Student Name: Karishma Kuria Student Id: 015947191

Individual Project Report

Problem Statement:

You will build a flight booking application using at least three design patterns. The application should maintain an internal, static database (inventory of flight details) (this may be developed using Hash Map's 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 have to be maintained. The data will contain the following tables and fields:

Table 1: Flights

- **Category** (Economy, Premium Economy, Business)
- **Flight number**
- The available **Seats** of each category
- **Price** of each seat
- **Arrival City**
- **Departure City**

- Input CSV file will contain booking details including booking name, flight number, seat category, number of seats, and the payment card number.

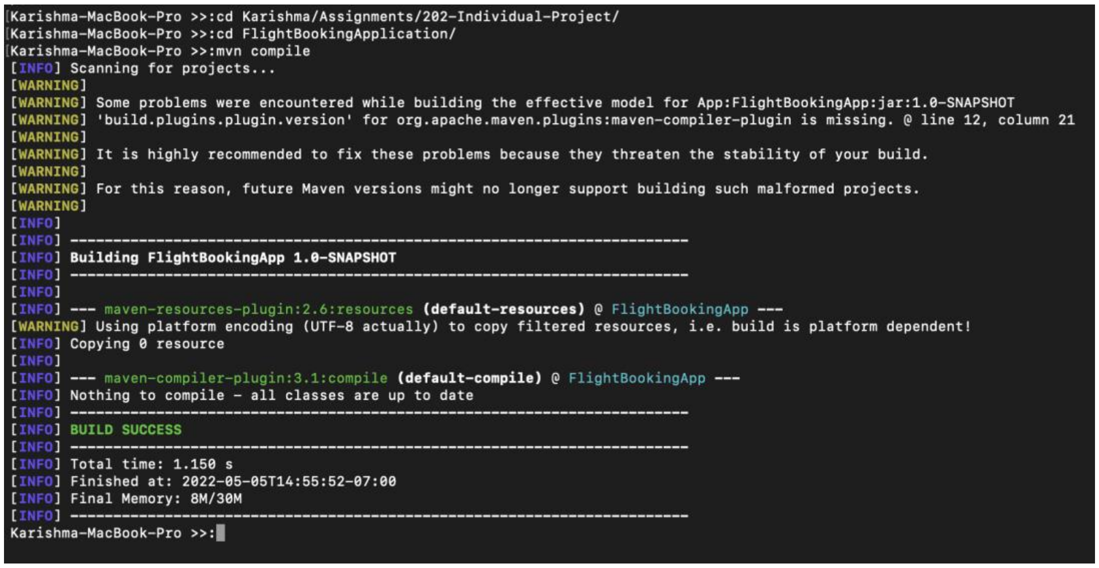
2. Input file should be processed as follows:

- Validate if the requested flight exists.
- If the flight exists, validate the number of seats requested for the category.
- After this validation, if the booking is valid, calculate the total price (NoOfSeats * price)
- Take the card number of the user and validate it using the given rules:
  - Visa card: has length either 13 or 16. It begins with a 4
  - MasterCard: has length 16. Begins with 5 and the 2nd digit begins from 1 to 5 inclusive
  - Discover: length 16, and the first 4 digits begins from 6011
  - Amex: has length 15 and starts with 3. 2nd digit must be 4 or 7
  - Any card greater than 19 or not satisfying above conditions is considered invalid.
- If the card is valid then modify the available seats for that category and flight number
- Then output the CSV list with booking name, flight number, Category, number of seats booked, total price.
- In case, it is an incorrect request at any of the steps, generate and output TXT file with the message "Please enter correct booking details for <booking name>:<reason>" and include the information with incorrect information. For example, please enter correct booking details for John: invalid flight number.

Student Name: Karishma Kuria   Student Id: 015947191

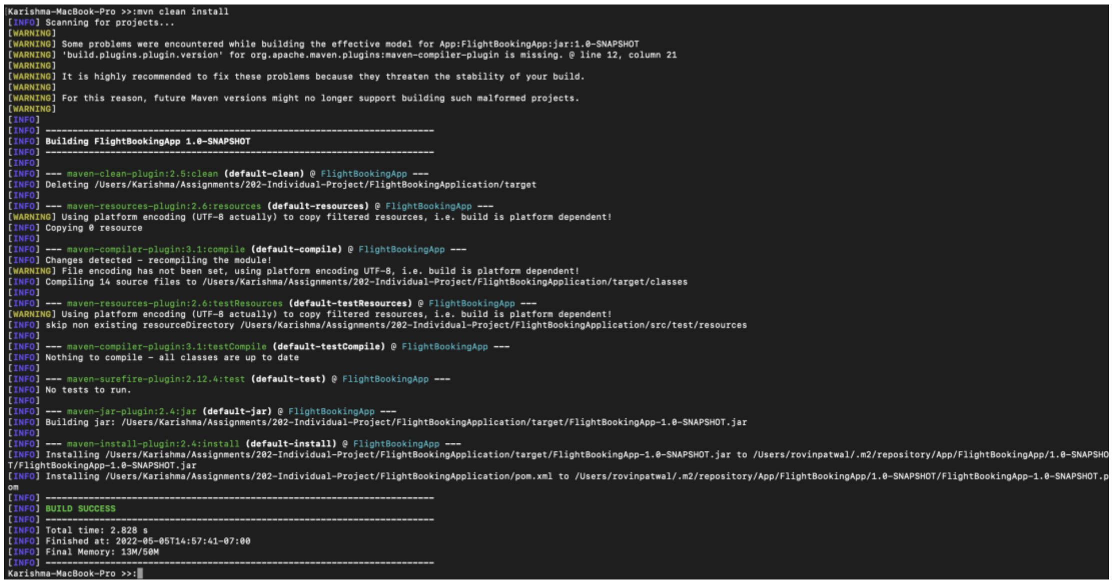
1. Detailed instructions of building the project and steps to execute the same.

