# PROJECT REPORT

## 1. Steps to Execute

Few steps to execute the project are as follows:

a. Navigate to the project directory
b. Execute following commands:
   i. ***mvn compile***
   ii. ***mvn clean install***
   iii. To run the project, enter path of the input files and output files - my input was stored in say: *"/Users/ashika/Downloads/FlightBookingApp/src/main/java/test/input/Sample.csv"*
   In the FlightBookingApp(project), in the root directory, there is an input folder where the input files can be stored
   Same for the output files as well:
   Example:
   *"/Users/ashika/Downloads/FlightBookingApp/src/main/java/test/output/Output.csv"*
   iv. ***mvn exec:java -Dexec.mainClass=test.RunClient -Dexec.args= "/Users/ashika/Downloads/FlightBookingApp/src/main/java/test/input/Sample.csv /Users/ashika/Downloads/FlightBookingApp/src/main/java/test/input/flights.csv /Users/ashika/Downloads/FlightBookingApp/src/main/java/test/output/Output.csv /Users/ashika/Downloads/FlightBookingApp/src/main/java/test/output/Output.txt"***
   Where: args[0] - Path to Bookings csv,
   args[1] - Path to Flights csv,
   args[2] - Path to Output csv and
   args[3] - Path to error txt files

## 2. Primary Problem

The primary problem I tried to solve is booking flights. There are various methods on how the data can be stored and accessed such as just creating a list of objects for both flights and booking data and then comparing each object in that booking to each object of flight and then validating the price, seats and category for each booking as per requirements. This is a tedious, time consuming and inefficient process.

We can also create a hashmap of flight to map each flight to its category but again it's an inefficient design as the mapping doesn't track the seats and price per seat. It has to be validated again or we need to create a hashmap of hashmap.

The most simple, efficient and valid design is to create a subclass for the flight with category, price and number of seats in each flight. This can be used to map flight to its

other features (category,seats and price per seat), when booking's flight number is validated against flight's flight number, it automatically retrieves all categories that are associated with the flight and when the category is validated, it just has to validate if within subclass there is a category that is same as that of booking and then use the details w.r.t to that category to get number of seats and price.

## 3. Secondary Problem

Secondary problem is validating all details and managing status. To validate a card number, first I need to validate if all of the booking details entered are correct. I have used enum to manage status if the flight number is valid, change status to validFlight, if category is validated, status changed to validCategory and so on. If all the conditions are met, the status is changed to success and output csv is populated, else if any of the conditions is not met, then entry into the error file with reason is added.

## 4. Design Pattern

Three design patterns I have used is:

1. Singleton

Singleton design pattern is used when the system has to ensure that only one instance of the class is created. I have stored flight information as static and created a single instance of the object which is accessed throughout the code.

```
public class readFlightDetails {
    // Singleton pattern - Only one instance of the class is created and used.
    4 usages
    public static ArrayList<flight> allFlights;
    1 usage
    public readFlightDetails(String file){...}
}
```

2. Chain of Responsibility

Chain of responsibility pattern is used when a set of objects handle the quest with a handler. On running the project, at first an object of flight is created, then an object of booking is created, both these objects are passed to the validation where the details are validated against each other and then entered into output or error files.

```java
// Chain of Responsibility
// Read booking data - Control to booking object
// Read flight data - Control to flight object
// Validate Booking - Control to Validate handler in RunClient
// Populate Output and error csv - Get output
public class RunClient {
    public static void main(String[] args) throws IOException {
        readBookingDetails rbd = new readBookingDetails(args[0]);
        readFlightDetails fbd = new readFlightDetails(args[1]);
        validateBooking.Validate(rbd,fbd,args[2],args[3]);
    }
}
```

3. Composite

Composite design pattern is used where hierarchical representation of objects is required. Class Flight has subclass categoryAndSeats.

```java
public class flight {
    2 usages
    public flight(String name, String category, Integer seats, Double price) {
        this.setFlightNumber(name);
        this.fullDetails = new ArrayList<>();
        this.setFullDetails(category, seats, price);
    }
    8 usages
    public enum Category {...}

    // Composite
    7 usages
    public class categoryAndSeats {
        1 usage
        public categoryAndSeats(Category type, Integer seats, Double price) {
            this.type = type;
            this.numSeats = seats;
            this.pricePerSeat = price;
        }
        3 usages
        public Category type;
        3 usages
        public Integer  numSeats;
        2 usages
        public Double  pricePerSeat;
    }
```

