**Part 1**

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

The primary problem that needs to be solved is **to identify the type of input file**. Followed by it, the read and write operations needs to be performed from that particular file.

It comprises of the below steps

1. Identify the type of file.
2. Follow the **strategy pattern** to use the object of a specific type of file and perform the read and write operations from the specific concrete strategy class.

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

The secondary problem that is being attempted to be solved here is to **detect the type of card**. The following steps needs to be adopted to solve it.

1. From the data that is read from the input file, extract the card number and pass it to the Card detector factory method class.
2. Use the **factory method** to create the object of the respective credit card class and pass it to the file handler class.

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

The design patterns used are as under.

1. Strategy Pattern

This pattern is utilised when a specific file type is identified to perform read and write operations

**Participants**

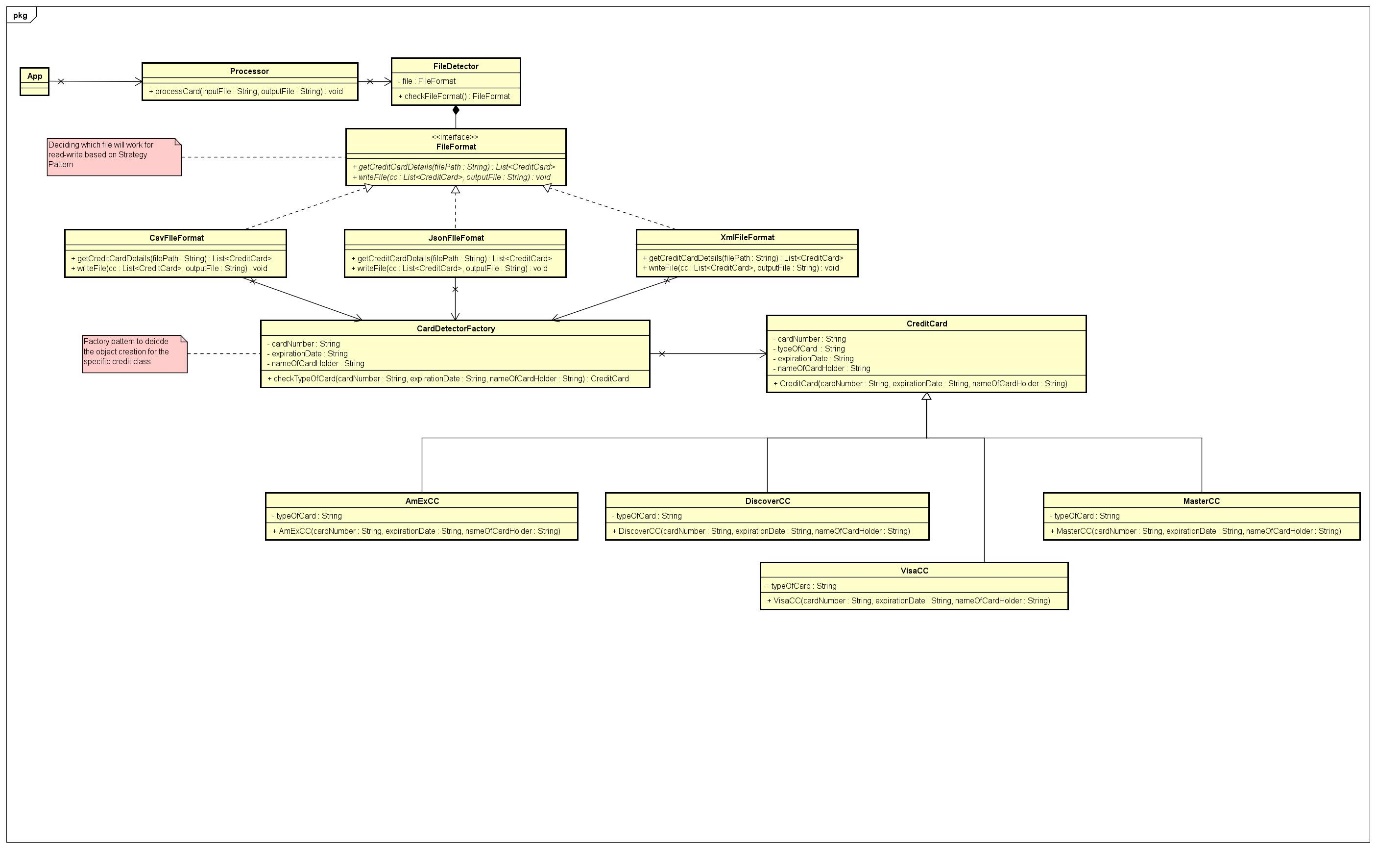
* **Strategy**: FileFormat.java interface declaring the functions for read and write operations
* **Concrete Strategy**: CsvFileFormat.java, JsonFileFormat.java, XmlFileFormat.java implements FileFormat interface and defines the body of the read and write operations
* **Context**: FileFactory.java is the context which maintains a reference to the strategy object

1. Factory Pattern

This pattern is used to identify the type of card and create the object corresponding to that class of which the card is detected to belong.

