CMPE 202 – Individual Project
Credit Card Problem

Part 1

Arpitha Gurumurthy - 014642290

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

The primary problem in this use case is to identify the card type given the card details in a CSV file.
Each credit card has:

- card number
- expiration date
- name of the card holder

Given these details of the credit card, we should find the type of credit card based on the below conditions:

- MasterCard if:
    1. first digit is a 5,
    2. second digit is in range 1 through 5 inclusive.
    3. length is 16 digits
- Visa if:
    1. First digit is a 4.
    2. Length is either 13 or 16 digits.
- AmericanExpress if:
    1. First digit is a 3
    2. second digit a 4 or 7.
    3. Length is 15 digits.
- Discover if:
    1. First four digits are 6011.
    2. Length is 16 digits

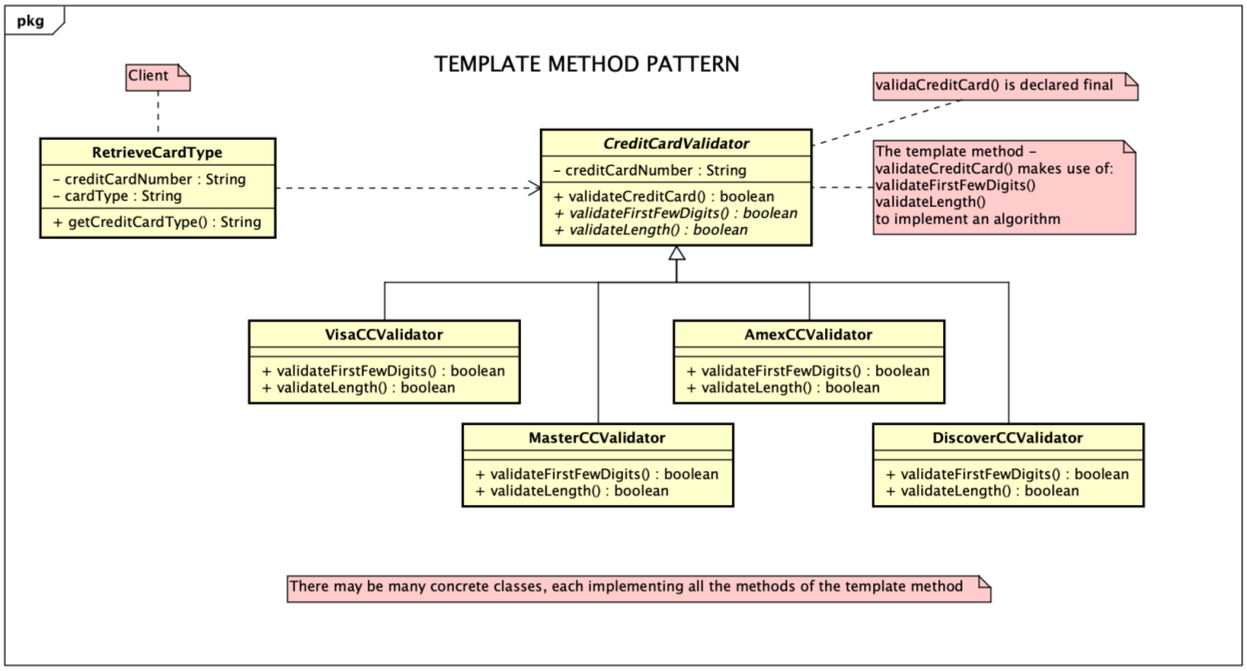
b. Describe what are the secondary problems you try to solve (if there are any).

Once we know the card type from the above given conditions, we need to create appropriate objects of the credit card. This can be considered as a secondary problem.

