program should output a confirmation of the purchase. It should then again prompt the user with the main menu.

If option 2 is chosen, the user should be prompted to enter how many shares they wish to sell and at what price. These shares should be removed from the queue of shares currently held, the capital gain/loss should be updated, and the program should output a confirmation of the sale. It should then prompt the user with the main menu again.

If option 3 is chosen, the total capital gain (or loss) from all transactions so far should be displayed. Then the user should be prompted with the main menu again.

If option 4 is chosen, the program should end.

To achieve this, your code should be organized into three classes.

1. The first is a standard Queue class (as discussed in class and in readings). You may choose whether you wish to implement the queue as an array or a singly linked list. You can test that your Queue operations are working properly from a main test method within the Queue class.
2. The second is a class that will handle and record all the transactions...

1. The CapGain class.
   
   - Fields:
     1. double totCapGain : the current total capital gain or loss in dollars
     2. Queue sharesHeld : a queue of the purchase prices of all shares currently held
   
   - Methods:
     1. void mainMenuPrompt() : print out the main menu
     2. void buy(int numShares, double price) : add numShares shares at price price to the queue
     3. void sell(int numShares, double price) : remove numShares shares from the queue, calculating gain/loss for each share (based on the price it was purchased for and the price it is being sold for), and updating the totCapGain.
     4. double getTotalCapGain() : return totCapGain.
3. And finally, a class for the main method that will drive the whole program (input/output)...

- The CapGainApp class. a main method that will do something along the lines of:
  
  create new CapGain object called cg
  create new Scanner object for I/O /*see Part A for more on Scanners*/
  do { /*yes this “do-while” loop is an actual Java loop construct you can use*/
    cg.mainMenuPrompt();
    get answer from user
    if answer is 1 {
      get numShares from user
      get price from user
      call cg.buy(numShares, price)
    } else if answer is 2 {
      ...
    } else if answer is 3 {
      nicely output cg.getTotalCapGain()
    } else {
      output farewell message
    }
  } while (user choice is between 1 and 3)

**note**: it is better to use iteration (as outlined above), not recursion, to keep bringing up the main menu until the user quits