1. Name:

Yu lin, Chou

2. **Project description:**

I built a eCommerce website which allows people to buy product that listed on the website, then when you click the “checkout” button, it would help you to generate the total amount based on the discount and different state tax.

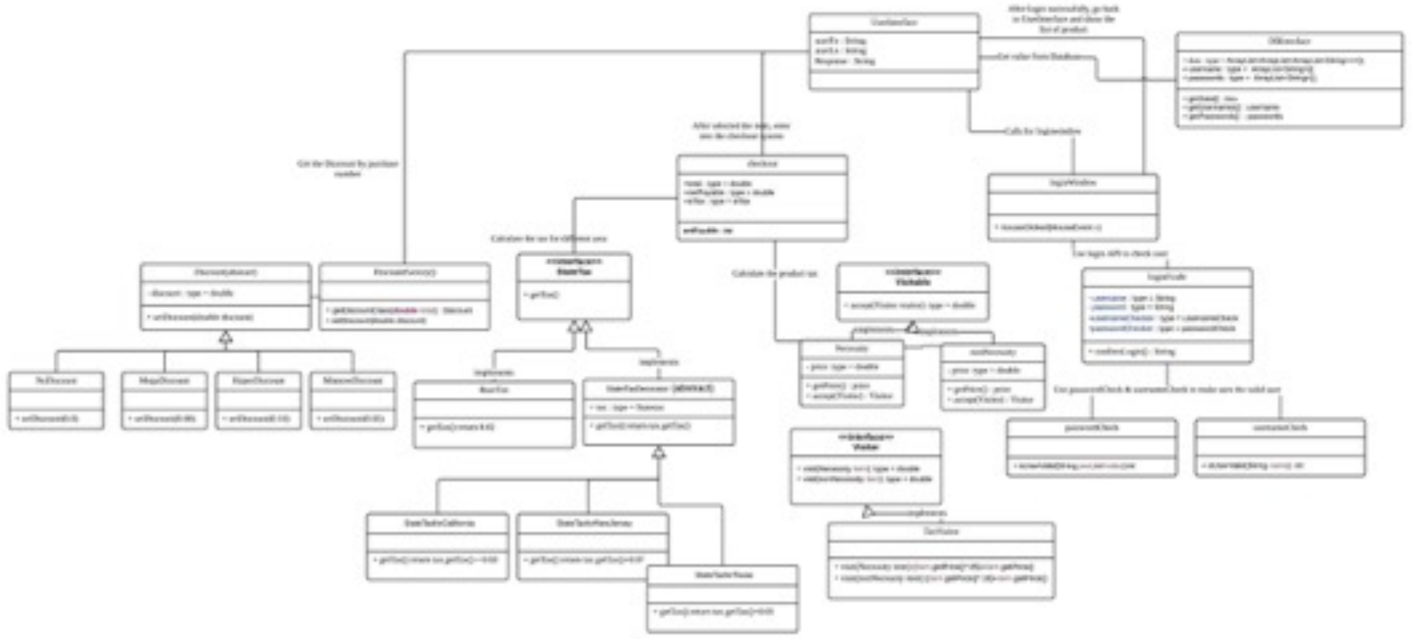
3. List the features that were implemented (table with ID and title).

ID	Features
1	login
2	add product into shopping list
3	checkout
4	calculate the state tax
5	calculate the available discount
6	calculate whether it is necessity item

4. List the features were not implemented (table with ID and title).

ID	Features
1	Modify the shopping cart
2	Send automatic email confirmation
3	Use credit card to pay for the product