1. Clone the code from github repo link.
2. Once the code is cloned in your local system, Open the command prompt and navigate to the directory where the pom file is located.
3. Run the below commands in the given sequence:
    1. mvn compile



```
Karishma-MacBook-Pro >>:cd Karishma/Assignments/202-Individual-Project/
Karishma-MacBook-Pro >>:cd FlightBookingApplication/
Karishma-MacBook-Pro >>:mvn compile
[INFO] Scanning for projects...
[WARNING]
[WARNING] Some problems were encountered while building the effective model for App:FlightBookingApp:jar:1.0-SNAPSHOT
[WARNING] 'build.plugins.plugin.version' for org.apache.maven.plugins:maven-compiler-plugin is missing. @ line 12, column 21
[WARNING]
[WARNING] It is highly recommended to fix these problems because they threaten the stability of your build.
[WARNING]
[WARNING] For this reason, future Maven versions might no longer support building such malformed projects.
[WARNING]
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building FlightBookingApp 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ FlightBookingApp ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ FlightBookingApp ---
[INFO] Nothing to compile - all classes are up to date
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.150 s
[INFO] Finished at: 2022-05-05T14:55:52-07:00
[INFO] Final Memory: 8M/30M
[INFO] ------------------------------------------------------------------------
Karishma-MacBook-Pro >>:
```

    2. mvn clean install



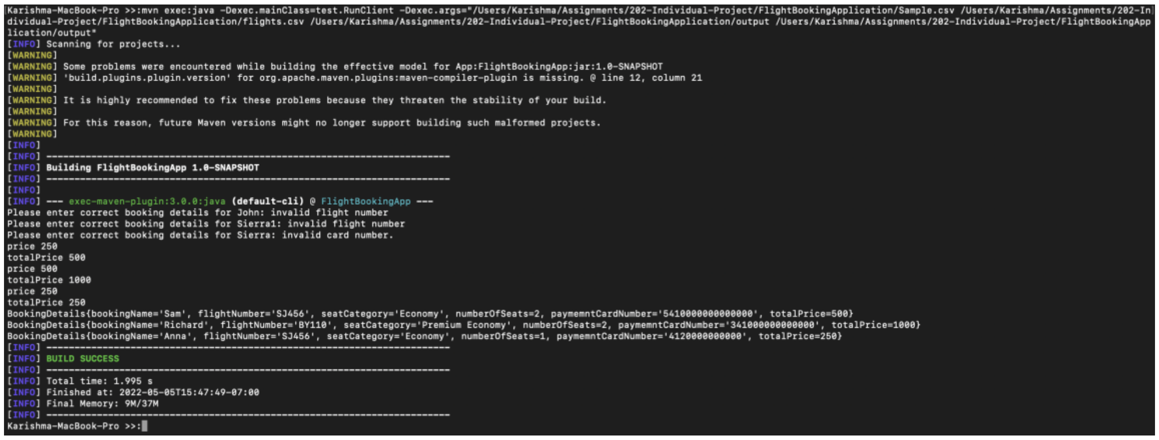
```
Karishma-MacBook-Pro >>:mvn clean install
[INFO] Scanning for projects...
[WARNING]
[WARNING] Some problems were encountered while building the effective model for App:FlightBookingApp:jar:1.0-SNAPSHOT
[WARNING] 'build.plugins.plugin.version' for org.apache.maven.plugins:maven-compiler-plugin is missing. @ line 12, column 21
[WARNING]
[WARNING] It is highly recommended to fix these problems because they threaten the stability of your build.
[WARNING]
[WARNING] For this reason, future Maven versions might no longer support building such malformed projects.
[WARNING]
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building FlightBookingApp 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ FlightBookingApp ---
[INFO] Deleting /Users/Karishma/Assignments/202-Individual-Project/FlightBookingApplication/target
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ FlightBookingApp ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ FlightBookingApp ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!
[INFO] Compiling 14 source files to /Users/Karishma/Assignments/202-Individual-Project/FlightBookingApplication/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ FlightBookingApp ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /Users/Karishma/Assignments/202-Individual-Project/FlightBookingApplication/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ FlightBookingApp ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ FlightBookingApp ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ FlightBookingApp ---
[INFO] Building jar: /Users/Karishma/Assignments/202-Individual-Project/FlightBookingApplication/target/FlightBookingApp-1.0-SNAPSHOT.jar
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ FlightBookingApp ---
[INFO] Installing /Users/Karishma/Assignments/202-Individual-Project/FlightBookingApplication/target/FlightBookingApp-1.0-SNAPSHOT.jar to /Users/rovinpatwal/.m2/repository/App/FlightBookingApp/1.0-SNAPSHO
T/FlightBookingApp-1.0-SNAPSHOT.jar
[INFO] Installing /Users/Karishma/Assignments/202-Individual-Project/FlightBookingApplication/pom.xml to /Users/rovinpatwal/.m2/repository/App/FlightBookingApp/1.0-SNAPSHOT/FlightBookingApp-1.0-SNAPSHOT.p
om
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.828 s
[INFO] Finished at: 2022-05-05T14:57:41-07:00
[INFO] Final Memory: 13M/50M
[INFO] ------------------------------------------------------------------------
Karishma-MacBook-Pro >>:
```

