# Individual Project

CMPE 202
016043105
Shakshi Richhariya

# Primary Problem:

The main challenge we're addressing involves accurately determining different types of cards (VisaCC, MasterCC, AmExCC, etc.) using the provided data. Our goal is to correctly identify legitimate card types when the inputted credit card number is valid. Conversely, if the number is invalid, we aim to display a suitable error message.

# Proposed Approach:

To tackle our issue, we propose using the factory design pattern. This approach is a solid strategy for creating objects, centralizing this function while hiding its complexities from users. By implementing a factory, we simplify our coding, improve maintainability, and increase scalability. Importantly, this method supports easy addition of new object types or changes in the creation process, reducing the reliance on specific implementations in client code.

The factory pattern brings several advantages:

- It hides the details of object creation from the user.
- It creates a separation between implementation and client classes through inheritance.
- The specific object created depends on the input received.

Employing the factory method results in:

- Fewer application-specific classes in our code.
- Simplified class extension or modification as needed.
-

To expand, the factory pattern is a creational design pattern that centralizes object creation. It encapsulates the process of creating objects, promoting adaptability and detaching the client code from specific creation details. This approach enhances code modularity and upkeep, allowing for the easy introduction of new object types or alterations in the creation process without affecting the client code.

For this project, I have created a foundational class for the credit card, with specific card types deriving from this class. A factory method interacts with these classes, generating the appropriate objects based on the input, corresponding to the relevant card type.

# Secondary Problem:

A key hurdle involves creating a specific credit card class upon verifying a credit card number.

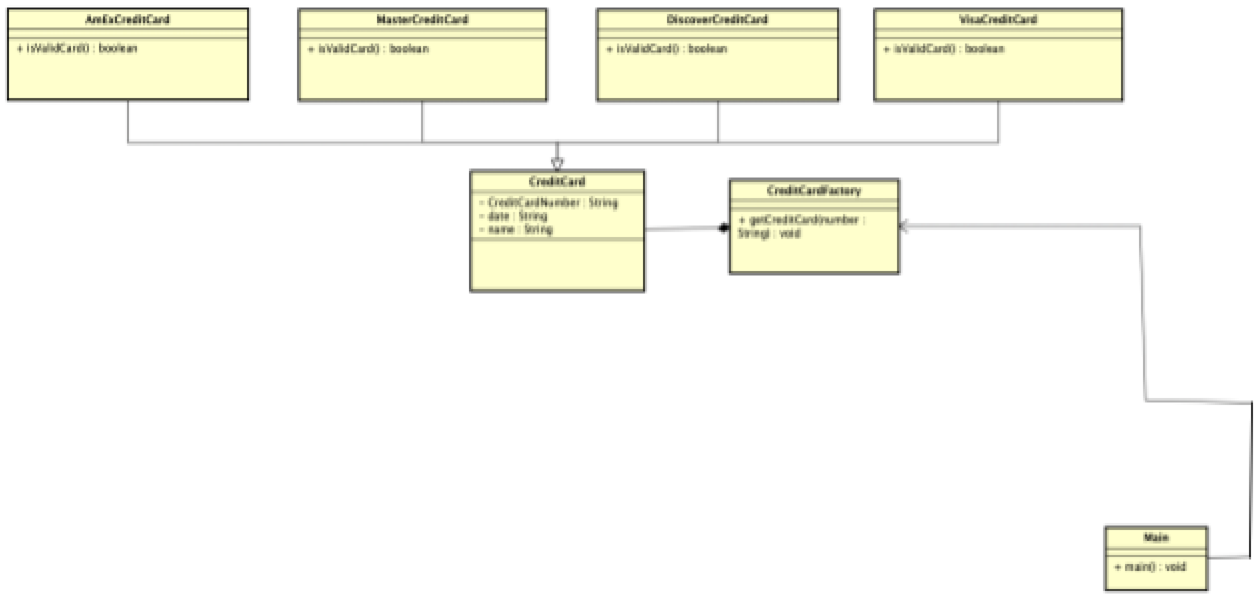
# Approach
The Factory design pattern provides a structured method to generate an object of the desired card type:

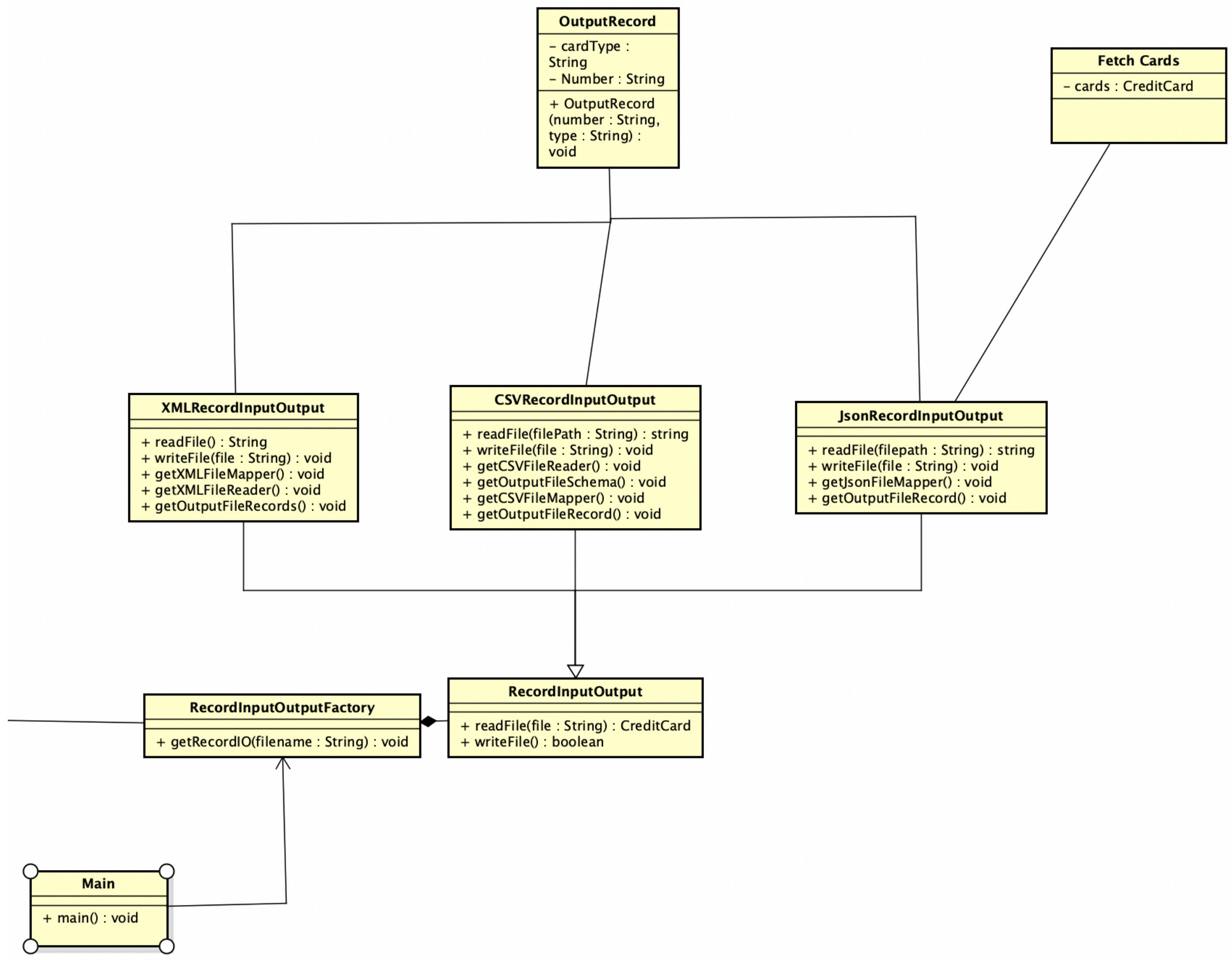
- Detection: The process begins by pinpointing a distinct feature, such as the initial digits of the credit card number, to classify the card type.
- Creation: Within the Factory class, a method is crafted to accept this distinct feature as input. This method employs a set of conditional statements to align the feature with the appropriate card type.
- Construction: Each conditional in this method triggers the generation of a particular credit card class object. The method then delivers this newly formed object.
- Implementation: The client code interacts solely with this Factory method, supplying the required features and in return, obtains a complete object of the correct type. The client remains oblivious to the specific subclasses or their creation methodology, enhancing modularity and maintainability.

# Consequences:

- Simplicity: Employing the Factory pattern eases the process of generating credit card objects from a CSV file. This layer of abstraction streamlines the client code, improving readability and maintainability.
- Flexibility: The Factory pattern is flexible to modifications. Introducing new card types into the Factory is straightforward and doesn't necessitate changes in the client code.

- Validation: Inclusion of validation logic within the Factory ensures that only legitimate credit card objects are constructed, reinforcing data accuracy and reducing errors.
- Consistency: This approach guarantees uniform creation of all credit card objects, simplifying debugging and upkeep.
- Decoupling: By employing the Factory pattern, the client code's dependence on specific credit card classes is eliminated, leading to a more modular and testable code structure.
- Validating Credit Card Numbers: Ensuring the credit card number adheres to specific rules for different card types (e.g., length, starting digits).
- Extensibility for Future Card Types: The system should be easily extendable to support additional credit card types without significant modifications to the existing code.
- Maintaining Readability and Manageability: As more card types are added, the system should remain easy to read and manage.


AmExCreditCard
+ isValidCard() : boolean
MasterCreditCard
+ isValidCard() : boolean
DiscoverCreditCard
+ isValidCard() : boolean
VisaCreditCard
+ isValidCard() : boolean
CreditCard
- CreditCardNumber : String
- date : String
- name : String
CreditCardFactory
+ getCreditCard(number : String) : void
Main
+ main() : void


OutputRecord
- cardType : String
- Number : String
+ OutputRecord (number : String, type : String) : void
Fetch Cards
- cards : CreditCard
XMLRecordInputOutput
+ readFile() : String
+ writeFile(file : String) : void
+ getXMLFileMapper() : void
+ getXMLFileReader() : void
+ getOutputFileRecords() : void
CSVRecordInputOutput
+ readFile(filePath : String) : string
+ writeFile(file : String) : void
+ getCSVFileReader() : void
+ getOutputFileSchema() : void
+ getCSVFileMapper() : void
+ getOutputFileRecord() : void
JsonRecordInputOutput
+ readFile(filepath : String) : string
+ writeFile(file : String) : void
+ getJsonFileMapper() : void
+ getOutputFileRecord() : void
RecordInputOutputFactory
+ getRecordIO(filename : String) : void
RecordInputOutput
+ readFile(file : String) : CreditCard
+ writeFile() : boolean
Main
+ main() : void

GitHub link:
https://github.com/gopinathsjsu/individual-project-shakshi