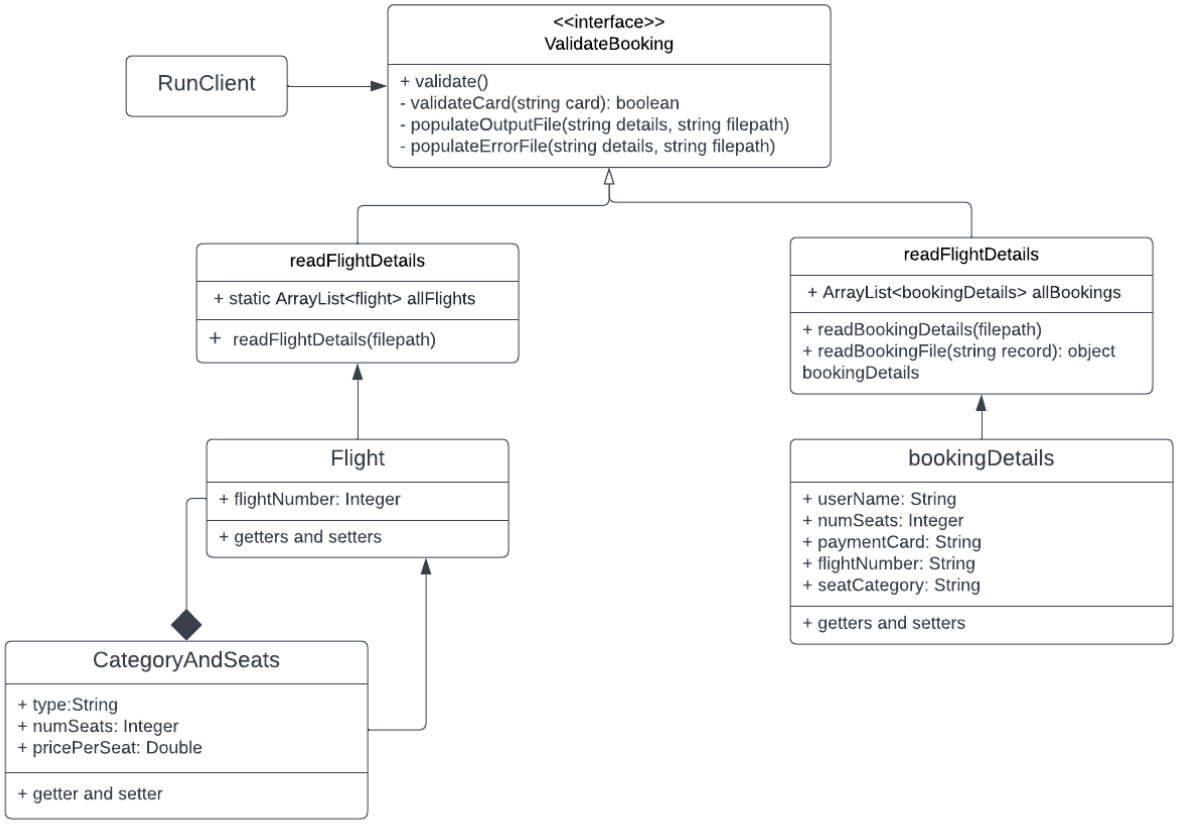
## 5. Consequences of using a design pattern

By using a singleton pattern, data is clean and only one instance of data is required which is created and used.

