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

Gayathri Pulagam (014629290)

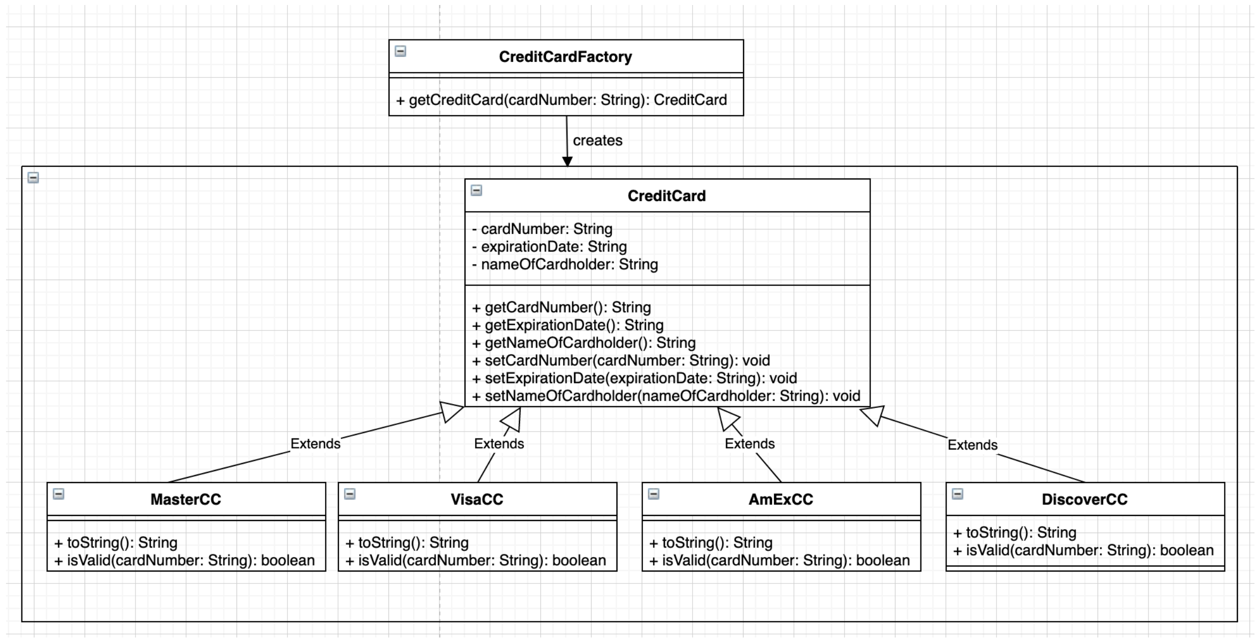
1. **Describe what is the primary problem you try to solve**
   The credit card inputs need to be validated and classified into a specific type based on some requirements such as the number of digits in the credit card number, the first digit and so on.

2. **Describe what are the secondary problems you try to solve (if there are any)**
   Then appropriate card objects are to be initialized based on the validation checks done in the first step. Given an input file of credit card numbers, need to write an output file in the desirable format with records of the cardtype, cardNumber and errorMessage if any.

