CMPE 202 – INDIVIDUAL PROJECT

Rajashekar Kommula - 016007043

PART 1

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

The credit cards must be verified and categorized into a particular category depending on certain criteria, such as the credit card number's length, the initial digit, and so on.

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

Based on the results of the validation checks performed in the first phase, the appropriate card objects should next be initialized. Write an output file in the desired format containing records of the cardType, cardNumber, and errorMessage using a given input file of credit card numbers.

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

I decided to utilize the factory pattern to handle this issue because it calls for the production of particular objects depending on the input type.

• The client provides an input file that includes credit card details and requests the creation of a particular kind of output file.

• An additional RecordIO object is created. Then, a suitable filetype RecordInputOutput object is constructed in RecordInputOutputFactory depending on the input file provided by the user.

• Specifically in the filetype RecordInputOutput subclasses (JsonRecordIOInputOutput, CsvRecordIOInputOutput, XmlRecordIOInputOutput) logic is written to take the card information from various sorts of input data and write the desired results to the designated output files. The CreditCardFactory class generates a new object of specific type.

• Depending on the inputs, many credit card types are constructed in the subclasses.

• The OutputRecord class is used to construct each output in the appropriate output file.

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

• The Factory Design pattern enables object creation while obviating the need for creation logic.

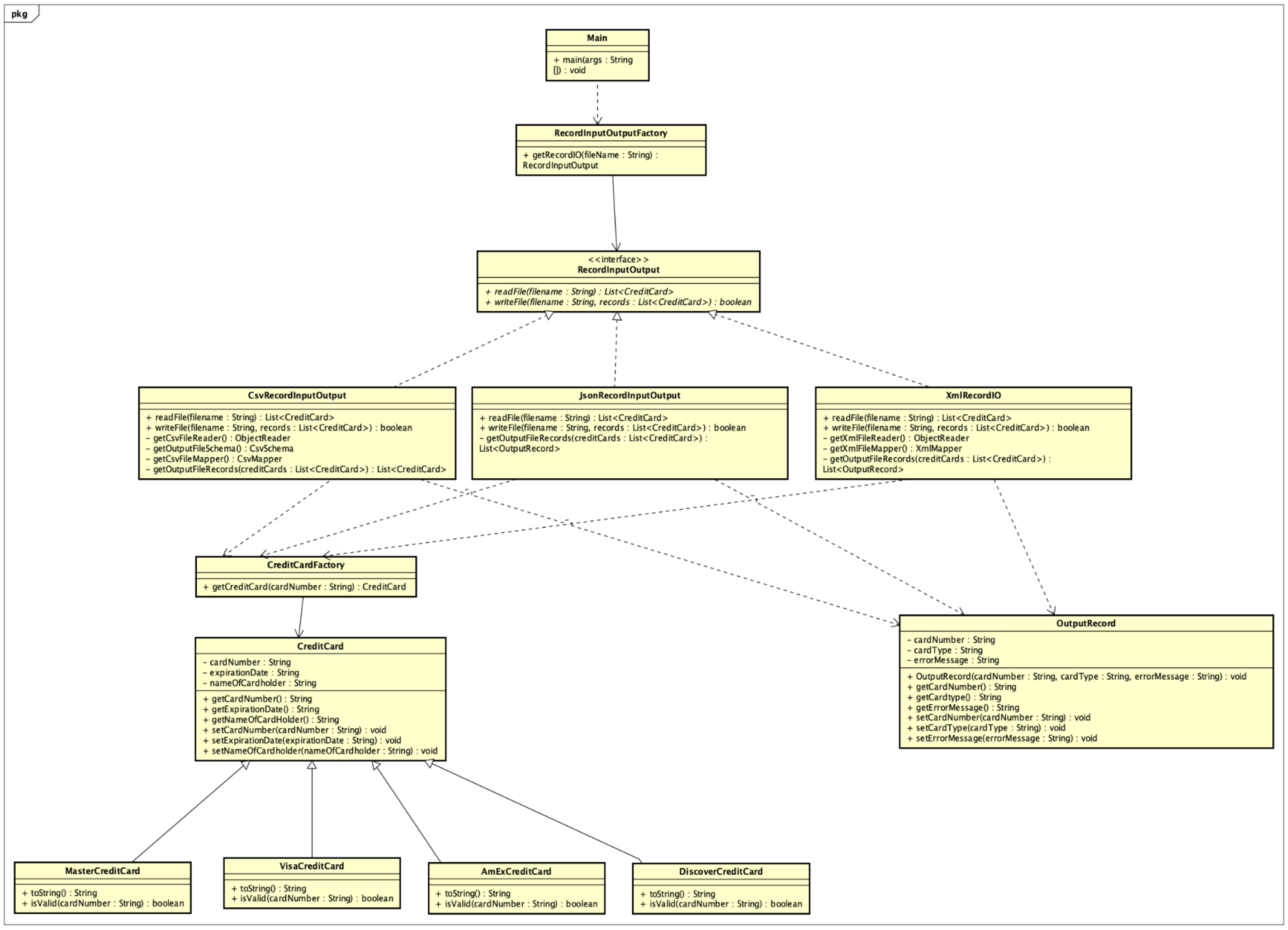
- By following this design, there is a probability that more subclass will be created in the future to handle new object types.

