CMPE202 Individual Project

Part-1:

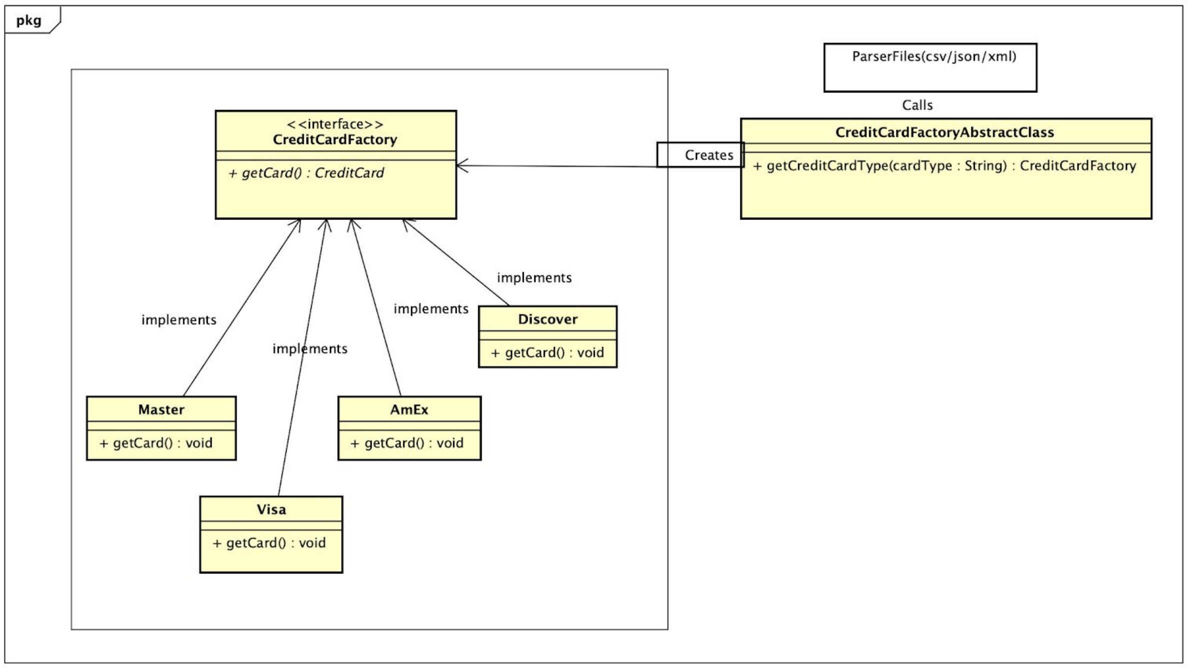
1. Describe what is the primary problem you try to solve?
    - ➢ The primary problem is how to create a workflow using design patterns that identify common communication patterns among objects to make sure that our application correctly identifies a credit card type or able to classify it as an invalid card.
    - ➢ Solved the problem using behavioural design patterns: Strategy Design Pattern and Chain of responsibility Design Pattern

2. Describe what are the secondary problems you try to solve (if there are any)?
    - ➢ The secondary problem was how to deal with object creation mechanisms, trying to create objects in a manner suitable to validate the credit card type. The basic form of object creation could result in design problems or in added complexity to the design.
    - ➢ Solved this problem using creating design pattern: Factory Design Pattern

