Object Oriented Software Engineering (COMP2003)

Assignment Semester 2, 2016

Student ID: 18249833

Name: Xhien Yi Tan (Xavier)

This submission includes the following Java Source Code:

- { Property, Company, Business, BankAccount, PrimaryCompany }.java
- { Event, Plan }.java
- { FileIO, IOBehaviour, PropertyIO, EventIO, PlanIO }.java
- { Subject, Observer, ProcessData }.java
- { YearlyTrainingData, YearlyEventData, YearlyProfitData, YearlyPlanData, YearlyInterestData }.java
- { TrainingSystem, TrainingSystemManager }.java
- { InvalidOperationException }.java

Also includes Others:

- UML – There are more than one uml png
- Report.pdf (This report)

Instruction on running my program:

Commands: java TrainingSystem *propertyfile eventfile planfile start_year end_year*

The calculation for each year happens in this order:

1) Process Events which affect the properties details
2) Calculate profit for each business and company
3) Buying or selling
4) Interest

Student ID: 18249833
Name: Xhien Yi Tan (Xavier)

Overview

My program will start from reading three files and it will start doing the operations for each year. For each year, it will print Company Name, Profit earned on previous year, Current bank balance for each company. The Design Pattern I have used in this assignment are **Strategy Pattern** and **Observer Pattern.**

Reasoning

Strategy Pattern: Answered in Q1.

Observer Pattern: I have used this pattern due to the result of the calculation of each year for this entire program relies on each other. It means that the TrainingData for each year is very dependent on previous year data. With that being said, here comes the explanation. Every year thing happens, and happens in a particular order. By using observer, changes that are made each year will notify these observers (YearlyEventData, YearlyProfitData, YearlyPlanData, YearlyInterestData) because each data is dependable on each other and theses dependent objects are to be notified automatically. Each observer then processes what it supposed to process.

Questions

Q1. Where have you used polymorphism, and why? Discuss any design patterns you've used that incorporate polymorphism.

Answer: According to my implementation, I have used Strategy Pattern to incorporate polymorphism. I have used it in to file io because there are 3 types of different file to be read. It makes sense for me to have used polymorphism here so that I can set different reading type in run-time. For example, reading Property, Event as well as Property.

Therefore, I have these classes for File IO:

- FileIO.java – It serves as the context for file io and it has an object type IOBehaviour
- IOBehaviour.java – It is an Interface that is responsible of reading different types of file
- PropertyIO.java, EventIO.java, PlanIO.java – All implements IOBehaviour.java Interface

Q2. How does your design achieve testability? That is, what have you done to make unit testing easier?

Answer: Making my design achieve testability, what I have done is to separate each important factors out and also unit test them separately. These factors are Event and Plan. By factoring each of them out, I can test my program separately without interfering each other.

Student ID: 18249833
Name: Xhien Yi Tan (Xavier)

Q3. Discuss two plausible alternative design choices, and explain their pros and cons.

Answer:

Template Method could be the plausible alternative here since it is similar to Strategy Pattern. The reason why I am not using template here because I don't have methods that can be served as template method since my FileIO (which is Strategy Pattern) has only one behaviour (reading behaviour) that is vary from each strategy object like Property, Event and also Plan. In template method, it requires more than one behaviour methods served as a template which I don't have. And also that the template method can't change the behaviour at run time.

Decorator could also be the other plausible alternative here. In the context of this assignment, I could have decorated property with plan or event so that the property's details can be affected by them in some way or in other word, provides additional functionality to existing object. But in this scenario, there are more than one property can be affected by event or plan. There is no point for me to decorate every events or plan on property just to add more and more functionality on them.