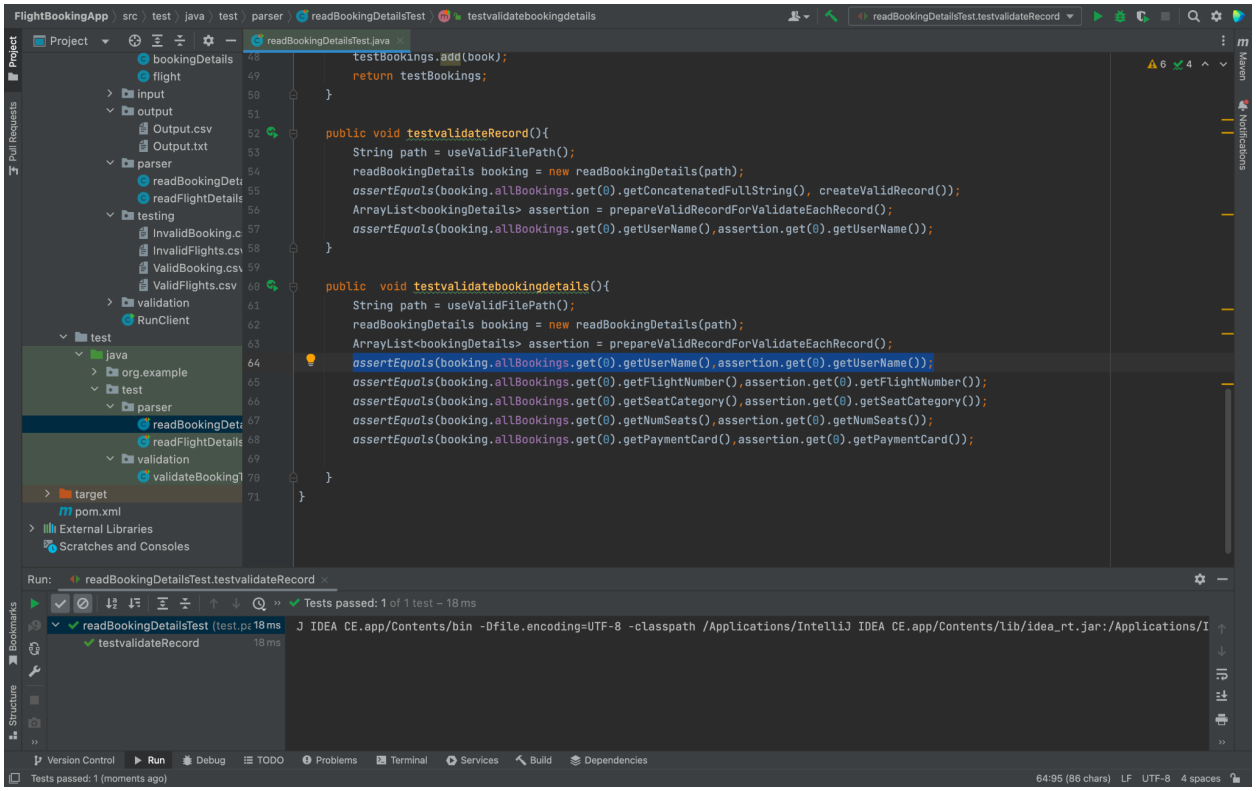
Using chain of responsibility we have a client called RunClient which has an interface to validate handler functions in the validation package where the function internally handles with 2 other objects of flight and booking.