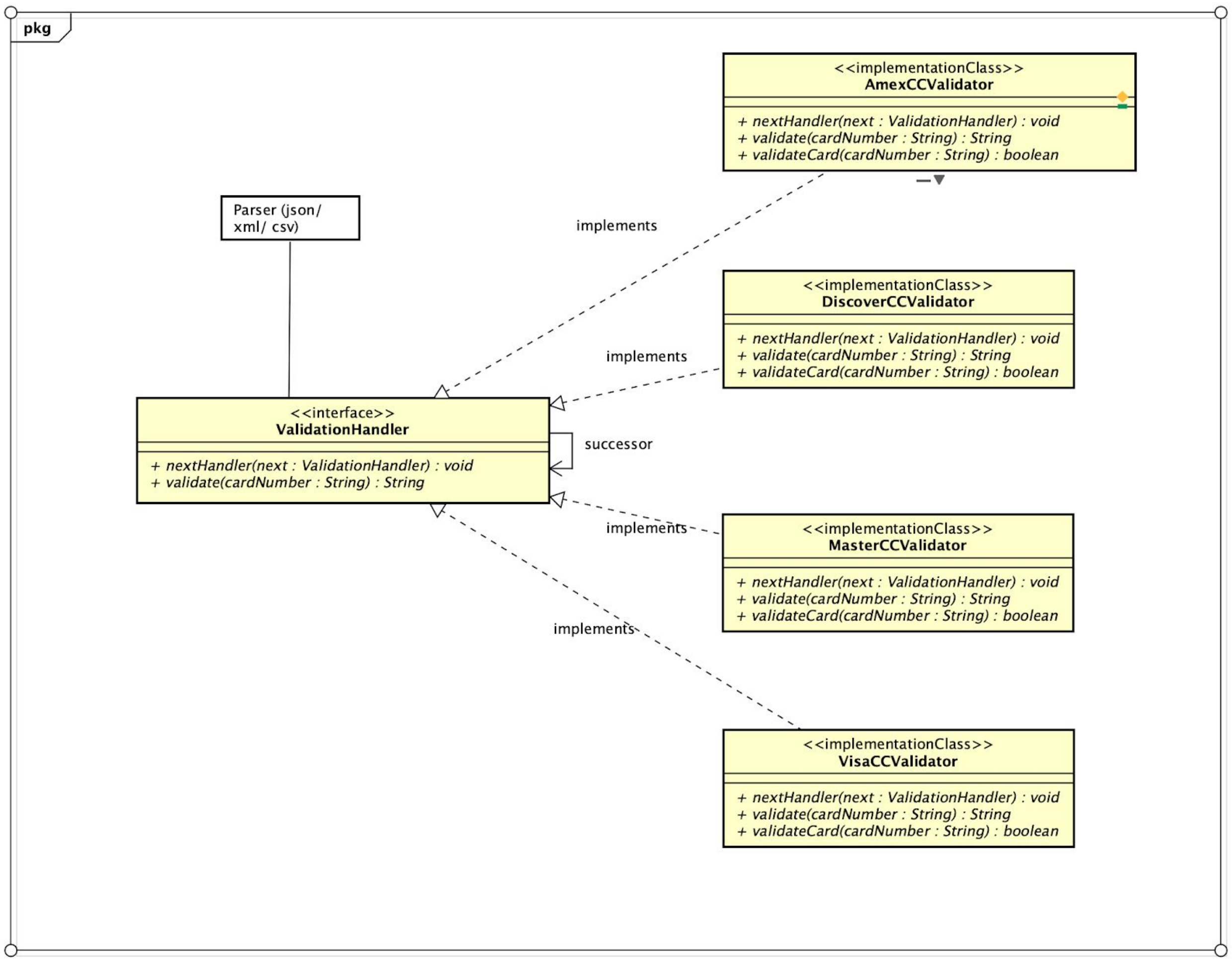
3. Describe what design pattern(s) you use and how (use plain text and diagrams)?

    Design patterns: Factory design pattern and Chain of responsibility design pattern.

    1. Factory design pattern:
        - ➢ After parsing the csv file, we get the credit card number in each entry in the csv file, now we have to validate that number for each of the four credit card categories and if it matches with any of the given regular expression patterns for each of the four categories, we have to someway communicate to the parser that this credit card belong to one of the following category.
        - ➢ For this, factory pattern can be used, it abstracts the credit card classes, but using an abstract class(CreditCardFactoryAbstractClass), we create an object of these credit cards using a factory(CreditCardFactory).
        - ➢ Based on the string we return from the parser, it creates a suitable object of corresponding credit card.



