README.md

# Problem Statement

### Git Link: https://github.com/gopinathsjsu/individual-project-harshanirudh

Requirements:

The application should maintain an internal, static database (inventory of stock) (this may be developed using HashMaps and/or other built-in Java Data structures). This means once we re-run the program, the changes to the data would not persist. We will provide the data that has to be maintained. The data will contain the following tables and fields:
(Sample input file and sample data set for the inventory would be provided separately)
Table 1: Items
Category (Essentials, Luxury, Miscellaneous)
Item for each category (Essentials - Clothes, soap, milk; Luxury - perfume, chocolates; Misc - Bedsheets, footwear)
The available Quantity of each item
Price of each item
Table 2: Cards
Card Numbers
2. Input CSV file will contain an order including Items, Quantity needed, and the payment card number.
3. Input file should be processed as follows:

- Validate if the requested quantity for each item is permissible. For example, if the request is to order 3 soaps, check the database if we have at least 3 soaps in our inventory.
- There will be a cap on the quantity of each category that can be ordered in one single order. For example, restrict Essentials to a maximum of 3, Luxury to 4, and Misc to 6. (This will be configured beforehand)
- In case it is an incorrect request, generate and output TXT file with message "Please correct quantities." and include the items with incorrect quantities

- After this validation, if the cart is valid, calculate prices for the cart.
- Take the card number of the user and if it is not present in DB add it.
  Output the CSV list with the total amount paid.

Please refer to the attached file for Inventory, Sample Input and output files. If you cannot proceed with an input transaction for any reasons mentioned above, generate an output TXT with a reason for the same.

## Steps to run the program

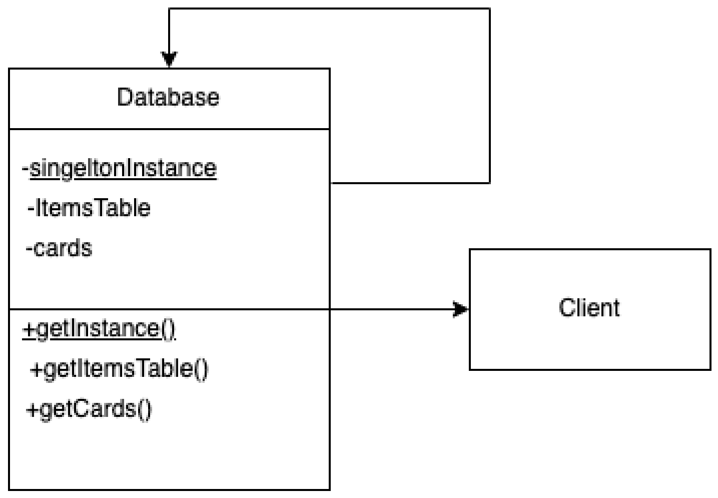
#### Java 8 needs to be installed

- Open CMD or Terminal
- Download the JAR file from the repository present in **bin** folder.
- Run the following command in terminal
  `java -jar <jar_name> <input_file_location>`
- For example: `java -jar Inventory-billing-0.0.1-SNAPSHOT.jar input.csv`
- Output files shall be created in **OUT** directory at the root of the folder
- It contains both the error and output files

