CMPE 202 – Individual Project – Credit Card Problem

Submitted By
Darshini Venkatesha Murthy Nag – 016668951

Part A Deliverables

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

The Primary problem which I am trying to solve is to verify the validity of the credit cards by checking if they belong to one of the accepted categories: Master Card, Visa, American Express and Discover. If a card does not belong to any of these categories, it should be marked as invalid. A file containing credit card records is available, it is crucial to ensure that the code is easily reusable and requires minimal maintenance.

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

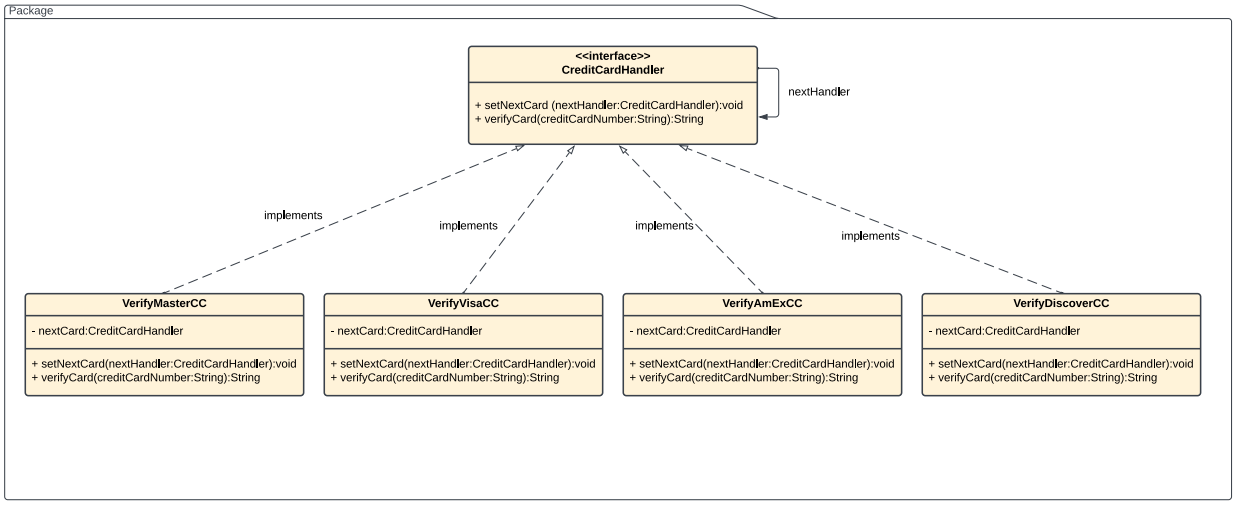
Once the credit card has been mapped to its relevant subclass, which could be Master Card, American Express, Visa or Discover, the next step is to generate a credit class object instance as per the specifications. To ensure that new credit card subclasses can be easily added in the future, it is important to implement an appropriate Creational design pattern for object creation; otherwise, the process of adding a new credit card subclass could be cumbersome.