2. Chain of responsibility design pattern:
   - Once you get credit card entry after parsing the CSV file, there should be a process in which the credit card number must be validated in order to know which of the four categories it belongs to.
   - Chain of responsibility pattern can be used to sequentially check whether the particular card number belongs to the present category, if not it is passed on to check the next category. This entire process is controlled by the ValidationHandler interface which passes the credit card number to the four validators - AmExCCValidator, MasterCCValidator, VisaCCValidator, DiscoverCCValidator.
   - All the four validators implement the interface and passon the card number to the next when it does not match with itself.


pkg
Parser (json/ xml/ csv)
<<interface>>
ValidationHandler
+ nextHandler(next : ValidationHandler) : void
+ validate(cardNumber : String) : String
successor
implements
implements
implements
implements
<<implementationClass>>
AmexCCValidator
+ nextHandler(next : ValidationHandler) : void
+ validate(cardNumber : String) : String
+ validateCard(cardNumber : String) : boolean
<<implementationClass>>
DiscoverCCValidator
+ nextHandler(next : ValidationHandler) : void
+ validate(cardNumber : String) : String
+ validateCard(cardNumber : String) : boolean
<<implementationClass>>
MasterCCValidator
+ nextHandler(next : ValidationHandler) : void
+ validate(cardNumber : String) : String
+ validateCard(cardNumber : String) : boolean
<<implementationClass>>
VisaCCValidator
+ nextHandler(next : ValidationHandler) : void
+ validate(cardNumber : String) : String
+ validateCard(cardNumber : String) : boolean

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

Consequences of Factory Design Pattern:

Pros:

1) It decoupled the business logic of creation of a card creation classes from the actual logic of finding the credit card type in the parsers
2) It gives scope to add new credit card types in future
○ The creator is not tightly coupled to any ConcreteProduct.
3) Allows you to change the design of your application more readily.
○ Makes our code more robust, less coupled and easy to extend.
4) Provides abstraction between implementation and client classes through inheritance.

Cons:

1) Parsers might have to subclass the CreditCardFactory interface just to create a particular credit card Object.
   ○ The client now must deal with another point of evolution.
2) Makes code more difficult to read as all of your code is behind an abstraction that may in turn hide abstractions.
3) Sometimes making too many objects often can decrease performance.

Consequences of Chain of responsibility Design Pattern:

Pros:

1) It decouples the file parsers and its corresponding credit card validators
   ○ frees an object from knowing which other object handles a request
   ○ both the receiver(file parsers) and the sender(validators) have no explicit knowledge of each other
2) Simplifies your object
   ○ it does not have to know the chain's structure to keep direct references to its members
   ○ keeps a single reference to their successor
3) gives you added flexibility in distributing responsibilities among objects. It allows you to add or remove responsibilities dynamically by changing the members or order of the chain.

Cons:

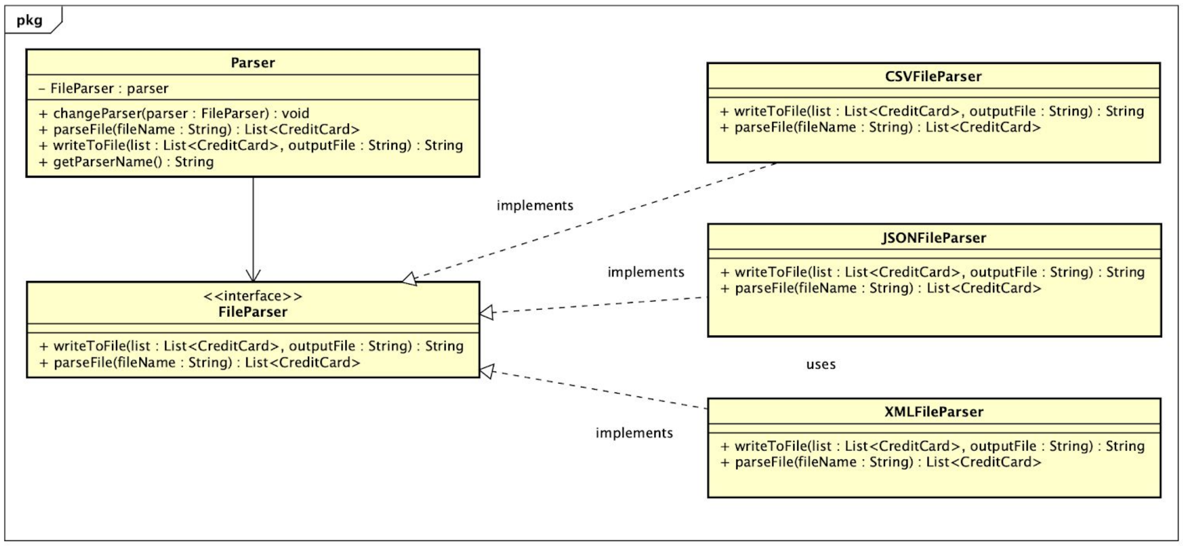
1) One drawback of this pattern is that the execution of the request is not guaranteed. It may fall off the end of the chain if no object handles it.
2) Another drawback is that it can be hard to observe and debug at runtime.

