NAME: Kapil Gulani
SJSU ID: 017461314

CMPE-202: Individual Project Documentation

Github Repository: https://github.com/gopinathsjsu/individual-project-kapilgulani/tree/main

Primary Problem

The primary problem your project solves is parsing and processing credit card records from different file formats (CSV, JSON, XML). Each record includes a credit card number, expiration date, and cardholder's name. The main challenge
is to read these records, validate the credit card number, determine the card issuer, and create an instance of the appropriate credit card class.

Secondary Problems

- File Format Flexibility: The system needs to handle various input file formats (CSV, JSON, XML) and potentially accommodate new formats in the future.
- Credit Card Validation: Each credit card number must be validated to check if it's a legitimate number and identify the card issuer (Visa, MasterCard, AmericanExpress, Discover).

Design Patterns Used

1. Factory Pattern:

- Used in CreditCardFactory to create instances of different credit card types based on the card number.

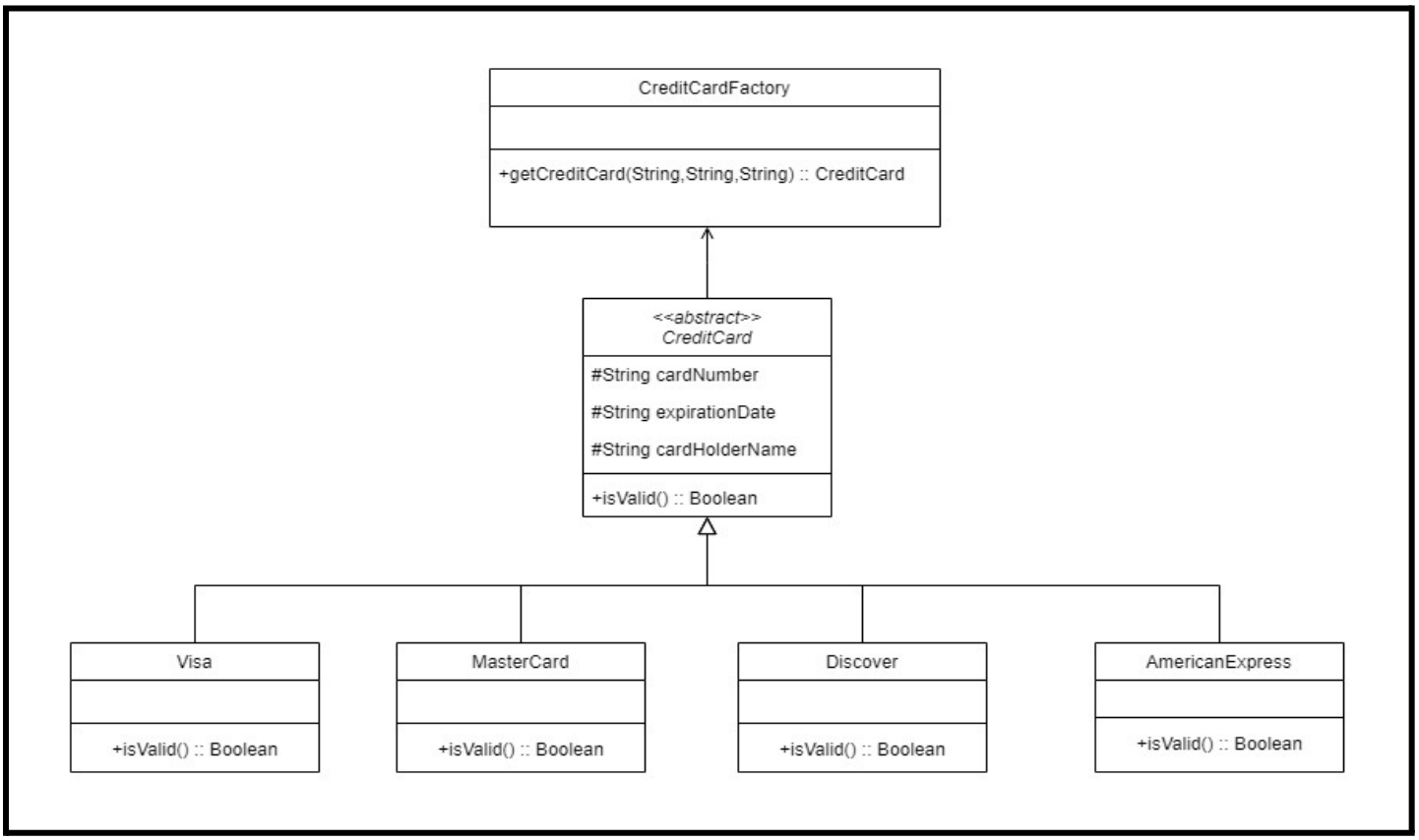
- This pattern allows easy extension to support new credit card types in the future.