**Participants**

* **Concrete Factory:** CardDetectorFactory.java is the factory class that identifies the type of card and creates the corresponding object of that specific class**.**
* **Concrete Product:** AmExCC.java, DiscoverCC.java, MasterCC.java, VisaCC.java are the concrete product whose objects are created by the Factory class.
* **Client:** CsvFileFormat.java, JsonFileFormat.java, XmlFileFormat.java that invokes the CardDetectorFactory.java

****

Class Diagram showing the design pattern

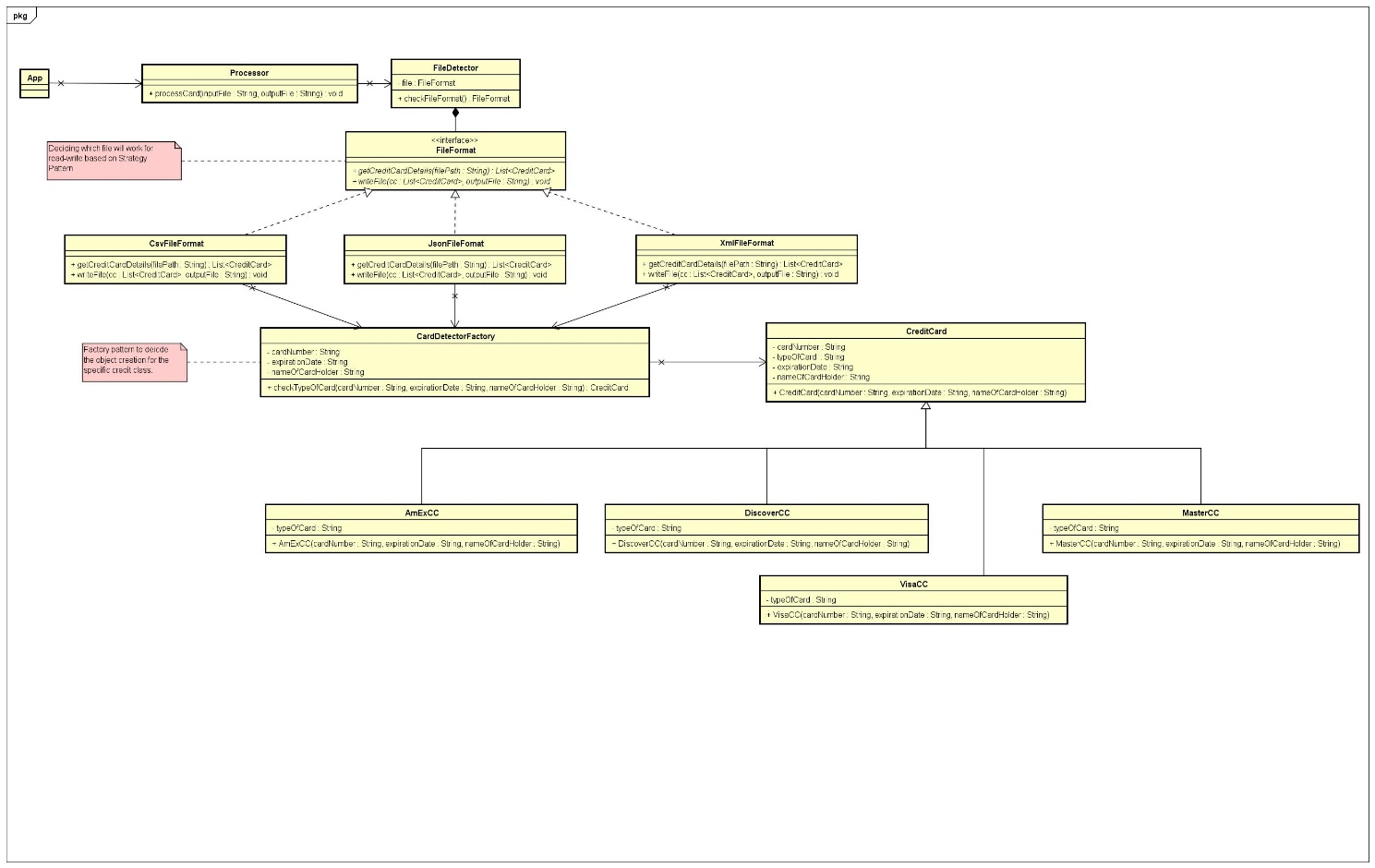
**Describe the consequences of using this/these pattern(s)**

The consequences of using the above showcased design patterns are as follows:

1. The strategy pattern has helped in fixing the operations for any type of file. If the new file type is included, a new Concrete Strategy implementation class would be written. There won’t be any change to other concrete.
2. The factory method pattern helps in identifying the type of card, so with the addition of any new type of card, only changed would be made is in the CreditCardDetector.java file with some add-on check conditions and a separate class extending from the parent class CreditCard.java would be written. Thus, existing classes won’t be affected.
3. It has helped in maintaining the code and also made it easy to understand the flow of the code.
4. The code is inline with SOLID principles thus, is open for extension also.

**Part 2**

**Design Pattern Implementation using class diagram**

****