Student Name: Karishma Kuria   Student Id: 015947191

3. Execute the below maven command with arguments which are the paths to the below files:

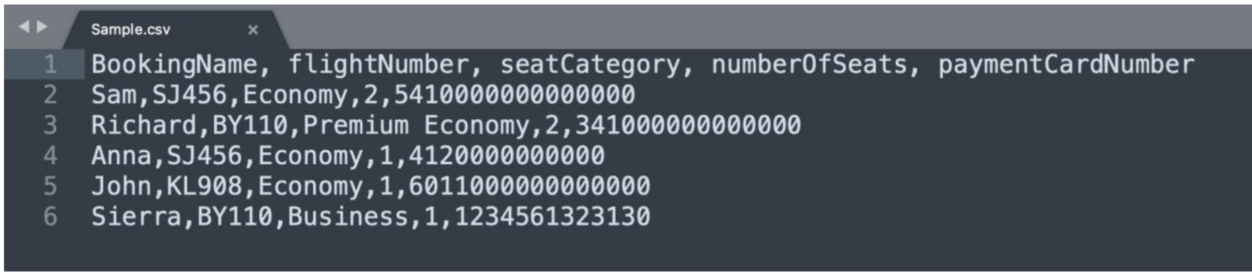
   mvn exec:java -Dexec.mainClass=test.RunClient -Dexec.args="<arg1> <arg2 > <arg3> <arg4>"

   - arg1 – path to the input file (Sample.csv)
   - arg2 – path to flight details to populate DB (flights.csv)
   - arg3 – path to Output.csv (Output file)
   - arg4 – path to Output.txt (Error file)



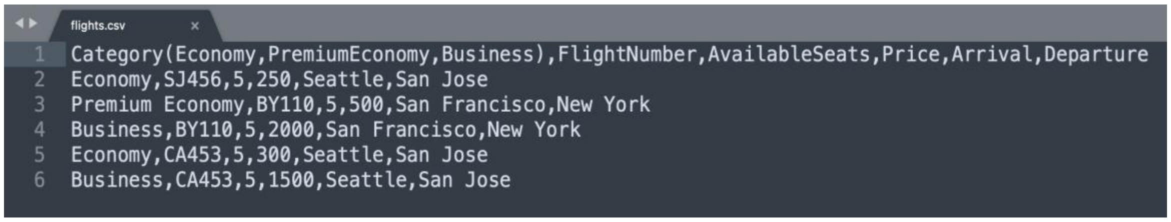
```
Karishma-MacBook-Pro >>:mvn exec:java -Dexec.mainClass=test.RunClient -Dexec.args="/Users/Karishma/Assignments/202-Individual-Project/FlightBookingApplication/Sample.csv /Users/Karishma/Assignments/202-In
dividual-Project/FlightBookingApplication/flights.csv /Users/Karishma/Assignments/202-Individual-Project/FlightBookingApplication/output /Users/Karishma/Assignments/202-Individual-Project/FlightBookingApp
lication/output"
[INFO] Scanning for projects...
[WARNING]
[WARNING] Some problems were encountered while building the effective model for App:FlightBookingApp:jar:1.0-SNAPSHOT
[WARNING] 'build.plugins.plugin.version' for org.apache.maven.plugins:maven-compiler-plugin is missing. @ line 12, column 21
[WARNING]
[WARNING] It is highly recommended to fix these problems because they threaten the stability of your build.
[WARNING]
[WARNING] For this reason, future Maven versions might no longer support building such malformed projects.
[WARNING]
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building FlightBookingApp 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- exec-maven-plugin:3.0.0:java (default-cli) @ FlightBookingApp ---
Please enter correct booking details for John: invalid flight number
Please enter correct booking details for Sierra1: invalid flight number
Please enter correct booking details for Sierra: invalid card number.
price 250
totalPrice 500
price 500
totalPrice 1000
price 250
totalPrice 250
BookingDetails{bookingName='Sam', flightNumber='SJ456', seatCategory='Economy', numberOfSeats=2, paymemntCardNumber='5410000000000000', totalPrice=500}
BookingDetails{bookingName='Richard', flightNumber='BY110', seatCategory='Premium Economy', numberOfSeats=2, paymemntCardNumber='341000000000000', totalPrice=1000}
BookingDetails{bookingName='Anna', flightNumber='SJ456', seatCategory='Economy', numberOfSeats=1, paymemntCardNumber='4120000000000', totalPrice=250}
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.995 s
[INFO] Finished at: 2022-05-05T15:47:49-07:00
[INFO] Final Memory: 9M/37M
[INFO] ------------------------------------------------------------------------
Karishma-MacBook-Pro >>:
```