2. Strategy Pattern:

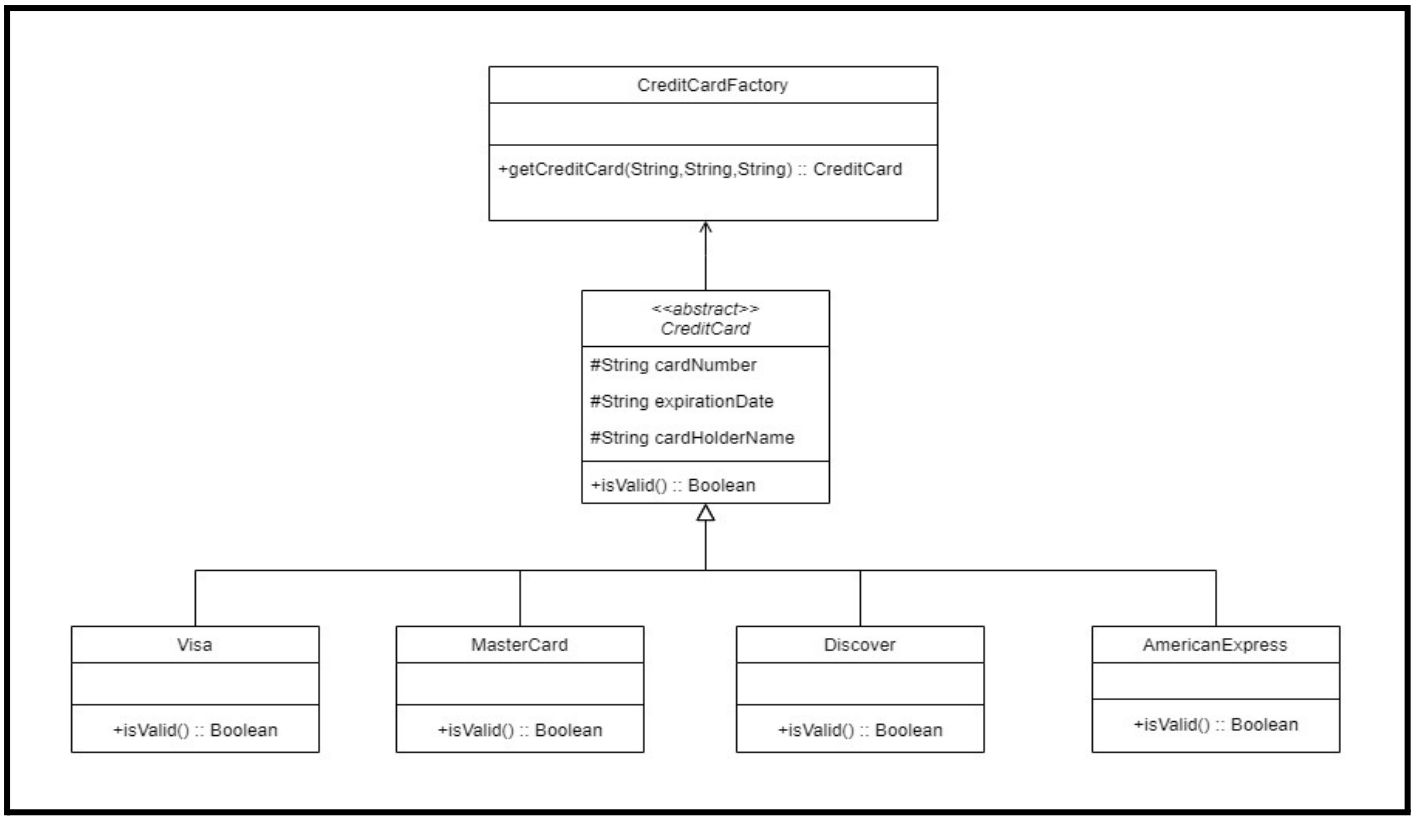
- Implemented through the FileParser interface with different strategies for parsing files (CSVParser, JSONParser, etc.).

- This pattern provides flexibility to add new file parsing strategies without modifying the existing code.

UML Class Diagram for Factory Pattern Implementation:



UML Class Diagram for Strategy Pattern Implementation:



Consequences of Using These Patterns:

Factory Pattern:
Pros: Simplifies object creation and centralizes logic.
Cons: The factory class may require modifications for new card types.

Strategy Pattern:
Pros: Enhances flexibility and scalability for file parsing.
Cons: Can introduce complexity with multiple strategy classes.