Part 1 & Part 2 (Design Patterns and Class Diagrams)

M.V.N.S.Prudhvi (016597581)

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

Credit card inputs must be checked and categorized based on certain criteria such as the number of digits in the credit card number, the initial digit,

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

Then, based on the validation checks performed in the first step, appropriate card objects must be initialized.

Given an input file of credit card numbers, create an output file in the desired format, including records for the cardtype, cardNumber, and any errorMessages.

3. Describe what design pattern(s) you use how (use plain text and diagrams) (My answer includes Part 1 & Part 2 under this question)

To solve this problem, I chose the Factory Pattern, which involves the creation of specific objects based on the input type.

- The client sends the credit card records input file and expects a specific type of output file to be written.
- A new RecordIO object is created.Then, in RecordIOFactory, an appropriate filetype RecordIO object is created based on the input filename specified by the client.
- There is logic in certain _filetype RecordIO subclasses (CsvRecordIO, JsonRecordIO, XmlRecordIO) to read the records in the input file and send the contents to an output file.

In addition, a new credit card object of that kind is produced in the CreditCardFactory class.

- By validating the inputs in concrete subclasses, various types of credit cards are created.
- Each output record in the output file is created using OutputRecord class

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

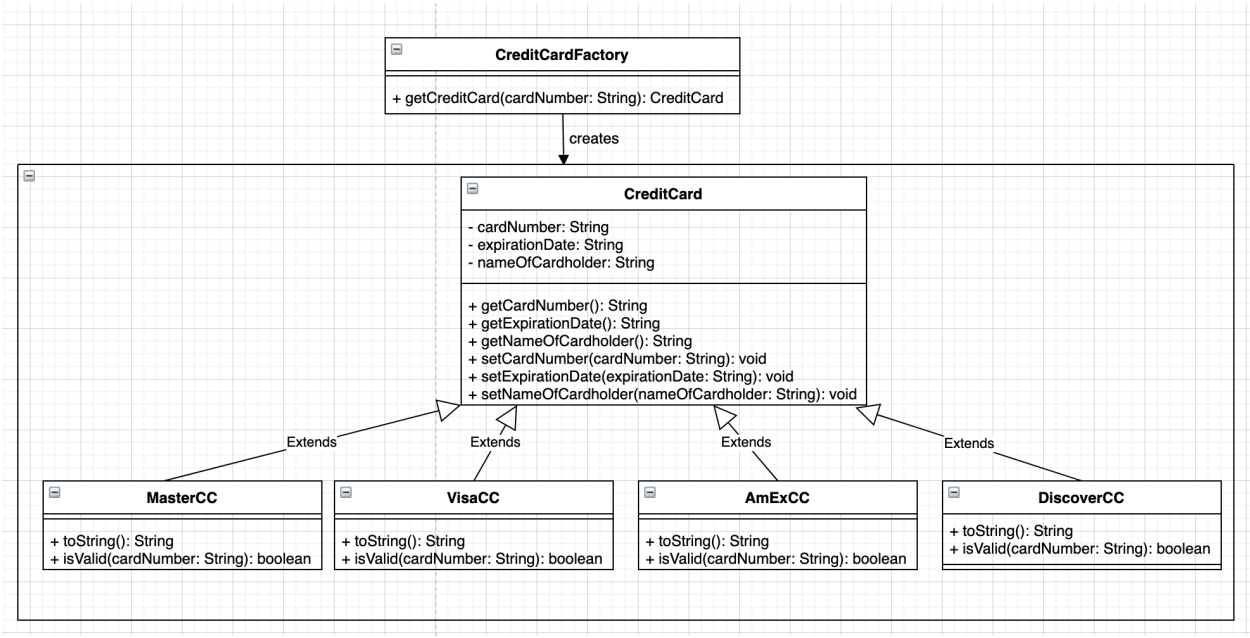
● The factory design facilitates object instantiation while abstracting the mechanism underlying creation.

