Threading

Stock.cs

```
    public class Stock
   {
      public event EventHandler<StockNotification> StockEvent;
      //public event StockNotify ProcessComplete;

      //Name of our stock.
      private string _name;
      //Starting value of the stock.
      private int _initialValue;
      //Max change of the stock that is possible.
      private int _maxChange;
      //Threshold value where we notify subscribers to the event.
      private int _threshold;
      //Amount of changes the stock goes through.
      private int _numChanges;
      //Current value of the stock.
      private int _currentValue;

      private readonly Thread _thread;
      public string StockName { get => _name; set => _name = value; }
      public int InitialValue { get => _initialValue; set => _initialValue = value;
}
      public int CurrentValue { get => _currentValue; set => _currentValue =
value; }
      public int MaxChange { get => _maxChange; set => _maxChange =
value; }
      public int Threshold { get => _threshold; set => _threshold = value; }
      public int NumChanges { get => _numChanges; set => _numChanges =
value; }
```

```
    public Stock(string name, int startingValue, int maxChange, int
threshold)
    {
        _name = name;
        _initialValue = startingValue;
        _currentValue = InitialValue;
        _maxChange = maxChange;
        _threshold = threshold;
        _thread = new Thread(new ThreadStart(Activate));
        _thread.Start();
    }

    public void Activate()
    {
        for (int i = 0; i < 25; i++)
        {
            Thread.Sleep(500);
            ChangeStockValue();
        }
    }

    public void ChangeStockValue()
    {
        var rand = new Random();
        CurrentValue += rand.Next(1, MaxChange);
        NumChanges++;
        if ((CurrentValue - InitialValue) > Threshold)
        {

            StockEvent?.Invoke(this, new StockNotification(StockName,
CurrentValue, NumChanges));
            //included process complete delegate
            //how to use delegate and event?
            //ProcessComplete?.Invoke(StockName, CurrentValue,
NumChanges);
        }
    }
```

```
}
```

StockBroker.cs

```
//threading StockBroker
public class StockBroker
{
    public string BrokerName { get; set; }

    //list to store broker's stocks
    public List<Stock> stocks = new List<Stock>();

    //create lock for writing to files
    public static ReaderWriterLockSlim myLock = new
ReaderWriterLockSlim();

    //txt file path
    readonly string destPath =
System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory,
"Lab1_Threading_Output.txt");
    //stock headers
    string titles = "Broker".PadRight(16) + "Stock".PadRight(16) +
"Value".PadRight(16) + "Changes".PadRight(10) + "Date and Time";

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

    //associate stock with broker
    public void AddStock(Stock stock)
```

```
    {
        stocks.Add(stock);

        //subscribe to stock's event using EventHandler
        stock.StockEvent += EventHandler;
    }

    //handles writing stockNotifications
    void EventHandler(Object sender, StockNotification e)
    {
        try
        {
            //enter the lock
            myLock.EnterWriteLock();

            //create newStock
            Stock newStock = (Stock)sender;

            //write stock info to console
            Console.WriteLine(BrokerName.PadRight(16) +
e.StockName.PadRight(16) + e.CurrentValue.ToString().PadRight(16) +
e.NumChanges.ToString().PadRight(10) + DateTime.Now.ToString());

            //write stock info to txt file
            using (StreamWriter outputFile = new StreamWriter(destPath,
true))
            {
                outputFile.WriteLine(BrokerName.PadRight(16) +
e.StockName.PadRight(16) + e.CurrentValue.ToString().PadRight(16) +
e.NumChanges.ToString().PadRight(10) + DateTime.Now.ToString());
            }
        }
        finally
        {
            //exit the lock.
            myLock.ExitWriteLock();
        }
```

```
    }
}
```