Part1 & Part2 Design

- Describe what is the primary problem you try to solve.

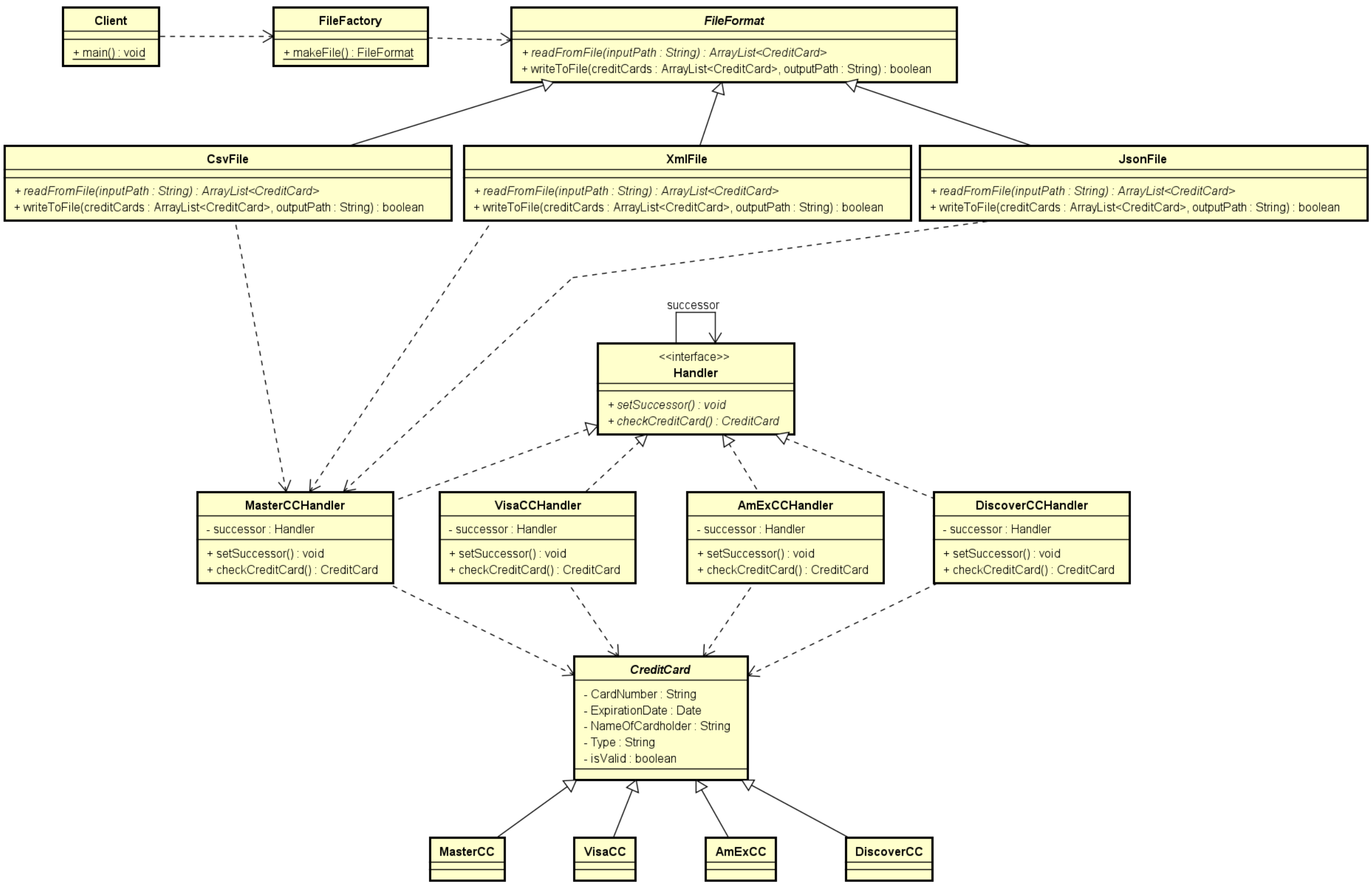
The primary problem is to figure out if the provided credit card number is valid. And if it is a valid number, we should determine the card issuer using behavior pattern.

- 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.