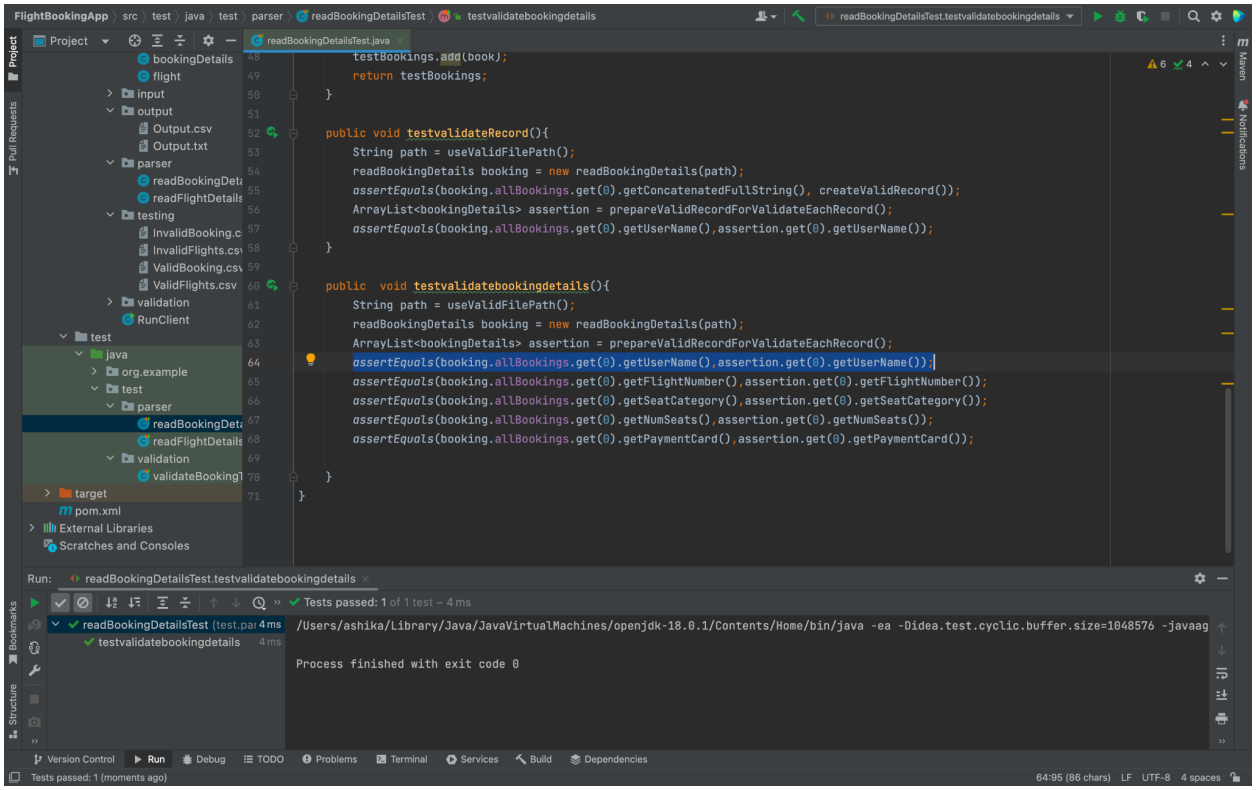
Using a composite design pattern we have created a scalable and efficient access control to the data within the flight. The class flight facilitates the subclass categoryAndSeats and the objects of subclass are nested within the objects of flight class.

