Lab Assignment 3

Jerry Belmonte
Keira Wong

CODE (x2 files)

Stock.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace StockLibrary
{
    public class Stock // PUBLISHER
    {
        // The .NET Framework defines a generic delegate called System.EventHandler<> that satisfies these rules:
        // public delegate void EventHandler<TEventArgs>(object source, TEventArgs e) where TEventArgs : EventArgs;
        // The "StockValueChanged" event is associated w/ the EventHandler delegate & is raised in a method, "OnStockValueChanged()"
        public event EventHandler<StockNotification> StockValueChanged; // (built-in Event-Handler Delegate ("EventHandler<StockNotification>") in place of creating a delegate explicitly)

        private readonly Thread _thread;

        public string StockName { get; set; }
        public int InitialValue { get; set; }
        public int CurrentValue { get; set; }
        public int MaxChange { get; set; }
        public int Threshold { get; set; }
        public int NumChanges { get; set; }

        public Stock(string name, int startingValue, int maxChange, int threshold)
        {
            StockName = name;
            InitialValue = startingValue;
            CurrentValue = InitialValue;
            MaxChange = maxChange;
            Threshold = threshold;
            NumChanges = 0;
```

```
            // (When a stock object is created, a thread is started.)
            _thread = new Thread(() => Activate());
            _thread.Start();
        }

        public void Activate()
        {
            int halfSecond = 500; // 1/2 second / (Every 500 milliseconds...)

            for (int i = 0; i < 25; i++)
            {
                Thread.Sleep(halfSecond);
                ChangeStockValue(); // (This stock's value is modified...)
            } // (& repeats 25 times after the stock object is created.)
        }

        public void ChangeStockValue() // StartProcess()
        {
            var rand = new Random(); // initialize a random number generator that will modify the stock value
            CurrentValue += rand.Next(MaxChange); // the range within a stock can change every time unit
            NumChanges++; // there will be a total of 25 changes to the stock from the Activate() method

            // Only raise the event of a stock value change when the change is above the threshold value
            if ((CurrentValue - InitialValue) > Threshold)
            {
                // Fire a StockNotification event with the name of the stock, current stock value, and the number
of changes.
                OnStockValueChanged(new StockNotification(StockName, CurrentValue, NumChanges));
            }
        }

        // The pattern requires that you write a protected virtual method that fires the event.
        // The name must match the name of the event, prefixed with the word "On",
        // and then accept a single EventArgs argument.
        protected virtual void OnStockValueChanged(StockNotification sn) {
            // that is why in the rand.Next() method, I opted for the single parameter function which is in the
range 0 to MaxChange.
            /* "StockValueChanged?.Invoke(this, sn);" is the same as the following code... ----------------------

            EventHandler handler = StockValueChanged;
            if (handler != null) // ("StockValueChanged?.": checks to make sure at least 1 listener is registered
to that event, "StockValueChanged")
            {
            handler(this, sn); // ("Invoke(this, sn)": "raises" the event ("StockValueChanged") ) / (Which stock?
-> "this" stock / What changed? -> "sn")
            }

            */ // https://stackoverflow.com/questions/12217632/calling-an-event-handler-in-c-sharp
---------------

            // ( passes the source of the event & the event's data ("sn"'s data) to be processed by the
EventHandler )
            // notifies all listeners who have registered w/ this StockNotification event, "StockValueChanged"
            StockValueChanged?.Invoke(this, sn);
```

```
            // To work robustly in multithreading scenerios, you need to assign the delegate to a temporary
variable before testing and invoking it:
            // var temp = StockValueChanged; if (temp != null) temp (this, sn);
            // We can achieve the same functionality without the temp variable with the null-conditional
operator:
            // StockValueChanged?Invoke(this, sn);
            // Being both thread-safe and succinct, this is the best general way to invoke events.

