**Coding Assignment 3**

I would like you to work in groups of 2.

**Problem Description**

A company has a *SalesData* class that reads sales data from different companies it owns and can print reports in different formats and headers. Soon, new companies will be bought and their data will need to be read. Also, new report formats and header formats are always a possibility. Redesign this code to:

1. Use the strategy pattern
2. Increase cohesion and decrease coupling
3. Utilize the design principle: separation of concerns (thus increasing modularity)

Thus, you can do anything you like to achieve the goals above.

An Eclipse project named *ca03.zip* is found on the Schedule which contains the class, some test code, and two text files.

Do the following:

1. Log your time. Place the time log(s) at the end of the write-up (next) or save as ca3\_lastName\_timelog.docx and place in the *refactored* package.
2. Provide a write-up that explains how you refactored the code, how the strategy pattern is used, how you have increased extensibility and maintainability. This should include and refer to a class diagram(s). Name this file: ca3\_lastName1\_lastName2.docx. Place this in the *refactored* package.
3. Refactor the code and do some brief testing testing.

**Deliverables**

1. Zip the *refactored* package into a file named: LastName1\_LastName2.zip and submit on Blazeview in the dropbox named, *CA 3*.

**Time Log**

If working in a group, each person should make a Time Log.

**Track your time using this table. Delete empty rows, add if needed. Put the total time at bottom. Delete everything above and submit (in your *prob1* folder) everything below saved as a word document.**

**Coding Assignment 3 - Time Log**

Name(s):

|  |  |
| --- | --- |
| **Time (hrs)** | **Task** |
| 1 | Brandon – discussed Strategy Pattern, looked at code for CA3 |
| 1 | Zack – discussed Strategy Pattern, looked at code for CA3 |
| 2 | Brandon – worked on interface for RDBFormatOne and RDBFormatTwo |
| 2 | Zack – worked on interface for RDBFormatOne and RDBFormatTwo |
| 1 | Brandon – added Company class with getter and setter methods |
| 1 | Zack - added Company class with getter and setter methods |
| 2 | Brandon – wrote interface for ReportBehavior |
| 2 | Zack - wrote interface for ReportBehavior |
| 1 | Brandon – updated class diagram and write-up |
| 1 | Zack – updated class diagram and write-up |
| **14** | **Total Time (hrs)** |

**Coding Assignment 3 – Write-Up**

We initially assessed the strategy pattern within the code that was given to us in CA3 by taking a good look at the program as it was and attempted to come up with our plan to write the first interface. After taking a look at the provided company text files, we decided that some of the dynamically changing content that needed to be interfaced first was the reports themselves. Rather than name the classes actual company names, we decided to name them by a particular format. Other companies that are added on later could be using the same data format in their txt files so naming the classes by a company would get confusing when dealing with future data.

We determined that we would need a Company class with getters and setters so that our interface could return an object. Returning an updated object to the main method allowed us to update particular datatypes and add on a second interface for dealing with the styling of the data.

The second interface that we added was the ReportBehavior interface. We determined that three different methods in the interface allowed us the best flexibility for switching out the displays based on passing the hType and updated Company object.

We considered making another interface for the header types because if you wanted to add on a different header later to the reports, you would have to add the different header into all the previously produced algorithms (Short(), Long(), LongPlusTaxes()). This would be good for us to consider updating.

We tested that the program was working correctly by including the main method in the SalesData class and added datatypes and interface usage. The program will run correctly when the path to the data.txt files is specified correctly in the RDBFormatOne and RDBFormatTwo classes. We also included commented lines showing where and what to change at runtime in order to get different results from running the program.

**Coding Assignment 3 – Class Diagram**