## Design Patterns Used:

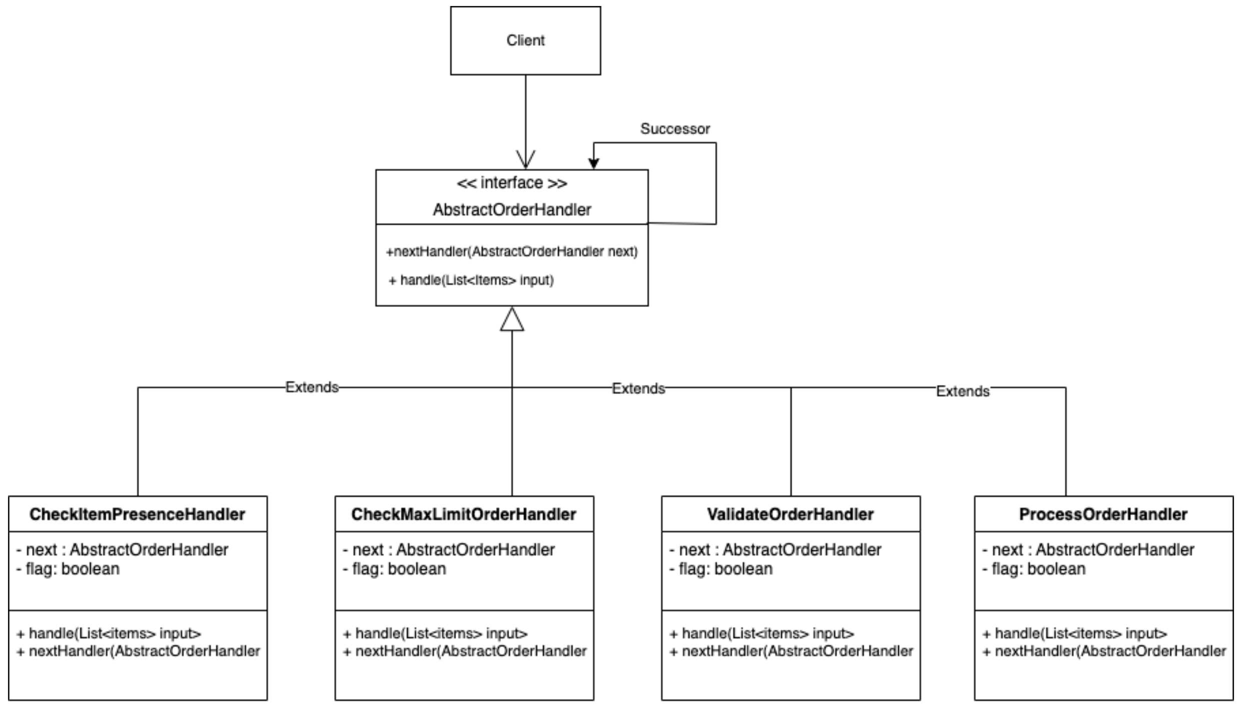
1. Singleton pattern

    - Singleton pattern is one of the simplest design patterns and it comes under creational design pattern
    - This pattern lets the client use the class only with one object.
    - Whenever an object is needed the same object is given.
    - Here, the inventory database has been implemented with singleton pattern, so that we can mimic a real database where it has single source of truth.
    - Whenever the database is required , a static call to `getInstance()` method gives the object.

