CMPE 202

Credit card problem

Git: https://github.com/gopinathsjsu/individual-project-sricharansjsu04

Sri Charan Reddy Mallu

017419779

Part 1: Designing the System

Primary Problem:

- The main challenge is to read credit card records from a CSV file, validate the card numbers, and instantiate the appropriate subclass of CreditCard based on the type of credit card (Visa, MasterCard, American Express, Discover).

Secondary Problems:

- Extensibility for Additional Card Types: The system should be designed to easily accommodate new credit card types.
- Validation of Card Numbers: Ensure that card numbers do not exceed 19 digits and adhere to issuer-specific formats.

Design Patterns:

- **Factory Method Pattern** (Creational Pattern):
    - Use for creating objects of CreditCard subclasses. This pattern allows the instantiation process to be deferred to subclasses, enabling the system to handle various types of credit cards dynamically.
- **Strategy Method Pattern**
    - The Strategy Pattern is suitable for situations where you want to define a family of algorithms, encapsulate each algorithm, and make them interchangeable. In this case, you need to define an algorithm for validating credit card numbers, and each credit card type can have its own validation strategy.

Consequences:

- **Factory Method**:
    - Provides flexibility in adding new credit card types without modifying existing code.
- **Strategy Pattern**:
    - Separates validation logic from the card objects, making it easy to modify or add new validation rules. However, it might overcomplicate the system for simple validation scenarios.

Part 2: Design Extension

Extend the design to handle multiple input file formats (CSV, JSON, XML) and produce corresponding outputs.

Extended Design:

Abstract Factory Pattern:

- This is used in conjunction with the Factory Pattern, in the factory package where different types of factories (*CSVOperationFactory*, *JSONOperationFactory*, *XMLOperationFactory*) are defined.
- The Abstract Factory Pattern provides an interface for creating families of related or dependent objects which are *createfileparser* and *createfilewriter* without specifying their concrete classes.

Strategy Pattern (Behavioral Pattern):

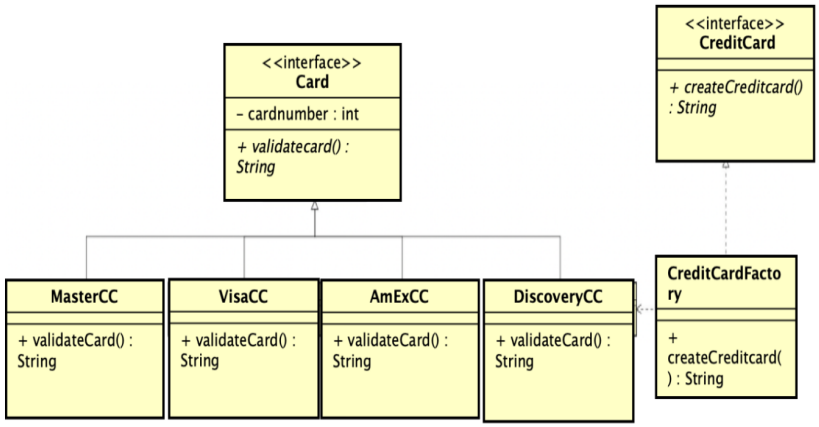
Due to the variety of file types, specific objects are created for each type and appropriate methods are employed depending on the input file.
To support various file formats, I employed a strategy design pattern. CsvFileparser, JsonFileParser, and XMLFileParse are the three interfaces I've developed. A file parser is added to all of these. As a result of the input, the behavior modifies.

Decorator Pattern:

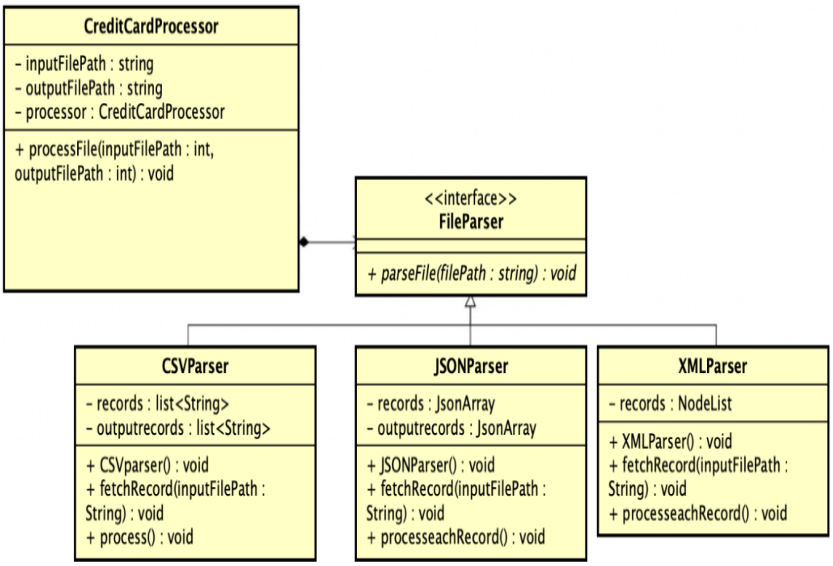
CreditCardWrapper wraps the *CreditCard* Class and acts as adapter when fileparses want to use the CreditCard for creating output objects in the required format.

