CMPE-202 Individual Project-Credit Card Problem

I. Part 1

1) Describe what is the primary problem you try to solve.

- In this problem, the primary problem statement is to identify what type of Credit Card does each record of the input file represent i.e., whether it's a MasterCard, Visa, American Express or Discover.
- For verifying this we must parse the given file for each record and verify that the corresponding credit card number is a valid account number or not. If the Credit Card number is invalid, handle it accordingly.
- If the card is a valid card, it needs to be processed by one of the four objects. Since the client doesn't know which object is the right one responsible for handling the request, we can use the Chain of Responsibility behavioral design pattern to issue a request to identify the type of Credit Card to one of the several concrete handlers without specifying the receiver explicitly.

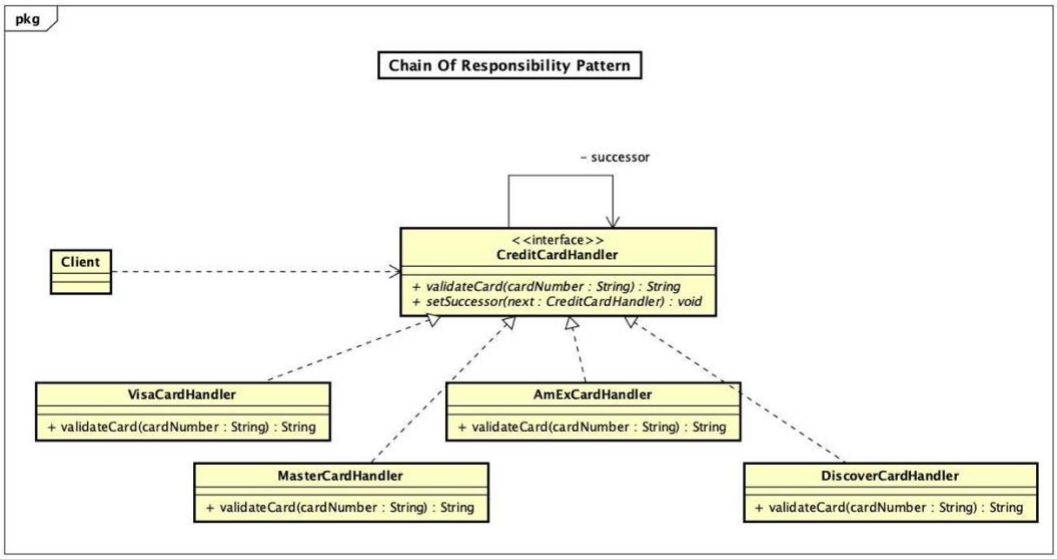
2) Describe what are the secondary problems you try to solve (if there are any).

The secondary problem is to create the appropriate objects of the Credit Card class once the type of card is identified after reading the record from the file. We can use Factory Method creational design pattern by defining an interface for creating Credit Card object and allowing the subclasses to create object of the identified Credit Card. This allows us to create the appropriate Credit Card objects at runtime.

3) Describe what design pattern(s) you use how (use plain text and diagrams).

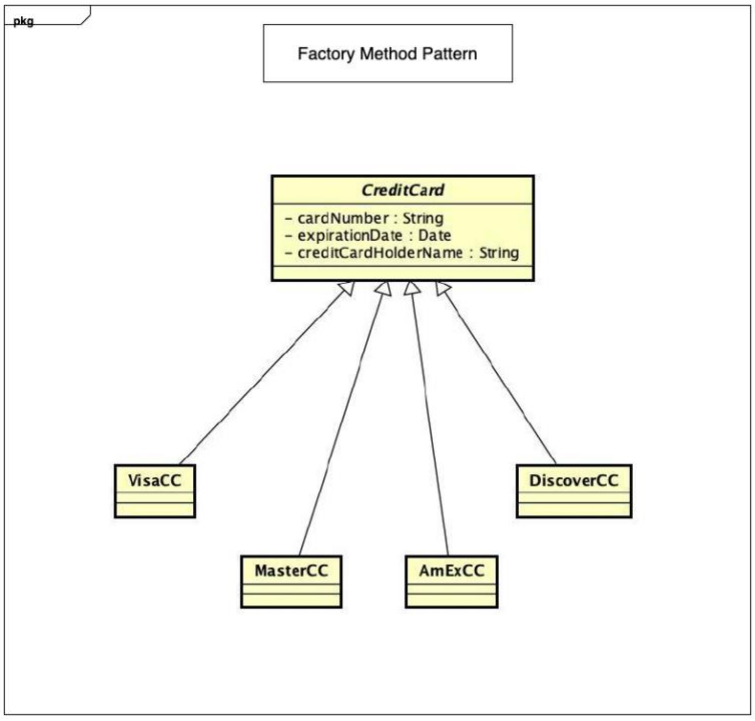
Chain of Responsibility:

- We use this behavioral pattern to identify the type of card inserted and to redirect it to a corresponding handler object for it.
- We define 4 concrete handlers to identify and validate the cards the 4 different types of cards.
- The request is passed to a chain of receiving objects until the appropriate credit card object handles it dynamically at runtime.
- This allows the request to validate the type of Credit Card to be issued to one of the several concrete handlers without specifying the receiver explicitly.
- In case, if new Credit Card types need to be added in future, we can add more handler classes without having to change code much (follows Open/Closed Principle)
- Thus, we can use chain of responsibility pattern to solve our primary problem to identify the type of Credit Card corresponding to a specific record in the file.



Factory Method Pattern:

- Once we identify the whether the Credit Card number is a valid account number or not and what is the appropriate Credit Card type, we can use Factory pattern to solve the secondary problem of creating appropriate objects of the credit card class at runtime without knowing beforehand what type of objects it needs to create or how to create it.
- The Abstract CreditCard class is the parent class for all the concrete Card classes (i.e. VisaCC, MasterCC, AmExCC and DiscoverCC). Each child class creates an object of that type CreditCard.
- Since there is no extra functionality required rather than creating specific cards, we have chosen not to create any extra creator classes for each card. We are using the concrete card classes for this purpose.
- The Credit Card class requires its subclasses (i.e. MasterCC, VisaCC, AmExCC, DiscoverCC) to specify the objects it creates at runtime. It delegates the creation of an appropriate Credit Card object to its subclasses and localize the logic of which helper subclass is the delegate.
- Moreover, we can add new subclasses in future without breaking the client code (follows Open/Closed Principle). Credit card object creation logic is hidden from the client and client can use the same interface to create a new Credit Card object in future.



4) Describe the consequences of using this/these pattern(s).

Chain of Responsibility Pattern:

- It allows us to create separate handlers for each Credit Card type and the request is passed to a chain of objects until an appropriate object handles it. If the handler cannot handle the request, then the Credit Card number can be declared as invalid.
- It follows Open/Closed Principle thereby allowing us to add new handlers in future without modifying the existing client code and therefore easily extendable if new Credit Card type is introduced in future.
- This allows loose coupling of the client's request to identify the Credit Card type and the receivers that perform the operations. (Single Responsibility Principle)
- The order of the request handling can be controlled
- We can add or remove responsibilities by changing the order of handlers in the chain.
- In order to forward the request every object in the chain has the reference to the next object and it doesn't have to know the entire chain structure thereby simplifying the object.
- Sometimes it is difficult to analyze and debug the runtime behavior
- Sometimes if no object handles the request it may fall off the end of the chain and some requests maybe unhandled.

Factory Method Pattern:

- It encapsulates the creation of Credit Card object.
- It provides approach to code to interface rather than implementation.
- It allows loose coupling of the concrete Products classes (MasterCC, VisaCC, AmExCC, DiscoverCC) and the creator class (CardCreator).
- The CreditCard class defers the instantiation to its subclasses at runtime.

- We can easily extend the code by adding new product types in future without breaking the existing client code (follows Open/Closed Principle).
- It moves the actual object creation code from client code thus making the code more robust, easier to support and less coupled. Thus, follows Single Responsibility Principle.
- It relies on inheritance by providing abstraction between client and implementation classes (follows Dependency Inversion Principle).

II. Part 2

So we need to extend our application to parse different file formats. For this we are using Strategy pattern. We check the extension of the input file and based on the input file context we define the strategy to parse the given document.
This will allow us to add more strategies in future as well. We just need to create a concrete strategy class like CsvFileParser, XmlFileParser and JsonFileParser.