ASYNC

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

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

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

    //dest path for txt file
```

```
    readonly string destPath =
System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDi
rectory, "Lab1_Async_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 async EventHandler
        stock.StockEvent += async (sender, e) => await
AsyncEventHandler(sender, e);
    }

    //async event handler Stock notification
    public async Task AsyncEventHandler(object sender,
StockNotification e)
    {
        Stock newStock = (Stock)sender;
```

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

        //call async write to txt file
        await WriteToFileAsync(BrokerName.PadRight(16) +
e.StockName.PadRight(16) +
e.CurrentValue.ToString().PadRight(16) +
e.NumChanges.ToString().PadRight(10) +
DateTime.Now.ToString());
    }

    //async method to write to txt file (continuously retries if
encounters exception)
    public async Task WriteToFileAsync(string line)
    {
        while (true)
        {
            try
            {
                using (StreamWriter outputFile = new
StreamWriter(destPath, true))
                {
                    await outputFile.WriteLineAsync(line);
                    //successfully wrote to file => exit the loop
                    return;
```

```
            }
        }
        catch
        {
            //sleep for 100 millisecs then continue retrying
            System.Threading.Thread.Sleep(100);
        }
    }
  }
}
```