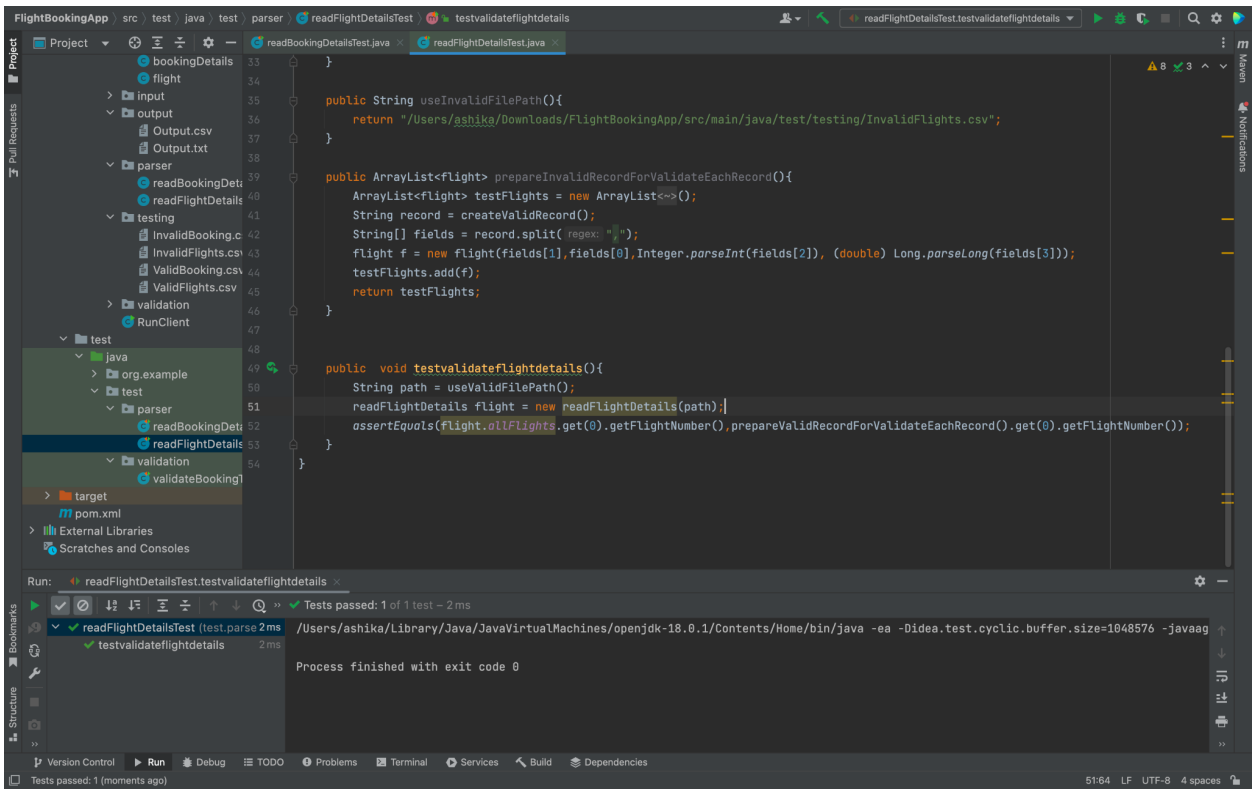
## 6. Class Diagram

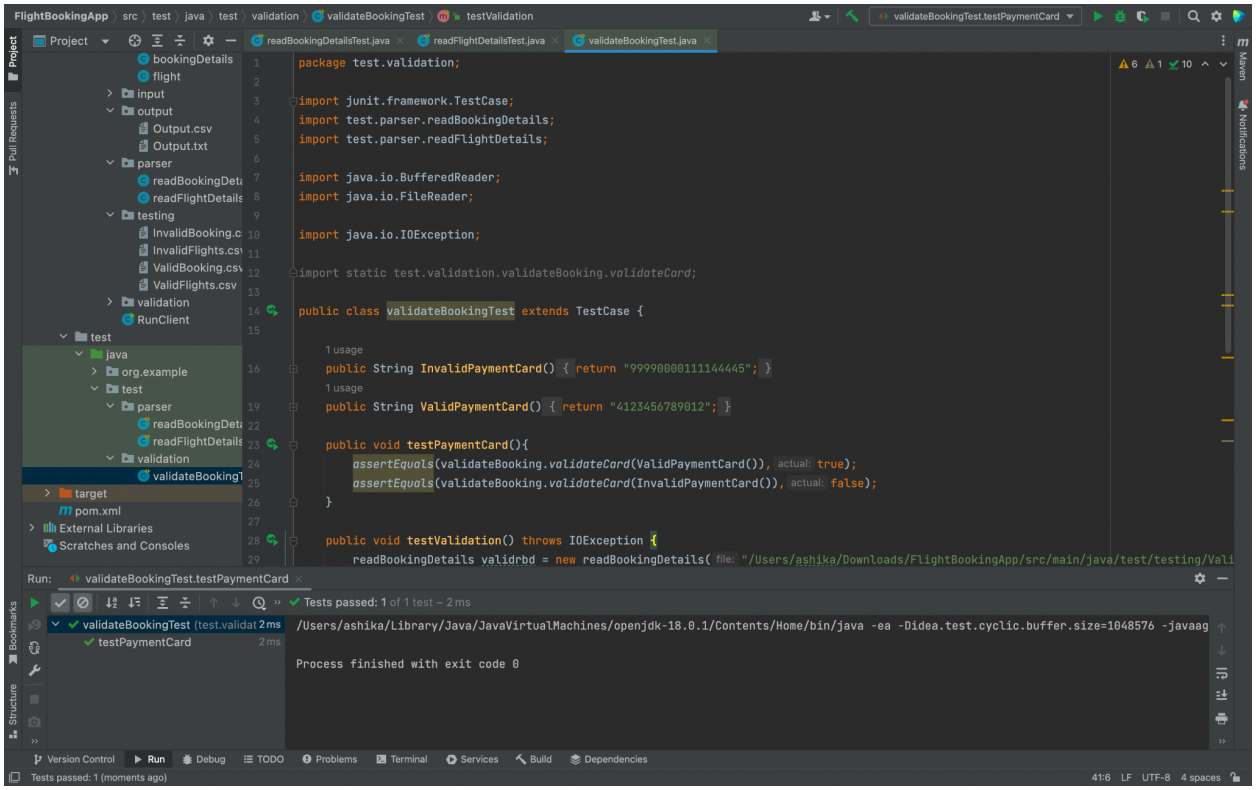


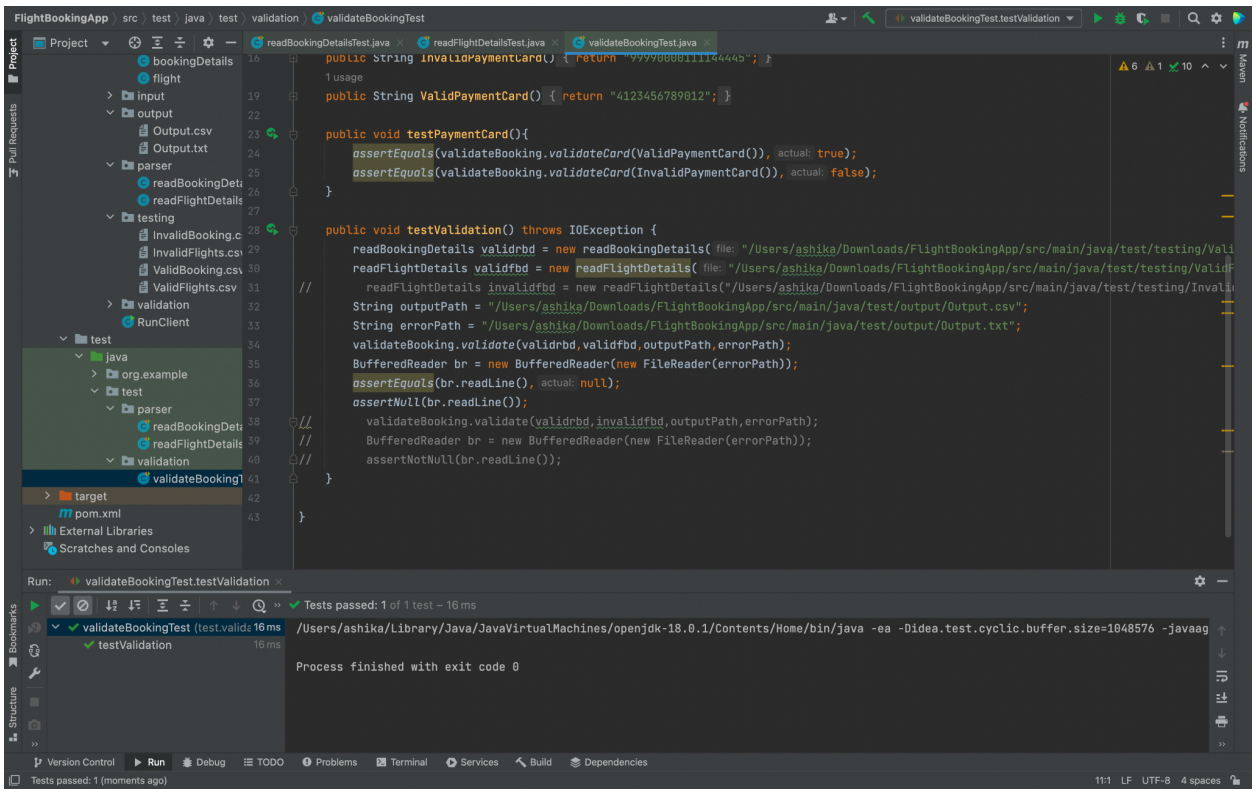
# 7. Junit tests




FlightBookingApp src test java test parser readBookingDetailsTest testvalidatebookingdetails
readBookingDetailsTest.testvalidatebookingdetails
Project
readBookingDetailsTest.java
bookingDetails
flight
input
output
Output.csv
Output.txt
parser
readBookingDeta
readFlightDetails
testing
InvalidBooking.c
InvalidFlights.csv
ValidBooking.csv
ValidFlights.csv
validation
RunClient
test
java
org.example
test
parser
readBookingDeta
readFlightDetails
validation
validateBooking1
target
pom.xml
External Libraries
Scratches and Consoles
testBookings.add(book);
return testBookings;
}
public void testvalidateRecord(){
String path = useValidFilePath();
readBookingDetails booking = new readBookingDetails(path);
assertEquals(booking.allBookings.get(0).getConcatenatedFullString(), createValidRecord());
ArrayList<bookingDetails> assertion = prepareValidRecordForValidateEachRecord();
assertEquals(booking.allBookings.get(0).getUserName(),assertion.get(0).getUserName());
}
public void testvalidatebookingdetails(){
String path = useValidFilePath();
readBookingDetails booking = new readBookingDetails(path);
ArrayList<bookingDetails> assertion = prepareValidRecordForValidateEachRecord();
