

Individual Project

<https://github.com/gopinathsjsu/individual-project-zisyang>

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 HashMap'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 has to be maintained. The data will contain the following tables and fields:

1: Table Flights

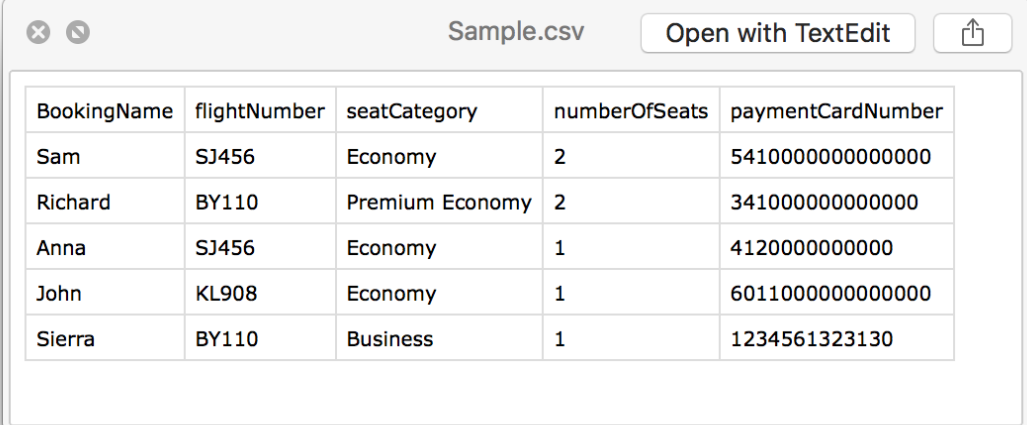
- **Category** (Economy, Premium Economy, Business)
- **Flight number**
- The available **Seats** of each category
- **Price** of each seat
- **Arrival City**
- **Departure City**

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

3. Input file should be processed as follows:

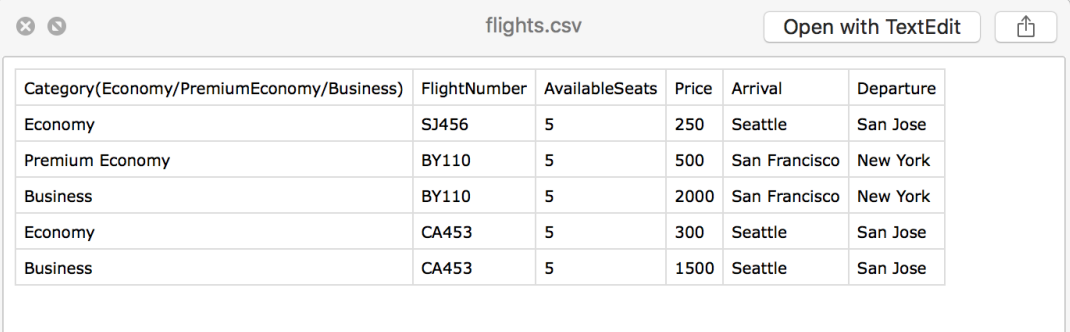
- Validate if the requested flight exists.
- If the flight exists, validate the number of seats requested for the category.
- After this validation, if the booking is valid, calculate the total price (NoOfSeats * price)
- Take the card number of the user and validate it using the given rules:
 - Visa 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 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.
- Sample input and output files:
- [Sample.csv](#)



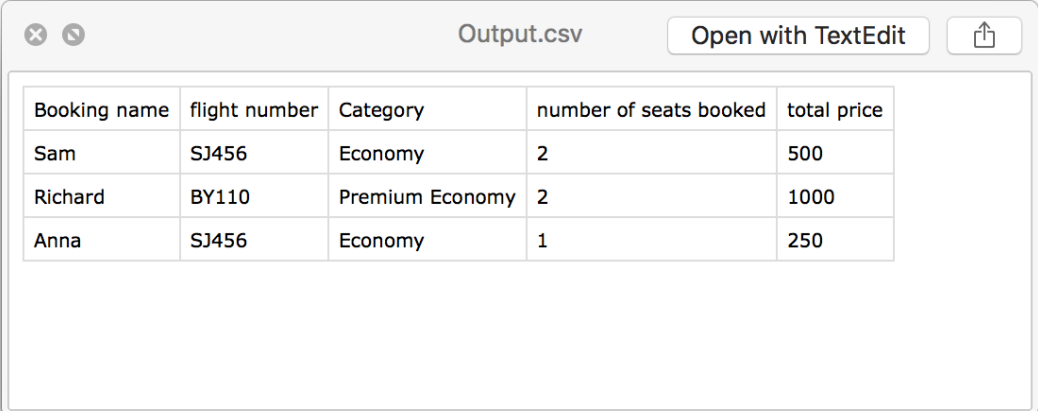
BookingName	flightNumber	seatCategory	numberOfSeats	paymentCardNumber
Sam	SJ456	Economy	2	5410000000000000
Richard	BY110	Premium Economy	2	3410000000000000
Anna	SJ456	Economy	1	4120000000000000
John	KL908	Economy	1	6011000000000000
Sierra	BY110	Business	1	1234561323130