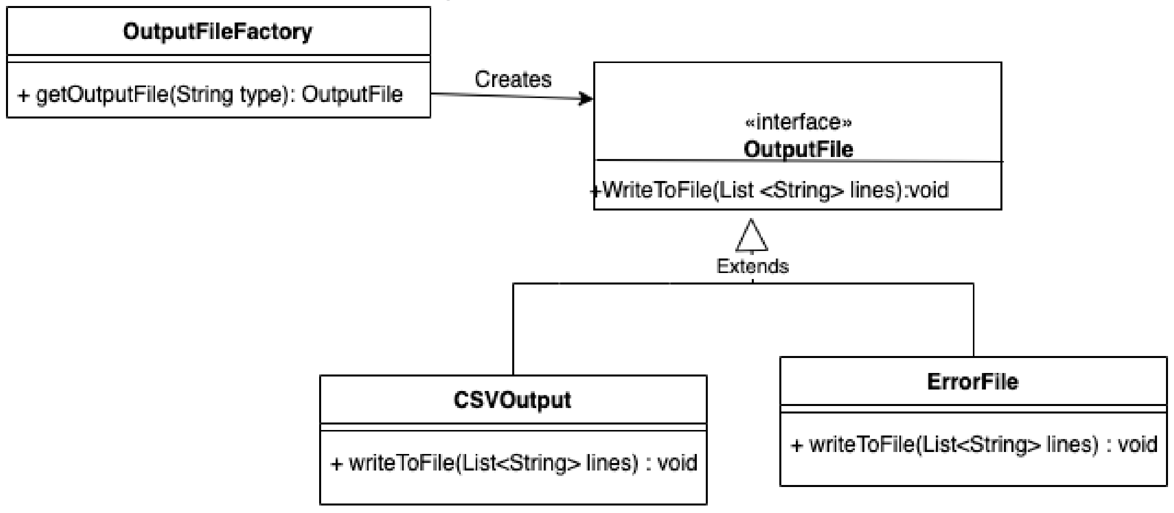


2. Chain of Responsibility pattern

   - This is behavioural pattern.
   - It is very useful pattern when the request has to be handled and prcoessed by multiple classes/objects.
   - Base class defines the methods signatures and the concerate classes override the behaviors and a chain is created .
   - In the chain the request is passed along till the appropriate class processes it and gives the result.
   - Here the base class is AbstractOrderHandler and we have 4 concerete handlers
     - **CheckItemPresenceHandler**: Check if items are present in inventory or not. If present passes to next handler in the chain.
     - **CheckMaxLimitOrderHandler**: Checks if the items in the order obey the maximum category as defined in the problem statement.If yes, passes down the chain
     - **ValidateOrderHandler**: Checks if the items quantity in the order is availabile in the inventory. If yes, passes the request down the chain.
     - **ProcessOrderHandler** : Finally this class processes the order and the request is fulfilled.


Client
Successor
<< interface >>
AbstractOrderHandler
+nextHandler(AbstractOrderHandler next)
+ handle(List<Items> input)
Extends
Extends
Extends
CheckItemPresenceHandler
CheckMaxLimitOrderHandler
ValidateOrderHandler
ProcessOrderHandler
- next : AbstractOrderHandler
- flag: boolean
+ handle(List<items> input>
+ nextHandler(AbstractOrderHandler

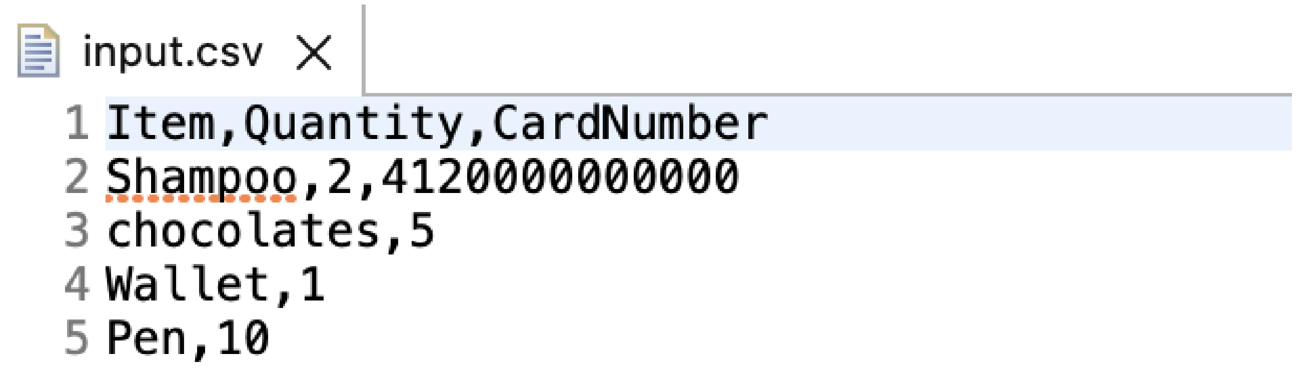
3. Factory Pattern
   - This is also a creational design pattern.
   - I have used this pattern to create output file and error file as required.
     - Base Class: OutputFile
     - Concrete Class 1: CSVOutput
     - Concrete Class 2: ErrorFile
   - Factory pattern allows sub classes to choose the type of objects to create.
   - it also promotes loose coupling


OutputFileFactory
+ getOutputFile(String type): OutputFile
Creates
«interface»
OutputFile
+WriteToFile(List <String> lines):void
Extends
CSVOutput
+ writeToFile(List<String> lines) : void
ErrorFile
+ writeToFile(List<String> lines) : void