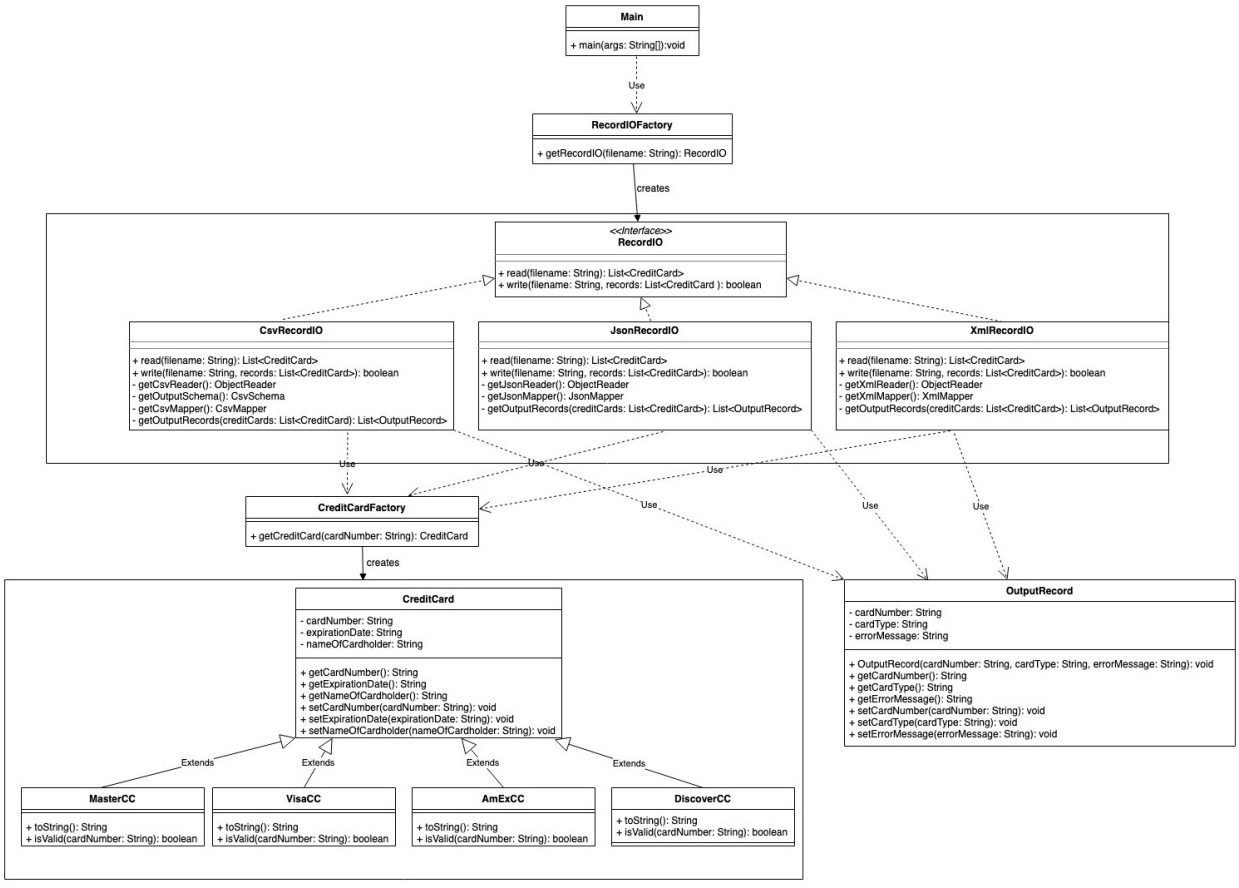
● This pattern allows for the addition of more concrete subclasses in the future to support new types of objects.

● Ensures that there is loose coupling due to subclasses' separation of logic and responsibilities.

This allows for easier modification and addition of subclasses in the future.

● One disadvantage of this pattern includes, it is difficult to read and understand the code because of the abstraction.


CreditCardFactory
+ getCreditCard(cardNumber: String): CreditCard
creates
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
Extends
Extends
Extends
Extends
MasterCC
+ toString(): String
+ isValid(cardNumber: String): boolean
VisaCC
+ toString(): String
+ isValid(cardNumber: String): boolean
AmExCC
+ toString(): String
+ isValid(cardNumber: String): boolean
DiscoverCC
+ toString(): String
+ isValid(cardNumber: String): boolean


Main
+ main(args: String[]):void
Use
RecordIOFactory
+ getRecordIO(filename: String): RecordIO
creates
<<Interface>>
RecordIO
+ read(filename: String): List<CreditCard>
+ write(filename: String, records: List<CreditCard): boolean
CsvRecordIO
+ read(filename: String): List<CreditCard>
+ write(filename: String, records: List<CreditCard>): boolean
- getCsvReader(): ObjectReader
- getOutputSchema(): CsvSchema
- getCsvMapper(): CsvMapper
- getOutputRecords(creditCards: List<CreditCard): List<OutputRecord>
JsonRecordIO
+ read(filename: String): List<CreditCard>
+ write(filename: String, records: List<CreditCard>): boolean
- getJsonReader(): ObjectReader
- getJsonMapper(): JsonMapper
- getOutputRecords(creditCards: List<CreditCard>): List<OutputRecord>
XmlRecordIO
+ read(filename: String): List<CreditCard>
+ write(filename: String, records: List<CreditCard>): boolean
- getXmlReader(): ObjectReader
- getXmlMapper(): XmlMapper
- getOutputRecords(creditCards: List<CreditCard>): List<OutputRecord>
Use
Use
Use
Use
Use
Use
CreditCardFactory
+ getCreditCard(cardNumber: String): CreditCard
creates
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
Extends
Extends
Extends
Extends
MasterCC
+ toString(): String
+ isValid(cardNumber: String): boolean
VisaCC
+ toString(): String
+ isValid(cardNumber: String): boolean
AmExCC
+ toString(): String
+ isValid(cardNumber: String): boolean
DiscoverCC
+ toString(): String
+ isValid(cardNumber: String): boolean
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