CMPE 202 Individual project
ID: 017455451

Part 1:

1. What is the primary problem you try to solve?

   The primary problem would be to create the appropriate card object at runtime based on the input value. We can create an interface or abstract class and subclasses for each credit card type but we would need to avoid using switches or if-else loops when we use these objects for code extensibility. Implementing switches for object creation would create an issue if any other types of cards need to be added in the future.

2. What are the secondary problems you try to solve?

   One of the secondary problems is using the above design patterns, how can we implement the logic to identify the card type.
   The other problem would be to implement the SOLID principle 'Open to enhancement closed to modification' for the input file process. The system should be able to include multiple file formats in the future without disturbing the code that already exists.

3. What design pattern(s) do you use and how (use plain text and diagrams)?

   The design patterns we can use for differentiating card types and allocating the correct card class would be: Factory and Chain of responsibility.

   The chain of responsibility can be used to determine the type of credit card at runtime and implement the appropriate logic to differentiate between various credit card numbers. We can simply put the logic in each credit card subclasses to check if the current number fits into this card category. If it does, we can return that class object, else it will pass the number to its successor. The factory class would simplify the setup of the chain and return appropriate objects, instead of implementing the COR multiple times.

Diagram:

https://drive.google.com/file/d/1UXmcJfaMrF0uHHWrI-lbikOrexe46j0p/view?usp=sharing

https://github.com/gopinathsjsu/individual-project-SayaliVB/blob/main/Class%20Diagram%20part%201.png

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

   By using the chain of responsibility design pattern, we can not only avoid using if-else rules and switches to create appropriate class objects, but also, the logic to determine whether the card number is of specific type is defined within the class itself. So, if we have to make any changes or add additional eligibility criteria, we do not have to modify the factory and add more if- else loops, but update the class itself, demonstrating the use of Single responsibility principle.
   We used factory pattern to avoid setting the successor chain or differentiating criteria for every time we have to determine a credit card number, in turn reducing the code repetition. This pattern also helps us achieve code extensibility as we can create a new type of credit class and it won't affect our entire code.