assertEquals(booking.allBookings.get(0).getUserName(),assertion.get(0).getUserName());
assertEquals(booking.allBookings.get(0).getFlightNumber(),assertion.get(0).getFlightNumber());
assertEquals(booking.allBookings.get(0).getSeatCategory(),assertion.get(0).getSeatCategory());
assertEquals(booking.allBookings.get(0).getNumSeats(),assertion.get(0).getNumSeats());
assertEquals(booking.allBookings.get(0).getPaymentCard(),assertion.get(0).getPaymentCard());
}
}
Run: readBookingDetailsTest.testvalidatebookingdetails
Tests passed: 1 of 1 test – 4 ms
readBookingDetailsTest (test.par 4 ms
testvalidatebookingdetails 4 ms
/Users/ashika/Library/Java/JavaVirtualMachines/openjdk-18.0.1/Contents/Home/bin/java -ea -Didea.test.cyclic.buffer.size=1048576 -javaag
Process finished with exit code 0
Version Control Run Debug TODO Problems Terminal Services Build Dependencies
Tests passed: 1 (moments ago)
64:95 (86 chars) LF UTF-8 4 spaces


FlightBookingApp src test java test parser readFlightDetailsTest testvalidateflightdetails
readFlightDetailsTest.testvalidateflightdetails
Project
readBookingDetailsTest.java
readFlightDetailsTest.java
bookingDetails
flight
input
output
Output.csv
Output.txt
parser
readBookingDeta
readFlightDetails
testing
InvalidBooking.c
InvalidFlights.csv
ValidBooking.csv
ValidFlights.csv
validation
RunClient
test
java