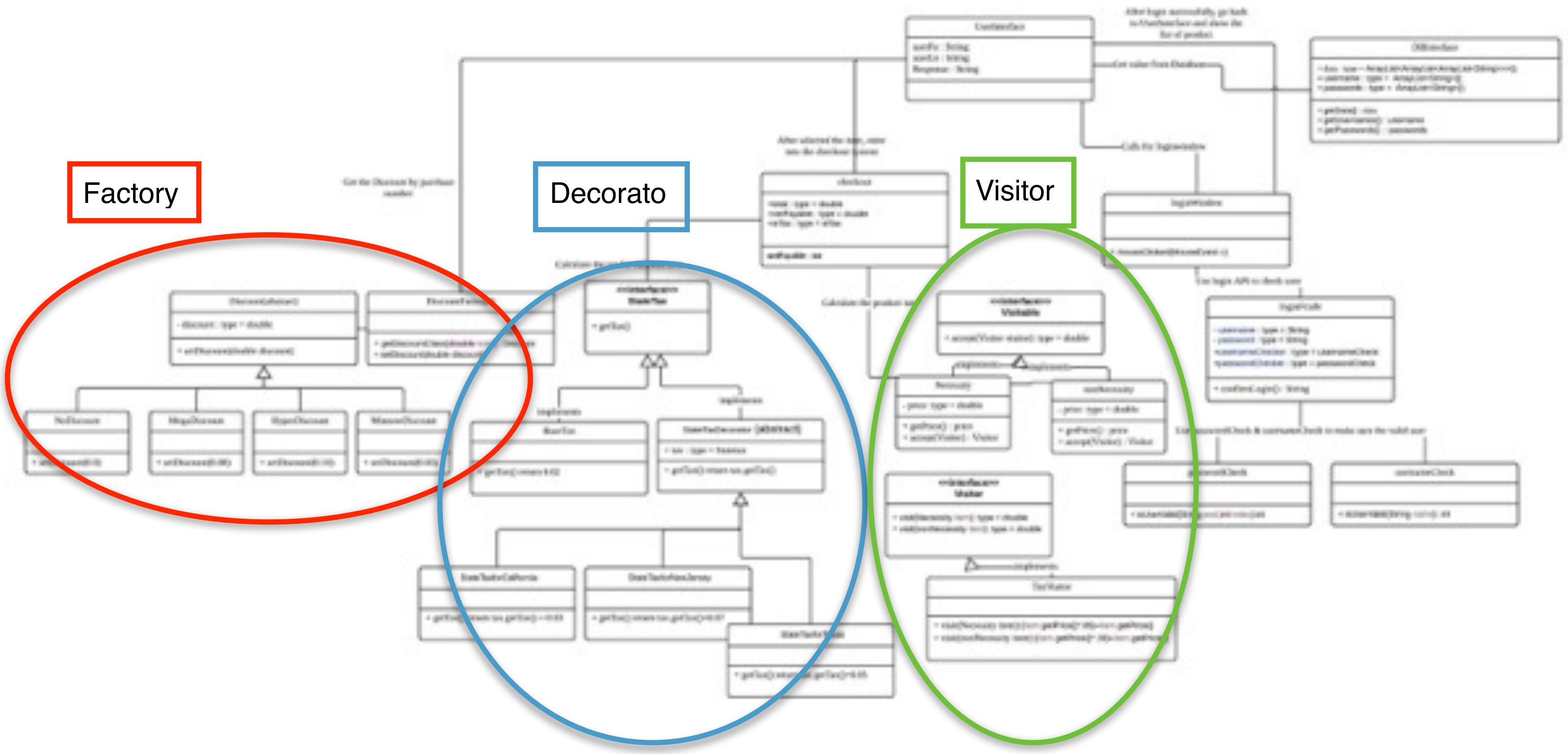
5. Final class diagram:



What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

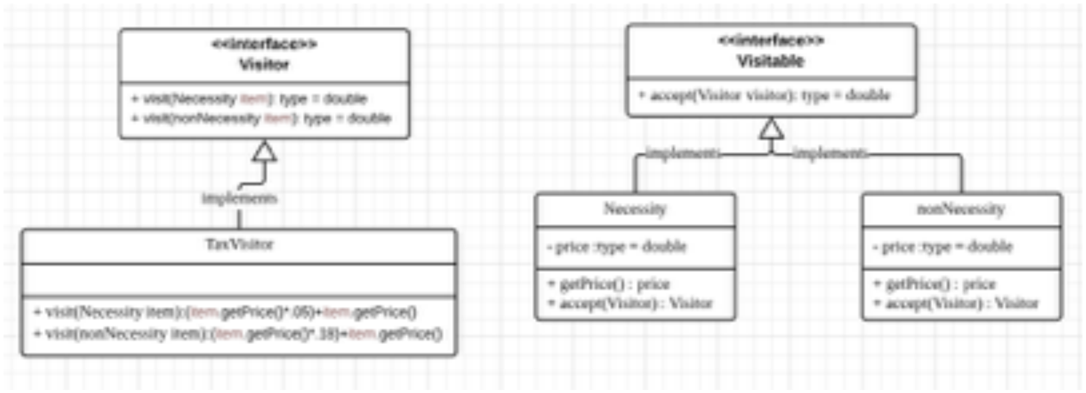
The changed part is that the final diagram is more completed and has more information about the design pattern, in the original one it does not include the design pattern concept and it only has few classes.