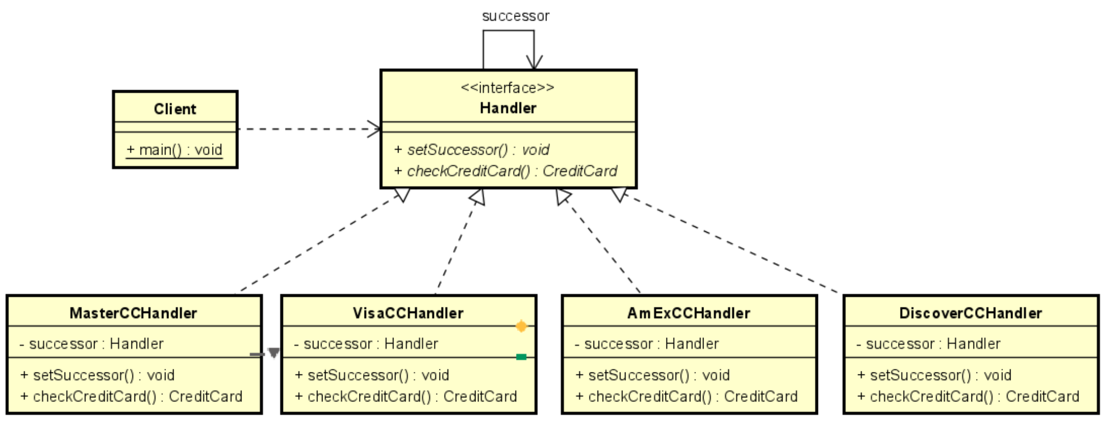
- Describe what design pattern(s) you use how (use plain text and diagrams).

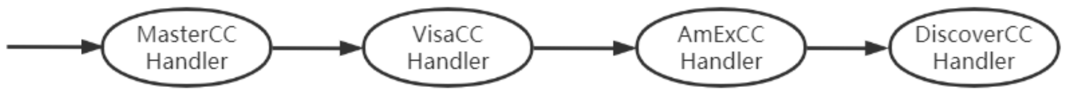
I choose to use Chain of Responsibility pattern to solve the first problem, and Factory Method pattern to solve the second problem.
The overall class diagram:


Client
+ main() : void
FileFactory
+ makeFile() : FileFormat
FileFormat
+ readFromFile(inputPath : String) : ArrayList<CreditCard>
+ writeToFile(creditCards : ArrayList<CreditCard>, outputPath : String) : boolean
CsvFile
+ readFromFile(inputPath : String) : ArrayList<CreditCard>
+ writeToFile(creditCards : ArrayList<CreditCard>, outputPath : String) : boolean
XmlFile
+ readFromFile(inputPath : String) : ArrayList<CreditCard>
+ writeToFile(creditCards : ArrayList<CreditCard>, outputPath : String) : boolean
JsonFile
+ readFromFile(inputPath : String) : ArrayList<CreditCard>
+ writeToFile(creditCards : ArrayList<CreditCard>, outputPath : String) : boolean
successor
<<interface>>
Handler
+ setSuccessor() : void
+ checkCreditCard() : CreditCard
MasterCCHandler
- successor : Handler
+ setSuccessor() : void
+ checkCreditCard() : CreditCard
VisaCCHandler
- successor : Handler
+ setSuccessor() : void
+ checkCreditCard() : CreditCard
AmExCCHandler
- successor : Handler
+ setSuccessor() : void
+ checkCreditCard() : CreditCard
DiscoverCCHandler
- successor : Handler
+ setSuccessor() : void
+ checkCreditCard() : CreditCard
CreditCard
- CardNumber : String
- ExpirationDate : Date
- NameOfCardholder : String
- Type : String
- isValid : boolean
MasterCC
VisaCC
AmExCC
DiscoverCC

Chain of Responsibility:

The Handler interface is defined to check the type of credit cards. And there are 4 concrete handlers (MasterCCHandler, VisaCCHandler, AmExCCHandler, DiscoverCCHandler) which implement the interface.
Each concrete handler can access its successor. When a credit card number is received from a file, it is passed to MasterCCHandler first, then go through a chain to determine the issuer of this card. If none of these 4 handlers is responsible of handling this number, this credit card is invalid.
If there are more subclasses for other types of credit card, the related concrete handler can be easily added at the end of the chain.