## Screenshots

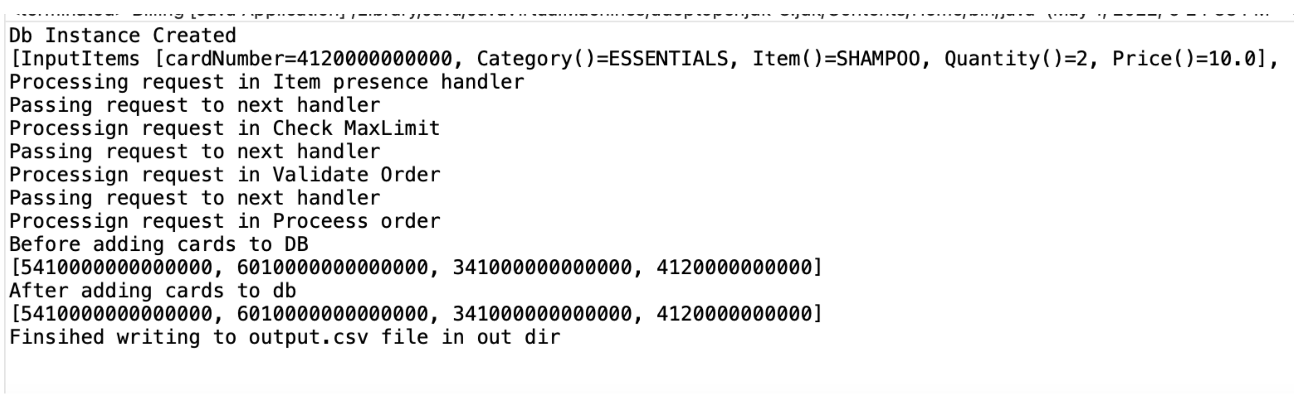
## Test Case 1

### Input

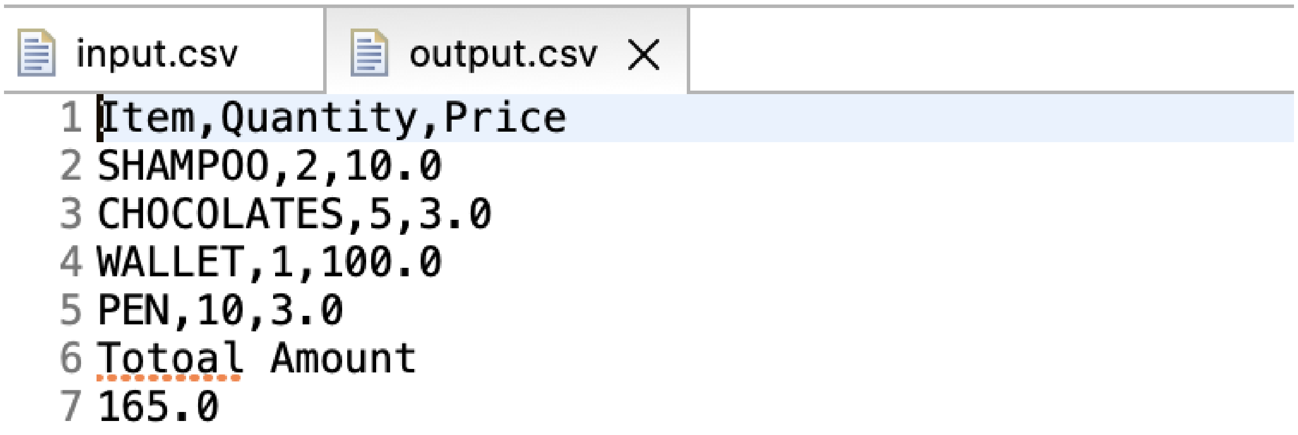

input.csv

```
Item,Quantity,CardNumber
Shampoo,2,4120000000000
chocolates,5
Wallet,1
Pen,10
```

### Output



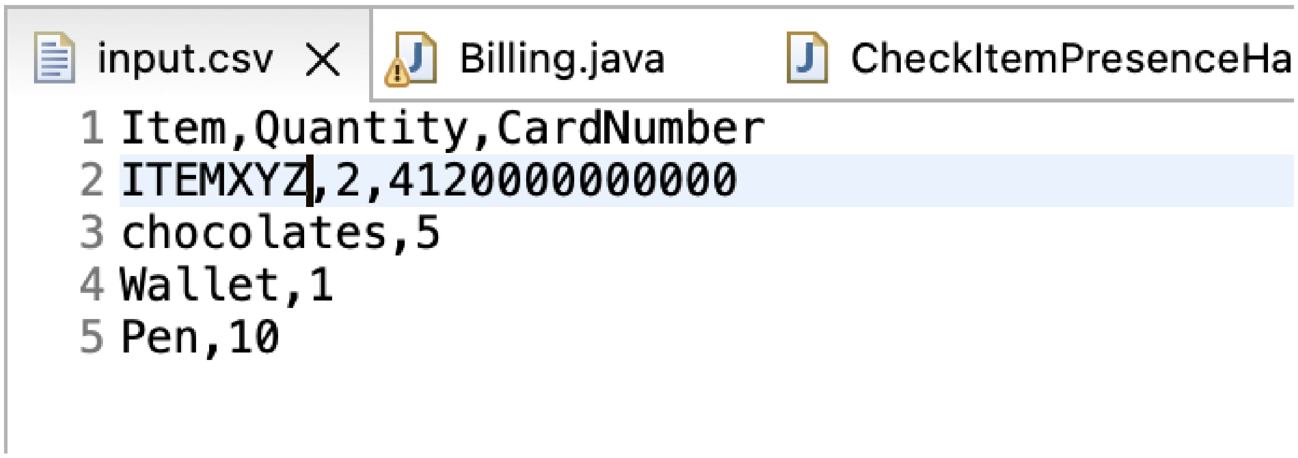
```
Db Instance Created
[InputItems [cardNumber=4120000000000, Category()=ESSENTIALS, Item()=SHAMPOO, Quantity()=2, Price()=10.0],
Processing request in Item presence handler
Passing request to next handler
Processign request in Check MaxLimit
Passing request to next handler
Processign request in Validate Order
Passing request to next handler
Processign request in Proceess order
Before adding cards to DB
[5410000000000000, 6010000000000000, 341000000000000, 4120000000000]
After adding cards to db
[5410000000000000, 6010000000000000, 341000000000000, 4120000000000]
Finsihed writing to output.csv file in out dir
```


