**Q1:** In the given scenario, our primary target is to create a robust and scalable system where Credit cards are segregated based on records coming from different files and each record falls under either a Credit Card type or it does not. Once the type has been identified, the system must return the appropriate instance of the Credit Card so that further operations can be performed. At the same time, we will have to make sure that the system follows Open Closed principle that states that the system should be open for extension and closed for modification. Following this principle, we will have to make sure that whenever a new Credit Card type is to be added or new conditions are added for segregation, there is no modification to the existing code.

**Q2.** There are 2 secondary problems that we need to take care while designing the system. The first involves making sure the classes are loosely coupled. This can be achieved using interfaces rather that concrete classes. Another problem which is basically an edge case. When the record in question does not fall under any category of credit cards, a default object needs to be returned.

**Q3 and Q4:** Since, there are 2 different problems we are trying to solve, we will be using 2 different design patterns that handles each of the scenarios very efficiently. The design patterns that we will be using are as follows:

1. **Chain of responsibility:**
   1. **Reason to Use:** The step in the process is to identify the type of the credit card. The chain of responsibility works in similar way checking handling a particular responsibility and if not satisfied, it passes the inputs to the next in chain.
   2. **Consequence**: Using this design pattern gives us the flexibility to add multiple responsibilities which can be added and removed as per our convenience. This will come handy when multiple types of Cards need to be added later.

A screenshot of a cell phone

Description automatically generated

1. **Abstract Factory:**
   1. **Reason to Use:** Once the type of card has been identified from the above step, we would need to return the appropriate instance that would be used further by the client. This is when we will call the appropriate factory and generate the exact instance of the credit card. The different factories will produce different types of instances of the card. Ex – The MasterCardFactory produces Mastercards.
   2. A picture containing text, map

      Description automatically generated**Consequence:** This will help to abstract all the card specific implementations from the client as well as insulates the client from modifying the implementation when new card types are added.

**New Interface / Class additions**

1. **Reader (Interface) –** Implemented by concrete classes CSVFileReader, XMLReader and JSONReader. These classes are responsible for reading specific file formats. Once the information are extracted from the files, they are put into an ArrayList<VO> which is then returned by the readFile function and used for further processing.
2. **Writer (Interface) –** Implemented by concrete classes CSVFileWriter, XMLWriter and JSONWriter. Once the specific credit cards have been generated using the different Credit Card Factories, they are put into an ArraytList<CreditCard> and fed as input to the write method of the Writer interface. This method is then responsible to write to the file in the same format as the input file.
3. **A picture containing screenshot

   Description automatically generatedUML diagrams for Reader and Writer**

**Program Flow Diagram**

**A screenshot of a cell phone

Description automatically generated**