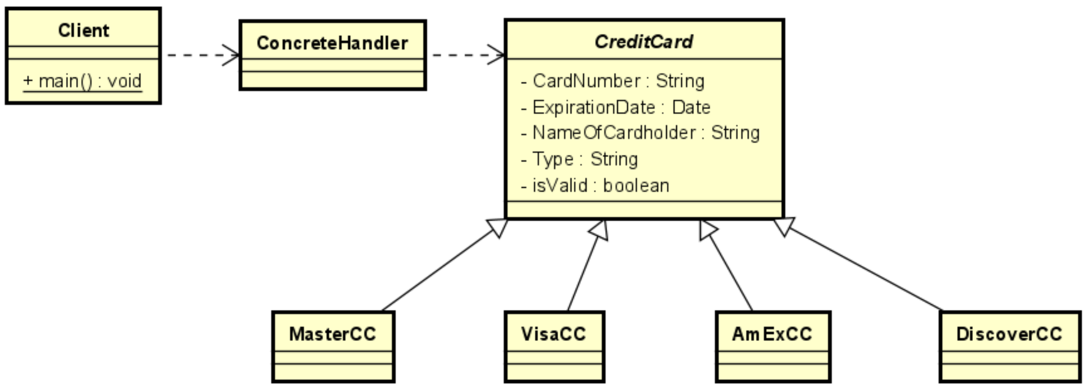


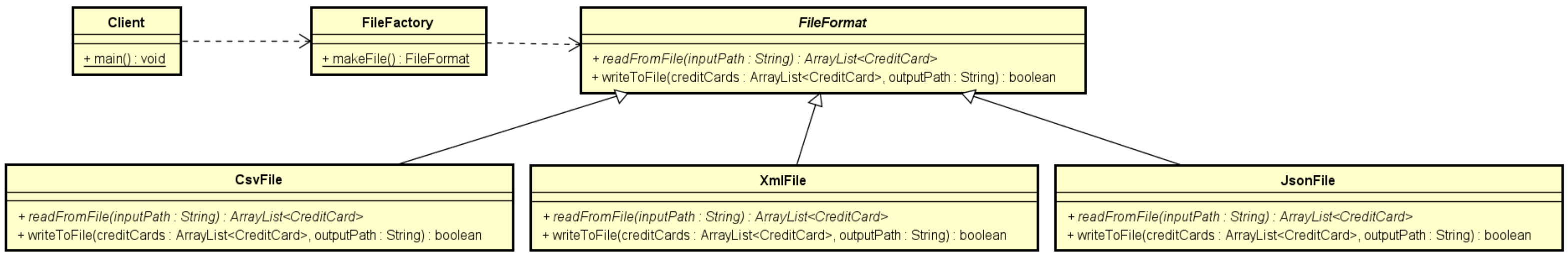


Factory Method:

1. For part1 creating card objects, we define an abstract class CreditCard, and there are 4 subclasses (MasterCC, VisaCC, AmExCC, DiscoverCC) which extend the CreditCard.
   When a ConcreteHandler finds out the card number from the file is exactly its type, it creates corresponding type of credit card object. For example, if MasterCCHandler figures out the card is Master credit card, it will create a MasterCC object. Therefore, the client does not know which subclass is created. And card objects are created by 4 subclasses.
2. For part2 creating File objects, we define a FileFactory class, and a FileFormat abstract class. There are 3 subclasses (CsvFile, XmlFile, JsonFile) which extend the FileFormat.
   When the client enters filename in the console, FileFactory can determine which file object to create base on the input. The FileFactory’s only job is to create

corresponding file object. Therefore, the client does not know which file object is going to create.




Client
+ main() : void
FileFactory
+ makeFile() : FileFormat
FileFormat
+ readFromFile(inputPath : String) : ArrayList<CreditCard>
+ writeToFile(creditCards : ArrayList<CreditCard>, outputPath : String) : boolean
CsvFile
+ readFromFile(inputPath : String) : ArrayList<CreditCard>
+ writeToFile(creditCards : ArrayList<CreditCard>, outputPath : String) : boolean
XmlFile
+ readFromFile(inputPath : String) : ArrayList<CreditCard>
+ writeToFile(creditCards : ArrayList<CreditCard>, outputPath : String) : boolean
JsonFile
+ readFromFile(inputPath : String) : ArrayList<CreditCard>
+ writeToFile(creditCards : ArrayList<CreditCard>, outputPath : String) : boolean

- Describe the consequences of using this/these pattern(s).

Chain of Responsibility:

1. The sender of the request does not need to know which handler object should handle the request.
2. Each handler object in the chain has a successor, if a handler figures out the card issuer, it does; otherwise it will pass the request to its successor. If none of the handlers could figure out the card type, the card is invalid.
3. The performance of this pattern is not guaranteed, since it may fall off the end of the chain if no handler object could handle it (invalid card number).
4. It is easy to add or remove credit card type dynamically by adding or removing members in the chain.

Factory Method:

1. The client does not know which class object to instantiate. The factory allows you to create object without specifying the exact class of object that will be created.
2. It is easy to add or remove file format in the future.