CMPE 202 - Credit Card Problem

Individual Project

Adesh Landge - 016190486

---

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

   The primary problem was to identify the different types of credit card based on the given logic for card types and come up with an optimal solution to create the necessary objects (Visa, MasterCard, Amex, Discover).

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

   The second problem was thinking of design patterns to solve the credit card problem. I went through all the design patterns that would be suitable for the given requirements. After identifying the design patterns, it was critical for me to apply them properly for creating the objects as well as reading different types of input files and writing the processed credit card data to different types of output files.

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

   I used two design patterns for solving the credit card problem. The design patterns used are Factory pattern for creating different credit card objects and Strategy pattern for dynamically changing the behavior of the credit card application to make use of the selected strategy to choose from different file types.

   1. **Factory Pattern:**

      Factory Method is a creational design pattern that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created. I used the factory pattern because it replaces direct object construction calls with calls to a special factory method which helps us solve the problem of creating objects dynamically for Amex, Visa, Discover, MasterCard. This also helps us to easily add any other extra objects to be created if they come up in future.

2. **Strategy Pattern**

   Strategy pattern is a behavioral design pattern that lets you define a family of algorithms, put each of them into a separate class, and make their objects interchangeable.

   In our case for the credit card problem, we have the need to parse different types of input files (JSON, XML and CSV). Writing all the parsing code in the same class or method will make it harder if there are new input files in future. To support parsing of newer input files and writing data to any new type of file in future, strategy pattern seemed to be an ideal option. In my code, I have created 2 different interfaces FileReaderStrategy and FileWriterStrategy for parsing and writing to files. And each of the three different classes JsonReader, XmlReader and CsvReader implements the interface FileReaderStrategy and each of of the three different classes JsonWriter, XmlWriter and CsvWriter implements the interface FileWriterStrategy.

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

   Strategy Pattern

   Pros:

   - We can swap algorithms for parsing and writing to files used inside an object at runtime.
   - We can add new strategies without having to change the context.

   Cons:

   - It could be difficult to identify the differences between strategies to be able to select a proper for our credit card app.
   - This pattern complicates the codebase with new classes and interfaces which are not really required in our case as we only have few input and output file formats for now.

   Factory Pattern

   Pros:

   - You can move the product creation code into one place in the program, making the code easier to support.

   Cons:

- The code may become more complicated since you need to introduce a lot of new subclasses to implement the pattern.