Part -2:

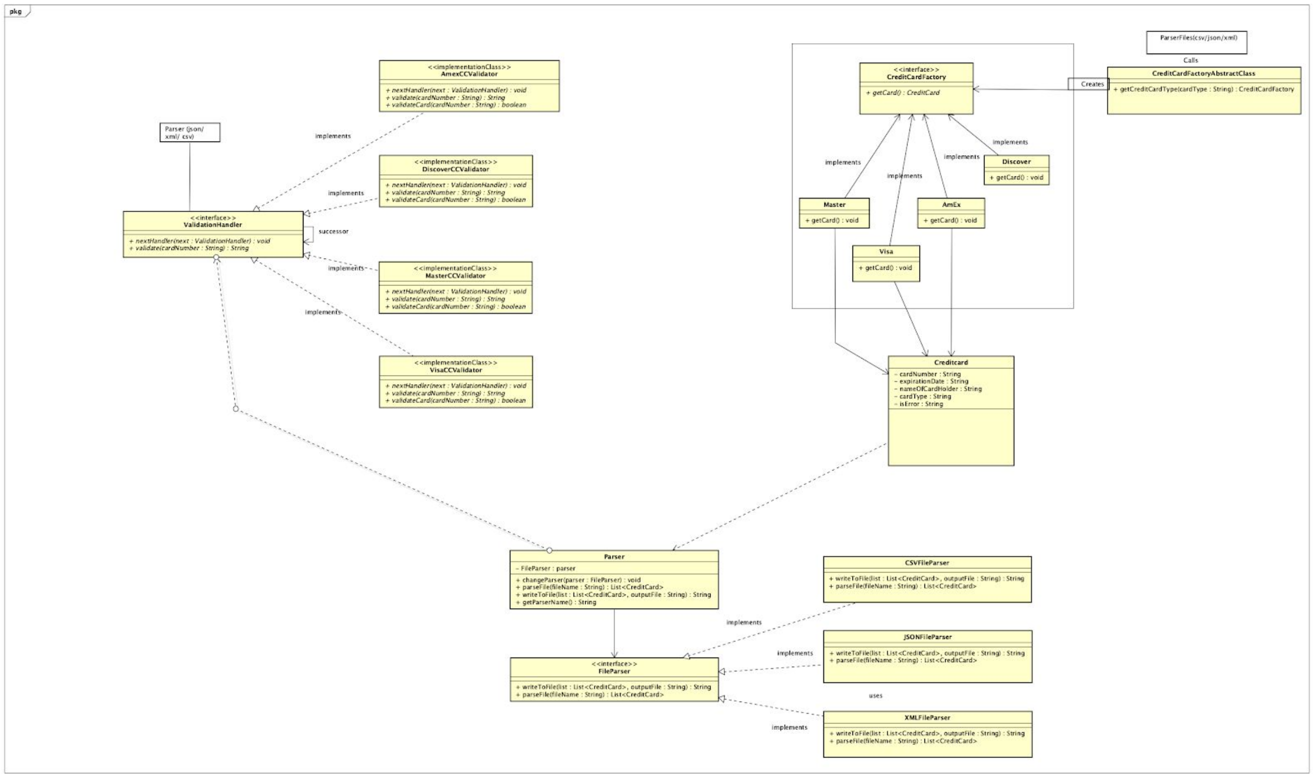
Extending the above design to xml and json formats:

➢ I used the Strategy design pattern to extend this file formatting to xml and json. It also allows us to implement this functionality to new file formats in the future.