Design Diagrams:

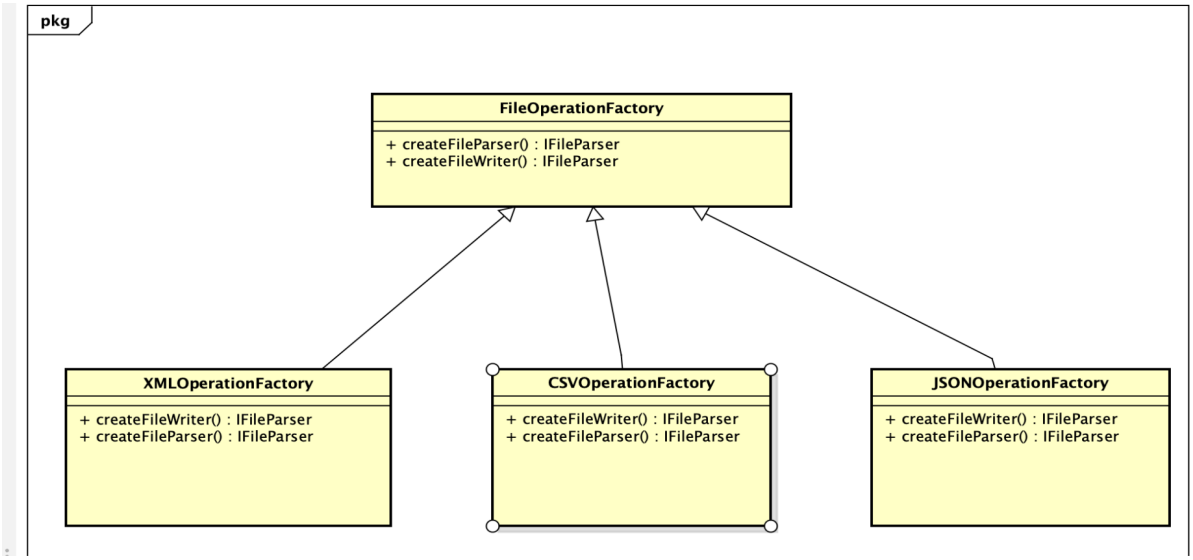
Factory Design Pattern



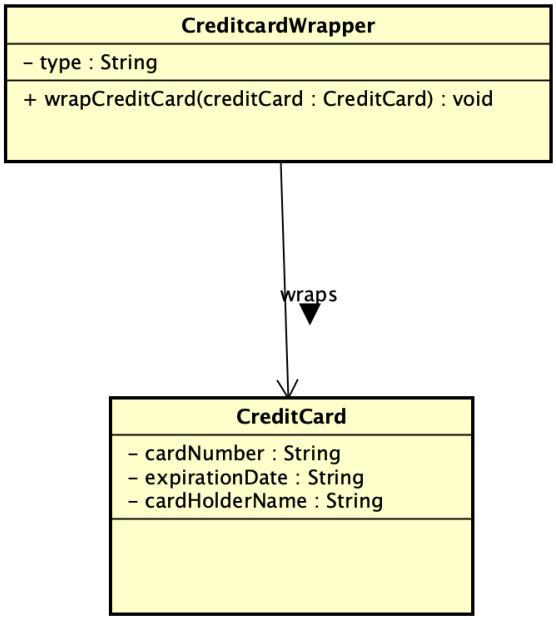
Strategy Design Pattern



Abstract Factory pattern:



Adapter pattern:


CreditcardWrapper
- type : String
+ wrapCreditCard(creditCard : CreditCard) : void
wraps
CreditCard
- cardNumber : String
- expirationDate : String
- cardHolderName : String

.