input.csv output.csv

```
Item,Quantity,Price
SHAMPOO,2,10.0
CHOCOLATES,5,3.0
WALLET,1,100.0
PEN,10,3.0
Totoal Amount
165.0
```

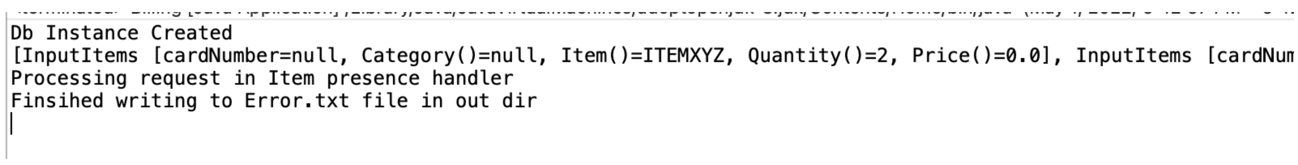
## Test Case 2

### Input

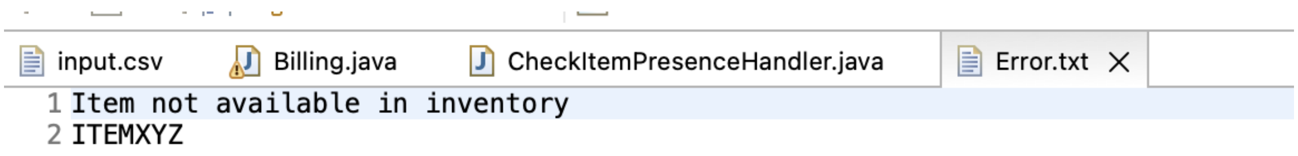


```
Item,Quantity,CardNumber
ITEMXYZ,2,4120000000000
chocolates,5
Wallet,1
Pen,10
```

## Output



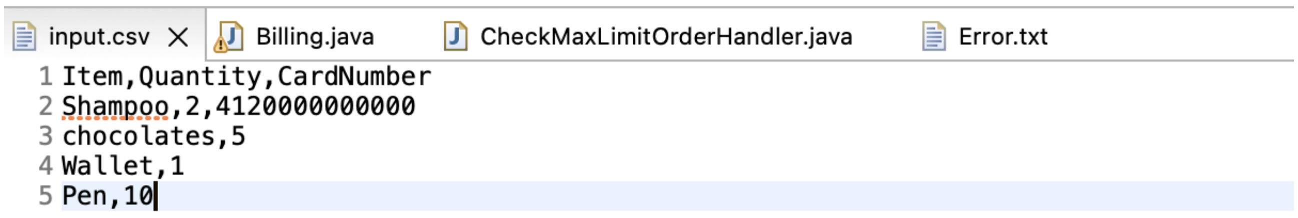
```
Db Instance Created
[InputItems [cardNumber=null, Category()=null, Item()=ITEMXYZ, Quantity()=2, Price()=0.0], InputItems [cardNum
Processing request in Item presence handler
Finsihed writing to Error.txt file in out dir
```



