Individual Project Deliverables

Part1:

Design the method(s) (and maybe additional classes) that reads a record from the file, verifies that the credit card number is a possible account number, and creates an instance of the appropriate credit card class

Primary Problem:

- Figure out what kind of card a specific record is about, as the input is dynamic contains multiple types of credit card type class records.
  Solution: pass all the input records to all classes like Master, Visa, Amex, discover in a sequence and implement separate handling logic at each class to validate the record.

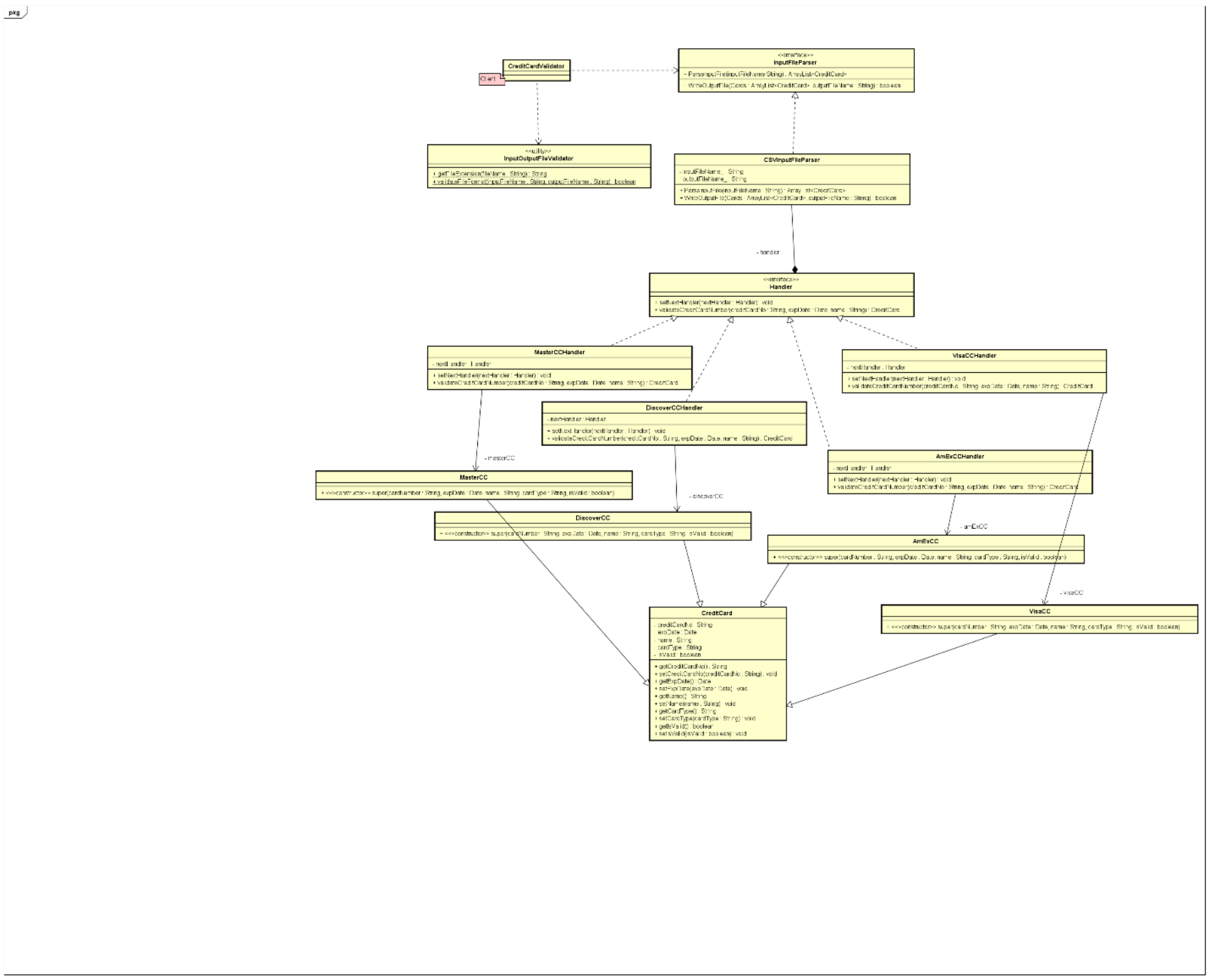
Secondary Problem:

- How to create the appropriate objects belonging to credit card type class.
- Solution: Let the handling logic itself instantiate respective class objects to initialize the credit card details and dump the data in output.