Sample Input file



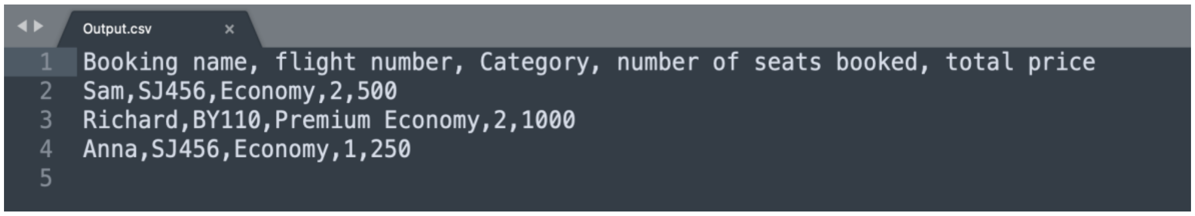
```
BookingName, flightNumber, seatCategory, numberOfSeats, paymentCardNumber
Sam,SJ456,Economy,2,5410000000000000
Richard,BY110,Premium Economy,2,341000000000000
Anna,SJ456,Economy,1,4120000000000
John,KL908,Economy,1,6011000000000000
Sierra,BY110,Business,1,1234561323130
```

Flight.csv



```
Category(Economy,PremiumEconomy,Business),FlightNumber,AvailableSeats,Price,Arrival,Departure
Economy,SJ456,5,250,Seattle,San Jose
Premium Economy,BY110,5,500,San Francisco,New York
Business,BY110,5,2000,San Francisco,New York
Economy,CA453,5,300,Seattle,San Jose
Business,CA453,5,1500,Seattle,San Jose
```

Output.csv

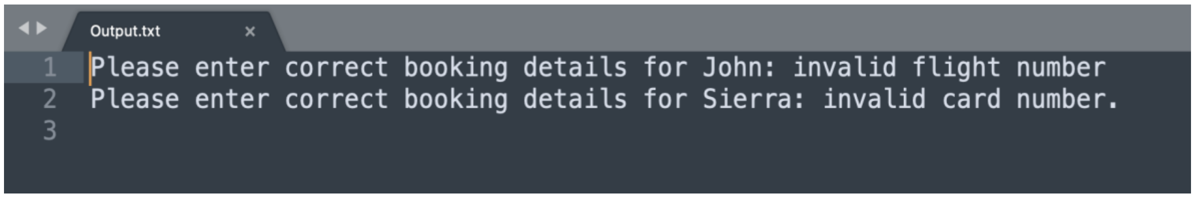


Output.csv

```
Booking name, flight number, Category, number of seats booked, total price
Sam,SJ456,Economy,2,500
Richard,BY110,Premium Economy,2,1000
Anna,SJ456,Economy,1,250

```

Output.txt



Output.txt

```
Please enter correct booking details for John: invalid flight number
Please enter correct booking details for Sierra: invalid card number.

```

2. Answer to the below question:

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

The primary problem is to build a flight booking application using at least 3 design patterns. This includes using a static database which can be implemented using any of the java structures such as HashMap. Static database means if the program is re-run the old changes made will not be there.
For the Static database, I have used HashMap. For making connection with the database I have implemented Singleton design pattern. The database contains the following tables and fields:

Table 1: Flights

- **Category** (Economy, Premium Economy, Business)
- **Flight number**
- **The available Seats of each category**
- **Price of each seat**
- **Arrival City**
- **Departure City**

The database is filled by reading the data from Flights.csv provided as part of the problem. We also have an input file which will be read and contains booking details, which includes booking name, flight number, seat category, number of seats and the payment card number. These details will be verified with the existing database and if the details are correct the booking is further processed and completed. Firstly, its verified that if the requested flight exists in the database. This is done by creating a connection with DB and verifying the existence of flight. Further if the flight exists then the number of seats requested is validated. If the requested flight and seats exists than total price (numberofseats * price) of the booking is calculated.

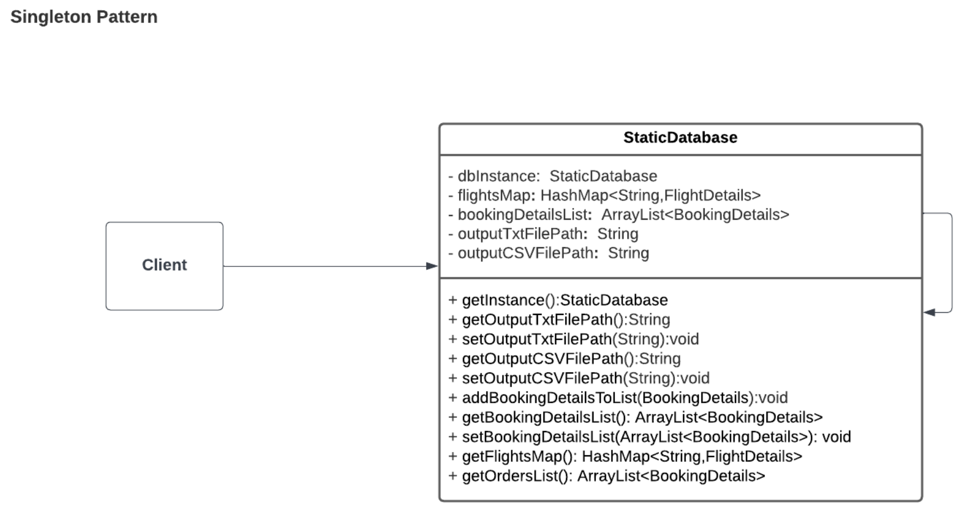
2. **Describe what are the secondary problems you try to solve**
   If the booking is valid one and the total cost is calculated then the secondary problem is to validate the card used for payment. For doing the validation on card below rules are used:
   - Visa card: has length either 13 or 16. It begins with a 4
   - Mastercard: has length 16. Begins with 5 and the 2nd digit begins from 1 to 5 inclusive.
   - Discover: length 16, and the first 4 digits begins from 6011
   - Amex: has length 15 and starts with 3. 2nd digit must be 4 or 7
   - Any card greater than 19 or not satisfying above conditions is considered invalid.