- ➢ Using this pattern, we can encapsulate interface(FileParser) details in a base class, and bury implementation details in derived classes(CSVFileParser, JSONFileParser, XMLFileParser)


pkg
Parser
− FileParser : parser
+ changeParser(parser : FileParser) : void
+ parseFile(fileName : String) : List<CreditCard>
+ writeToFile(list : List<CreditCard>, outputFile : String) : String
+ getParserName() : String
<<interface>>
FileParser
+ writeToFile(list : List<CreditCard>, outputFile : String) : String
+ parseFile(fileName : String) : List<CreditCard>
CSVFileParser
+ writeToFile(list : List<CreditCard>, outputFile : String) : String
+ parseFile(fileName : String) : List<CreditCard>
JSONFileParser
+ writeToFile(list : List<CreditCard>, outputFile : String) : String
+ parseFile(fileName : String) : List<CreditCard>
XMLFileParser
+ writeToFile(list : List<CreditCard>, outputFile : String) : String
+ parseFile(fileName : String) : List<CreditCard>
implements
implements
uses
implements

- ➢ So to build the entire system which finds the credit card type, I used Strategy, Chain of Responsibility and Factory design patterns. Check below the entire system design of the credit card type finder.
- ➢ This entire design will allow to add a new credit card type/ new file format with ease without changing much of the code, just by creating classes and their corresponding validators.
- ➢ This design makes an abstraction between these modules and gives flexibility to the creators.


pkg
<<implementationClass>>
AmexCCValidator
+ nextHandler(next : ValidationHandler) : void
+ validate(cardNumber : String) : String
+ validateCard(cardNumber : String) : boolean
Parser (json/ xml/ csv)
implements
<<implementationClass>>
DiscoverCCValidator
+ nextHandler(next : ValidationHandler) : void
+ validate(cardNumber : String) : String
+ validateCard(cardNumber : String) : boolean
implements
<<interface>>
ValidationHandler
+ nextHandler(next : ValidationHandler) : void
+ validate(cardNumber : String) : String
successor
implements
<<implementationClass>>
MasterCCValidator
+ nextHandler(next : ValidationHandler) : void
+ validate(cardNumber : String) : String
+ validateCard(cardNumber : String) : boolean
implements
<<implementationClass>>
VisaCCValidator
+ nextHandler(next : ValidationHandler) : void
+ validate(cardNumber : String) : String
+ validateCard(cardNumber : String) : boolean
ParserFiles(csv/json/xml)
Calls
<<interface>>
CreditCardFactory
+ getCard() : CreditCard
Creates
CreditCardFactoryAbstractClass
+ getCreditCardType(cardType : String) : CreditCardFactory
implements
implements
implements
implements
Discover
+ getCard() : void
Master
+ getCard() : void
AmEx
+ getCard() : void
Visa
+ getCard() : void
Creditcard
- cardNumber : String
- expirationDate : String
- nameOfCardHolder : String
- cardType : String
- isError : String
Parser
- FileParser : parser
+ changeParser(parser : FileParser) : void
+ parseFile(fileName : String) : List<CreditCard>
+ writeToFile(list : List<CreditCard>, outputFile : String) : String
+ getParserName() : String
CSVFileParser
+ writeToFile(list : List<CreditCard>, outputFile : String) : String
+ parseFile(fileName : String) : List<CreditCard>
implements
<<interface>>
FileParser
+ writeToFile(list : List<CreditCard>, outputFile : String) : String
+ parseFile(fileName : String) : List<CreditCard>
implements
JSONFileParser
+ writeToFile(list : List<CreditCard>, outputFile : String) : String
+ parseFile(fileName : String) : List<CreditCard>
uses
implements
XMLFileParser
+ writeToFile(list : List<CreditCard>, outputFile : String) : String
+ parseFile(fileName : String) : List<CreditCard>