DELIVERABLES, DESIGN PATTERNS & CLASS DIAGRAMS

Anudeep Goud Kotagiri

SJSU ID: 017407377

Github Repo:
https://github.com/gopinathsjsu/individual_project_Anudeep_Goud

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

   We need to classify the credit card into a designated category and verify it according to provided specifications, such as the initial digits and the overall digit count corresponding to various credit card types.

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, following the validation tests conducted in the preceding step. Generating an output file in the specified format is essential, including entries with credit card type, credit card number, and error messages. Error messages will be triggered if the input file contains credit card numbers with non-numeric characters.

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

Given the need to construct specific objects based on input types, I opted to employ the factory pattern to address this issue.

- The client requests the generation of a particular output file and provides an input file containing credit card numbers.

- A fresh RecordIO object is instantiated. Subsequently, a suitable _filetype_RecordIO object is crafted in the RecordIOFactory, depending on the client-provided input filename.

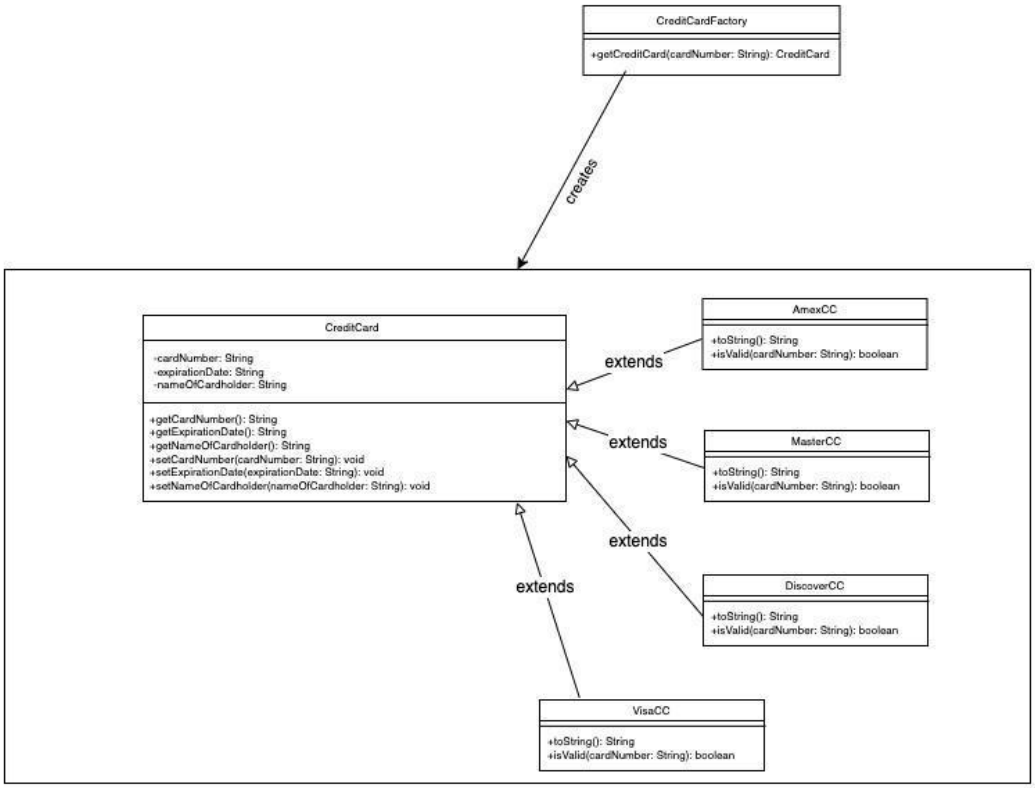
- Certain logic exists within specific _filetype_RecordIO subclasses (CsvRecordIO, JsonRecordIO, and XmlRecordIO) to read records from the input file using appropriate readers and then write the contents to an output file.

- Furthermore, the CreditCardFactory class generates a new object of the pertinent credit card type.

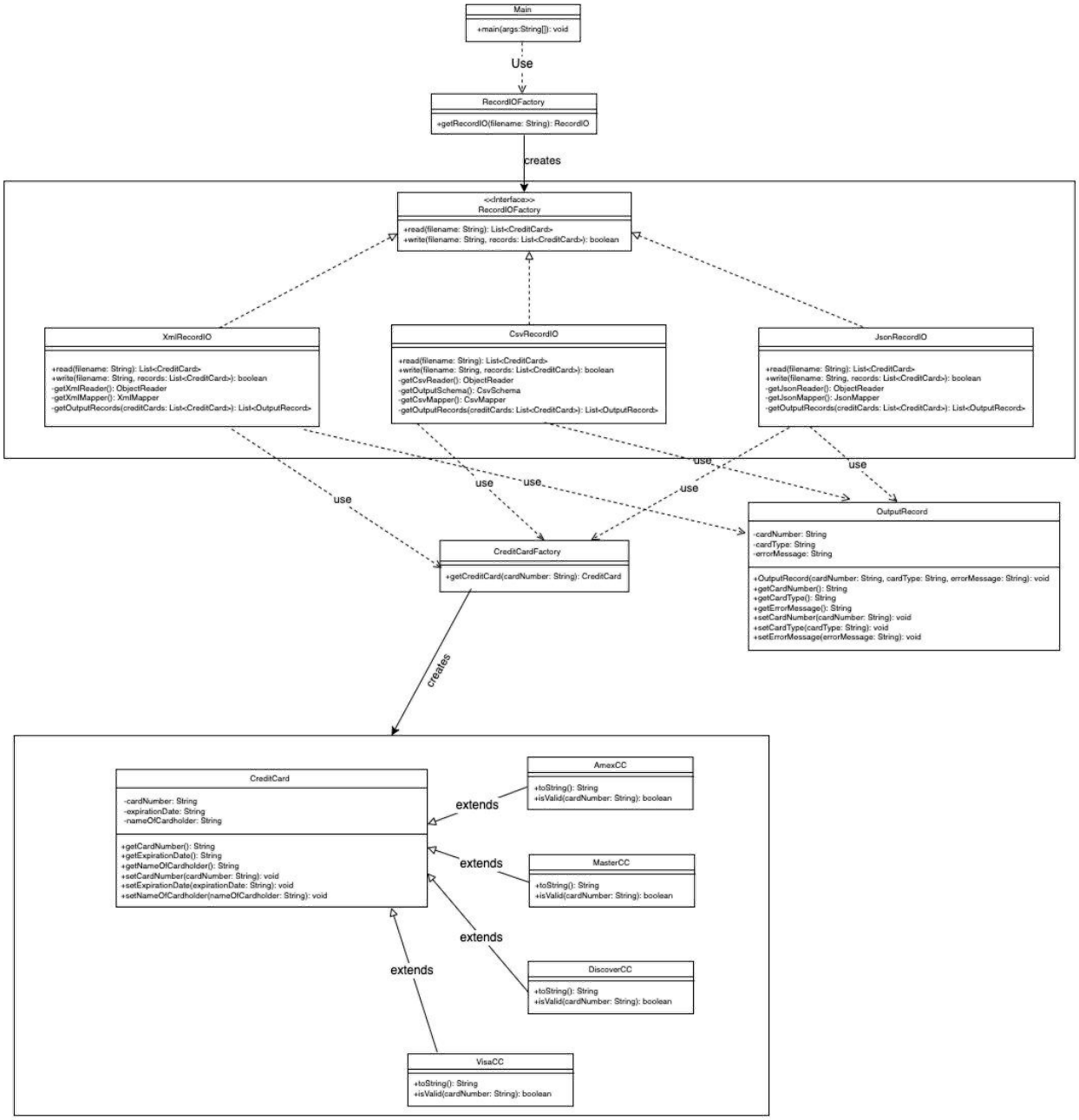
- By validating inputs in specific subclasses (AmExCC, DiscoverCC, MasterCC, VisaCC), various types of credit cards are generated.

- Using the OutputRecord class, each output record in the output file is generated.

Credit Card Object Instantiation Class Diagram:



Application Class Diagram



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

- The factory design simplifies the process of creating objects by abstracting the creation logic.

- This pattern allows for the seamless incorporation of new concrete subclasses in the future to manage additional types of objects.
- This ensures loose coupling as a result of the division of duties and responsibilities among subclasses. This facilitates future modifications and subclass additions that may be required.
- One drawback of this approach is that the abstraction makes the code challenging to read and comprehend.