# Part 1:

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

The primary problem to solve here is to apply design patterns to find out which card type a particular card number is and then to create an object of that particular card type class. Also the design should be such that further addition of any card types will not violate the Open-closed principle and it should be hassle free.

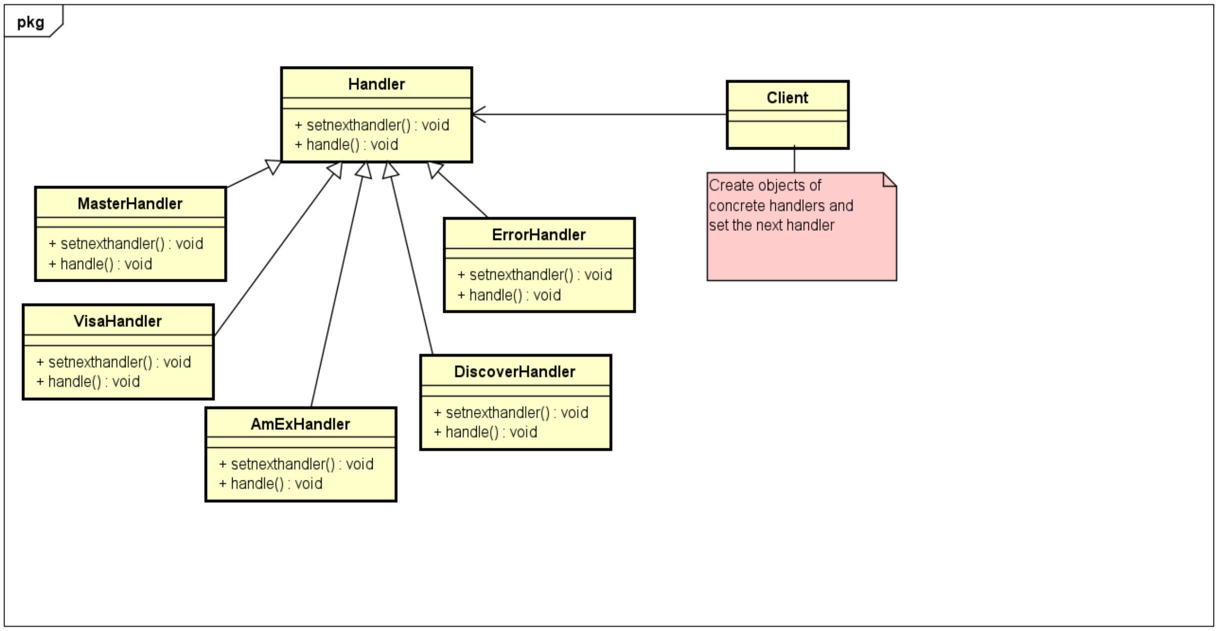
## 2. Describe what are the secondary problems you try to solve (if there are any).

Some secondary problems are how to parse the csv file and how to retrieve card number from the file.

