Name: Olexandr Goloshchapov Date: 6/29/2020

Lab 1 – Strategy Pattern

Objectives

Part 1 – Refactoring for Strategy Pattern

Part 2 – Extend for more data

Reflection

Background/Scenario

In this lab you will refactor an application that provides a single stock market report so that it can provide multiple reports. Currently, the application looks for high stock swings and outputs the raw data to the console. You are going to add a report that outputs exceptionally high volume days. You could be a cut and paste pirate and modify the existing report, but you know that doesn't work in a Patterns class. Remember the credo—separate what varies from what stays the same. In this case it's the filter for deciding which trading days are present in the report. Separate these areas of code by using the strategy pattern. The stockData.csv file can be located anywhere, but if you don't provide a relative or full path, it needs to be in the same directory as the .exe file.

Required Resources

- Visual Studio
- Internet Connection

Part 1: Refactoring For Strategy Pattern

Step 1: Familiarize yourself with the code StrategyLabStarterCode.zip--and the output.

The csvRow.Split(',') in the TradingDay() constructor is C#'s version of strtok(). Once you are comfortable with the code and the output, consider the first goal: separating the code that iterates over each trading day from the code that "determines" if the day should be part of the report. In other words, the logic before the "if" statement inside the "foreach" of PrintHighDailySwings()

Step 2: Rename PrintHighDailySwings()

The strategy will become part of the "if" statement in PrintHighDailySwings(). Which means we can rename PrintHighDailySwings() to a more generic name of ReportTradingDays(). You might as well do this now. Keep in mind, you will have to pass an interface class parameter of the strategy you want to incorporate (discussed below) in addition to the StockMarket parameter.

You will eventually implement ReportTradingDays() similar to the pseudo code below:

```
For each tradingDay in tradingDays
    If ( Include( tradingDay ) = true ) Then
        Output Trading Day
    End  If
End
```

Step 3: Create a new Class Library (DLL)

Add a class library (dll) to your solution. “Name it StockReportStrategies.” Add two new class files named StockMarket and TradingDay respectively and copy the functionality of the StockMarket and TradingDay files in the StrategyLabStarterCode project. Remove the original files from the StrategyLabStarterCode project. Make both classes public. Make sure you can do a successful build. You will not modify the code in either one of these files.

Step 4: Create the IFilterStrategy Interface

Rename class1.cs to IFilterStrategy.cs. In this interface add method called Include() that takes a trading day as a parameter and returns a bool true/false depending on the strategy implemented.

Step 5: Implement the interface

Let’s start by implementing our original strategy of percent swing. In the dll, add a class called HighDailySwing that derives from IFilterStrategy and implements Include() to provide the same output of the original code.

Step 6: Update Program.cs with the new strategy

Modify Program.cs to create an instance of HighDailySwing and pass the instance, along with an instance of StockMarket to ReportTradingDays(). Implement ReportTradingDays() relative to the pseudo code above. Verify that the output is the same as the original code.

Step 7: Implement a second strategy

Implement a second strategy that shows days with high volume. This strategy will locate days where more than 20 million shares have been traded. Demonstrate that your new strategy works. Note—If you are programming to interfaces, this should not result in any change to the ReportTradingDays method.

Something to think about in the future—as simple as these strategies are, delegates would have been a more practical approach. We will cover delegates soon!

Part 2: Extend for more data

So far in the lab, you have used my dataset csv. Let’s pretend that we just got a new vendor for stock data. Everything seems alright at first but the higher ups didn’t do a good job at negotiating with the new vendor and we have some data in a different format. Everything looks good except for one minor difference. We will use the Strategy Pattern to fix this.

For this next part, I want you to download your own stock dataset from whatever company you choose. The data can be downloaded from here:

https://finance.yahoo.com/quote/GOOG/history?p=GOOG

Google is an example, but feel free to use whatever company you want.

Once you view the page, click the Download CSV button and save the csv alongside the lab’s original csv file.

Open both the original CSV file and the new CSV file you have just downloaded side by side. What are the differences? What do you think would happen if you tried to load up the new CSV file into your code with no changes?

The spreadsheets have Volume and Adjusted Close in different columns. This will likely result in incorrect parsing of the file by StockMarket class. GOOG volume appears to be almost always above 20,000,000, so the report will not filter anything out of the spreadsheet.

Step 1: Implement a new CSV parsing strategy

Keeping with the spirit of the Strategy Design Pattern, create a brand new strategy to pass along the csv to fix the code so both the original csv and your downloaded csv files work at the same time.

Note: You will create a new interface and two new classes for this part.

Step 2: Modify the original filter strategies

Depending on the company, they might not even come close to 20 million shares traded in a day. Modify the HighDailySwing and HighVolume strategies to take in a **constructor argument** to customize the swing percentage or volume limit. Make it so that if **no parameter is passed**, it will default to their **original values**.

Reflection

1. Did you run into any trouble with the lab?