org.example
test
parser
readBookingDeta
readFlightDetails
validation
validateBooking1
target
pom.xml
External Libraries
Scratches and Consoles
}
public String useInvalidFilePath(){
return "/Users/ashika/Downloads/FlightBookingApp/src/main/java/test/testing/InvalidFlights.csv";
}
public ArrayList<flight> prepareInvalidRecordForValidateEachRecord(){
ArrayList<flight> testFlights = new ArrayList<>();
String record = createValidRecord();
String[] fields = record.split( regex: ",");
flight f = new flight(fields[1],fields[0],Integer.parseInt(fields[2]), (double) Long.parseLong(fields[3]));
testFlights.add(f);
return testFlights;
}
public void testvalidateflightdetails(){
String path = useValidFilePath();
readFlightDetails flight = new readFlightDetails(path);
assertEquals(flight.allFlights.get(0).getFlightNumber(),prepareValidRecordForValidateEachRecord().get(0).getFlightNumber());
}
}
Run: readFlightDetailsTest.testvalidateflightdetails
Tests passed: 1 of 1 test – 2 ms
readFlightDetailsTest (test.parse 2 ms
testvalidateflightdetails 2 ms
/Users/ashika/Library/Java/JavaVirtualMachines/openjdk-18.0.1/Contents/Home/bin/java -ea -Didea.test.cyclic.buffer.size=1048576 -javaag
Process finished with exit code 0
Version Control Run Debug TODO Problems Terminal Services Build Dependencies
Tests passed: 1 (moments ago)
51:64 LF UTF-8 4 spaces