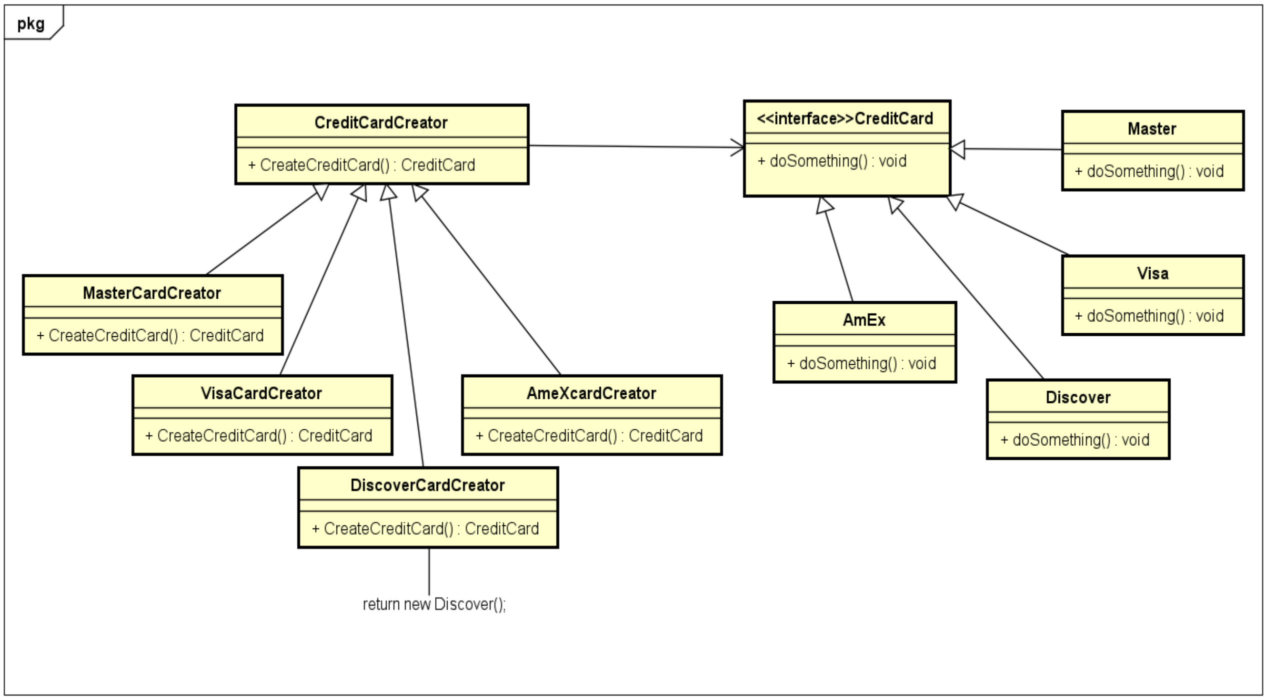
## 3. Describe what design pattern(s) you use how (use plain text and diagrams).

The design patterns used in my solution are:

1. **Chain of Responsibility**: This behavioural design pattern would enable to find out which card type a particular card number is. For different card types there is a handler like masterCardHandler, visaCardHandler etc. Each handler will process the card number to check if it can handle it, else will pass to the next handler. A card number will pass through the handlers until a card is of the available card types and if it does not belong to any handlers, then a final errorCardHandler will process it.


pkg
Handler
+ setnexthandler() : void
+ handle() : void
Client
Create objects of concrete handlers and set the next handler
MasterHandler
+ setnexthandler() : void
+ handle() : void
ErrorHandler
+ setnexthandler() : void
+ handle() : void
VisaHandler
+ setnexthandler() : void
+ handle() : void
DiscoverHandler
+ setnexthandler() : void
+ handle() : void
AmExHandler
+ setnexthandler() : void
+ handle() : void