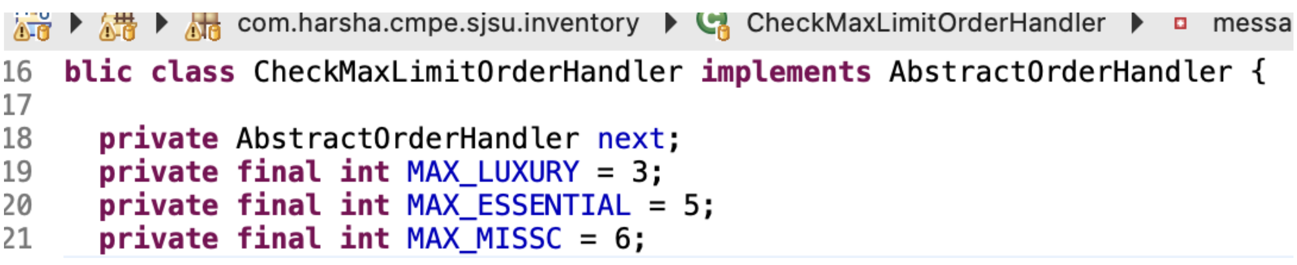
```
Item not available in inventory
ITEMXYZ
```

## Test Case 3

## Input



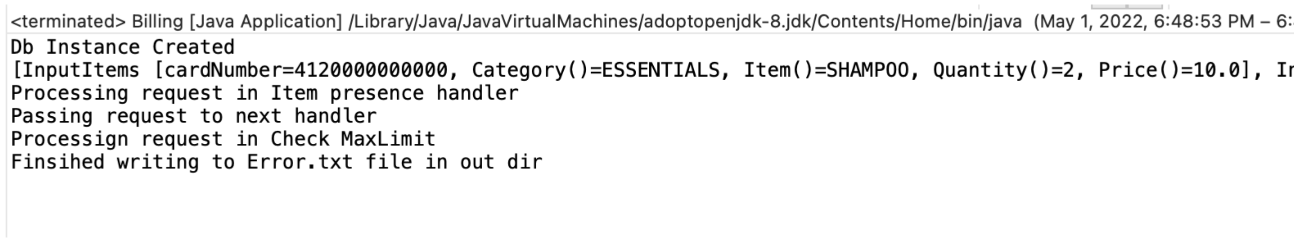
```
Item,Quantity,CardNumber
Shampoo,2,4120000000000
chocolates,5
Wallet,1
Pen,10
```



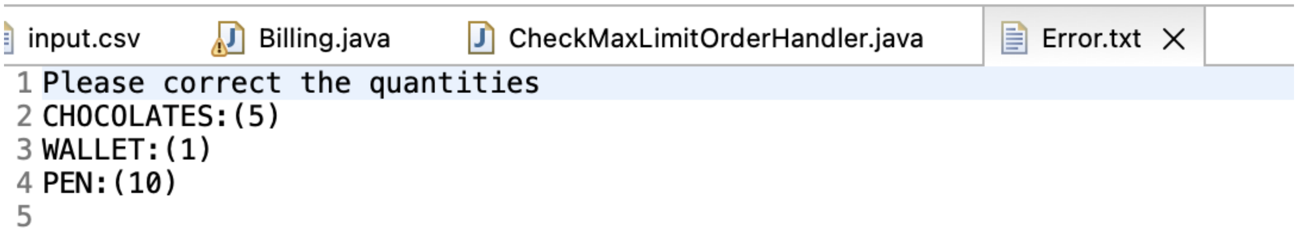
```
blic class CheckMaxLimitOrderHandler implements AbstractOrderHandler {

  private AbstractOrderHandler next;
  private final int MAX_LUXURY = 3;
  private final int MAX_ESSENTIAL = 5;
  private final int MAX_MISSC = 6;
```

## Output



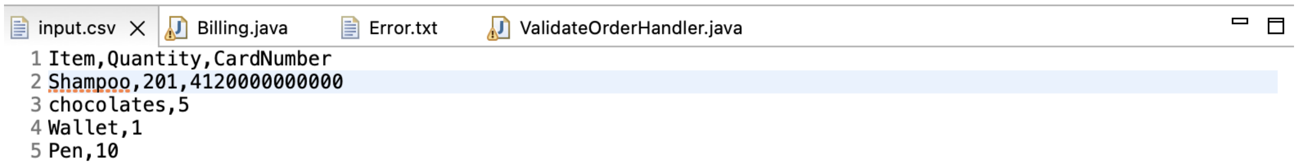
```
<terminated> Billing [Java Application] /Library/Java/JavaVirtualMachines/adoptopenjdk-8.jdk/Contents/Home/bin/java  (May 1, 2022, 6:48:53 PM – 6:
Db Instance Created
[InputItems [cardNumber=4120000000000, Category()=ESSENTIALS, Item()=SHAMPOO, Quantity()=2, Price()=10.0], In
Processing request in Item presence handler
Passing request to next handler
Processign request in Check MaxLimit
Finsihed writing to Error.txt file in out dir
```



```
Please correct the quantities
CHOCOLATES:(5)
WALLET:(1)
PEN:(10)
```

