CMPE 202 Individual Project

Link to GitHub - https://github.com/Dhrithi432/202IndividualProject/tree/main

1.Describe what is the primary problem you are trying to solve.

The primary problem is to figure out if the provided credit card number is valid. And if it is a valid number, we should determine the card issuer using a design pattern.

2.Describe what are the secondary problems you try to solve.

The secondary problem is to choose a creational pattern to create appropriate card objects. For part2, we need to use a creational pattern to create different file format objects.

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

1. Strategy Pattern

Problem Addressed:

- Handling different file formats (CSV, JSON, XML, etc.) for input and output.
- Flexibility to add support for new file formats in the future.

How It's Used:

- Create a FileParserStrategy interface.
- Implement concrete strategies - CSVParserStrategy, JSONParserStrategy, and XMLParserStrategy.

Consequences:

- **Pros**: Enhances flexibility and reusability. Adding a new file format involves creating a new strategy without altering existing code.
- **Cons**: Can introduce a number of classes into the system, potentially increasing complexity.

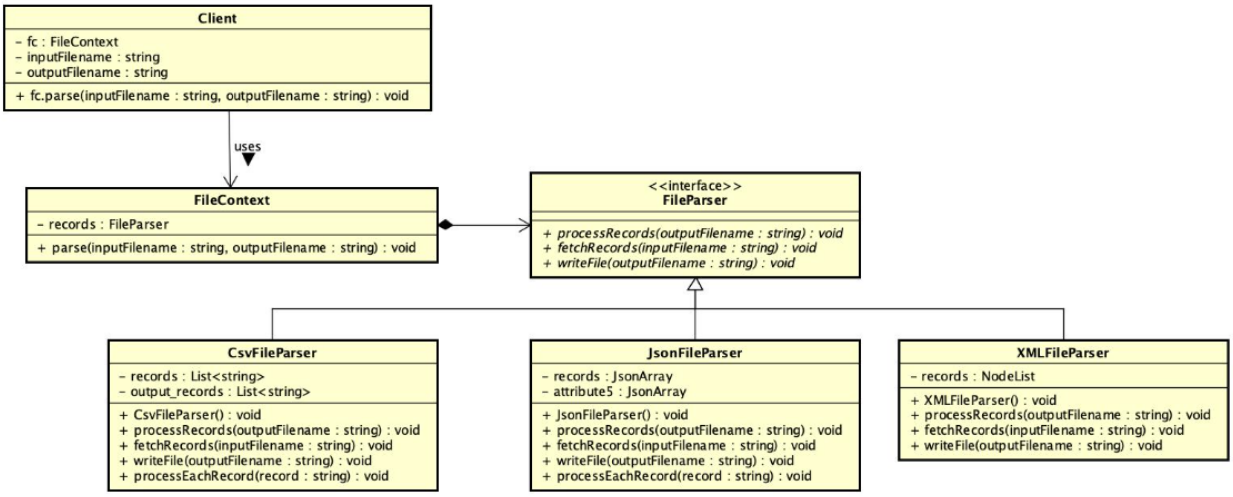

Figure 1. Strategy Pattern

2. Iterator Pattern

Problem Addressed:

Iterating over a collection of credit card records in the file without exposing its underlying representation.

How It's Used:

- Implement an Iterator interface with methods like hasNext() and next().
- Create a CreditCardIterator class that implements the Iterator interface, used to iterate over credit card records.
- The FileParserStrategy implementations return an iterator for traversing credit card records.

Consequences:

- **Pros**: Decouples the collection of credit card records from the algorithm that processes them. Supports different ways of traversing the collection.
- **Cons**: Can be overkill if your collection structure is simple and unlikely to change.