   Once the card is declared as a valid one, the available seats are modified for the specific flight number of that requested category. This refers to the confirmation of the booking and an Output.csv is generated listing booking name, flight number, Category, number of seats booked, total price.

   If there is any incorrect request at any step in the above process than an output txt file is generated with the message describing the issue in the request.

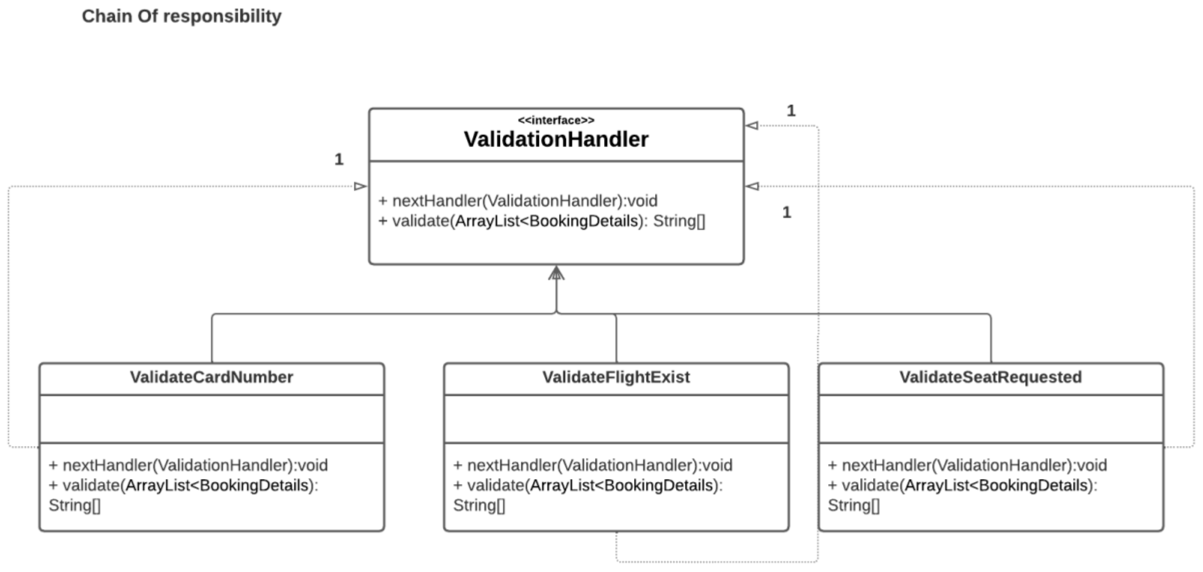
   I have used Factory pattern to handle the output and error file creation. I have implemented Change of responsibility design pattern to do card number validation.

3. **Describe what design pattern(s) you use how (use plain text and diagrams)**
   1. **Singleton:** Since it's a creational design pattern which makes sure that only one instance of the class is used which has global access. In our case, there is only 1 database which holds the flight details and is static so singleton pattern fits perfectly here. In the entire process wherever we need the database instance *getInstance()* will be called which will have a global access and can be reused.


Singleton Pattern
Client
StaticDatabase
- dbInstance: StaticDatabase
- flightsMap: HashMap<String,FlightDetails>
- bookingDetailsList: ArrayList<BookingDetails>
- outputTxtFilePath: String
- outputCSVFilePath: String
+ getInstance():StaticDatabase
+ getOutputTxtFilePath():String
+ setOutputTxtFilePath(String):void
+ getOutputCSVFilePath():String
+ setOutputCSVFilePath(String):void
+ addBookingDetailsToList(BookingDetails):void
+ getBookingDetailsList(): ArrayList<BookingDetails>
+ setBookingDetailsList(ArrayList<BookingDetails>): void
+ getFlightsMap(): HashMap<String,FlightDetails>
+ getOrdersList(): ArrayList<BookingDetails>

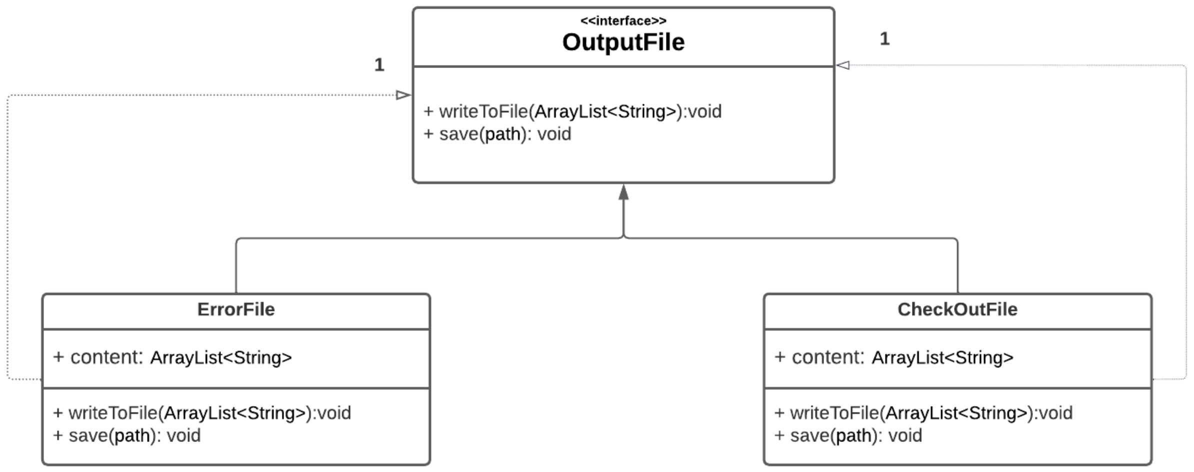
2. **Chain of Responsibility**: This design pattern is a behavioral design pattern which uses multiple handlers to process the request. The request is transferred between the handlers in the chain based on the implementations inside the handlers.
   I have used this design pattern to do the following:
   - Validate if the requested flight exists.
   - Validate if requested seat exists.
   - Validate if the card number is valid based on the rules provided in the problem statement.

The following files are used as part of the implementation:


Chain Of responsibility
<<interface>>
ValidationHandler
+ nextHandler(ValidationHandler):void
+ validate(ArrayList<BookingDetails>): String[]
1
1
1
ValidateCardNumber
+ nextHandler(ValidationHandler):void
+ validate(ArrayList<BookingDetails>): String[]
ValidateFlightExist
+ nextHandler(ValidationHandler):void
+ validate(ArrayList<BookingDetails>): String[]
ValidateSeatRequested
+ nextHandler(ValidationHandler):void
+ validate(ArrayList<BookingDetails>): String[]

3. **Factory Pattern:** This is a creational design pattern which is used to create objects in superclass providing subclasses the ability to alter the type of object to be created. Since in this case there are 2 types of files are created as the output which is the output csv containing the confirmation details once the booking is done and the other is the output.txt which contains the errors encountered when making an invalid request. So, I have implemented factory pattern using *OutputFile* interface which contains methods to write to a file and save the file at a given location. This interface is implemented by *CheckOutFile* and *ErrorFile*.

Factory Pattern


<<interface>>
OutputFile
+ writeToFile(ArrayList<String>):void
+ save(path): void
1
1
ErrorFile
+ content: ArrayList<String>
+ writeToFile(ArrayList<String>):void
+ save(path): void
CheckOutFile
+ content: ArrayList<String>
+ writeToFile(ArrayList<String>):void
+ save(path): void

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

Singleton: As described in the problem statement that we only have 1 database which is static database which means every time the program is rerun it will not persist the old data. So, it made sense to use singleton pattern here since only one instance of the database will be reused in the entire process because only one database is to be modified and it should not hold the old data.

We don't have to create the instance multiple times, this reduces the amount of runtime memory space used.

The instance used will have a global access. This reduces the number of global variables used inside the code.

Chain of responsibility:

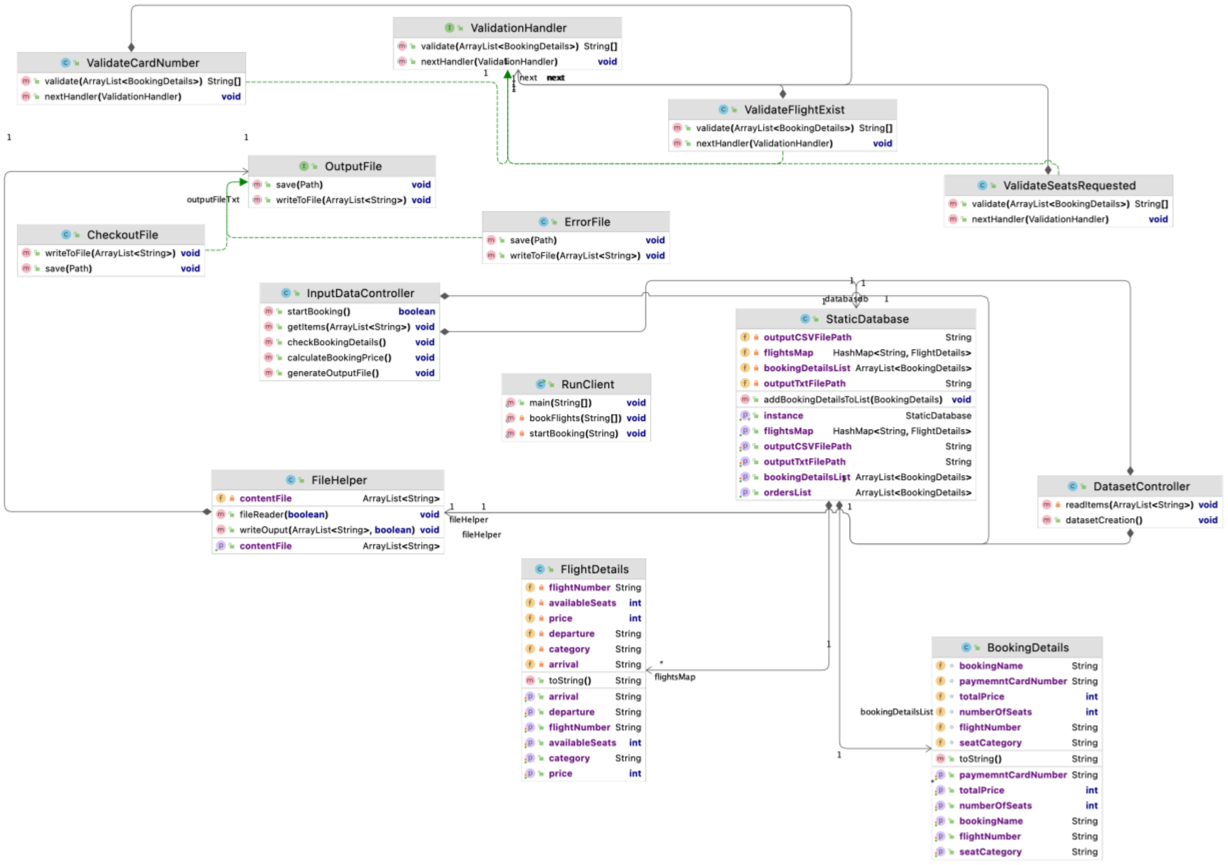
There is a very loose coupling in the code, for instance the developer wants to add or update a rule in card validation class then there is no impact on the other classes of the chain.

He/she will just have to update the respective class. The end user and the receiver have no knowledge of each other.
Since the entire process of booking a flight goes through multiple validation steps it becomes easy to implement it with this design pattern.
In this the validations are performed in a given sequence and error is reported if encountered at any step.

Factory: Since the requirement was to generate 2 different files for output. Factory design patterns fits this requirement. In this the handler decides if the output should be a checkoutfile or a errorfile.
It's always more flexible to create object inside the main method rather than creating an object directly. The *OutputFile* interface has provided privilege to the classes implementing it to decide on the type of output file to be generated.

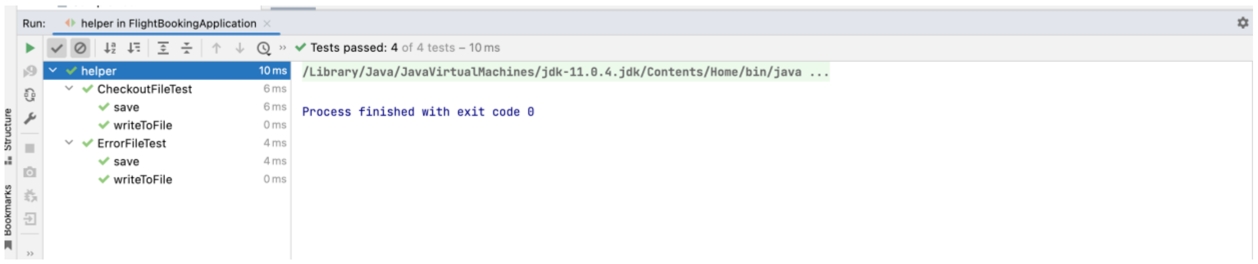
5. Class diagram

Student Name: Karishma Kuria Student Id: 015947191



ValidationHandler
validate(ArrayList<BookingDetails>) String[]
nextHandler(ValidationHandler) void
ValidateCardNumber
validate(ArrayList<BookingDetails>) String[]
nextHandler(ValidationHandler) void
ValidateFlightExist
validate(ArrayList<BookingDetails>) String[]
nextHandler(ValidationHandler) void
ValidateSeatsRequested
validate(ArrayList<BookingDetails>) String[]
nextHandler(ValidationHandler) void
next
OutputFile
save(Path) void
writeToFile(ArrayList<String>) void
outputFileTxt
CheckoutFile
writeToFile(ArrayList<String>) void
save(Path) void
ErrorFile
save(Path) void
writeToFile(ArrayList<String>) void
InputDataController
startBooking() boolean
getItems(ArrayList<String>) void
checkBookingDetails() void
calculateBookingPrice() void
generateOutputFile() void
RunClient
main(String[]) void
bookFlights(String[]) void
startBooking(String) void
StaticDatabase
outputCSVFilePath String
flightsMap HashMap<String, FlightDetails>
bookingDetailsList ArrayList<BookingDetails>
outputTxtFilePath String
addBookingDetailsToList(BookingDetails) void
instance StaticDatabase
flightsMap HashMap<String, FlightDetails>
outputCSVFilePath String
outputTxtFilePath String
bookingDetailsList ArrayList<BookingDetails>
ordersList ArrayList<BookingDetails>
databasedb
DatasetController
readItems(ArrayList<String>) void
datasetCreation() void
FileHelper
contentFile ArrayList<String>
fileReader(boolean) void
writeOuput(ArrayList<String>, boolean) void
contentFile ArrayList<String>
fileHelper
fileHelper
FlightDetails
flightNumber String
availableSeats int
price int
departure String
category String
arrival String
toString() String
arrival String
departure String
flightNumber String
availableSeats int
category String
price int
flightsMap
BookingDetails
bookingName String
paymemntCardNumber String
totalPrice int
numberOfSeats int
flightNumber String
seatCategory String
toString() String
paymemntCardNumber String
totalPrice int
numberOfSeats int
bookingName String
flightNumber String
seatCategory String
bookingDetailsList
1
*