- Due to the division of logic and duties among many subclasses, guarantees that there is a loose coupling. This makes it easier to modify existing subclasses and create new ones as needed in the future.

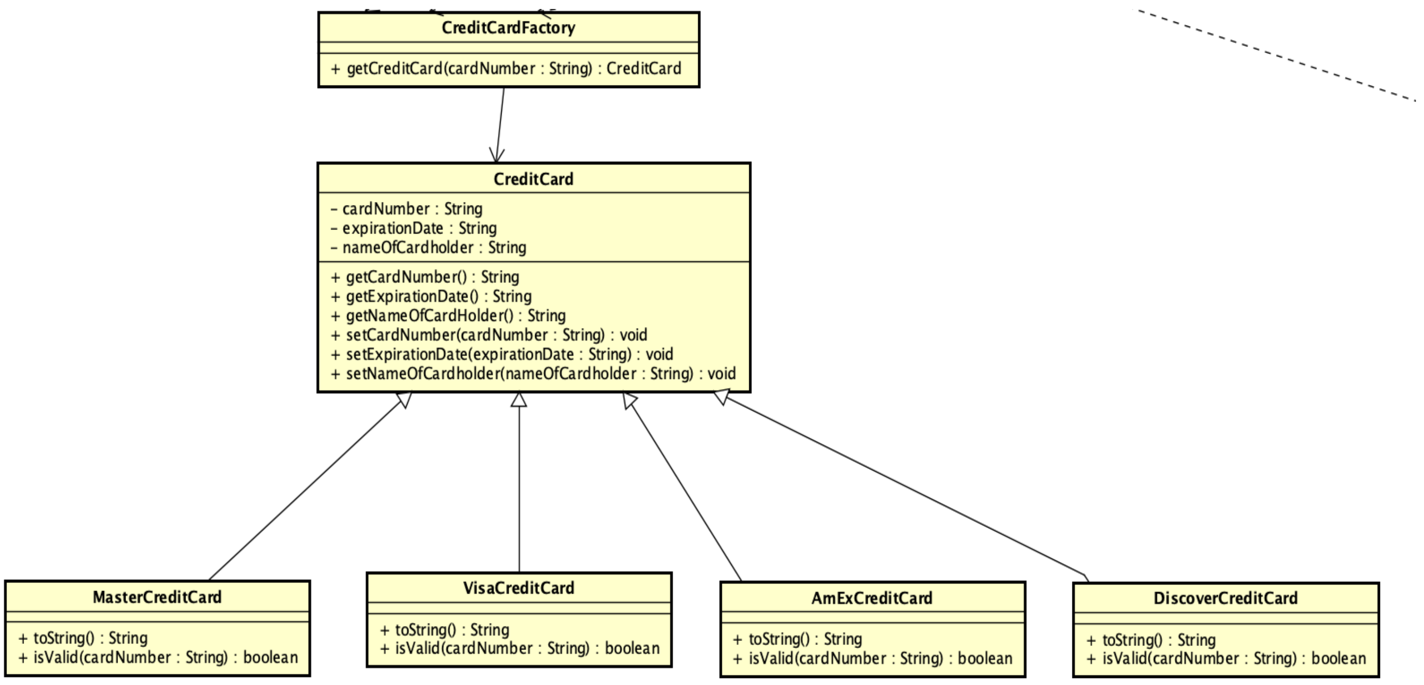
- The abstract code makes it difficult to read and comprehend.