            //([sender].[OnEventOccurred] += [] to subscribe, -= to unsubscribe)
        }
    }
}
```

StockNotification.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace StockLibrary
{
    // KKKKK Custom EventArgs Class:
    // a predefined Framework class with no members (other than the static Empty property).
    // EventArgs is a base class for conveying information for an event.
    // We are subclassing EventArgs to convey a stock price change event being fired.
    public class StockNotification : EventArgs
    {
        // KKKKK Allows for 1+ data values to be passed to the "On_____()"
        public string StockName { get; set; }
        public int CurrentValue { get; set; }
        public int NumChanges { get; set; }


        /// <summary>
        ///     Stock notification attributes that are set and changed
        /// </summary>
        /// <param name="stockName">Name of stock</param>
        /// <param name="currentValue">Current value of the stock</param>
        /// <param name="numberChanges">Number of changes the stock goes through</param>
        public StockNotification(string stockName, int currentValue, int numberChanges)
        {
            StockName = stockName;
            CurrentValue = currentValue;
            NumChanges = numberChanges;
        }
    }
}
```

StockBroker.cs

```
using System;
```

```
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.Threading;

namespace StockLibrary
{
   public class StockBroker // SUBSCRIBER
   {
      public string BrokerName { get; set; }

      public List<Stock> stocks = new List<Stock>();

      // Represents a lock that is used to manage access to a resource, allows one thread to be
      // in write mode with exclusive ownership of the lock.
      public static ReaderWriterLockSlim myLock = new ReaderWriterLockSlim();
      readonly string docPath = @"C:\Users\keira\Desktop\Spring 2020\CECS 475\Labs\Lab Assignment
#3\output.txt";
      public string titles = "Broker".PadRight(10)
                           + "Stock".PadRight(15)
                           + "Value".PadRight(10)
                           + "Changes".PadRight(10)
                           + "Date and Time";

      public StockBroker(string brokerName)
      {
         BrokerName = brokerName;
      }

      public void AddStock(Stock stock)
      {
         stocks.Add(stock);

         // registers StockBroker as a subscriber to the event, "StockValueChanged"
         // to be notified whenever the value of the stock being added is changed.
         // set this Stock's EventHandler delegate to this StockBroker's EventHandler
         stock.StockValueChanged += stock_StockValueChanged;
         // publisher.EventInterestedIn += subscriber's EventHandler
      }

      // Event Handler
      void stock_StockValueChanged(Object sender, StockNotification sn) // REVIEW (The "Notify"
Method?), (EventArgs e -> StockNotification sn?)
      {
         myLock.EnterWriteLock(); // The EventerWriteLock method is used to enter the lock in write mode.
         // When a thread is in write mode, no other thread can enter the lock in any mode.
         try
         {
            Stock newStock = (Stock) sender;
            string statement = BrokerName.PadRight(10);
            statement += sn.StockName.PadRight(15);
            statement += sn.CurrentValue.ToString().PadRight(10);
            statement += sn.NumChanges.ToString().PadRight(10);
            statement += DateTime.Now.ToString();

            // Create the file to write to with the titles if it does not exist yet.
```

