DELIVERABLES, DESIGN PATTERNS & CLASS DIAGRAMS

SUNDARA SAI KIRAN
PSID: 017409028

1. **Describe what is the primary problem you try to solve.**
   We need to organize credit cards into specific groups and verify their categorization using given criteria, like the initial digits and the total number of digits for various types of credit cards

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

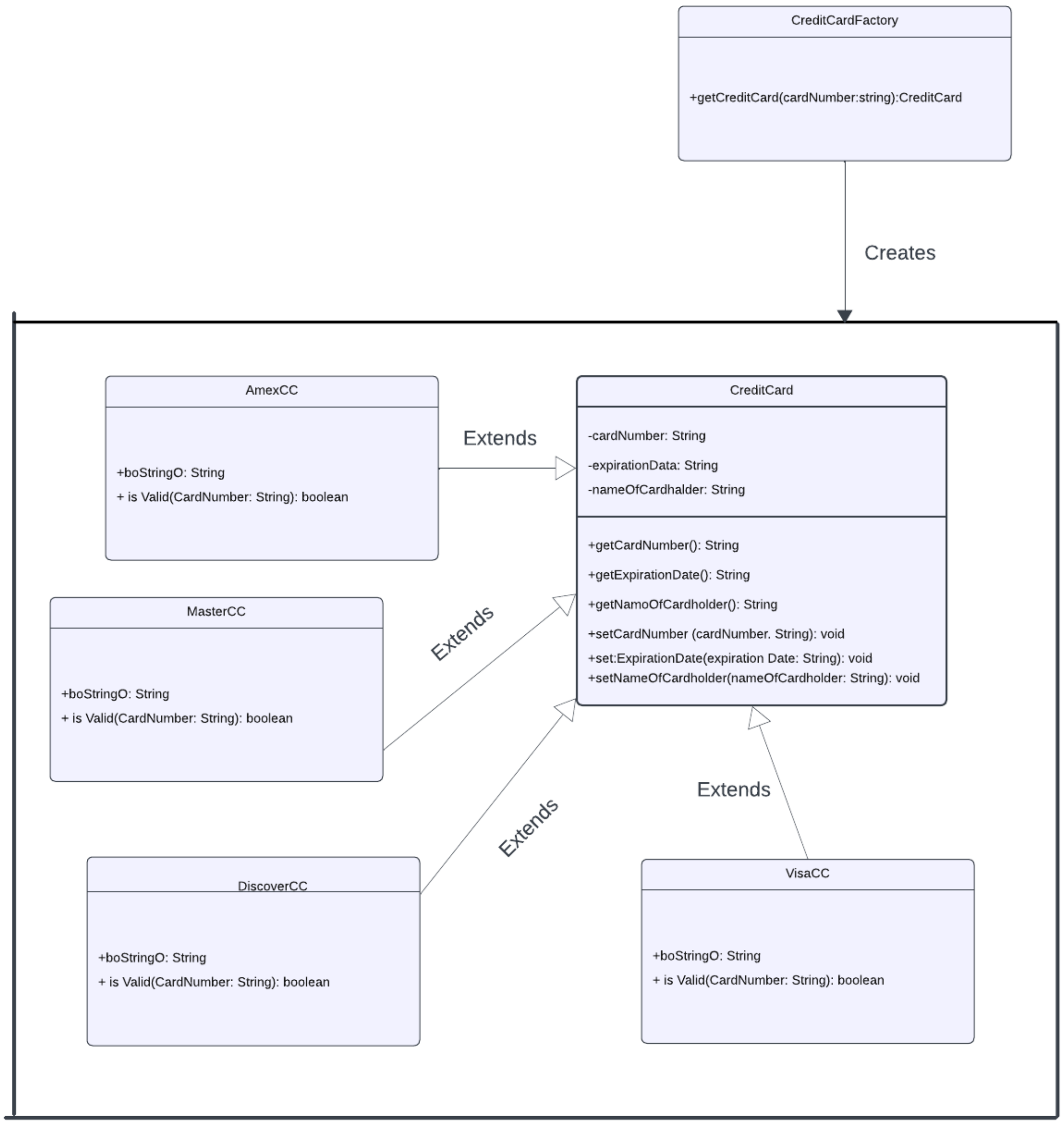
   We need to set up the credit card objects correctly by considering the validation tests conducted in the previous step. When given an input file containing different credit card numbers, we are required to generate an output file in the specified format. Each entry in the output file should include the credit card type, credit card number, and any error messages. Error messages will be shown if the input file contains non-numeric characters in the credit card numbers..

