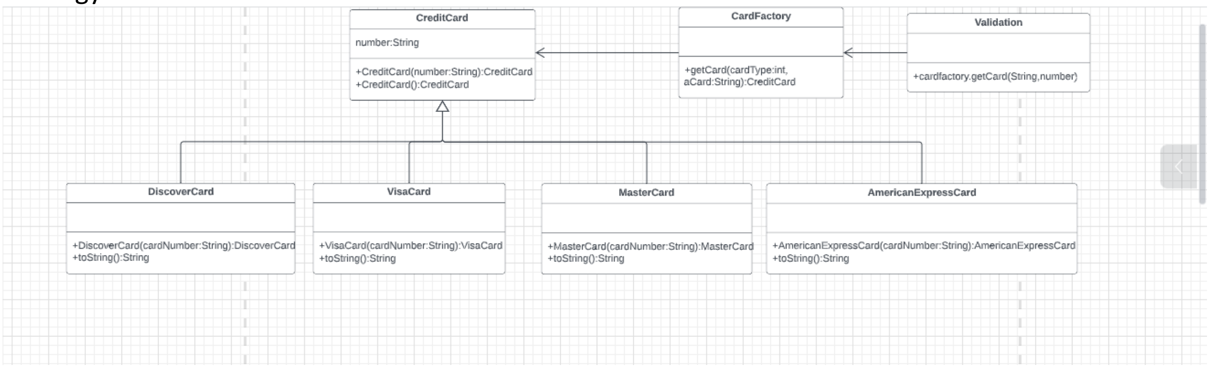
CMPE-202- Individual Project

Student Name – Sourya Prateek Jammula

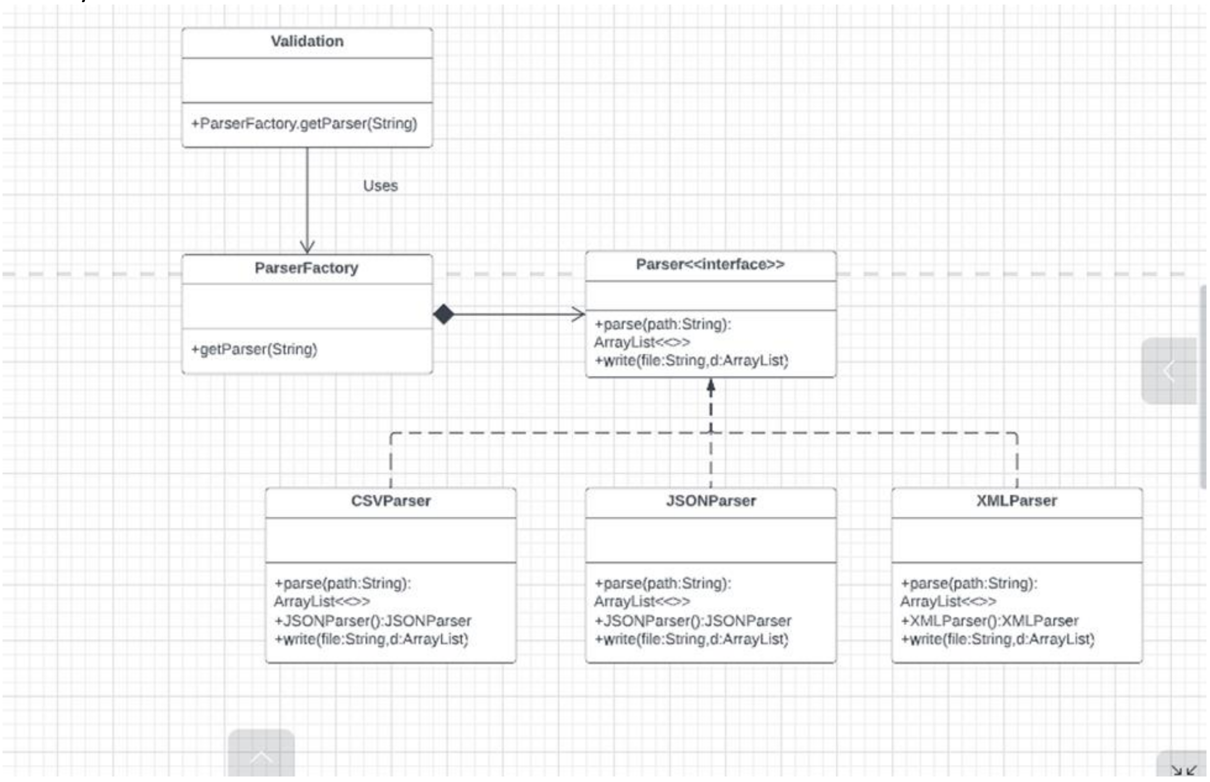
Student ID – 016702062

Design Patterns

1) Strategy Pattern


CreditCard
number:String
+CreditCard(number:String):CreditCard
+CreditCard():CreditCard
CardFactory
+getCard(cardType:int, aCard:String):CreditCard
Validation
+cardfactory.getCard(String,number)
DiscoverCard
+DiscoverCard(cardNumber:String):DiscoverCard
+toString():String
VisaCard
+VisaCard(cardNumber:String):VisaCard
+toString():String
MasterCard
+MasterCard(cardNumber:String):MasterCard
+toString():String
AmericanExpressCard
+AmericanExpressCard(cardNumber:String):AmericanExpressCard
+toString():String

2) Factory Pattern


Validation
+ParserFactory.getParser(String)
Uses
ParserFactory
+getParser(String)
Parser<<interface>>
+parse(path:String): ArrayList<<>>
+write(file:String,d:ArrayList)
CSVParser
+parse(path:String): ArrayList<<>>
+JSONParser():JSONParser
+write(file:String,d:ArrayList)
JSONParser
+parse(path:String): ArrayList<<>>
+JSONParser():JSONParser
+write(file:String,d:ArrayList)
XMLParser
+parse(path:String): ArrayList<<>>
+XMLParser():XMLParser
+write(file:String,d:ArrayList)

- Describe what is the primary problem you try to solve.
    - The main problem we are trying to solve is that we need to identify the type of the file format (i.e, json , csv or xml) and read it to find the cardNumber from the input file and checking each number to be valid or invalid based on Luhn's algorithm(self-chosen). Furthermore, we

need to identify the type fo acrd i.e, MasterCard, Visa or Discover based on the validations given in problem structure. Ther should be specific design pattern followed inorder to figure out what kind of card a specific record is about, the other one with how you create the appropriate objects.

- Describe what are the secondary problems you try to solve (if there are any).
    - The secondary problem is to identify the card is valid or invalid according to a algorithm, here I have chosen Luhn's algorithm for the same. There is another problem to be resolved that is to write the type of card if valid into specific file formats in which it comes in.
