CMPE - 202 Software System Design
202 Individual Project
Amika Mehta (015957695)

Part - 1

1. Describe the primary problem you are trying to solve
   Through the above problem statement I have realized a primary issue. I.e. To figure out if the given card is valid or not and to check it against the various credit cards available -
   - Master Card
   - Visa
   - American Express
   - Discover

   Another issue is if the card is valid then to determine the card issuer based on the credit card number.
   Lastly, to find the relevant objects based on card types.

2. Secondary Problem
   The secondary problem is that we need to identify a proper design class in order to accommodate additional credit card classes in the future for the various credit card types.For which we need to use Cretional pattern is to be used to create different file format objects.

Part - 2

3. Describe what design patterns you used ?
   Design patterns used - Chain of Responsibility, Strategy and Factory Method

Factory Method Design Pattern

1. For Part -1 We are creating an abstract class CreditCard and 4 subclasses for each credit card type which extend the CreditCard class. Each concrete handler can access its successor. Whenever a Credit card number is received a ConcreteHandler would find the exact type of card and create a corresponding credit card object. For instance if a MasterCardhandler figures out that it's a MasterCard it will create a MasterCard object. Therefore the end user doesn't know which subclass is exactly created.
2. For Part-2, while creating File Objects we define a FileFormat abstract class. There are 3 subclasses (CSV file, XML file, JSON file) which extend the file format. Therefore whenever an end user gives any file as an input internally FileFactory would determine which file object to create based on the input file given. The File Factory's only job is to create corresponding File Object without the client knowing as to which File Object is going to be created.

   Diagram

Chain of Responsibility Design Pattern

1. After parsing the input file (which is in either in xml, json, csv) we need to first check as to which format does the credit card belong to among the 4 credit card types - MasterCard, Visa, American Express or Discover for which the handler interface is defined and we have 4 concrete handlers(MasterCardHandler, VisaCardHandler, AmExCardHandler and DiscoverCardHandler).
2. Once the exact type of credit card has been validated. We use the Chain of Responsibility Design Pattern because we first start off with the MasterCard handler.
3. If the given input is not MasterCard then it would get passed along to the next type i.e Visa followed by Amex(American Express) and lastly Discover.
4. While doing so we need to moderate the entire process for which we are using Handlers, which pass the input file along the 4 classes and if nothing an error message is displayed on the screen.
5. All the 4 credit card handlers implement the main handler and then pass the file along.

Strategy Design Pattern

1. Strategy Design Pattern allows us to change the behavior of an application based on the selected strategy without writing extra code.
2. Since there are different file types, in order to address them different objects have been created and appropriate strategies are used based on the file types.
3. I have used the Strategy Design Pattern to support the different file formats (xml, csv or json) based on the input file type. There are 3 different Handlers which are being used - Reader, Writer and CreditCardHandler.
4. The Read Interface - It has a read method which will read the input file based on the type of the input method given(xml,csv,json)
5. The Writer Interface - using the writeToFile method it will take the given input and write in an output file in the same file format as the input file.
6. The CreditCardHandler Interface - It has a creditCardType interface which would take the parsed input and check as to in which format does the input fall - MasterCard or Visa or Discover or Amex.
7. This design pattern helps in swapping behavior at runtime and also accommodates newer files without writing any additional code as it follows the open/close principle. The main benefit being it doesn’t affect the end user.

4. Describe the consequences of using this / these design patterns.

Chain of Responsibility

Pros:

1. Separates the sender of the request from the recipient. I.e. the recipient doesn't need to know which handler object is going to be used to handle the request.
   With this design pattern the client doesn't need to worry about the chain structure as each handler object in the chain has a successor and it maintains a direct reference to

its methods and just in case the handler couldn’t figure out the type of card then it’s declared invalid.
2. It is easy to add/remove/change the order of the chain (credit card type) dynamically by adding or removing the members in the chain.

Cons:

1. Hard to observe the run-time characteristics and debug.
2. The performance of the pattern is not guaranteed since it may fall off the end of the chain if no handler object could handle it - the credit card type.

Strategy Design Pattern

Pros:

1. Strategy design pattern has the ease of adding any number of strategies dynamically as and when required.

Cons:

1. The User should be aware of multiple types of strategies and what's the exact difference between them in order to use them.
2. At any time there is a possibility of having more objects than it can handle and hence making it cumbersome and repetitive.

Factory Method

Pros:

1. The client doesn't need to know which class object to instantiate. The Factory method allows you to create an object without exactly specifying the exact class of object that will be created.
2. It is easy and flexible to add or remove the different file formats in the future.

Class Diagram