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

Design patterns that can be applied are:

- ***Template Method Pattern:***
  To verify that the credit card number is a possible account number

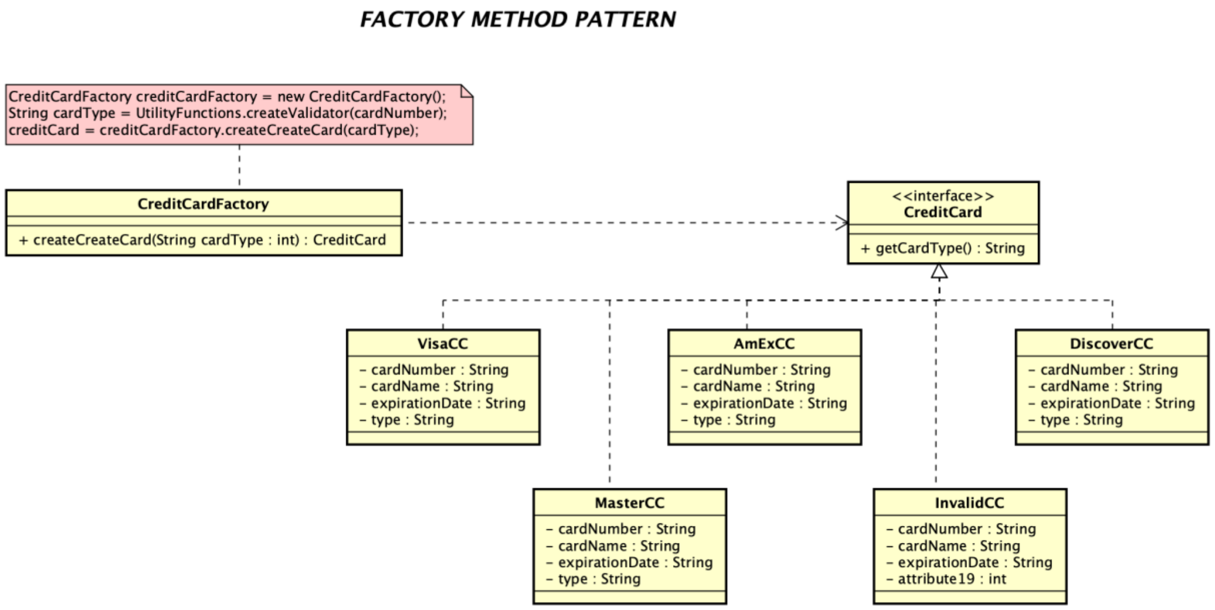

pkg
TEMPLATE METHOD PATTERN
Client
validaCreditCard() is declared final
The template method – validateCreditCard() makes use of: validateFirstFewDigits() validateLength() to implement an algorithm
RetrieveCardType
– creditCardNumber : String
– cardType : String
+ getCreditCardType() : String
CreditCardValidator
– creditCardNumber : String
+ validateCreditCard() : boolean
+ validateFirstFewDigits() : boolean
+ validateLength() : boolean
VisaCCValidator
AmexCCValidator
MasterCCValidator
DiscoverCCValidator
+ validateFirstFewDigits() : boolean
+ validateLength() : boolean
There may be many concrete classes, each implementing all the methods of the template method

- The client is our scenario is "RetrieveCardType" class. When getCreditCardType() is invoked, the validateCreditCard() method of CreditCardValidator is invoked.
- This validateCreditCard() is our template method which defines an algorithm to verify the card type of a given card number. The algorithm has 2 steps each of which is defined as a method inside the template method: validateFirstFewDigits(), validateLength().
- Each of the subclasses of CreditCardValidator implements their version of validateFirstFewDigits() and validateLength().
- When the CreditCardValidator receives an input, each of the subclasses are responsible for validating the card and return the appropriate card type.

***NOTE**:*
- CreditCardValidator is an abstract class
- validateFirstFewDigits(), validateLength() are abstract methods.
- validateCreditCard() is a final method.

