INDIVIDUAL PROJECT

PART 1 DELIVERABLES

NAME: REVANTH KRISHNA MADDULA
SJSU ID: 014605201

- Describe what is the primary problem you try to solve.

The primary problem which I am trying to solve is to check whether a card is valid or not by checking with the valid types of credit cards: Master Card, Visa, American Express and Discover, and also validates with the right checksum. If the card doesn't correspond to any category, the specific credit card record should be considered as invalid. Since a CSV file with credit card records is presented, it is highly necessary to make the code reusable without high maintenance.

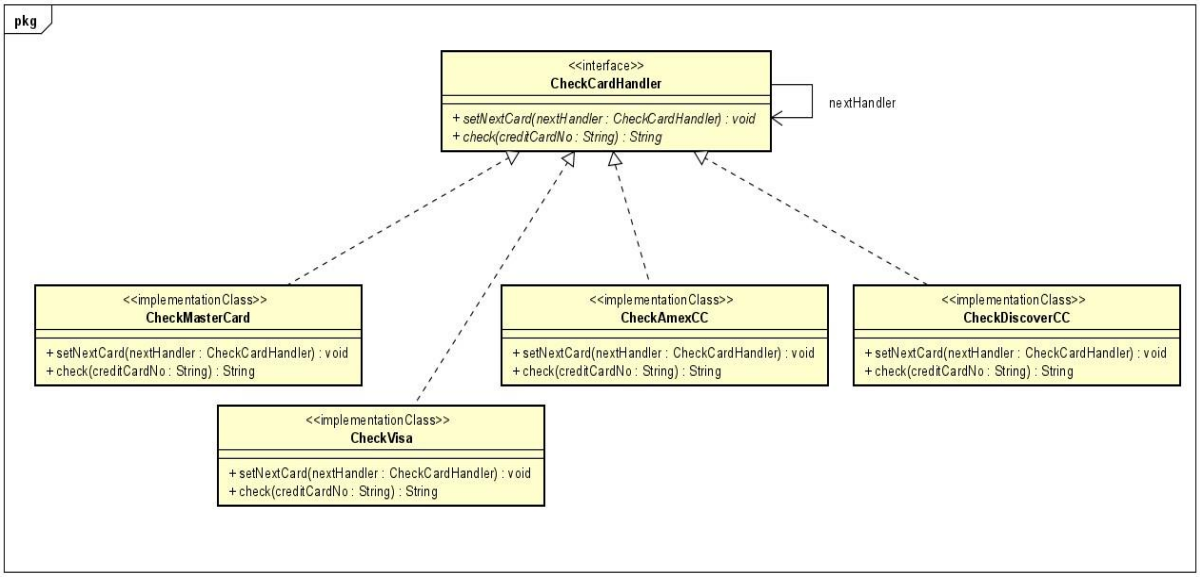
- Describe what are the secondary problems you try to solve (if there are any).

After mapping the credit card to the respective sub class of the Credit card (can be Master Card, American Express, Visa, Discover), the following task is to create an instance of the credit class object as specified. A proper creation design pattern is necessary, to accommodate newer credit card classes in the future, or else it would be very hectic to add a new credit card subclass.

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

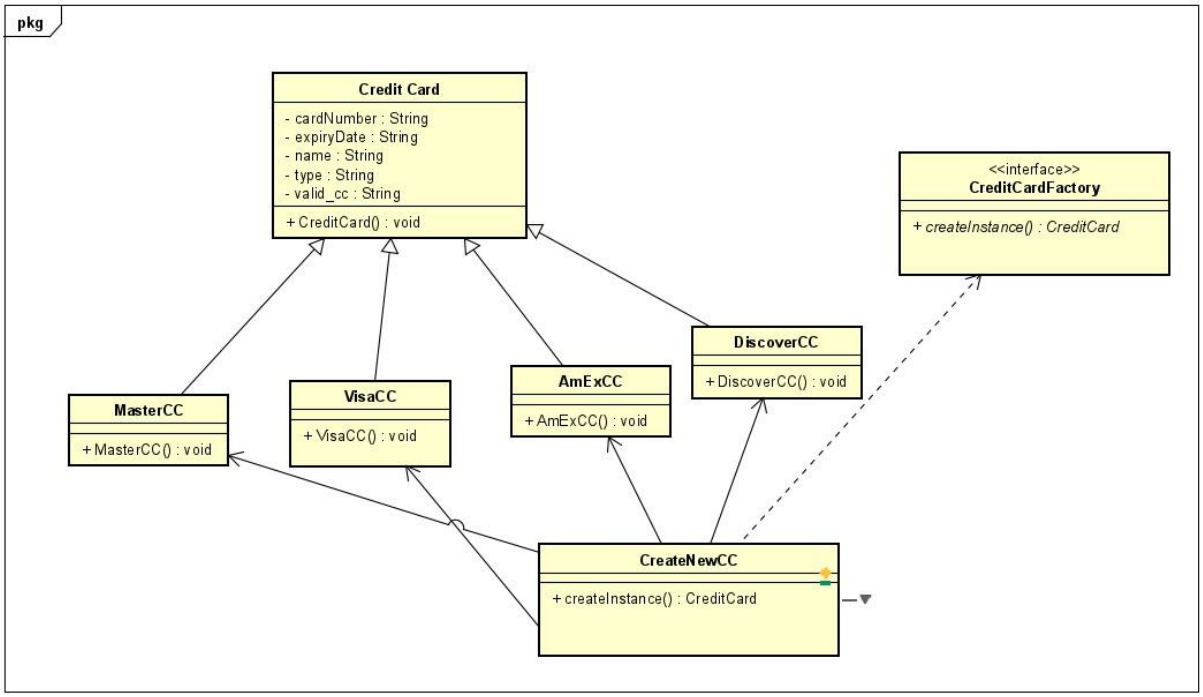
Two design patterns can be considered for the mentioned problems:

1) Chain of Responsibility:
In order to check which class a Credit Card belongs to, "Chain of Responsibility" behavioural pattern can be used to pass the requests along the handlers in a sequential manner: Each handler class is used to check if the credit card belongs to that class, if it is not: the handler is passed to the next class. This process is followed until the card is done verifying with all of the credit classes. If a handler matches with the class, a String is returned matching to that class name, or else the control is passed to the next handler.



2) Factory:

After finding the subclass of the credit card type, the creational design pattern “factory” can be used for creation of an instance for the particular card type. This pattern is preferred because the creation logic is not exposed to the user. Using a common interface, newly created objects can be referred.



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

Consequences of using Chain of Responsibility pattern:
Advantages:

- Decrease the coupling degree, by decoupling the classes that invoke operations.
- Improve the flexibility of the functions assigned to objects.
- The object is simplified as it does not need to know the structure of the chain.
- It is easy to control the order of request handling. In the future, adding new handlers to the existing chain also becomes easy.

Disadvantages:

- The request at a recipient is not always confirmed for sure.
- In the case of mishandling of the chain, requests might get dropped, or it can also create a cycle within the chain.
- It might not be simplified to observe the features of the operation.

Consequences of using Factory pattern:
Advantages:

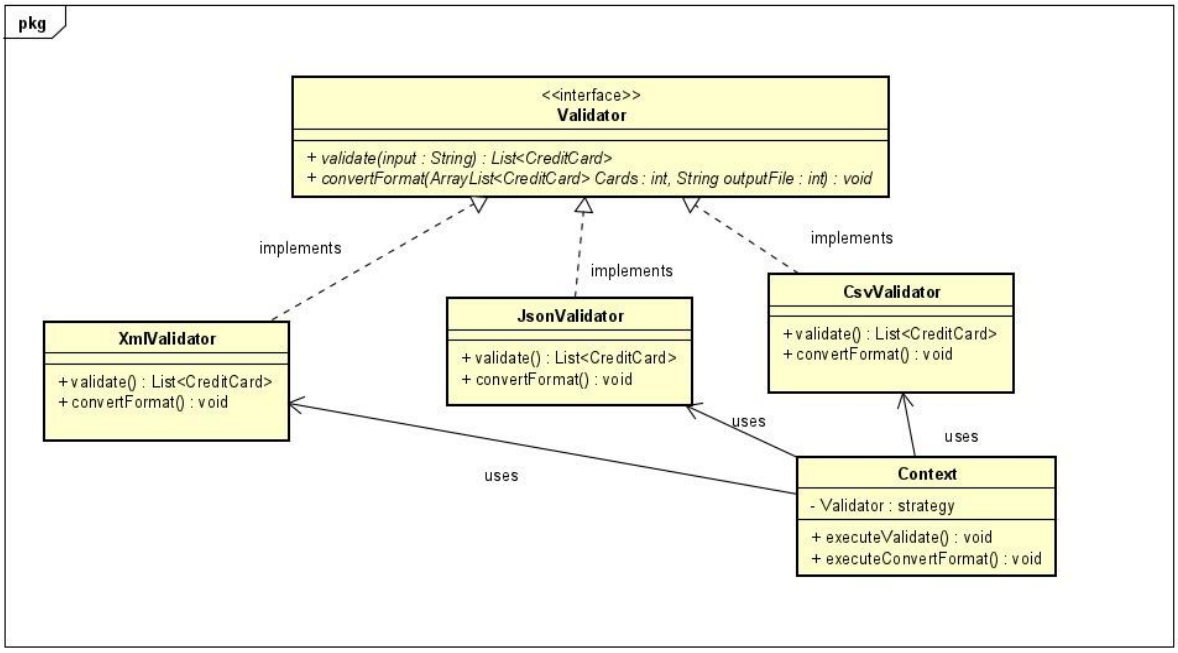
- Supports Single responsibility principle, by making the code easy for support by moving the creation of new instance code to only one place in the program of execution.
- Avoid the strong coupling between the concrete classes and the creator.
- Support open-closed principle, by paving the way to create new types of instances without dismantling the existing code.

Disadvantages:

- Creation of a subclass is required just to instantiate the object where converting it as a subclass is not really necessary.
- Code becomes more complicated due to the addition of many new subclasses into the system.

PART 2

Strategy pattern for parsing different input file formats:



Extended class diagram: