CMPE 202 – INDIVIDUAL PROJECT

SAKRUTHI AVIRINENI – 016009929

PART 1 + CLASS DIAGRAMS

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

   Based on the given requirements, I found that the primary problem is as following:

   - Check whether the credit card details are valid or not based on the different types of credit cards available (Master Card, Visa, American Express, Discover).
   - To validate the type of credit card based on the given requirements like the credit card number length, should be fixed number or 12 or 14 digits.

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

   - The secondary problem I tried to solve is to identify the proper design pattern to support the addition of new credit class for the different types of credit cards in future.

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

   I choose **Factory design pattern**, because this design pattern creates objects based on the type of input.

   - The client sends the input file that contains credit card details and expects the creation of a particular kind of input file.

   - A new **RecordIO** object is created. Based on the type of file name specified by the client, an appropriate **_filetype_RecordIO** object is created in the RecordIOFactory.

   - In the specific _filetype_RecordIO subclasses (JsonRecordIO, CsvRecordIO, XmlRecordIO) there is a logic to read the card details in different types of the input file and write the expected output to that specified output file. A new credit card object of that particular type is created in the **CreditCardFactory** class.

   - In the subclasses, different types of credit cards are created based on the inputs. (DiscoverCC, MasterCC, VisaCC, AmExCC).

- Each output is created in the specified output file is created using OutputRecord class.

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

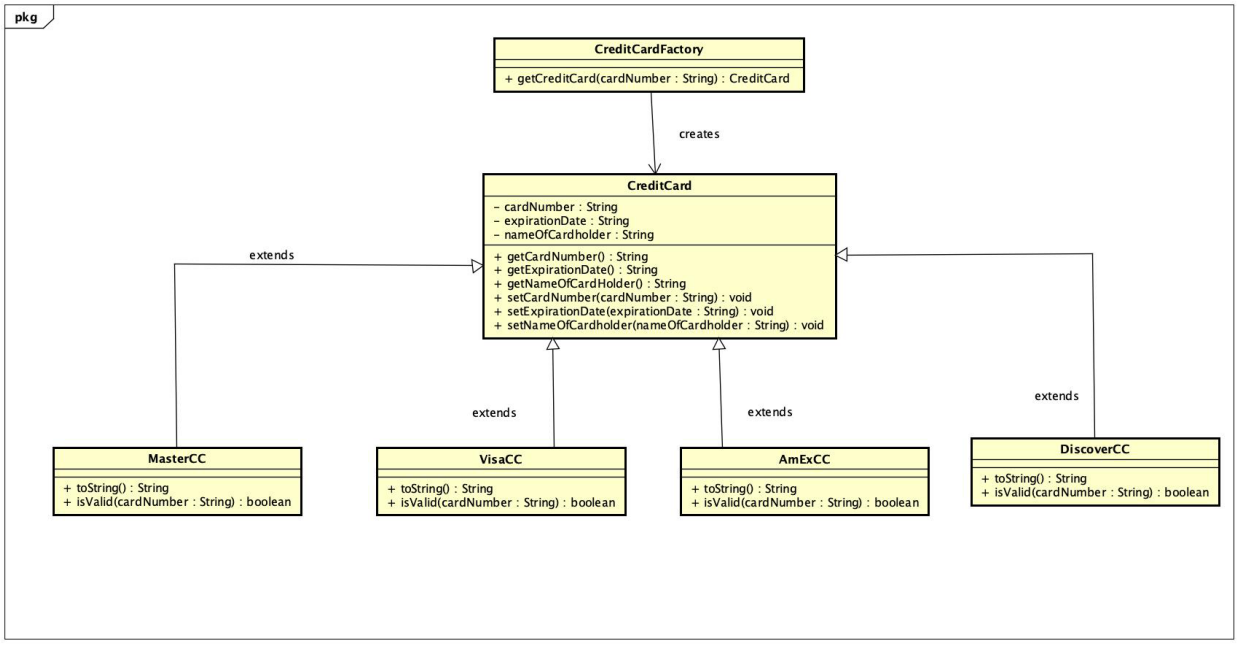
- Factory Design pattern helps in the creation of the object while removing the logic behind the creation.

- By using this pattern there is a chance of adding more subclasses to support new object types in the future.

- Ensures that there is a loose coupling because of the separation of logic and responsibilities among different subclasses. This facilitates modification and addition of subclasses that may be required in the future.

- One drawback of this approach is that the abstraction makes the code challenging to read and understand.

Class diagram of credit card object instantiation.



Class diagram of entire Application