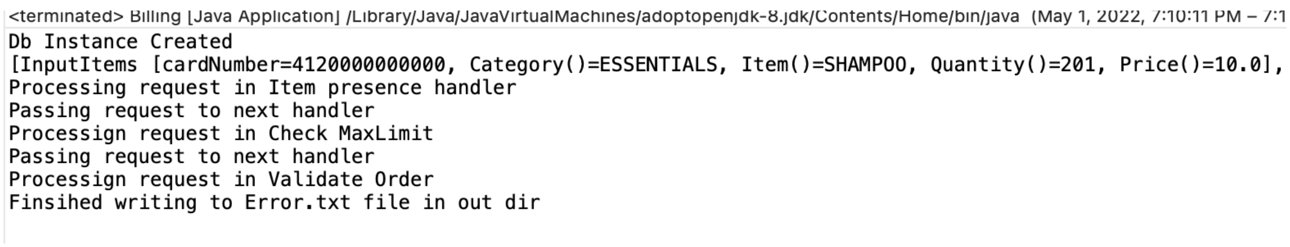
## Test Case 4

## Input

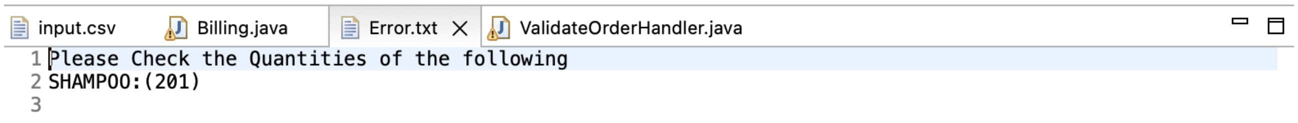


```
Item,Quantity,CardNumber
Shampoo,201,4120000000000
chocolates,5
Wallet,1
Pen,10
```

