Credit Card Problem

Student Name: Raviteja Gannarapu
Student ID: 016650972

Design patterns:

- Chain of Responsibility.
- Factory method.

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

- **Determining credit card number** is the primary problem. Each credit card has its own specific criteria. Though current problem lists only a few credit card types, design should support any number of credit cards.

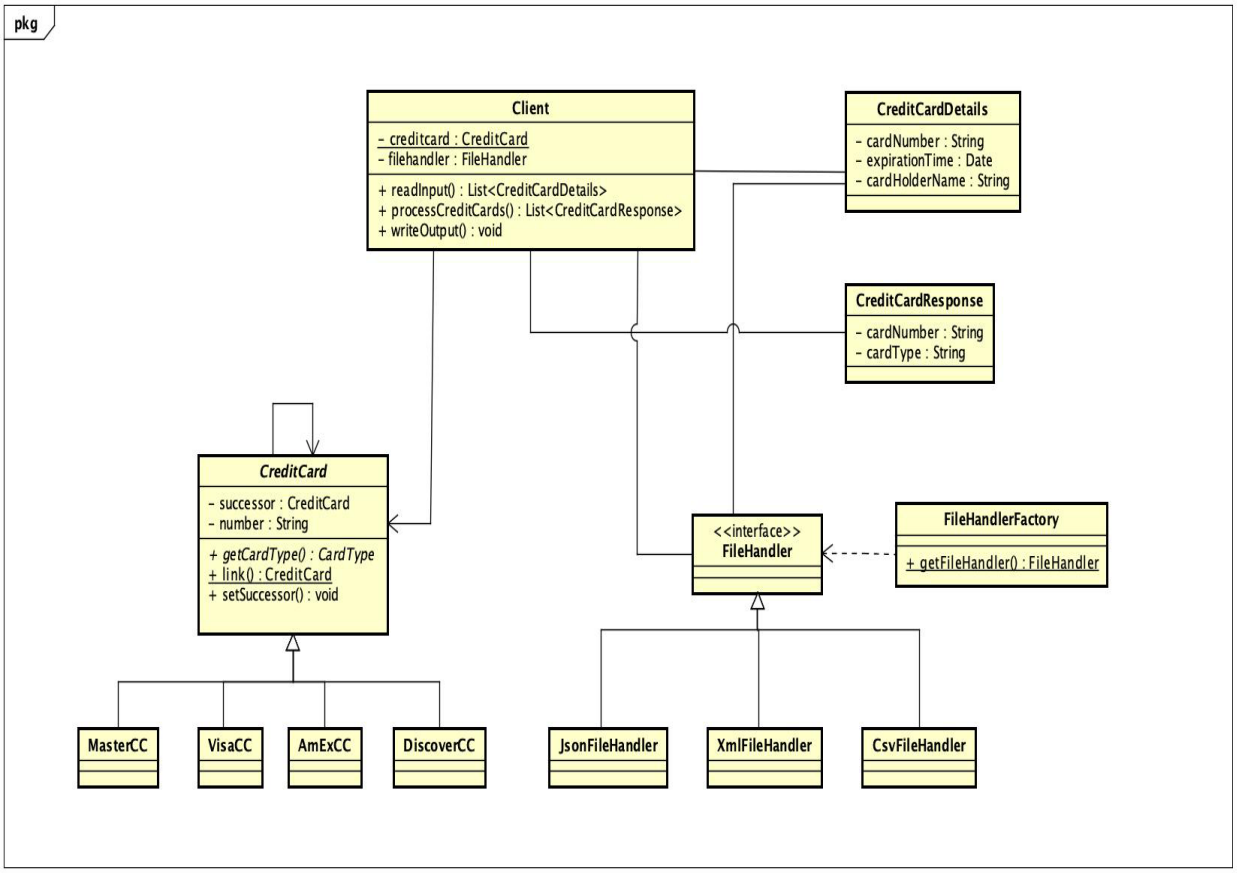
2. **Describe what are the secondary problems you try to solve (if there are any).**

- Application should **accept multiple file formats**. It should support multiple file parsers or handlers. And they are determined on runtime.

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

- Application is implemented using **Chain of Responsibility** Behavioral pattern and **Factory method** creational design pattern.
- Primary problem of determining credit card type is tackled with Chain of Responsibility.
- Created abstract class CreditCard with subclasses of different credit cards like MasterCC, VisaCC, AmExCC and DiscoverCC. While CreditCard acts as Handler with link to successor CreditCard.
- Each subclass overrides a Handler method getCardType to define their own criteria using which creditcard type is determined.
- CreditCard class has a method link to define chain of objects.
- For the secondary problem, Factory method is implemented. Interface FileHandler is created which declares methods to read from and write to files. Similarly, there exists a filehandler for each file format. A filehandler is determined based on input and output file's extension and factory method is used to create handler class object.

Class Diagram:



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

- Using these patterns, application can be easily updated to supports any number of different credit cards. Also, newer file formats can be easily added.
- Since each credit card defines its own characteristics, defining identity becomes its own responsibility.
- Reduced coupling: An object is not required to know which another object will respond. Only that a request will be treated "appropriately" must be known by an object. A link in the chain doesn't have to be aware of the structure, and neither the sender nor the recipient has any explicit knowledge of the other. Chain of Responsibility can therefore make object interconnections simpler. Object keep a single reference to their successor rather than maintaining references to all potential receivers.
- Added flexibility in assigning responsibilities to objects.

- *Provides hooks for subclasses. There is never a situation when generating an object directly is more versatile than using a factory method inside a class. Subclasses are given a hook by the factory method to provide an expanded version of an object.*