- Describe what design pattern(s) you use how (use plain text and diagrams).
    - I have used Strategy design pattern to know the file format input file comes in(JSON, CSV, XML). It also allows to use this implementation in the future for newer file formats.In this patter, we encapsulate the Parser interface deatils and hide the implementation details in the derived classes (CSVParser, XMLParser, JSONParser). After parsing these, we extract the card number from the input files and now we need to identify the type of the credit card that is the MasterCard, Visa, Discover(by matching the regular expressions) . For identifying the type of card, we use the Factory Design pattern. This is done by using CardFactory class object for the Card class. Based on the argumnets given it creates the particular credit card type.
- Describe the consequences of using this/these pattern(s).
    - Strategy Pattern:
      pros:This pattern can helps to swap algorithms used inside an object at runtime. Also, it can isolate the implementation details of an algorithm from the code that client use it.
      cons: If we only have a couple of algorithms and they rarely change, there’s no real reason to overcomplicate the program with new classes and interfaces that come along with the pattern. Also, Clients must be aware of the differences between strategies to be able to select a proper one.
    - Factory Pattern:
      pros: You avoid tight coupling between the creator and the concrete products. Single Responsibility Principle. You can move the product creation code into one place in the program, making the code easier to support. Open/Closed Principle. You can introduce new types of products into the program without breaking existing client code.

cons: The code may become more complicated since you need to introduce a lot of new subclasses to implement the pattern. The best case scenario is when you’re introducing the pattern into an existing hierarchy of creator classes.