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

Following two Design Patterns can be used to solve above problems

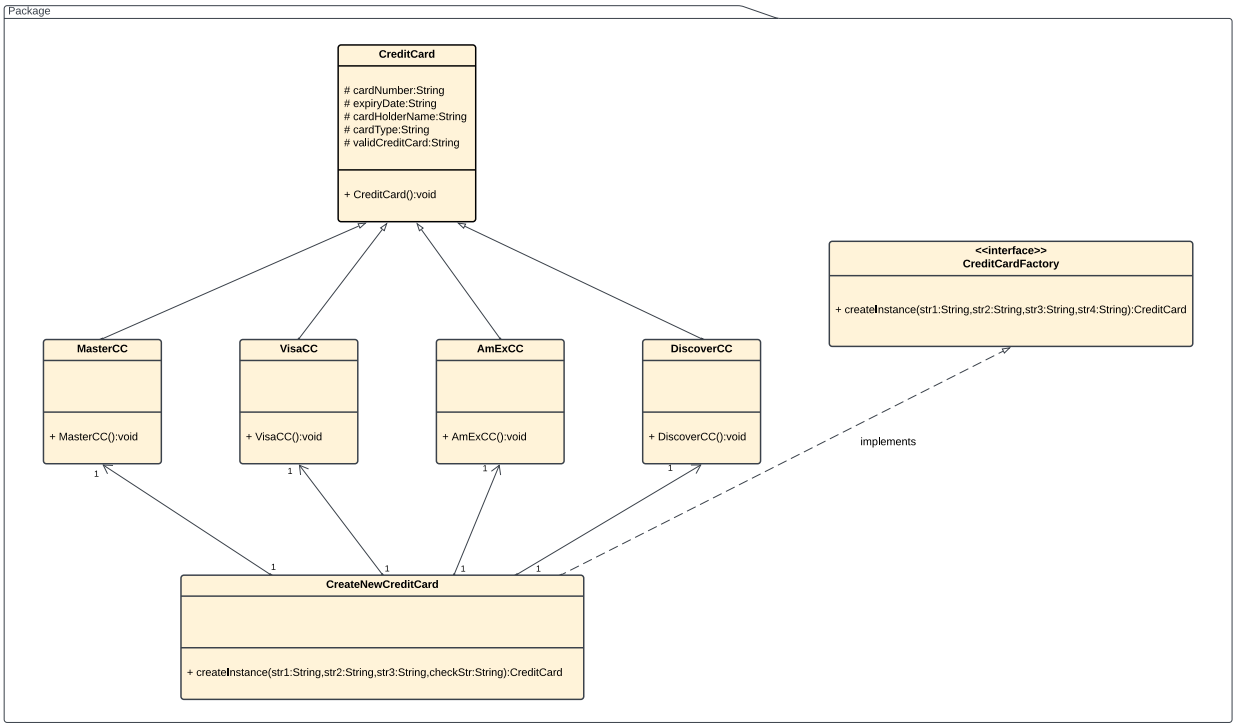
1. Chain Of Responsibility

The “Chain of Responsibility” behavioral pattern can be employed to determine the class of a credit card. This pattern involves sequentially passing requests through various handlers to determine which handler can verify the credit card's class. Each handler is responsible for checking if the credit card belongs to its class. If it doesn't, the handler passes the request to the next handler in the chain. This process continues until the credit card is verified with all available credit card classes. If a handler matches the credit card's class, it returns a string that matches the class name. If not, control passes to the next handler.



2. Factory Method

Once the subclass of the credit card type has been identified, the “Factory Method” creational design pattern can be employed to generate an instance for that specific type of card. This pattern is favored because it keeps the creation logic hidden from the user. Newly created objects can be referred to through a common interface.



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

Consequences of using Chain of Responsibility pattern

Advantages:

1. Single Responsibility Principle - Reduce the level of coupling by decoupling the classes that trigger operations.
2. The pattern can simplify the object design by reducing the number of direct connections between objects, leading to a more manageable and maintainable codebase.
3. Can control the order of request handling.
4. Open/Closed Principle -The pattern allows for easy extension of the system by adding new handlers that can handle new types of requests without affecting the existing code.

Disadvantages:

1. The pattern can increase the complexity of the code, especially when dealing with a large number of handlers in the chain. This can make it difficult to debug and maintain the code.
2. If no handler is able to handle a request, it may go unhandled, leading to unexpected behavior or errors.
3. Each request must be passed through the chain until a suitable handler is found, which can result in a performance overhead, especially if the chain is long or the request processing is complex.

Consequences of using Factory Method pattern

Advantages:

1. 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.
2. Avoids the strong coupling between the concrete classes and the creator.
3. Supports open-closed principle, by paving the way to create new types of instances without breaking the existing code.
4. It promotes code reusability, by allowing to reuse the same Factory class to create objects in different parts of the code.

Disadvantages:

1. Implementing the Factory Method Pattern can add complexity to the code, especially if there are many different types of objects to create.
2. It can add overhead to the code, due to the addition of many new sub classes.
3. It can be inflexible if we had to create objects with complex dependencies or if we need to change the way objects are created dynamically at runtime.

Part B Deliverables

Strategy Pattern has been used for parsing different Input file formats

1. Strategy pattern has advantage for swapping the algorithms used at runtime, which is the use case here for different file formats.
2. It isolates the implementation details of an algorithm from the code that is being used.
3. Replacement of Inheritance with Composition.
4. Open/Closed Principle – New Strategies can be introduced without having to change the context

Extended Class Diagram