Individual Project CMPE 202
Flight Application

Name - Harika Nalam
SJSU ID - 015939963

Steps to Run the application

1. Download the Project by clicking Code-> Download ZIP on the GitHub repository or clone the project
2. Once the project is downloaded unzip the file.
3. Open the Command line and got to the project folder location
4. Cd individual-project-flightApp
5. Run “mvn compile”
6. Run “mvn clean install”
7. Place input csv flight and customer details under the project folder and execute the project.
8. To run project use cmd - *mvn exec:java -Dexec.mainClass=test.RunClient -Dexec.args="sample.csv flights.csv Output.csv Output.txt"*
    a. argos=”<Customer Input Filepath> <Flight Input Filepath> <Path to .csv flight Booking details> <path to Error log .txt file>”
9. To run test cases - “mvn test”

Problem Statement:

The objective of this project is to build a JAVA flight application. The application books the flight tickets for the customer. The flight information is maintained in an internal static database. I have used hashmap to store the flight information which is read from a .csv file.
Customer requests to book a flight ticket by providing the flight number, number of seats and category of seats for booking, and credit card number to complete the booking. After successful booking Itinerary for the customer is saved to a .csv file.

Flight DB attributes

a. Available Seat details (SeatType, Number of seats, SeatPrice)
b. Flight Number (this is the unique identifier)
c. Arrival
d. Departure

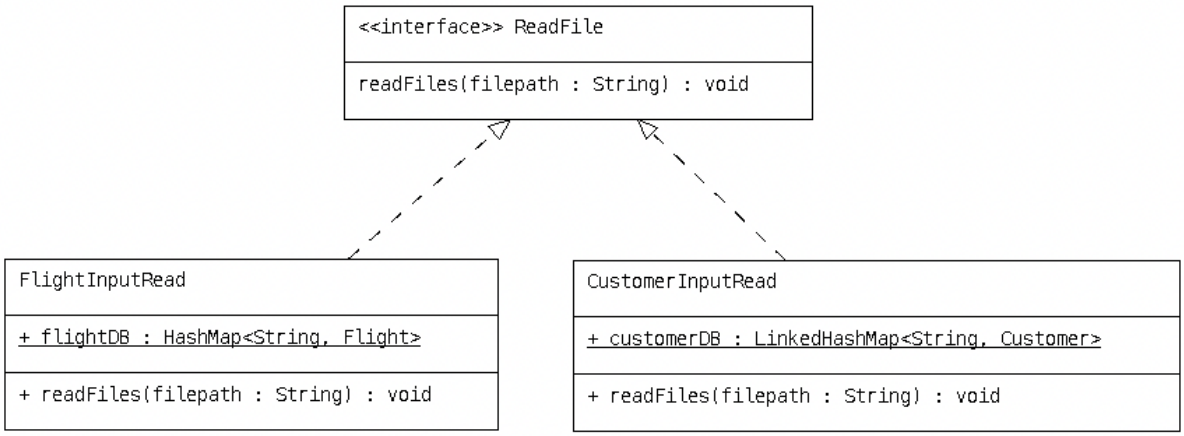
Customer DB attributes

a. Customer name

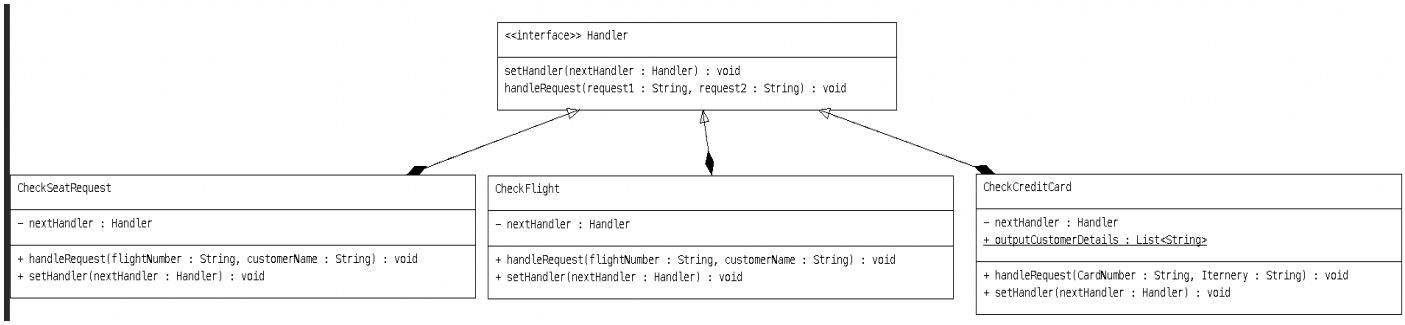
b. Flight number
c. Number of seats for booking
d. Seat Type
e. Credit card number

Design Patterns

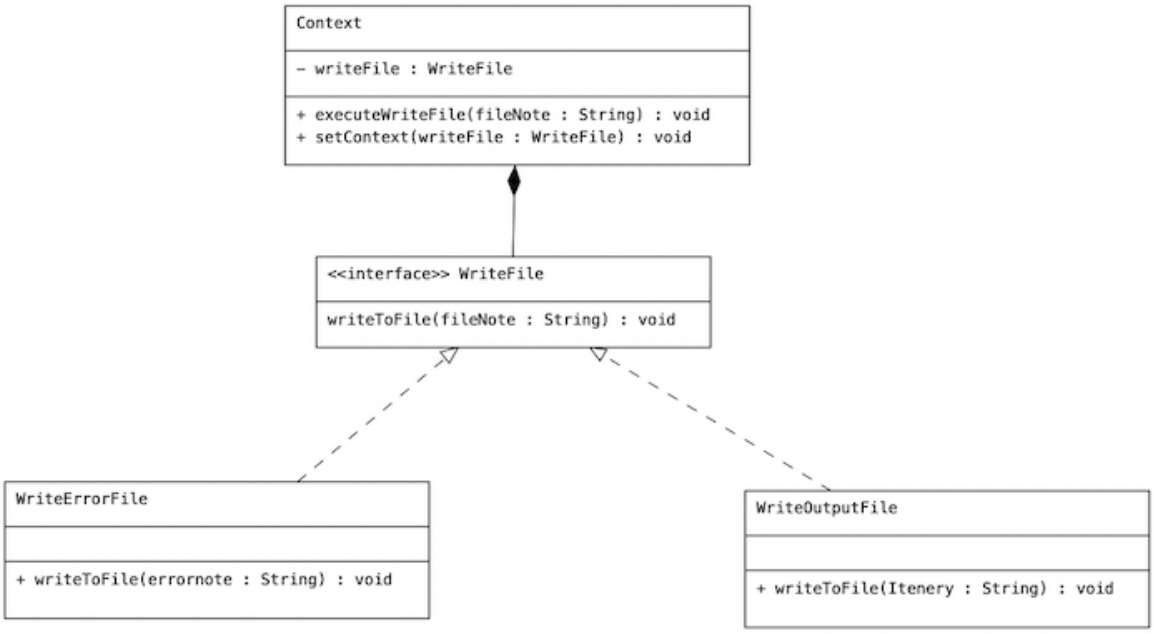
1. Factory Method Pattern
    a. This pattern is used to identify and read the dataset file or read the input file. CustomerInputRead class is used to read customer details files, return customer hashmap, and the FlightInputRead is used to read flight details files and store the flight details in a hashmap and return flight DB.



2. Chain of Responsibilities
    a. This pattern is used to validate a chain of operation in order before booking flight tickets. The chain operations include checkFlight , checkSeatRequest, checkCreditCard. First, the flight number customer requested for booking is validated(if the flight is present in the “flightDB”) and check if there are sufficient tickets present for the requested seat category, and last validate the credit card number.

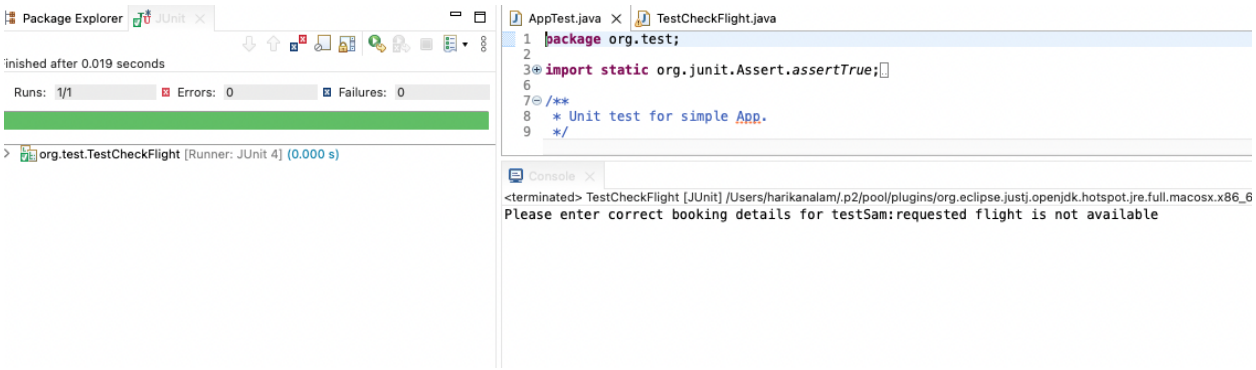


3. Strategy
    a. A strategy pattern is used to write the output to the file. In this application if there is a successful booking then the customer itinerary details are added to ".csv" file and for an unsuccessful booking an error note is added to ".txt" file with the error message". This pattern is used to dynamically change the file to which output is added. Context object is instantiated and based on the booking outcome the method "writeToFIle" is called from "writeOutputFile" and "writeErrorFile"

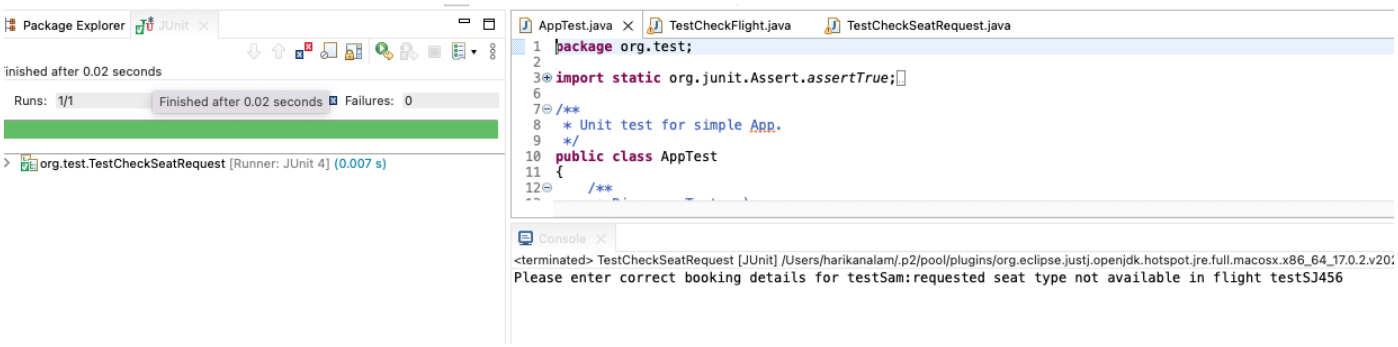


JUnit Tescase Screenshots.

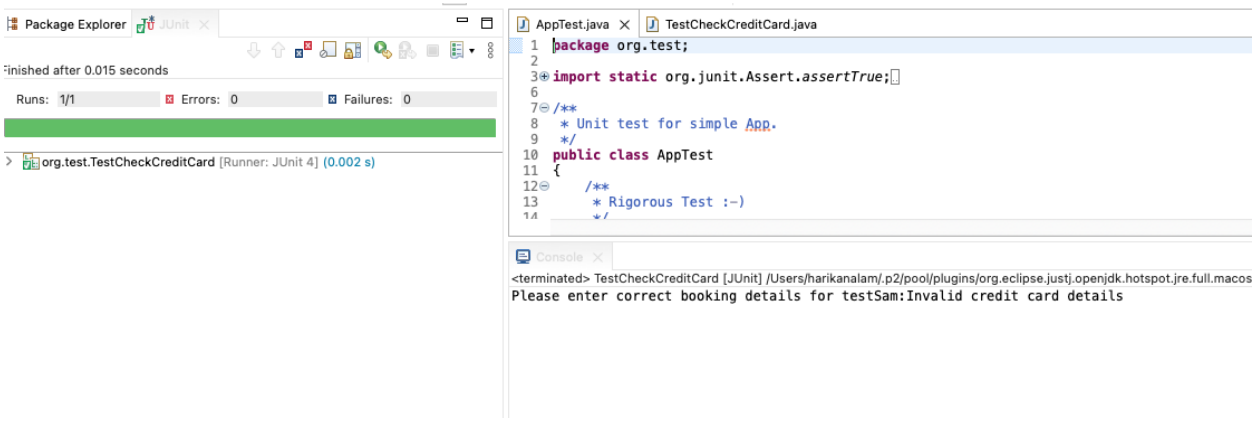
1. Check if the flight is present in DB requested for booking by the customer



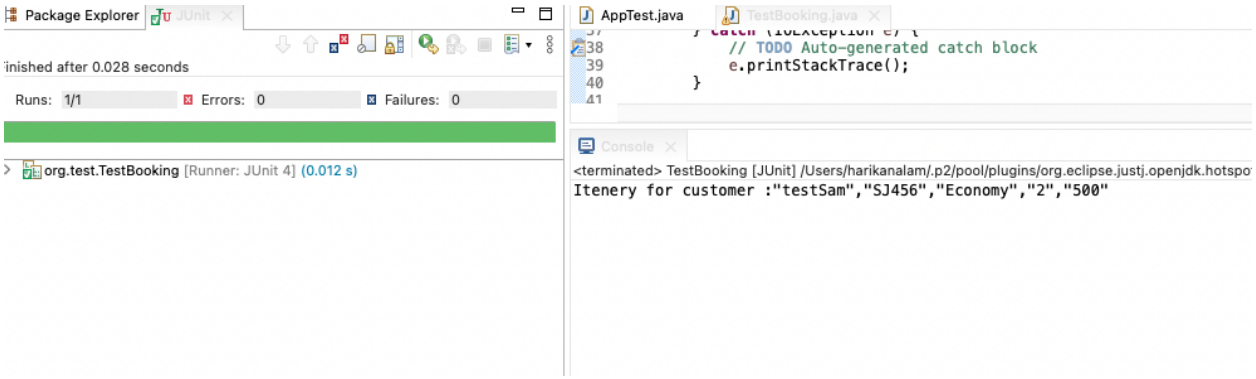
2. Check if the seat type requested by the customer is available



3. Check if credit card details are valid



4. Check if all the given details by the customer are valid then give out the Itinerary to the customer with all the required details



Class Diagram:

Saved as FlightApplicationUMLDiagram.png in GitHub repository.