6. For each design pattern implemented
Show the classes from your class diagram that implement each design pattern.



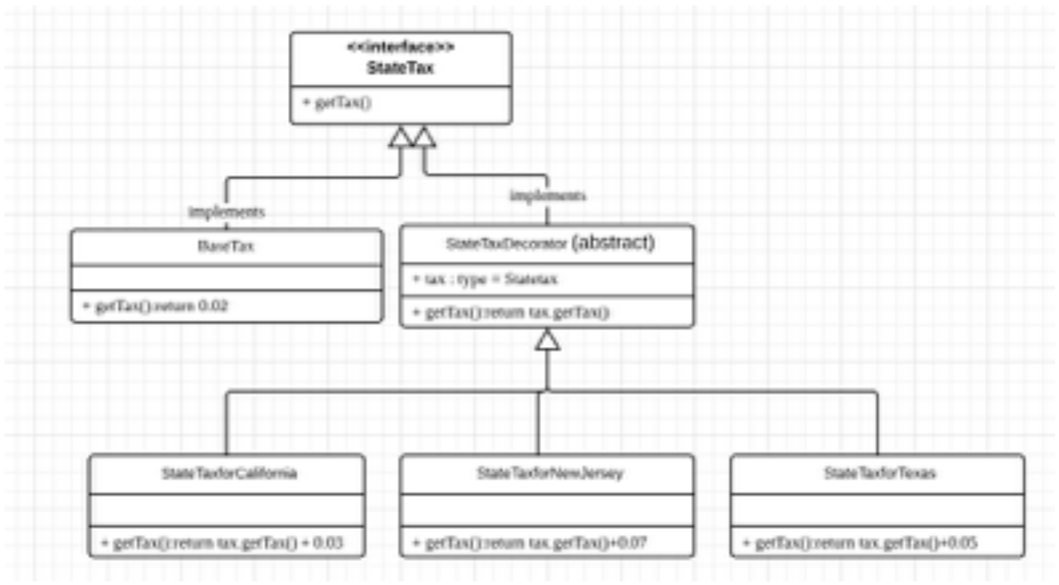
Show the class diagram for the design pattern.

Visitor =>



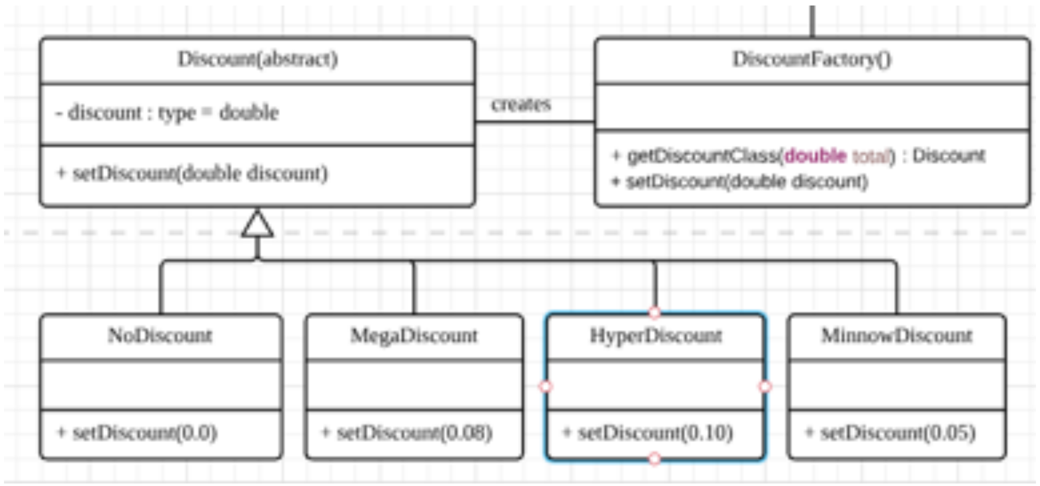
<<interface>>
Visitor
+ visit(Necessity item): type = double
+ visit(nonNecessity item): type = double
implements
TaxVisitor
+ visit(Necessity item):(item.getPrice()*.05)+item.getPrice()
+ visit(nonNecessity item):(item.getPrice()*.18)+item.getPrice()
<<interface>>
Visitable
+ accept(Visitor visitor): type = double
implements
implements
Necessity
- price :type = double
+ getPrice() : price
+ accept(Visitor) : Visitor
nonNecessity
- price :type = double
+ getPrice() : price
+ accept(Visitor) : Visitor