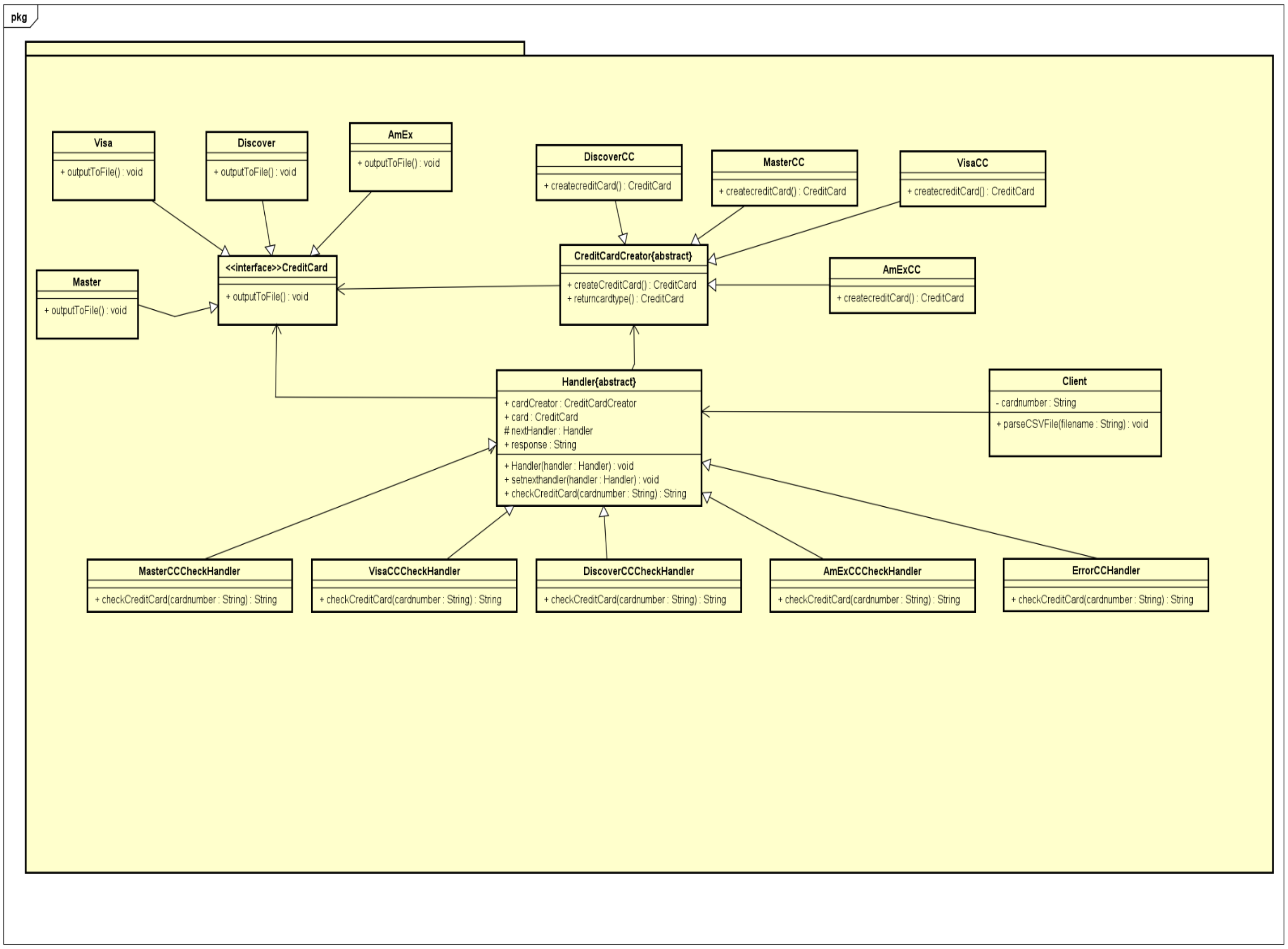
2. **Factory method :** This creational design pattern is used to create the objects of the particular card class type. There are cardCreators for each card type. Once the handler will find out which card type a particular card number is, it will call the cardCreator to create the object of that card type.



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

COR- It follows Open-Closed Principle as for new card types handling in future, we can introduce new handlers without breaking the existing client code. Also it follows Single Responsibility as each handler is focussed on one task.

Factory Method: For new card types object creation, we need to add more subclasses and it will not require modification in the existing client code.It will be an extension. And each creator is concerned for only their object creation. Thus it follows Open-Closed and Single Responsibility principle. A negative consequence is as card types increase we will have more subclasses.


pkg
Visa
+ outputToFile() : void
Discover
+ outputToFile() : void
AmEx
+ outputToFile() : void
DiscoverCC
+ createcreditCard() : CreditCard
MasterCC
+ createcreditCard() : CreditCard
VisaCC
+ createcreditCard() : CreditCard
Master
+ outputToFile() : void
<<interface>>CreditCard
+ outputToFile() : void
CreditCardCreator{abstract}
+ createCreditCard() : CreditCard
+ returncardtype() : CreditCard
AmExCC
+ createcreditCard() : CreditCard
Handler{abstract}
+ cardCreator : CreditCardCreator
+ card : CreditCard
# nextHandler : Handler
+ response : String
+ Handler(handler : Handler) : void
+ setnexthandler(handler : Handler) : void
+ checkCreditCard(cardnumber : String) : String
Client
- cardnumber : String
+ parseCSVFile(filename : String) : void
MasterCCCheckHandler
+ checkCreditCard(cardnumber : String) : String
VisaCCCheckHandler
+ checkCreditCard(cardnumber : String) : String
DiscoverCCCheckHandler
+ checkCreditCard(cardnumber : String) : String
AmExCCCheckHandler
+ checkCreditCard(cardnumber : String) : String
ErrorCCHandler
+ checkCreditCard(cardnumber : String) : String