3. Junit Test case:
Factory Design Pattern test execution results:


Run: helper in FlightBookingApplication

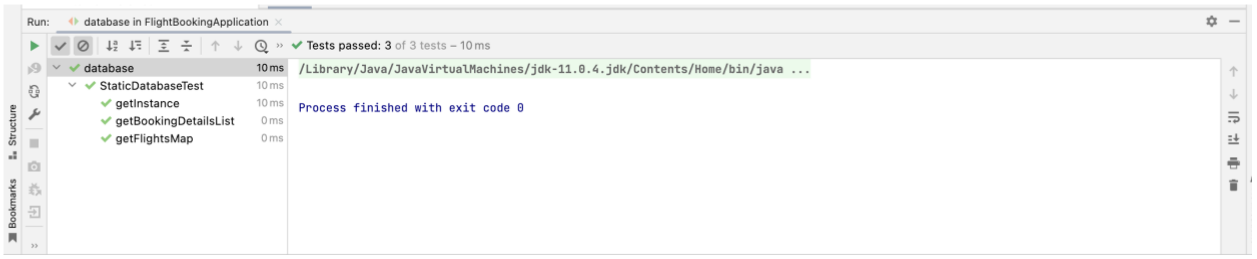
Tests passed: 4 of 4 tests – 10 ms

```
helper                10 ms
  CheckoutFileTest     6 ms
    save               6 ms
    writeToFile        0 ms
  ErrorFileTest        4 ms
    save               4 ms
    writeToFile        0 ms
```

```
/Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

Singleton Design Pattern test execution results:



Run: database in FlightBookingApplication

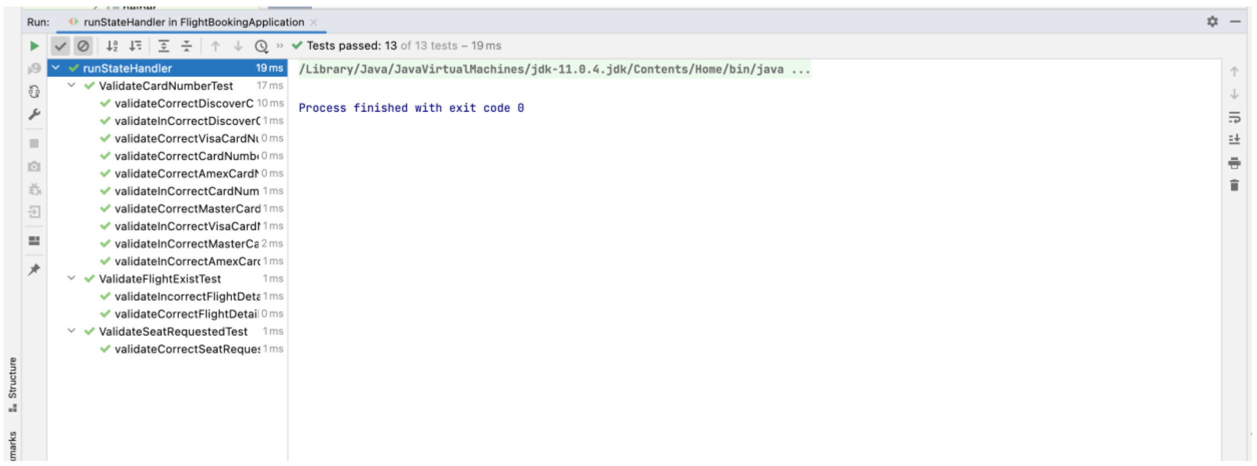
Tests passed: 3 of 3 tests – 10 ms

```
database                     10 ms
  StaticDatabaseTest         10 ms
    getInstance              10 ms
    getBookingDetailsList     0 ms
    getFlightsMap             0 ms
```

```
/Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

Chain of Responsibility Design Pattern test execution results:



Run: runStateHandler in FlightBookingApplication

Tests passed: 13 of 13 tests – 19 ms

```
runStateHandler                    19 ms
  ValidateCardNumberTest           17 ms
    validateCorrectDiscoverC       10 ms
    validateInCorrectDiscover(      1 ms
    validateCorrectVisaCardNu       0 ms
    validateCorrectCardNumb         0 ms
    validateCorrectAmexCardN        0 ms
    validateInCorrectCardNum        1 ms
    validateCorrectMasterCard       1 ms
    validateInCorrectVisaCardI      1 ms
    validateInCorrectMasterCa       2 ms
    validateInCorrectAmexCar        1 ms
  ValidateFlightExistTest           1 ms
    validateIncorrectFlightDeta     1 ms
    validateCorrectFlightDetai      0 ms
  ValidateSeatRequestedTest         1 ms
    validateCorrectSeatReques       1 ms
```

```
/Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```