-
- [flights.csv](#)



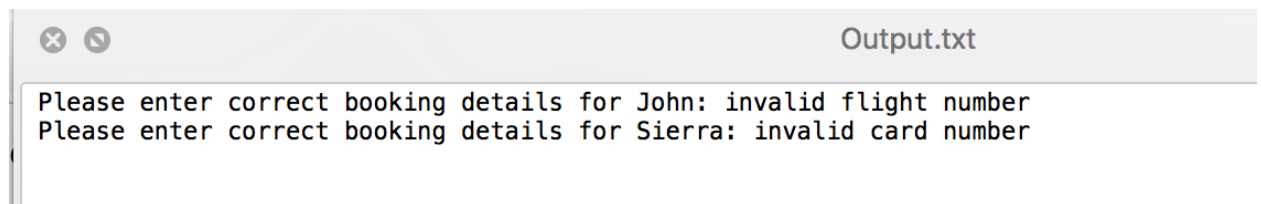
Category(Economy/PremiumEconomy/Business)	FlightNumber	AvailableSeats	Price	Arrival	Departure
Economy	SJ456	5	250	Seattle	San Jose
Premium Economy	BY110	5	500	San Francisco	New York
Business	BY110	5	2000	San Francisco	New York
Economy	CA453	5	300	Seattle	San Jose
Business	CA453	5	1500	Seattle	San Jose

- [Output.csv](#)



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](#)



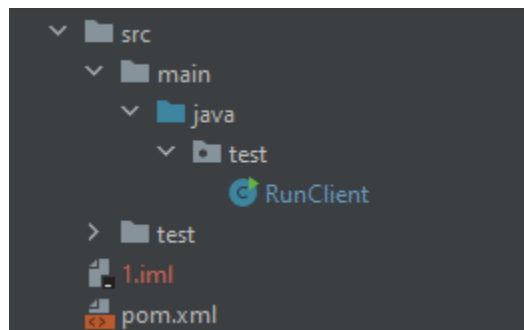
```

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

```

Project structure and build instructions to be followed:

- It should be a maven project with dependencies declared in the pom file.
- Folder structure would look like below after creating maven project.



- Main class should be named as RunClient placed under src/main/java/test/RunClient.
- All the Junit test cases should be placed under src/test.

Steps to execute:

- Open the command line where the project directory is located and execute below steps
 - mvn compile
 - mvn clean install
 - Execute the below maven command to execute with arguments (Path to where the input file is located and path to where output file should be located) passed via command line
 - mvn exec:java -Dexec.mainClass=test.RunClient -Dexec.args="<arg1> <arg2 > <arg3> <arg4>"
 - arg1 – path to the input data (Sample.csv)
 - arg2 – path to flight details to populate DB (flights.csv)
 - arg3 – path to Output.csv
 - arg4 – path to Output.txt

REPORT

The report will have three sections:

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

Answers to the following questions:

1. Describe what is the primary problem you try to solve
 2. Describe what are the secondary problems you try to solve
 3. Describe what design pattern(s) you use how (use plain text and diagrams)
 4. Describe the consequences of using this/these pattern(s).
 5. Class diagram.
3. Junit test cases should be written for all the design pattern classes/methods. Screenshots of test execution and result should be included.

Note: Please submit your report on canvas (doc or pdf) and commit your source code in Github. Please do not use any external dependencies and ensure your project executes using the above-mentioned steps of execution.