Morton Ashley Project 2 Project 1 Analysis

1. **Calculate coupling ratio for the design components for the use cases specified above.**

4/5 (.8) Going off of my first project diagram being 4 sub-assemblies and 5 classes based off of the class diagram. If I were to find the coupling ratio for the actual program then the ratio I would be one: 1 sub-assembly over one class. These ratios show that changing or updating the program would be hard to do. With my re-designed diagram I have calculated a coupling ration of 9/8 (1.125) which shows slight improvement.

1. **Calculate cohesion value for the design components for the use cases specified above.**

**Create Sale Transaction—**

H(CreateSaleTransaction) = 0; This class was non-existent in my first project. Since the cohesion is lower than .5 it is obviously better to split things up a little. I created everything in my main method which was too much.

**Enter Rebate--**

H(EnterRebate) = 0; Once gain this class was non-existent in my first project. Everything was in my Main method which ultimately creates a terrible cohesion calculation.

I know this isn’t called for but I want to do the cohesion calculation for my main method, so:

H(Main) = 1(method) + 0(references) / 0(fields) \* 1(method) = 0; Still terrible because I shoved everything into one thing. It’s as if I just fixed a car with duct tape and phone wire: things will eventually go south.

1. **Determine if each of the classes in your design that provide the functionality for the use cases specified above violate the SRP.**

**Create Sale Transaction—**I guess technically speaking, this class does not violate the SRP. Why? Because if it does not exist then how is it violating anything?

**Enter Rebate—**Once again I can’t say that this is technically violating anything.

**Main—**that being said: My main method is violating the SRP tremendously. If it were following the SRP then it would only have one purpose and only play a single part to the program. Instead, since it encapsulates the entire program, it contains every portion of the program and has multiple parts it is in charge of. I would have to split it up into multiple classes that each have a specific job in order to not violate the SRP.

1. **Determine if each of the classes in our design that provide the functionality for the use cases specified above violate the OCP.**

**Create Sale Transaction—**Once again a class that doesn’t exist can’t technically violate anything.

**Enter Rebate—**my whole project was pathetic and I’m sorry you had to see that.

**Main—**my main method, which encapsulates the entire program, violates the OCP tremendously. Changing any portion of my code would cause it to worsen, if being worse is even possible at this point. My program is like the old Windows game application: Pinball. The code is so horrendously typed that there is no hope in updating it or fixing it. The only way to fix it is to throw it away and start over with a better approach.

1. **Modify one class/module that violates the SRP, so that it does not violates the principle anymore.**

I didn’t get much done code-wise but I hope my UML shows that I am trying to fix my violations and better my program. I have restarted from scratch because there were so many issues with my first project that it was easier to just start over.

1. **Modify one class/module (different that the one in 5) that violates the OCP, so that it does not violates the principle anymore.**

Likewise with this one, I started over from scratch and I hope you can see from what I have so far that this will have better coupling and cohesion ratios at the end of the day.

\*\* In addition to the above I want to add in that because my first project was so terribly written I decided to start over from scratch. Because of this I will include my first project and my remake (PA2) for your viewing. Although the coding is not finished I hope that viewing my UML class-diagram will show you that I have noticed my mistakes from the first project and am learning a better way to code.