CMPE – 202 Individual Project Assignment

Team Name ***: Data Miners***

Student Name ***: Poojashree NS***

SJSU ID ***: 015979795***

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 HashMap 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:

Table 1: Flights

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

1. 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.

Instructions to build and execute the project:

1. Clone or download/unzip zip file from repository https://github.com/gopinathsjsu/individual-project-poojashreeNS/tree/main , Open the command line where the project directory is located.

   cd <path_to_downloaded_repo>/CMPE-202-IP/flightreservation

2. Execute the following command to build the project

   mvn compile

```
E:\CMPE-202-IP\flightreservation>mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] --------------------< cmpe202:flightreservation >----------------------
[INFO] Building flightreservation 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ flightreservation ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory E:\CMPE-202-IP\flightreservation\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ flightreservation ---
[INFO] Nothing to compile - all classes are up to date
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.619 s
[INFO] Finished at: 2022-05-03T15:39:49-07:00
[INFO] ------------------------------------------------------------------------
```

   mvn clean install

```
E:\CMPE-202-IP\flightreservation>mvn clean install
[INFO] Scanning for projects...
[INFO]
[INFO] --------------------< cmpe202:flightreservation >----------------------
[INFO] Building flightreservation 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ flightreservation ---
[INFO] Deleting E:\CMPE-202-IP\flightreservation\target
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ flightreservation ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory E:\CMPE-202-IP\flightreservation\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ flightreservation ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 12 source files to E:\CMPE-202-IP\flightreservation\target\classes
[INFO]
```

3. 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>"
```

```
E:\CMPE-202-IP\flightreservation>mvn exec:java -Dexec.mainClass=test.RunClient -D
exec.args="E:\CMPE-202-IP\Sample.csv E:\CMPE-202-IP\flightsdata.csv E:\CMPE-202-I
P\output.csv E:\CMPE-202-IP\error.txt"
[INFO] Scanning for projects...
[INFO]
[INFO] ---------------------< cmpe202:flightreservation >----------------------
[INFO] Building flightreservation 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec-maven-plugin:3.0.0:java (default-cli) @ flightreservation ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.704 s
[INFO] Finished at: 2022-05-03T15:43:35-07:00
[INFO] ------------------------------------------------------------------------
```

where,

- arg1 – path to the input data (*Sample booking file :"CMPE-202-IP\Sample.csv"*)
- arg2 – path to flight details to populate DB (*Sample database.csv file : "CMPE-202-IP\flights.csv"*)
- arg3 – path to Output.csv (*Sample output.csv file : "CMPE-202-IP\output.csv"*)
- arg4 – path to Output.txt (*Sample error file : "CMPE-202-IP\error.txt"*)

Primary problem and solution:

- Store the data read from the csv to a database which has only a single instance for accessing the data.
- Implemented **Singleton design pattern** for the database for this application.

Secondary problem and solution:

- Validate request at each stage if request holds good proceed with the flow of scenario or to stop the execution of the application.
- To implement this scenario, I used a **Chain of Responsibility** where each request is processed using handlers, if the request holds good, then the request is passed on to the next handler.

- Also, each card has a different strategy or algorithms to check if it is valid or not. I have implemented that using a **Strategy design pattern**.

Design Patterns Used:

Singleton:

One of the creational design patterns. It ensures that a class has only one instance, while providing a global access point to this instance.

- Make the default constructor private, to prevent other objects from using the new operator with the Singleton class.
- Create a static creation method that acts as a constructor. Under the hood, this method calls the private constructor to create an object and saves it in a static field. All following calls to this method return the cached object.

```
//Getting instance of the singleton object instead of creating it again and again
public static FlightDatabase getInstance() {
    if (db_instance == null)
        db_instance = new FlightDatabase();

    return db_instance;
}
```

Chain of Responsibility:

It is a behavioral design pattern that lets us pass requests along a chain of handlers. Upon receiving a request, each handler decides either to process the request or to pass it to the next handler in the chain.

- In my project, each check should be extracted to its own class with a single method that performs the check. The request, along with its data, is passed to this method as an argument.
- The best part is a handler can decide not to pass the request further down the chain and effectively stop any further processing.