- ***Factory Method pattern***

  To create an instance of the appropriate credit card class –


FACTORY METHOD PATTERN
CreditCardFactory creditCardFactory = new CreditCardFactory();
String cardType = UtilityFunctions.createValidator(cardNumber);
creditCard = creditCardFactory.createCreateCard(cardType);
CreditCardFactory
+ createCreateCard(String cardType : int) : CreditCard
<<interface>>
CreditCard
+ getCardType() : String
VisaCC
AmExCC
DiscoverCC
MasterCC
InvalidCC
– cardNumber : String
– cardName : String
– expirationDate : String
– type : String
– attribute19 : int

- Once the template method returns the credit card type after validation, the client would want to create an object of that class.
- First, we define a createCreditCard() method in the CreditCardFactory class that returns new objects based on the card type.
- CreditCard declares an interface which is common to all objects that can be produced by the CreditCardFactor.
- Concrete credit cards – VisaCC, AmExCC, DiscoverCC, MasterCC, InvalidCC are different implementations of the interface.

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

<u>Template Method:</u>

The benefits of using template method pattern for validating the card type include:

- Template method pattern serves as a template for an algorithm. The algorithm in this scenario includes the steps to validate the card.
- Each method in the algorithm verifies one condition for a given card.
- If the subclasses want to override some methods of the template pattern, such methods are defined as abstract in the parent.
- The template method itself contains all the generic steps of the algorithm and is defined as “final” in the parent class.
- The implementation of the validateCreditCard() method is abstracted from the client.

Factory Method:

The benefits of using factory method pattern for validating the card type include:

- Using the factory method to instantiate appropriate objects would lessen the burden for the client.
- Factory pattern gives us the ability to postpone the object creation, so clients do not have the overhead of creating subclass type objects.
- Also adding new card types into the use case would be easy, client code does not require changes and only the factory code would need minor changes.

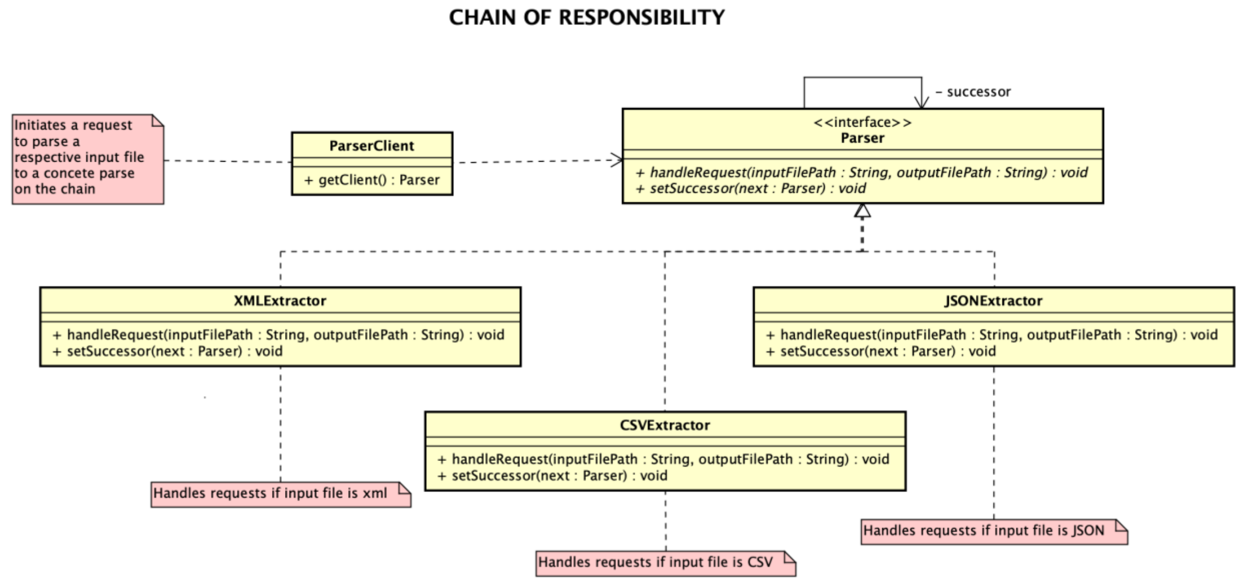
CMPE 202 – Individual Project
Credit Card Problem