Above problems are solved by implementing below stated design pattern

Design Pattern:

Chain of Responsibility: As we the application requires to validate credit card records, one by one and these records are contained in a input file. The application is unaware of what type of data records each line in file has. So, I have implemented chain of responsibility as my choice of design pattern to solve both primary and secondary problems. We have a Handler that defines an interface for handling requests. Also, it implements the successor link. When a client (CreditCardValitor the main class) issues a request, the request propagates along the chain until a ConcreteHandler(MasterCC Handler, DiscoverCC Handler etc) object takes responsibility for handling it. If the ConcreteHandler can handle the request, it does so; otherwise it forwards the request to its successor. Each concrete handler has each credit card type validation logic and calls the specific constructor CreditCard class like MasterCC, DiscoverCC, VisaCC etc to create objects.


pkg
CreditCardValidator
Client
<<interface>>
InputFileParser
+ ParseInputFile(inputFileName:String) : ArrayList<CreditCard>
+ WriteOutputFile(Cards : ArrayList<CreditCard>, outputFileName : String) : boolean
<<utility>>
InputOutputFileValidator
+ getFileExtension(fileName : String) : String
+ validateFileFormat(inputFileName : String, outputFileName : String) : boolean
CSVInputFileParser
- inputFileName_ : String
- outputFileName_ : String
+ ParseInputFile(inputFileName : String) : ArrayList<CreditCard>
+ WriteOutputFile(Cards : ArrayList<CreditCard>, outputFileName : String) : boolean
- handler
<<interface>>
Handler
+ setNextHandler(nextHandler : Handler) : void
+ validateCreditCardNumber(creditCardNo : String, expDate : Date, name : String) : CreditCard
MasterCCHandler
- nextHandler : Handler
+ setNextHandler(nextHandler : Handler) : void
+ validateCreditCardNumber(creditCardNo : String, expDate : Date, name : String) : CreditCard
VisaCCHandler
- nextHandler : Handler
+ setNextHandler(nextHandler : Handler) : void
+ validateCreditCardNumber(creditCardNo : String, expDate : Date, name : String) : CreditCard
DiscoverCCHandler
- nextHandler : Handler
+ setNextHandler(nextHandler : Handler) : void
+ validateCreditCardNumber(creditCardNo : String, expDate : Date, name : String) : CreditCard
AmExCCHandler
- nextHandler : Handler
+ setNextHandler(nextHandler : Handler) : void
+ validateCreditCardNumber(creditCardNo : String, expDate : Date, name : String) : CreditCard
- masterCC
MasterCC
+ <<constructor>> super(cardNumber : String, expDate : Date, name : String, cardType : String, isValid : boolean)
- discoverCC
DiscoverCC
+ <<constructor>> super(cardNumber : String, expDate : Date, name : String, cardType : String, isValid : boolean)
- amExCC
AmExCC
+ <<constructor>> super(cardNumber : String, expDate : Date, name : String, cardType : String, isValid : boolean)
- visaCC
VisaCC
+ <<constructor>> super(cardNumber : String, expDate : Date, name : String, cardType : String, isValid : boolean)
CreditCard
- creditCardNo : String
- expDate : Date
- name : String
- cardType : String
- isValid : boolean
+ getCreditCardNo() : String
+ setCreditCardNo(creditCardNo : String) : void
+ getExpDate() : Date
+ setExpDate(expDate : Date) : void
+ getName() : String
+ setName(name : String) : void
+ getCardType() : String
+ setCardType(cardType : String) : void
+ getIsValid() : boolean
+ setIsValid(isValid : boolean) : void

Part 2:

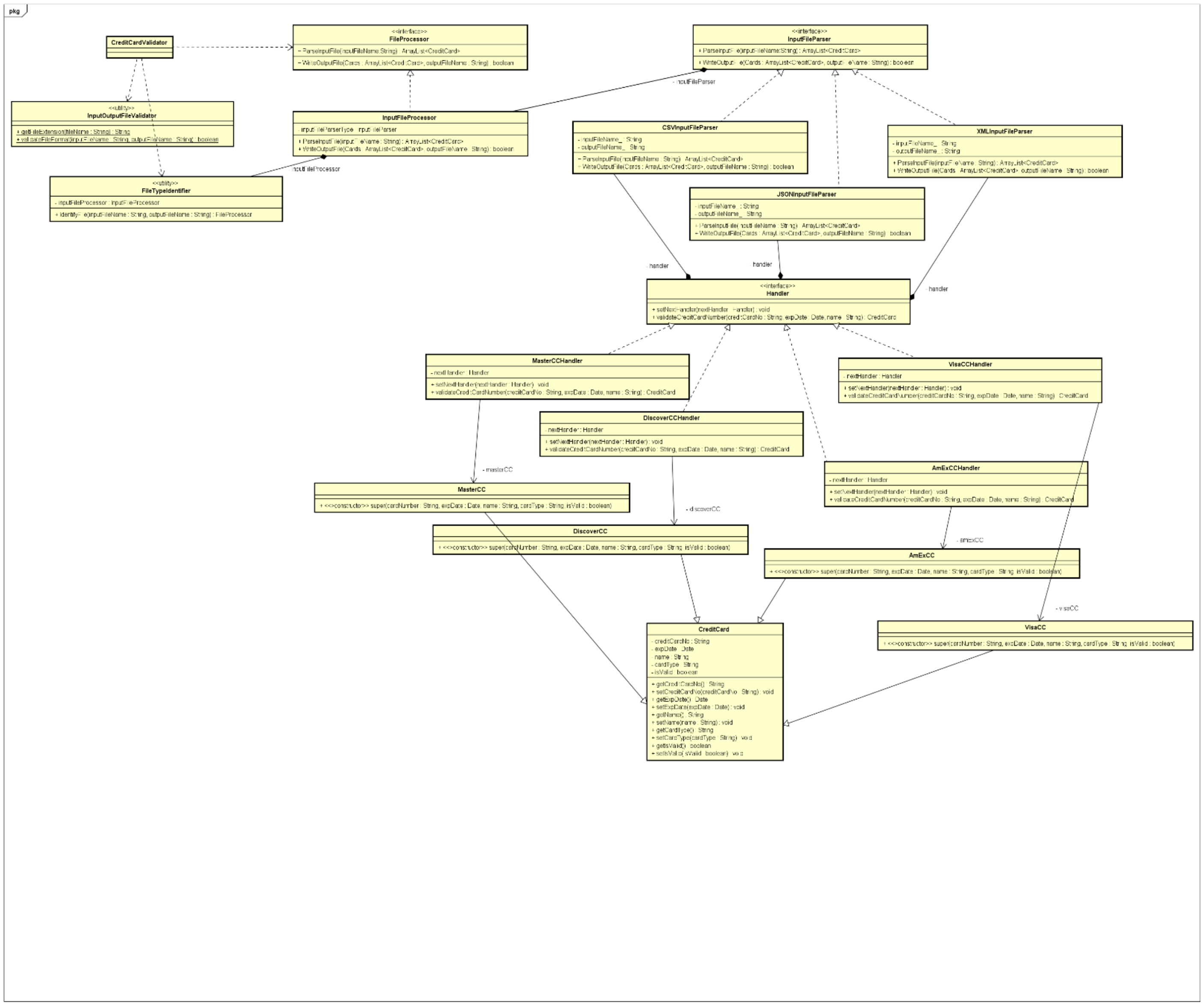
Problem Statement:

Continue with the design from Part 1 and extend it to parse different input file formats (json, xml, csv) and detect the type of credit card and then output to a file

Now that in part 1 we know how to pass the input to an handler and handler will take the responsibility of validating and creating objects of CreditCardClass.

Modified Factory Pattern: (Factory +Strategy)

To solve the issue of determining the file type. I have combined Factory and Strategy Pattern such that, client first calls the a file processor **FileProcessor** factory which inturn Asks **InputFileProcessor (Factory pattern design)** parser to get **inputFileParserType** instance depending on extension (csv,json,xml). I have created each concrete strategy like CSVInputFileParser, XMLInputFileParser, JSONInputFileParser which will intract with handler to pass the request and validate the credit cards. So, by just adding few interfaces and classes I could easily extended the design pattern to support multiple file formats. Combination of factory and strategy and chain of responsibility pattern are pillars for entire application design.