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

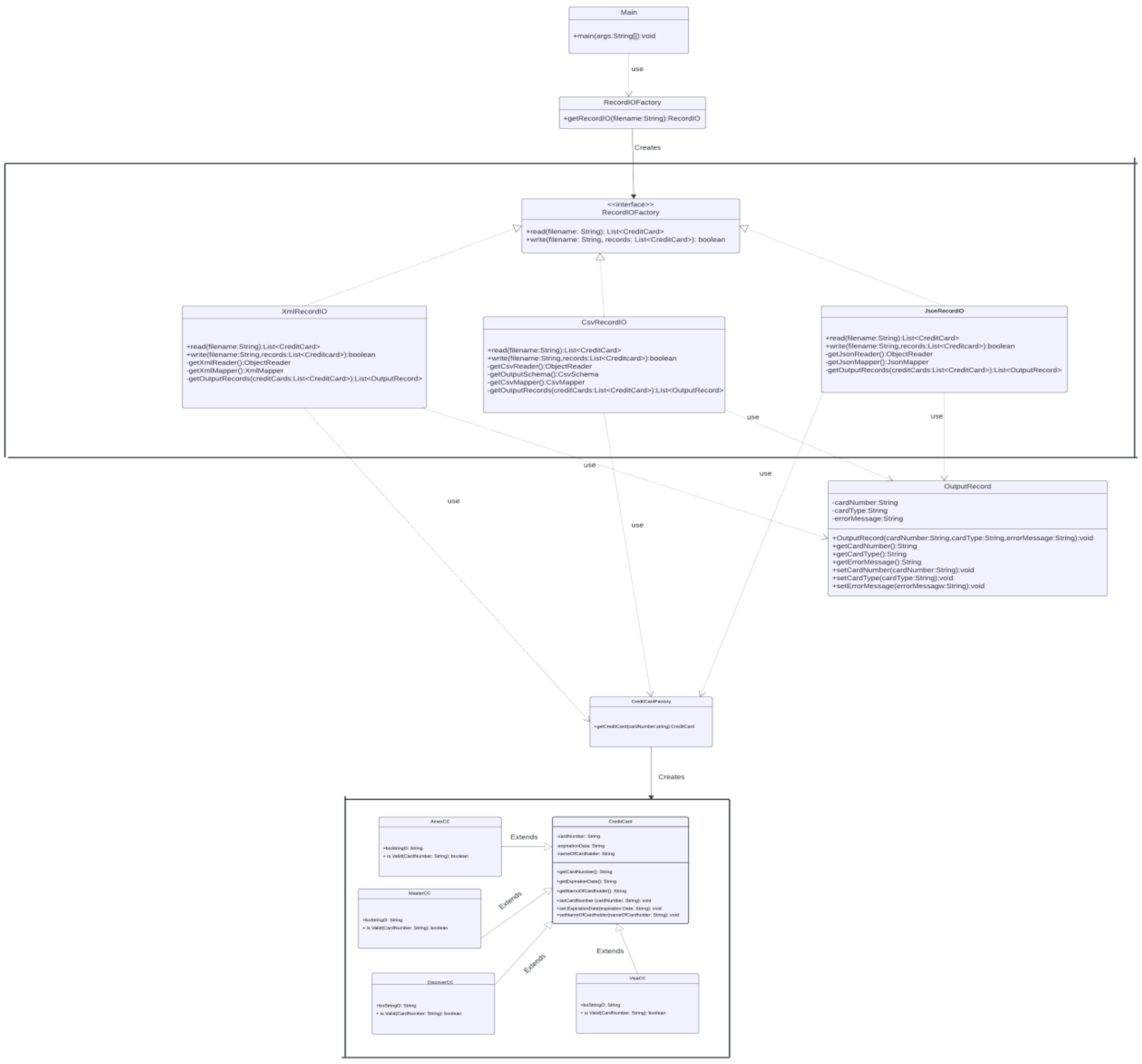
In tackling this issue that involves creating specific objects based on input types, I opted to employ the factory pattern for a solution.