Part 2

Arpitha Gurumurthy - 014642290

Chain of Responsibility

To design the methods that parses different file formats –


CHAIN OF RESPONSIBILITY
Initiates a request to parse a respective input file to a concete parse on the chain
ParserClient
+ getClient() : Parser
– successor
<<interface>> Parser
+ handleRequest(inputFilePath : String, outputFilePath : String) : void
+ setSuccessor(next : Parser) : void
XMLExtractor
+ handleRequest(inputFilePath : String, outputFilePath : String) : void
+ setSuccessor(next : Parser) : void
JSONExtractor
+ handleRequest(inputFilePath : String, outputFilePath : String) : void
+ setSuccessor(next : Parser) : void
CSVExtractor
+ handleRequest(inputFilePath : String, outputFilePath : String) : void
+ setSuccessor(next : Parser) : void
Handles requests if input file is xml
Handles requests if input file is CSV
Handles requests if input file is JSON

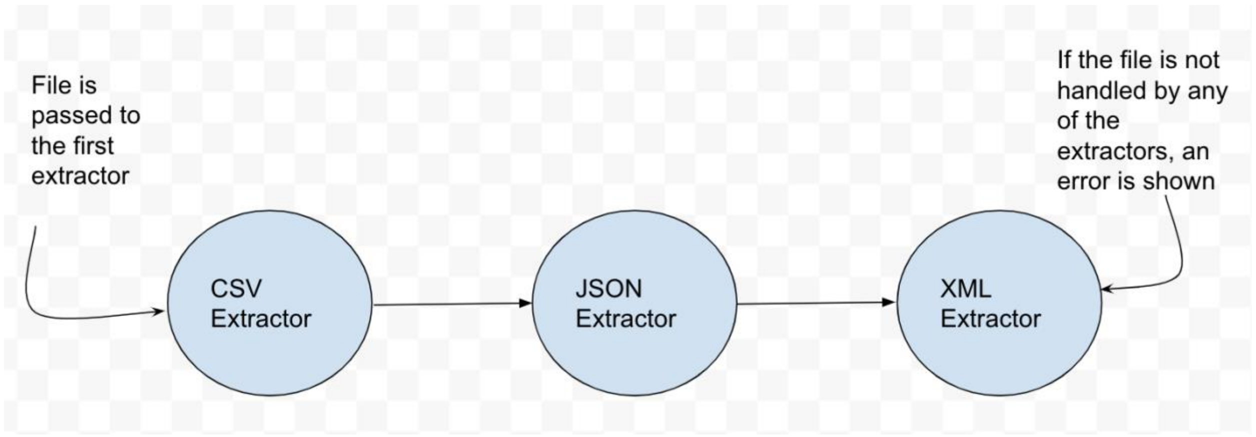
PARTICIPANTS:

1. ***Parser*** – Interface
- Defines an interface for handling requests to parse xml, csv and json as of now. More types can be accommodated in the future.
- Implements the successor link using the method setSuccessor(Parser next).

2. ***CSVExtractor, JSONExtractor and XMLExtractor***
   Handles requests based on the file type. If it cannot handle it, it calls handleRequest() method on the successor if present.

3. ***ParserClient***

Initiates the request to the parser and the request will be handled appropriately.


File is passed to the first extractor
CSV Extractor
JSON Extractor
XML Extractor
If the file is not handled by any of the extractors, an error is shown

The consequences of using chain of responsibility:

- The sender of the request does not know which object is responsible to parse the input file.
- It saves client overhead since it is handled by the service layer.