FlightBookingApp › src › test › java › test › validation › validateBookingTest › testValidation
validateBookingTest.testPaymentCard
Project
readBookingDetailsTest.java
readFlightDetailsTest.java
validateBookingTest.java
bookingDetails
flight
input
output
Output.csv
Output.txt
parser
readBookingDeta
readFlightDetails
testing
InvalidBooking.c
InvalidFlights.csv
ValidBooking.csv
ValidFlights.csv
validation
RunClient
test
java
org.example
test
parser
readBookingDeta
readFlightDetails
validation
validateBookingT
target
pom.xml
External Libraries
Scratches and Consoles
package test.validation;
import junit.framework.TestCase;
import test.parser.readBookingDetails;
import test.parser.readFlightDetails;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import static test.validation.validateBooking.validateCard;
public class validateBookingTest extends TestCase {
1 usage
public String InvalidPaymentCard() { return "9999800011144445"; }
1 usage
public String ValidPaymentCard() { return "4123456789012"; }
public void testPaymentCard(){
assertEquals(validateBooking.validateCard(ValidPaymentCard()), actual: true);
assertEquals(validateBooking.validateCard(InvalidPaymentCard()), actual: false);
}
public void testValidation() throws IOException {
readBookingDetails validrbd = new readBookingDetails( file: "/Users/ashika/Downloads/FlightBookingApp/src/main/java/test/testing/Vali
Run:
validateBookingTest.testPaymentCard
Tests passed: 1 of 1 test – 2 ms
validateBookingTest (test.validat 2 ms
testPaymentCard 2 ms
/Users/ashika/Library/Java/JavaVirtualMachines/openjdk-18.0.1/Contents/Home/bin/java -ea -Didea.test.cyclic.buffer.size=1048576 -javaag
Process finished with exit code 0
Version Control
Run
Debug
TODO
Problems
Terminal
Services
Build
Dependencies
Tests passed: 1 (moments ago)
41:6
LF
UTF-8
4 spaces


FlightBookingApp › src › test › java › test › validation › validateBookingTest
validateBookingTest.testValidation
public String InvalidPaymentCard() { return "9999800011144445"; }
1 usage
public String ValidPaymentCard() { return "4123456789012"; }
public void testPaymentCard(){
assertEquals(validateBooking.validateCard(ValidPaymentCard()), actual: true);
assertEquals(validateBooking.validateCard(InvalidPaymentCard()), actual: false);
}
public void testValidation() throws IOException {
readBookingDetails validrbd = new readBookingDetails( file: "/Users/ashika/Downloads/FlightBookingApp/src/main/java/test/testing/Vali
readFlightDetails validfbd = new readFlightDetails( file: "/Users/ashika/Downloads/FlightBookingApp/src/main/java/test/testing/ValidF
// readFlightDetails invalidfbd = new readFlightDetails("/Users/ashika/Downloads/FlightBookingApp/src/main/java/test/testing/Invali
String outputPath = "/Users/ashika/Downloads/FlightBookingApp/src/main/java/test/output/Output.csv";
String errorPath = "/Users/ashika/Downloads/FlightBookingApp/src/main/java/test/output/Output.txt";
validateBooking.validate(validrbd,validfbd,outputPath,errorPath);
BufferedReader br = new BufferedReader(new FileReader(errorPath));
assertEquals(br.readLine(), actual: null);
assertNull(br.readLine());
// validateBooking.validate(validrbd,invalidfbd,outputPath,errorPath);
// BufferedReader br = new BufferedReader(new FileReader(errorPath));
// assertNotNull(br.readLine());
}
}
Run:
validateBookingTest.testValidation
Tests passed: 1 of 1 test – 16 ms
validateBookingTest (test.valida 16 ms
testValidation 16 ms
/Users/ashika/Library/Java/JavaVirtualMachines/openjdk-18.0.1/Contents/Home/bin/java -ea -Didea.test.cyclic.buffer.size=1048576 -javaag
Process finished with exit code 0
Tests passed: 1 (moments ago)
11:1
LF
UTF-8
4 spaces