Braden Hicks

3/20/2019

Clearent

Single Responsibility Principle

To convey the single responsibility principle, I created the following classes:

- Person
- Wallet
- Credit Card
- Visa
- MasterCard
- Discover
- CalculatePersonTotalInterest
- CalculateWalletTotalInterest

And the interface:

- ICalculateCreditCardInterest

Each class and interface have only one job to do. The Person, Wallet, CreditCard, Visa, MasterCard, and Discover classes only instantiate the objects with their attributes accordingly. The CalculatePersonTotalInterest class only calculates the total interest a person has, and the CalculateWalletTotalInterest class only calculates the total interest a single wallet has. The ICalculateCreditCardInterest interface only calculates interest for credit cards.

Open/Closed Principle

I exemplified the open/closed principle with the CreditCard, Visa, MasterCard, Discover, and ICalculateCreditCard interest. Visa, MasterCard, and Discover all inherit from CreditCard, which has two fields, interest and balance. Each card implements the ICalculateCreditCardInterest, which has a function called calculate, which calculates interest. Each CreditCard subclass has its own calculate function which calculates the interest rate appropriate to each card. This leaves room for extension, as we can add more CreditCard subclasses necessary which can implement the ICalculateCreditCardInterest interface as long as it has a corresponding calculate function as well.

Liskov Substitution Principle

This principle is shown with the CreditCard class and the corresponding subclasses, Visa, MasterCard, and Discovery, and the Wallet class. The Wallet class has an attribute which is a list of type CreditCard. As seen in the test cases, we can instantiate and add the CreditCard subclasses to the list of CreditCards without any problem, proving the principle.

Interface Segregation Principle

This principle can be seen in the ICalculateCreditCardInterest and the CreditCard subclasses. The idea is to rid the interface of unnecessary functions or attributes in an interface which some classes that implement the interface will not touch. In other words, create smaller, tighter interfaces with only important attributes and functions. In the case for the ICalculcateCreditCardInterest and the CreditCard subclasses, the only important field in the interface is the calculate function, which each subclass will use.

Dependency Inversion Principle

The dependency inversion principle is shown through the relationship between the CreditCard subclasses and the ICalculateCreditCardInterest interface. The interface is not dependent on the type of CreditCard that is being use, it will still calculate the interest whether the card is Visa, MasterCard, or Discover. We can add plenty of other CreditCard subclasses and it still will not care what type of card it is, as long as it has a corresponding calculate method, it will perform the calculate function.