Decorator =>



<<interface>>
StateTax
+ getTax()
implements
implements
BaseTax
+ getTax():return 0.02
StateTaxDecorator (abstract)
+ tax : type = Statetax
+ getTax():return tax.getTax()
StateTaxforCalifornia
+ getTax():return tax.getTax() + 0.03
StateTaxforNewJersey
+ getTax():return tax.getTax()+0.07
StateTaxforTexas
+ getTax():return tax.getTax()+0.05

Factory =>



Explain how you implemented the design pattern, why you selected that design pattern.

decorator design pattern =>

Why : I selected this design pattern because the decorator pattern will be used to calculate the cost of an order. The pattern will be helpful in adjusting the tax rate dynamically when a user opts for checkout. If the user is from a certain state, a particular tax is to be applied on the purchase, all these selections are to be made dynamically, so the decorator pattern will be used in place of simple inheritance to handle this behavior.

How to implement : I created an interface called StateTax, and then I created concrete class BaseTax implementing the interface so that I can set basic condition, then created an abstract decorator class StateTaxDecorator implementing the StateTax interface and having StateTax object as its instance variable. After I finished these process, I can generate few customized classes with inheritance from StateTaxDecorator to adjust different situation(state tax from California, New Jersey and Texas)

visitor design pattern =>

Why : In the database the products will be listed as necessity items or not, the items that are not a necessity include liquor, tobacco etc. And should be taxed additionally. The visitor pattern will be used to handle this behavior in particular. Since the necessity and non necessity items has distinct operations(One required extra tax, one is not) to perform across a structure of objects. This avoids adding in code throughout your object structure that is better kept separated.

How to implement : The core of this pattern is the Visitor interface. This interface defines a visit operation for each type of ConcreteElement in the object structure. Meanwhile, the TaxVisitor implements the operations defined in the Visitor interface. The tax visitor will store local state, typically as it traverses the set of elements. The Visitable interface simply defines an accept method to allow the visitor to run some action over that element - the Necessity & nonNecessity will implement this accept method.

factory design pattern =>

Why : The holiday season is around and our store has decided to give every customer, some discount based on their purchases. Now the discounts are completely dynamic and are based on the amount of purchase a customer does. For simplicity, we have 3 types of discounts

1. Minnow, 5% discount – For customers who purchase between $500-$1500
2. Mega, 8% discount – For customers who purchase between $1500-$3000

3. Hyper, 10% discount – For customers who purchase for more than $3000

Why we use the factory pattern ? We don't know ahead of time which class of discount the user will fall in. It only depends on his purchase. All the potential discount classes fall under the same category of discounts. We want the program to select classes on the runtime.

How to implement : I created a Discount abstract class and concrete classes implementing the it, my project has four types of different discount. A factory class DiscountFactory is defined as a next step. our demo will use DiscountFactory to get a Discount object. It will pass information to DiscountFactory to get the type of object it needs.

7. What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

I learned that Design pattern is really useful in the real world software development, if we do not use the pattern like decorator or visitor, we would be forced to generate a lot of unnecessary code to handle different situation, it would also make the program more difficult to manage. However, with the help of design pattern, the program would be easier to track code since everything is organized.