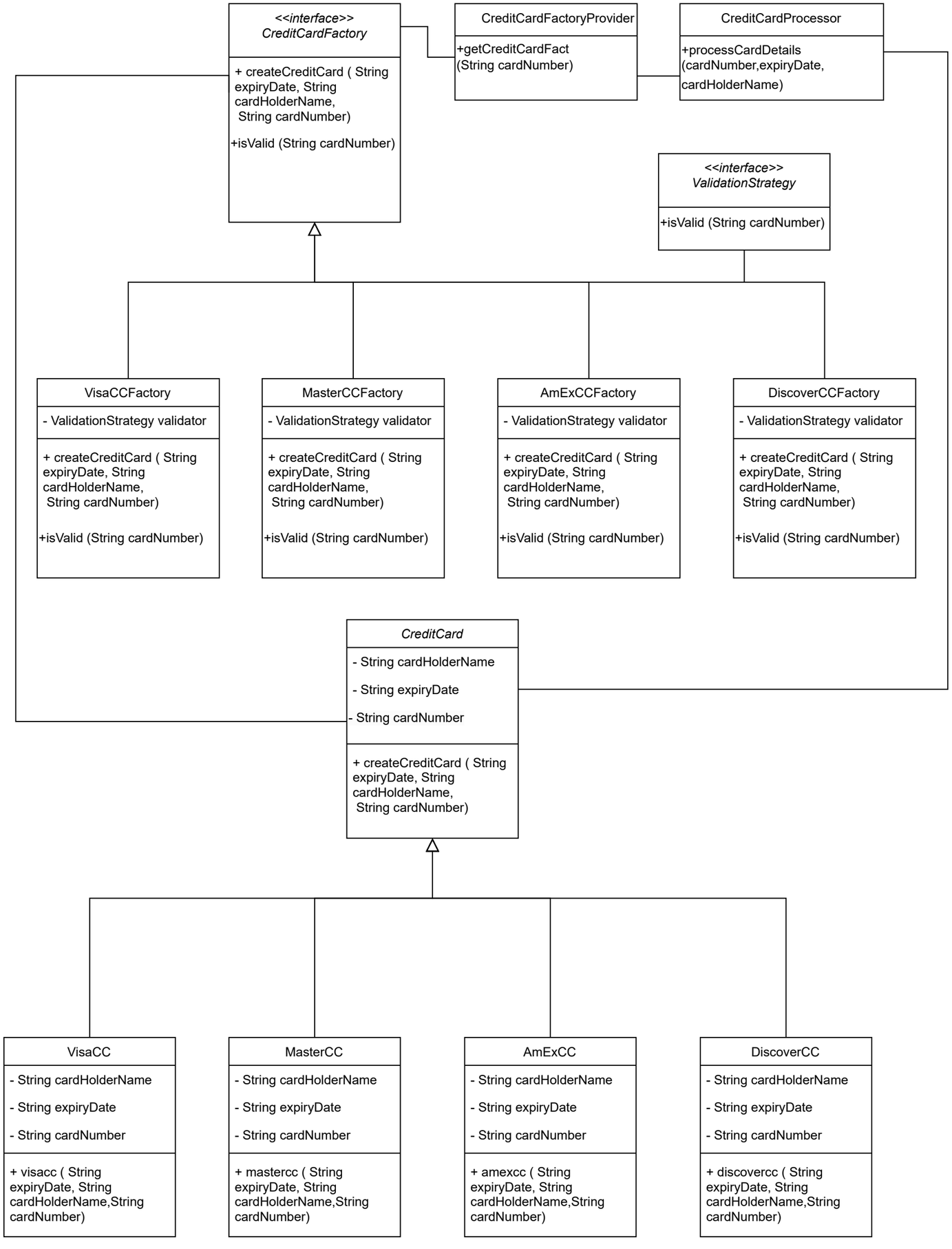

Primary Problem to Solve:

The core problem is to correctly parse a record from a CSV file that contains credit card details, ensure the card number adheres to the rules specific to each card issuer, and then create an instance of the corresponding subclass of **Credit Card.**

Secondary Problems to Solve:

Validation: Implementing validation logic for credit card numbers based on the issuer's rules.
Scalability: Designing a system that can easily accommodate new card issuer subclasses in the future without significant refactoring.
Object Creation: Ensuring the correct subclass of CreditCard is instantiated based on the credit card number.

Design Patterns to Use:

Factory Pattern:

The Factory pattern would be useful to encapsulate the logic of creating objects of the type **CreditCard**. This pattern will help in instantiating a subclass of **CreditCard** based on the validation of the card number.

Strategy Pattern:
The Strategy pattern would be ideal for encapsulating the validation logic for different types of credit cards into interchangeable algorithms.

Consequences of Using These Patterns:

Pros:

- **Flexibility**: New card types can be added with minimal changes to the existing code base.
- **Maintainability**: Separation of concerns makes it easier to maintain and update validation logic or object creation code.
- **Reusability**: The validation strategies can be used elsewhere if needed.

Cons:

- **Complexity**: Introducing multiple patterns and classes can complicate the system's structure.
- **Class proliferation**: Each new card type may require new classes for both the factory and validation strategy, leading to a large number of classes.