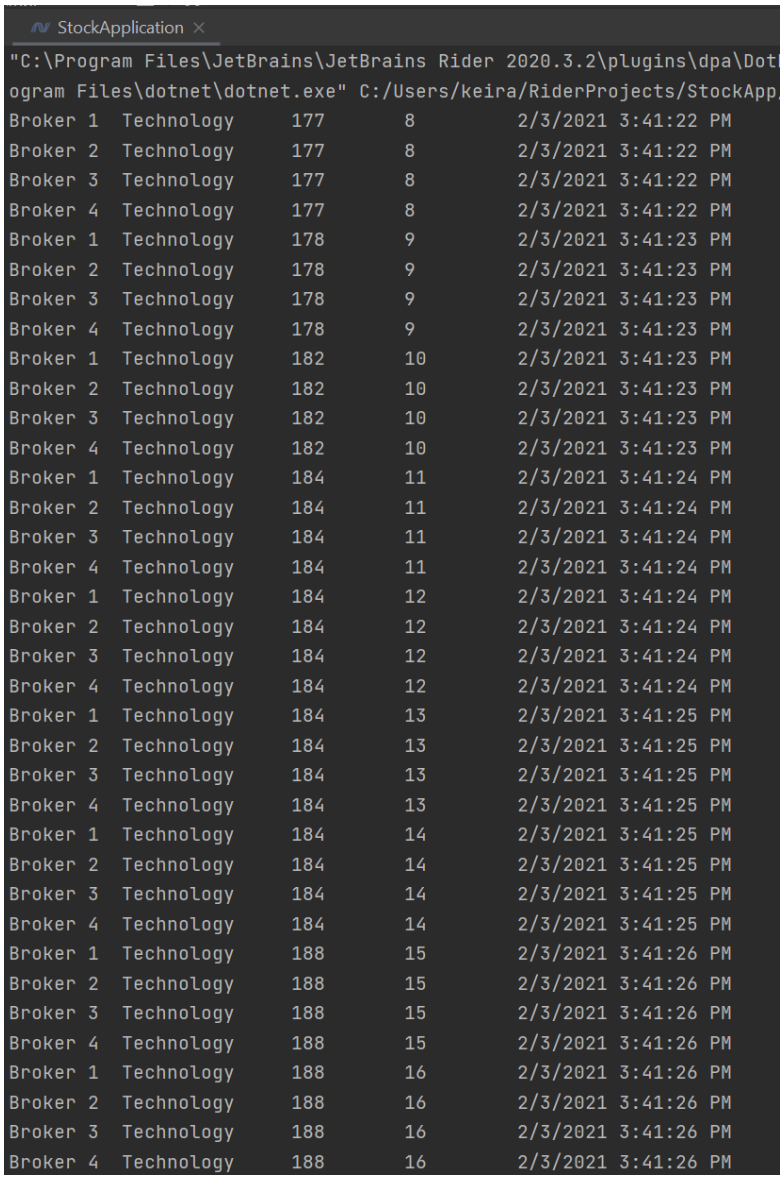
```
            if (!File.Exists(docPath))
            {
                using (StreamWriter sw = File.CreateText(docPath))
                {
                    sw.WriteLine(titles);
                    Console.WriteLine(titles);
                }
            }

            // Writes to a text file the statement message received from the Publisher.
            using (StreamWriter outputFile = new StreamWriter(docPath, true))
            {
                outputFile.WriteLine(statement);
                Console.WriteLine(statement);
            }
        }
        finally
        {
            myLock.ExitWriteLock();
        }
    }
  }
}
```

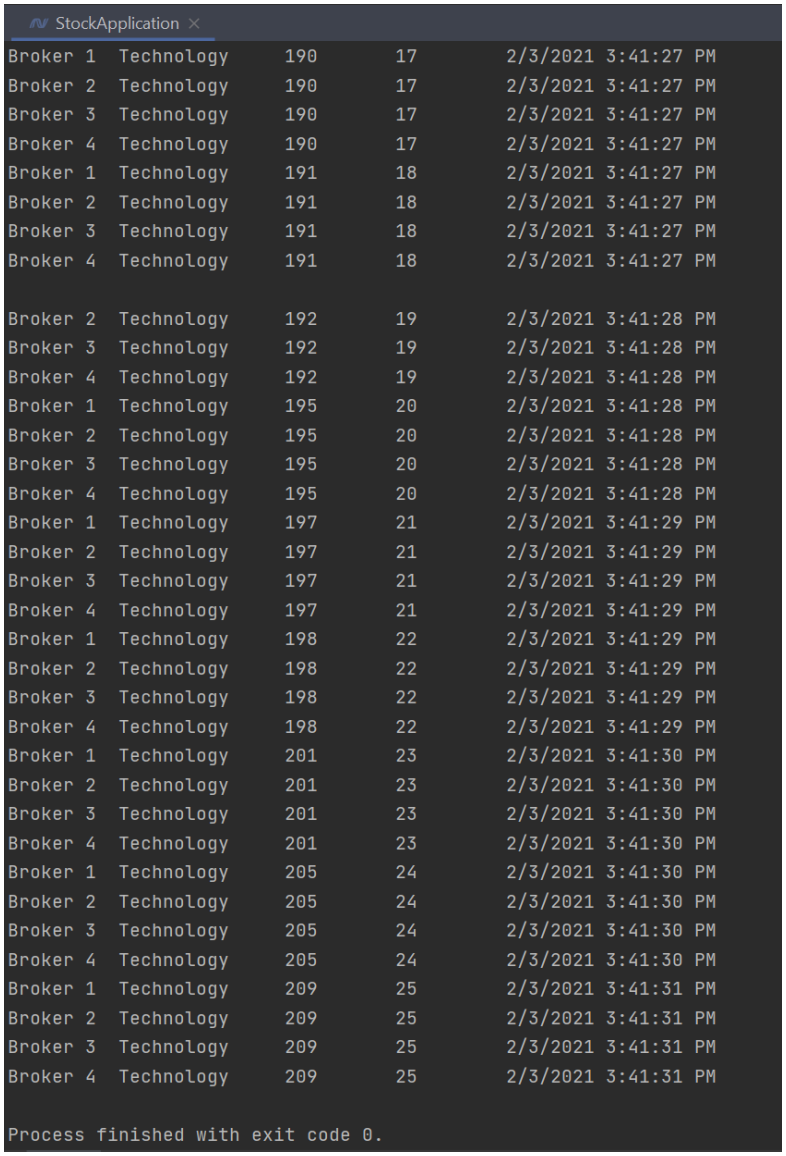
Program.cs

```
using System;
using StockLibrary;

namespace StockApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      Stock stock1 = new Stock("Technology", 160, 5, 15);
      Stock stock2 = new Stock("Retail", 30, 2, 6);
      Stock stock3 = new Stock("Banking", 90, 4, 10);
      Stock stock4 = new Stock("Commodity", 500, 20, 50);

      StockBroker b1 = new StockBroker("Broker 1");
      b1.AddStock(stock1);
      b1.AddStock(stock2);

      StockBroker b2 = new StockBroker("Broker 2");
      b2.AddStock(stock1);
      b2.AddStock(stock3);
      b2.AddStock(stock4);

      StockBroker b3 = new StockBroker("Broker 3");
      b3.AddStock(stock1);
      b3.AddStock(stock3);

      StockBroker b4 = new StockBroker("Broker 4");
```

```
            b4.AddStock(stock1);
            b4.AddStock(stock2);
            b4.AddStock(stock3);
            b4.AddStock(stock4);
        }
    }
}
```

OUTPUT


StockApplication
"C:\Program Files\JetBrains\JetBrains Rider 2020.3.2\plugins\dpa\Dot
ogram Files\dotnet\dotnet.exe" C:/Users/keira/RiderProjects/StockApp
Broker 1 Technology 177 8 2/3/2021 3:41:22 PM
Broker 2 Technology 177 8 2/3/2021 3:41:22 PM
Broker 3 Technology 177 8 2/3/2021 3:41:22 PM
Broker 4 Technology 177 8 2/3/2021 3:41:22 PM
Broker 1 Technology 178 9 2/3/2021 3:41:23 PM
Broker 2 Technology 178 9 2/3/2021 3:41:23 PM
Broker 3 Technology 178 9 2/3/2021 3:41:23 PM
Broker 4 Technology 178 9 2/3/2021 3:41:23 PM
Broker 1 Technology 182 10 2/3/2021 3:41:23 PM
Broker 2 Technology 182 10 2/3/2021 3:41:23 PM
Broker 3 Technology 182 10 2/3/2021 3:41:23 PM
Broker 4 Technology 182 10 2/3/2021 3:41:23 PM
Broker 1 Technology 184 11 2/3/2021 3:41:24 PM
Broker 2 Technology 184 11 2/3/2021 3:41:24 PM
Broker 3 Technology 184 11 2/3/2021 3:41:24 PM
Broker 4 Technology 184 11 2/3/2021 3:41:24 PM
Broker 1 Technology 184 12 2/3/2021 3:41:24 PM
Broker 2 Technology 184 12 2/3/2021 3:41:24 PM
Broker 3 Technology 184 12 2/3/2021 3:41:24 PM
Broker 4 Technology 184 12 2/3/2021 3:41:24 PM
Broker 1 Technology 184 13 2/3/2021 3:41:25 PM
Broker 2 Technology 184 13 2/3/2021 3:41:25 PM
Broker 3 Technology 184 13 2/3/2021 3:41:25 PM
Broker 4 Technology 184 13 2/3/2021 3:41:25 PM
Broker 1 Technology 184 14 2/3/2021 3:41:25 PM
Broker 2 Technology 184 14 2/3/2021 3:41:25 PM
Broker 3 Technology 184 14 2/3/2021 3:41:25 PM
Broker 4 Technology 184 14 2/3/2021 3:41:25 PM
Broker 1 Technology 188 15 2/3/2021 3:41:26 PM
Broker 2 Technology 188 15 2/3/2021 3:41:26 PM
Broker 3 Technology 188 15 2/3/2021 3:41:26 PM
Broker 4 Technology 188 15 2/3/2021 3:41:26 PM
Broker 1 Technology 188 16 2/3/2021 3:41:26 PM
Broker 2 Technology 188 16 2/3/2021 3:41:26 PM
Broker 3 Technology 188 16 2/3/2021 3:41:26 PM
Broker 4 Technology 188 16 2/3/2021 3:41:26 PM