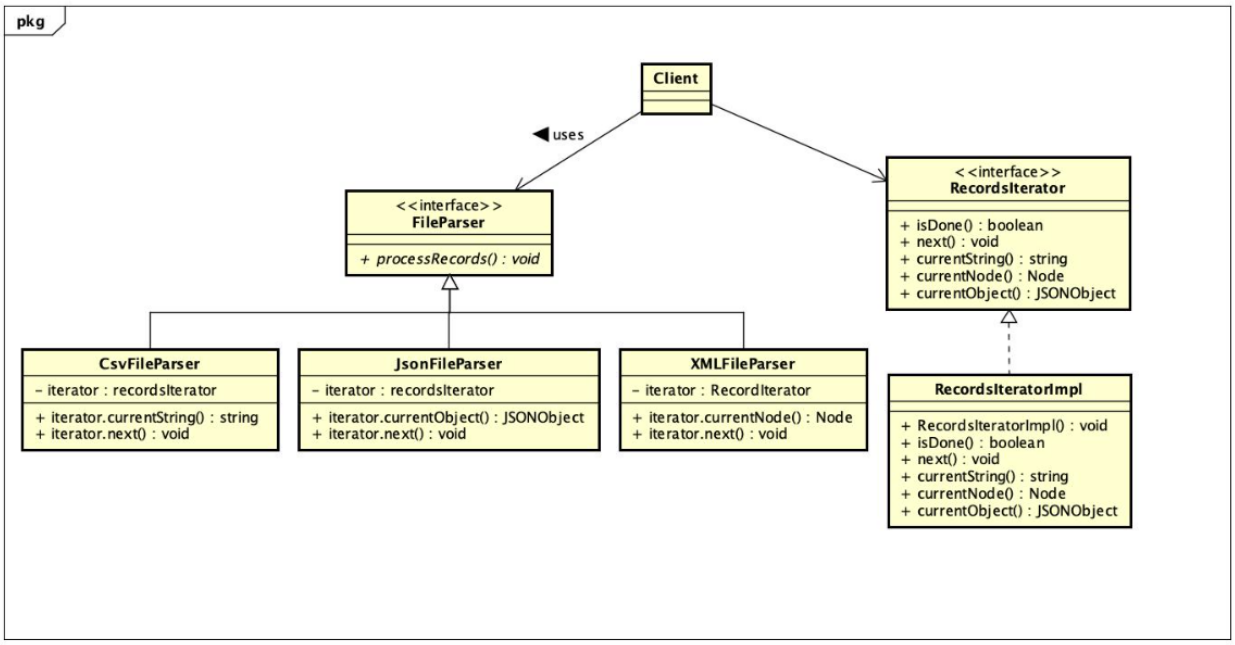

Figure 2. Iterator Pattern

3. Factory Method Pattern

Problem Addressed:

Creating instances of different CreditCard subclasses (like VisaCC, MasterCC, AmExCC) based on the credit card number.

How It's Used:

- Define a CreditCardFactory interface with a method like createCreditCard(String cardNumber).
- Implement concrete factories for each credit card type, which contain the logic to validate and instantiate the appropriate CreditCard subclass.
- Use these factories in your system to create CreditCard objects without specifying the exact class of the object that will be created.

Consequences:

- **Pros:** Provides a way to encapsulate object creation logic, making the system more modular and easier to maintain. Easy to add new types of credit cards.
- **Cons:** The number of classes in the system increases as each new credit card type requires a new factory class.

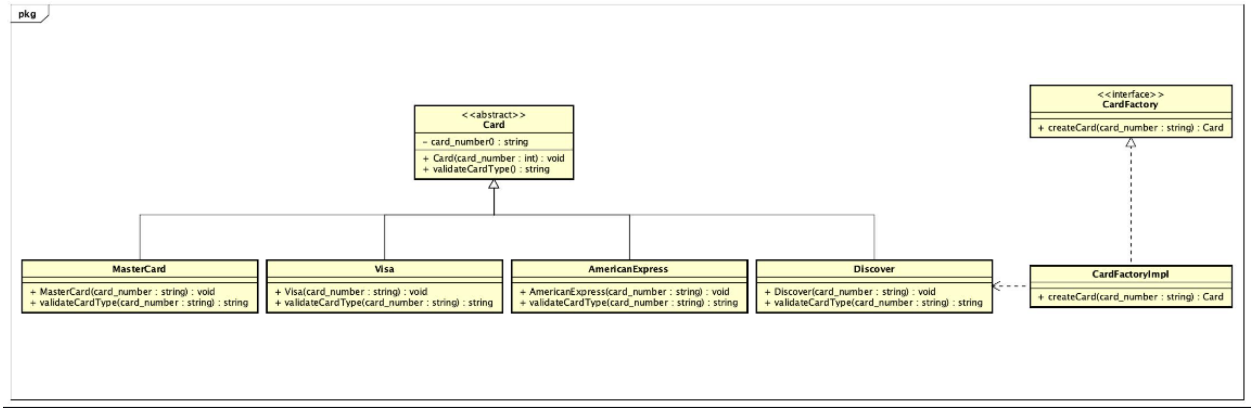

Figure 3. Factory Method Pattern

Integration of Patterns

These patterns work together to create a flexible, extensible system. The Strategy pattern allows for easy handling of different file formats, the Iterator pattern efficiently manages traversal of credit card records, and the Factory Method pattern effectively handles the creation of specific credit card type objects based on the credit card number. This design makes your system robust and adaptable to future requirements, such as adding support for new credit card types or file formats.