pkg
CreditCardValidator
<<interface>>
FileProcessor
~ ParseInputFile(inputFileName:String) : ArrayList<CreditCard>
~ WriteOutputFile(Cards : ArrayList<CreditCard>, outputFileName : String) : boolean
<<utility>>
InputOutputFileValidator
+ getFileExtension(fileName : String) : String
+ validateFileFormat(inputFileName : String, outputFileName : String) : boolean
InputFileProcessor
- inputFileParserType : InputFileParser
+ ParseInputFile(inputFileName : String) : ArrayList<CreditCard>
+ WriteOutputFile(Cards : ArrayList<CreditCard>, outputFileName : String) : boolean
- inputFileProcessor
<<utility>>
FileTypeIdentifier
- inputFileProcessor : InputFileProcessor
+ identifyFile(inputFileName : String, outputFileName : String) : FileProcessor
<<interface>>
InputFileParser
+ ParseInputFile(inputFileName:String) : ArrayList<CreditCard>
+ WriteOutputFile(Cards : ArrayList<CreditCard>, outputFileName : String) : boolean
- inputFileParser
CSVInputFileParser
- inputFileName_ : String
- outputFileName_ : String
~ ParseInputFile(inputFileName : String) : ArrayList<CreditCard>
~ WriteOutputFile(Cards : ArrayList<CreditCard>, outputFileName : String) : boolean
XMLInputFileParser
- inputFileName_ : String
- outputFileName_ : String
+ ParseInputFile(inputFileName : String) : ArrayList<CreditCard>
+ WriteOutputFile(Cards : ArrayList<CreditCard>, outputFileName : String) : boolean
JSONInputFileParser
- inputFileName_ : String
- outputFileName_ : String
+ ParseInputFile(inputFileName : String) : ArrayList<CreditCard>
+ WriteOutputFile(Cards : ArrayList<CreditCard>, outputFileName : String) : boolean
- handler
- handler
- handler
<<interface>>
Handler
+ setNextHandler(nextHandler : Handler) : void
+ validateCreditCardNumber(creditCardNo : String, expDate : Date, name : String) : CreditCard
MasterCCHandler
- nextHandler : Handler
+ setNextHandler(nextHandler : Handler) : void
+ validateCreditCardNumber(creditCardNo : String, expDate : Date, name : String) : CreditCard
VisaCCHandler
- nextHandler : Handler
+ setNextHandler(nextHandler : Handler) : void
+ validateCreditCardNumber(creditCardNo : String, expDate : Date, name : String) : CreditCard
DiscoverCCHandler
- nextHandler : Handler
+ setNextHandler(nextHandler : Handler) : void
+ validateCreditCardNumber(creditCardNo : String, expDate : Date, name : String) : CreditCard
AmExCCHandler
- nextHandler : Handler
+ setNextHandler(nextHandler : Handler) : void
+ validateCreditCardNumber(creditCardNo : String, expDate : Date, name : String) : CreditCard
- masterCC
MasterCC
+ <<constructor>> super(cardNumber : String, expDate : Date, name : String, cardType : String, isValid : boolean)
- discoverCC
DiscoverCC
+ <<constructor>> super(cardNumber : String, expDate : Date, name : String, cardType : String, isValid : boolean)
- amExCC
AmExCC
+ <<constructor>> super(cardNumber : String, expDate : Date, name : String, cardType : String, isValid : boolean)
- visaCC
VisaCC
+ <<constructor>> super(cardNumber : String, expDate : Date, name : String, cardType : String, isValid : boolean)
CreditCard
- creditCardNo : String
- expDate : Date
- name : String
- cardType : String
- isValid : boolean
+ getCreditCardNo() : String
+ setCreditCardNo(creditCardNo : String) : void
+ getExpDate() : Date
+ setExpDate(expDate : Date) : void
+ getName() : String
+ setName(name : String) : void
+ getCardType() : String
+ setCardType(cardType : String) : void
+ getIsValid() : boolean
+ setIsValid(isValid : boolean) : void

Consequences of using Chain of Responsibility:

Decouples sender and receiver of the requests. Client does not need to know which handler to call or which class to interact to get his validations. The chain structure will take care of the object creation. We can easily add or remove the responsibility by adding or remove handlers of concrete handler classes.
Since this is sequence and chain operations, it will take more execution time at worst case O(n) where n = number of handlers. Or sometimes it can perform at best case if the first-class handler itself is the key logic that client wants to validate.
Can handle dynamic inputs.

Consequences of using Factory Pattern + Strategy:

Factory Method lets a class defer instantiation to subclasses
separate knowledge of “how to use” from knowledge of “what exactly to create”. Client knows first part, factory second part. Strategy pattern provides flexibility to add any number of new file types to parse. Additionally, the factory will have the ability to abstract away all of the polymorphism from main credit card validation logic.

Note: Diagrams are added in Git Repo for clear visibility