StockApplication
Broker 1 Technology 190 17 2/3/2021 3:41:27 PM
Broker 2 Technology 190 17 2/3/2021 3:41:27 PM
Broker 3 Technology 190 17 2/3/2021 3:41:27 PM
Broker 4 Technology 190 17 2/3/2021 3:41:27 PM
Broker 1 Technology 191 18 2/3/2021 3:41:27 PM
Broker 2 Technology 191 18 2/3/2021 3:41:27 PM
Broker 3 Technology 191 18 2/3/2021 3:41:27 PM
Broker 4 Technology 191 18 2/3/2021 3:41:27 PM
Broker 2 Technology 192 19 2/3/2021 3:41:28 PM
Broker 3 Technology 192 19 2/3/2021 3:41:28 PM
Broker 4 Technology 192 19 2/3/2021 3:41:28 PM
Broker 1 Technology 195 20 2/3/2021 3:41:28 PM
Broker 2 Technology 195 20 2/3/2021 3:41:28 PM
Broker 3 Technology 195 20 2/3/2021 3:41:28 PM
Broker 4 Technology 195 20 2/3/2021 3:41:28 PM
Broker 1 Technology 197 21 2/3/2021 3:41:29 PM
Broker 2 Technology 197 21 2/3/2021 3:41:29 PM
Broker 3 Technology 197 21 2/3/2021 3:41:29 PM
Broker 4 Technology 197 21 2/3/2021 3:41:29 PM
Broker 1 Technology 198 22 2/3/2021 3:41:29 PM
Broker 2 Technology 198 22 2/3/2021 3:41:29 PM
Broker 3 Technology 198 22 2/3/2021 3:41:29 PM
Broker 4 Technology 198 22 2/3/2021 3:41:29 PM
Broker 1 Technology 201 23 2/3/2021 3:41:30 PM
Broker 2 Technology 201 23 2/3/2021 3:41:30 PM
Broker 3 Technology 201 23 2/3/2021 3:41:30 PM
Broker 4 Technology 201 23 2/3/2021 3:41:30 PM
Broker 1 Technology 205 24 2/3/2021 3:41:30 PM
Broker 2 Technology 205 24 2/3/2021 3:41:30 PM
Broker 3 Technology 205 24 2/3/2021 3:41:30 PM
Broker 4 Technology 205 24 2/3/2021 3:41:30 PM
Broker 1 Technology 209 25 2/3/2021 3:41:31 PM
Broker 2 Technology 209 25 2/3/2021 3:41:31 PM
Broker 3 Technology 209 25 2/3/2021 3:41:31 PM
Broker 4 Technology 209 25 2/3/2021 3:41:31 PM
Process finished with exit code 0.

TEAM MEMBER’S WORK

Jerry Belmonte

- Threading & Reading/Writing to the File

Keira Wong

- Events