Diagrams

Complete Class Diagram


pkg
Main
+ main(args : String []) : void
RecordInputOutputFactory
+ getRecordIO(fileName : String) : RecordInputOutput
<<interface>>
RecordInputOutput
+ readFile(filename : String) : List<CreditCard>
+ writeFile(filename : String, records : List<CreditCard>) : boolean
CsvRecordInputOutput
+ readFile(filename : String) : List<CreditCard>
+ writeFile(filename : String, records : List<CreditCard>) : boolean
- getCsvFileReader() : ObjectReader
- getOutputFileSchema() : CsvSchema
- getCsvFileMapper() : CsvMapper
- getOutputFileRecords(creditCards : List<CreditCard>) : List<CreditCard>
JsonRecordInputOutput
+ readFile(filename : String) : List<CreditCard>
+ writeFile(filename : String, records : List<CreditCard>) : boolean
- getOutputFileRecords(creditCards : List<CreditCard>) : List<OutputRecord>
XmlRecordIO
+ readFile(filename : String) : List<CreditCard>
+ writeFile(filename : String, records : List<CreditCard>) : boolean
- getXmlFileReader() : ObjectReader
- getXmlFileMapper() : XmlMapper
- getOutputFileRecords(creditCards : List<CreditCard>) : List<OutputRecord>
CreditCardFactory
+ getCreditCard(cardNumber : String) : CreditCard
CreditCard
- cardNumber : String
- expirationDate : String
- nameOfCardholder : String
+ getCardNumber() : String
+ getExpirationDate() : String
+ getNameOfCardHolder() : String
+ setCardNumber(cardNumber : String) : void
+ setExpirationDate(expirationDate : String) : void
+ setNameOfCardholder(nameOfCardholder : String) : void
OutputRecord
- cardNumber : String
- cardType : String
- errorMessage : String
+ OutputRecord(cardNumber : String, cardType : String, errorMessage : String) : void
+ getCardNumber() : String
+ getCardtype() : String
+ getErrorMessage() : String
+ setCardNumber(cardNumber : String) : void
+ setCardType(cardType : String) : void
+ setErrorMessage(errorMessage : String) : void
MasterCreditCard
+ toString() : String
+ isValid(cardNumber : String) : boolean
VisaCreditCard
+ toString() : String
+ isValid(cardNumber : String) : boolean
AmExCreditCard
+ toString() : String
+ isValid(cardNumber : String) : boolean
DiscoverCreditCard
+ toString() : String
+ isValid(cardNumber : String) : boolean

Credit Card Class Diagram


CreditCardFactory
+ getCreditCard(cardNumber : String) : CreditCard
CreditCard
– cardNumber : String
– expirationDate : String
– nameOfCardholder : String
+ getCardNumber() : String
+ getExpirationDate() : String
+ getNameOfCardHolder() : String
+ setCardNumber(cardNumber : String) : void
+ setExpirationDate(expirationDate : String) : void
+ setNameOfCardholder(nameOfCardholder : String) : void
MasterCreditCard
+ toString() : String
+ isValid(cardNumber : String) : boolean
VisaCreditCard
+ toString() : String
+ isValid(cardNumber : String) : boolean
AmExCreditCard
+ toString() : String
+ isValid(cardNumber : String) : boolean
DiscoverCreditCard
+ toString() : String
+ isValid(cardNumber : String) : boolean