Exercise 5

Setup

- RAM: `16GB`
- CPU: `4x2.3GHz`
- OS: `Windows 10 Home`
- IDE: `Visual Studio 2022`

Stock Data Visualization

Sequential Implementation

The code-snippet below shows a parallel implementation of the data retrieval. Using the `.ContinueWith()` function one can define a sequence with which the tasks should be executed.

```
private void TaskImplementation() {
    var tasks = names.Select( // for each stock in list (start of async/parallel)
        name => RetrieveStockDataAsync(name) // retrieve data from service
            .ContinueWith((taskObj) => taskObj.Result.GetValues())
            .ContinueWith((t) => {
                var values = t.Result;
                var toSeriesTask = new[] {
                    GetSeriesAsync(values, name), GetTrendAsync(values, name)
                };

                Task.WaitAll(toSeriesTask);
                return toSeriesTask.Select(x => x.Result);
            })).ToArray();

    Task.WhenAll(tasks).ContinueWith((t) => {
        DisplayData(t.Result.SelectMany(x => x).ToList());
        SaveImage("chart");
    });
}
```

Async Implementation

Since async/await are just syntactic sugar making it easier for the developer, the functionality itself doesn't change, but the code is much shorter and better to read.

```
private async Task AsyncImplementation() {
    var tasks = names.Select(async (name) => {
        var stockData = await RetrieveStockDataAsync(name);
        var values = await stockData.GetValuesAsync();

        var seriesTask = GetSeriesAsync(values, name);
```

~~~
        var trendTask = GetTrendAsync(values, name);

        return new[] {
        await seriesTask, await trendTask
        };
    });

    var results = await Task.WhenAll(tasks);
    var seriesList = results.SelectMany(x => x).ToList();

    DisplayData(seriesList);
    SaveImage("chart async");
}
```
~~~