Here's how the process works:
1. The client requests the creation of a particular output file and provides an input file containing credit card numbers.
2. A new RecordIO object is instantiated. The RecordIOFactory is then utilized to construct a suitable _filetype_RecordIO object based on the input filename provided by the client.
3. Certain _filetype_RecordIO subclasses (CsvRecordIO, JsonRecordIO, and XmlRecordIO) contain logic to read records from the input file using the appropriate readers and subsequently write the contents to an output file.
4. Additionally, the CreditCardFactory class is responsible for generating a new object representing the relevant type of credit card.
5. The validation of inputs is performed in specific subclasses (AmExCC, DiscoverCC, MasterCC, VisaCC), leading to the creation of different types of credit cards
By utilizing the OutputRecord class, we create individual entries for each record in the output file.

Credit Card Object Instantiation Class Diagram:


CreditCardFactory
+getCreditCard(cardNumber:string):CreditCard
Creates
AmexCC
+boStringO: String
+ is Valid(CardNumber: String): boolean
Extends
CreditCard
-cardNumber: String
-expirationData: String
-nameOfCardhalder: String
+getCardNumber(): String
+getExpirationDate(): String
+getNamoOfCardholder(): String
+setCardNumber (cardNumber. String): void
+set:ExpirationDate(expiration Date: String): void
+setNameOfCardholder(nameOfCardholder: String): void
MasterCC
+boStringO: String
+ is Valid(CardNumber: String): boolean
Extends
Extends
Extends
DiscoverCC
+boStringO: String
+ is Valid(CardNumber: String): boolean
VisaCC
+boStringO: String
+ is Valid(CardNumber: String): boolean

Application Class Diagram:


Main
+main(args:String[]):void
use
RecordIOFactory
+getRecordIO(filename:String):RecordIO
Creates
<<interface>>
RecordIOFactory
+read(filename: String): List<CreditCard>
+write(filename: String, records: List<CreditCard>): boolean
XmlRecordIO
+read(filename:String):List<CreditCard>
+write(filename:String,records:List<Creditcard>):boolean
-getXmlReader():ObjectReader
-getXmlMapper():XmlMapper
-getOutputRecords(creditCards:List<CreditCard>):List<OutputRecord>
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>
use
use
use
use
use
use
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(errorMessagw:String):void
CreditCardFactory
+getCreditCard(cardNumber:string):CreditCard
Creates
AmexCC
+toStringO: String
+ is Valid(CardNumber: String): boolean
Extends
CreditCard
-cardNumber: String
-expirationData: String
-nameOfCardholder: String
+getCardNumber(): String
+getExpirationDate(): String
+getNameOfCardholder(): String
+setCardNumber (cardNumber: String): void
+set ExpirationDate(expiration Date: String): void
+setNameOfCardholder(nameOfCardholder: String): void
MasterCC
+toStringO: String
+ is Valid(CardNumber: String): boolean
Extends
DiscoverCC
+toStringO: String
+ is Valid(CardNumber: String): boolean
Extends
VisaCC
+toStringO: String
+ is Valid(CardNumber: String): boolean
Extends

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

- The factory design makes it easier to create objects by hiding the details of how they are constructed.
- Using this approach allows for the potential addition of new specific subclasses to manage different types of objects later on. This promotes flexibility and easy adaptation to changes.
- While this method has its advantages, one drawback is that the abstraction can make the code less straightforward and harder to understand.

GitHub Link: https://github.com/gopinathsjsu/individual_project_sundarasaikiran