## Output

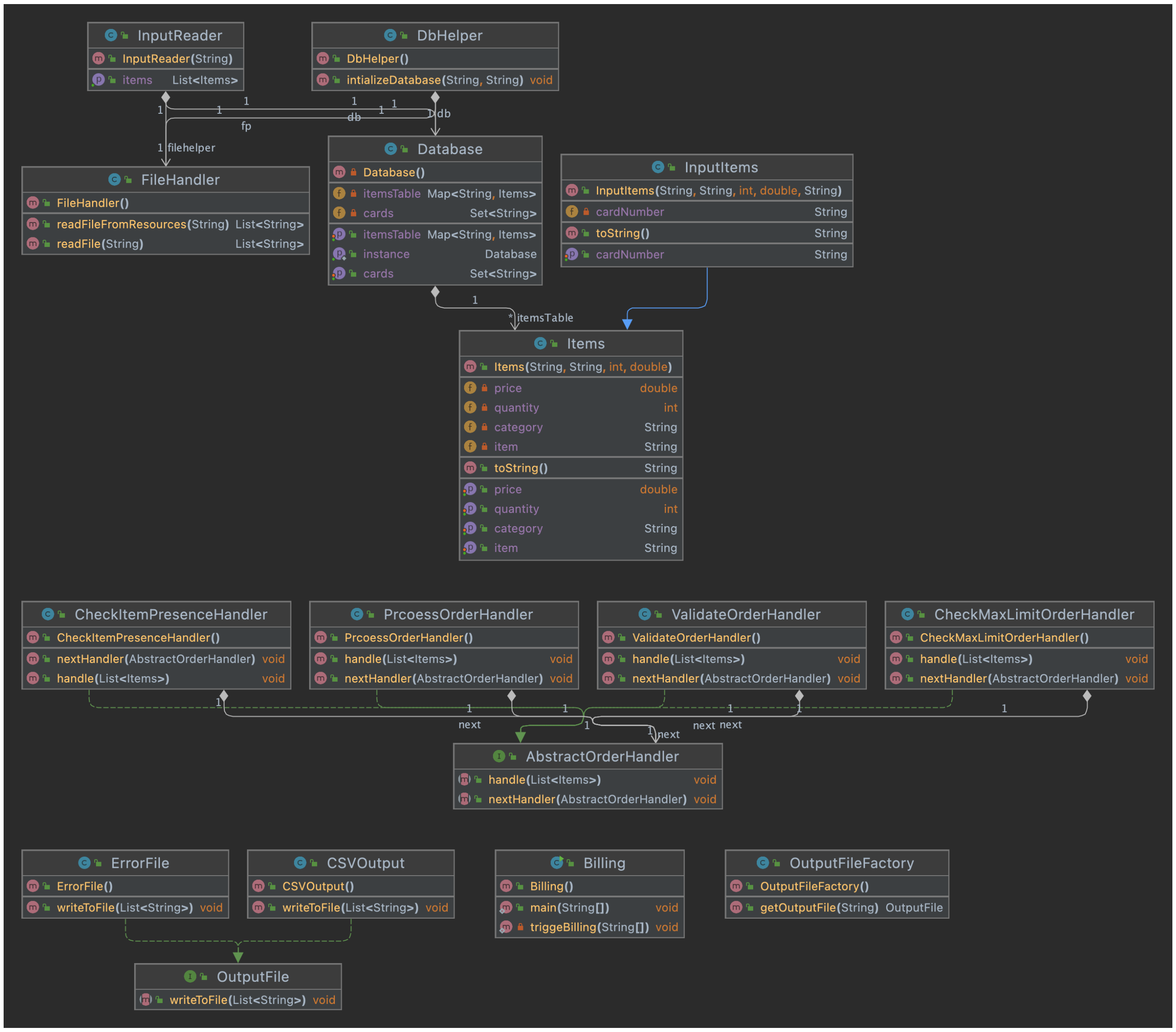


```
<terminated> Billing [Java Application] /Library/Java/JavaVirtualMachines/adoptopenjdk-8.jdk/Contents/Home/bin/java  (May 1, 2022, 7:10:11 PM – 7:1
Db Instance Created
[InputItems [cardNumber=4120000000000, Category()=ESSENTIALS, Item()=SHAMPOO, Quantity()=201, Price()=10.0],
Processing request in Item presence handler
Passing request to next handler
Processing request in Check MaxLimit
Passing request to next handler
Processign request in Validate Order
Finsihed writing to Error.txt file in out dir
```



```
Please Check the Quantities of the following
SHAMPOO:(201)
```

## Class Diagram of Entire Project


InputReader
InputReader(String)
items List<Items>
DbHelper
DbHelper()
intializeDatabase(String, String) void
1
db
fp
1 filehelper
FileHandler
FileHandler()
readFileFromResources(String) List<String>
readFile(String) List<String>
Database
Database()
itemsTable Map<String, Items>
cards Set<String>
itemsTable Map<String, Items>
instance Database
cards Set<String>
InputItems
InputItems(String, String, int, double, String)
cardNumber String
toString() String
cardNumber String
1
* itemsTable
Items
Items(String, String, int, double)
price double
quantity int
category String
item String
toString() String
price double
quantity int
category String
item String
CheckItemPresenceHandler
CheckItemPresenceHandler()
nextHandler(AbstractOrderHandler) void
handle(List<Items>) void
PrcoessOrderHandler
PrcoessOrderHandler()
handle(List<Items>) void
nextHandler(AbstractOrderHandler) void
ValidateOrderHandler
ValidateOrderHandler()
handle(List<Items>) void
nextHandler(AbstractOrderHandler) void
CheckMaxLimitOrderHandler
CheckMaxLimitOrderHandler()
handle(List<Items>) void
nextHandler(AbstractOrderHandler) void
next
next next
AbstractOrderHandler
handle(List<Items>) void
nextHandler(AbstractOrderHandler) void
ErrorFile
ErrorFile()
writeToFile(List<String>) void
CSVOutput
CSVOutput()
writeToFile(List<String>) void
Billing
Billing()
main(String[]) void
triggeBilling(String[]) void
OutputFileFactory
OutputFileFactory()
getOutputFile(String) OutputFile
OutputFile
writeToFile(List<String>) void