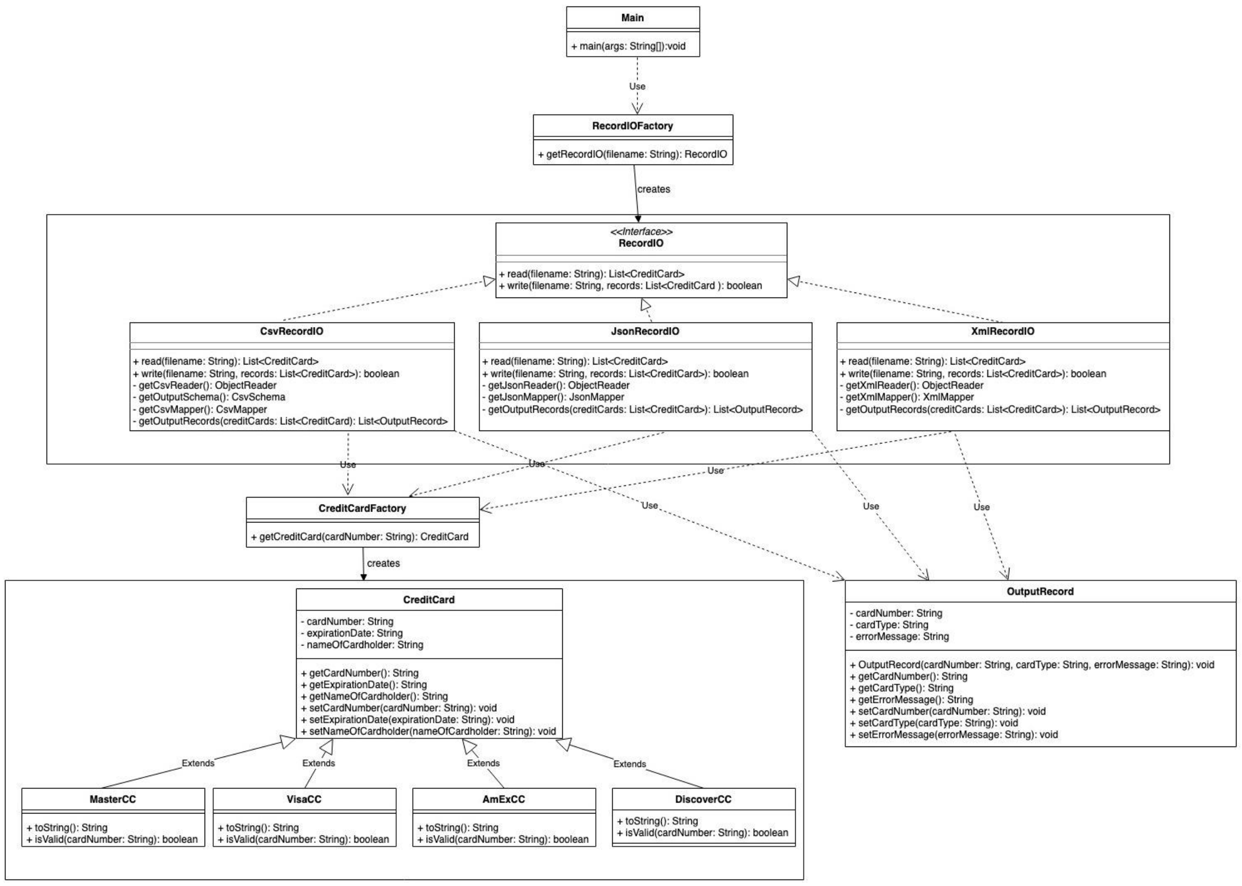
3. **Describe what design pattern(s) you use how (use plain text and diagrams) (My answer includes Part 1 & Part 2 under this question)**
   I chose to use **Factory Pattern** to solve this problem as this involves creation of specific objects based on the input type.
   - The client sends the input file containing credit card records and expects a specific type of output file to be written.
   - A new **RecordIO** object is instantiated. Then based on the input filename specified by the client, an appropriate **_filetype_RecordIO** object is created in **RecordIOFactory**.
   - In specific **_filetype_RecordIO** subclasses (CsvRecordIO, JsonRecordIO, XmlRecordIO), there is logic to read the records in the input file using appropriate readers and write the contents to an output file. Along with this, a new credit card object of it's type is created in **CreditCardFactory** class.
   - Different type of credit cards are created by validating the inputs in concrete subclasses (AmExCC, DiscoverCC, MasterCC, VisaCC)
   - Each output record in the output file is created using **OutputRecord** class

4. **Describe the consequences of using this/these pattern(s)**
   - Factory pattern helps in instantiation of objects while abstracting the logic behind creation.
   - With this pattern, there is a scope of adding more concrete subclasses in the future to support new types of objects.
   - Makes sure that there is loose coupling because of the separation of logic and responsibilities amongst subclasses. This helps in easier modification and addition of subclasses as necessary in the future.
   - One disadvantage of this pattern includes, it is difficult to read and understand the code because of the abstraction.

## Class diagram of the credit card object instantiation


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

## Class diagram of the entire application


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