```
package test;

public abstract class ValidateRequest {

    protected ValidateRequest nextRequest;

    public ValidateRequest setNextRequest(ValidateRequest nextRequest) {
        this.nextRequest = nextRequest;
        return nextRequest;
    }

    public abstract void validateRequest(Data nextRequest) throws Exception;

}
```

```java
// FlightDB handler : passes the request to next handler if the Flight is valid or
@Override
public void validateRequest(Data data) throws Exception {
    if (data.flightNumber != null && isFlightExsist(data.flightNumber)) {
        data.csvList.add(flightNumber);
        flight = getFlightInfo(data.flightNumber);
        validateRequestObject = setNextRequest(flight);
        validateRequestObject.validateRequest(data);
    } else {
        data.printErrorToFile(data, " : invalid flight number" + "\n");
        throw new Exception();
    }
}
```

Handler 1 : Check for Flight

```java
// Flight handler : passes the request to next handler if the category is valid
@Override
public void validateRequest(Data data) throws Exception {
    if (data.seatCategory != null && isSeatCategoryExsist(data.seatCategory)) {
        data.csvList.add(data.seatCategory);
        validateRequestObject = setNextRequest(categorySeatmapper.get(data.seatCategory));
        validateRequestObject.validateRequest(data);
    } else {
        data.printErrorToFile(data, " : invalid seat category" + "\n");
        throw new Exception();
    }
}
```

Handler 3 : Check for Category

```java
@Override
public void validateRequest(Data data) throws Exception {
    int seats = getAvaliableSeats();
    double price = getFlightPrice();
    String BookingSeat = data.numberOfSeats;

    if (BookingSeat.matches("-?(0|[1-9]\\d*)") && Integer.parseInt(BookingSeat) <= seats) {
        double totalPrice = Integer.parseInt(BookingSeat) * price;
        data.csvList.add(data.numberOfSeats);
        data.csvList.add(String.valueOf(totalPrice));
    } else {
        data.printErrorToFile(data, " : Seats not avaliable or seat request invalid" + "\n");
        throw new Exception();
    }
    setNextRequest(null);
}
```

Handler 4 : Check for Seats

```java
// Payment handler : passes the request to next handler if the card is valid
@Override
public void validateRequest(Data data) throws Exception {
    if (validateCardDetails(data.paymentCardNumber)) {
        setNextRequest(null);
    } else {
        data.printErrorToFile(data, " : invalid card" + "\n");
        throw new Exception();
    }
}
```

Handler 4 : Check for payment

<u>Strategy Pattern:</u>

It is a behavioral design pattern that lets us define a family of algorithms, put each of them into a separate class, and make their objects interchangeable.

- I have considered each card validation aspect to be a different algorithm and constructed a class for each of the cards.
- Upon satisfying the requirements of card length we pass on the validation request to a specific card class.

```java
package test;

public abstract class Card extends ValidateRequest{
    //ValidateCardDetails : abstract method validate card
    public abstract boolean validateCardDetails(String cardInfo);
}
```

Card Abstract class

```java
package test;

public class AmexCard extends Card {

    public AmexCard() {
    }

    // Amex Card : Card length should be 15 and second digit of the card should be 4
    // or 7
    public boolean validateCardDetails(String cardInfo) {
        return (cardInfo.length() == 15 && (cardInfo.substring(0, 2) == "4" || cardInfo.substring(0, 2) == "7"));
    }

    // Payment handler : passes the request to next handler if the card is valid
    @Override
    public void validateRequest(Data data) throws Exception {
        if (validateCardDetails(data.paymentCardNumber)) {
            setNextRequest(null);
        } else {
            data.printErrorToFile(data, " : invalid card" + "\n");
            throw new Exception();
        }
    }
}
```

Amex card class that extends card

```java
package test;

public class AmexCard extends Card {

    public AmexCard() {
    }

    // Amex Card : Card length should be 15 and second digit of the card should be 4
    // or 7
    public boolean validateCardDetails(String cardInfo) {
        return (cardInfo.length() == 15 && (cardInfo.substring(0, 2) == "4" || cardInfo.substring(0, 2) == "7"));
    }

    // Payment handler : passes the request to next handler if the card is valid
    @Override
    public void validateRequest(Data data) throws Exception {
        if (validateCardDetails(data.paymentCardNumber)) {
            setNextRequest(null);
        } else {
            data.printErrorToFile(data, " : invalid card" + "\n");
            throw new Exception();
        }
    }
}
```

Discovery card class that extends card

```java
package test;

public class AmexCard extends Card {

    public AmexCard() {
    }

    // Amex Card : Card length should be 15 and second digit of the card should be 4
    // or 7
    public boolean validateCardDetails(String cardInfo) {
        return (cardInfo.length() == 15 && (cardInfo.substring(0, 2) == "4" || cardInfo.substring(0, 2) == "7"));
    }

    // Payment handler : passes the request to next handler if the card is valid
    @Override
    public void validateRequest(Data data) throws Exception {
        if (validateCardDetails(data.paymentCardNumber)) {
            setNextRequest(null);
        } else {
            data.printErrorToFile(data, " : invalid card" + "\n");
            throw new Exception();
        }
    }
}
```

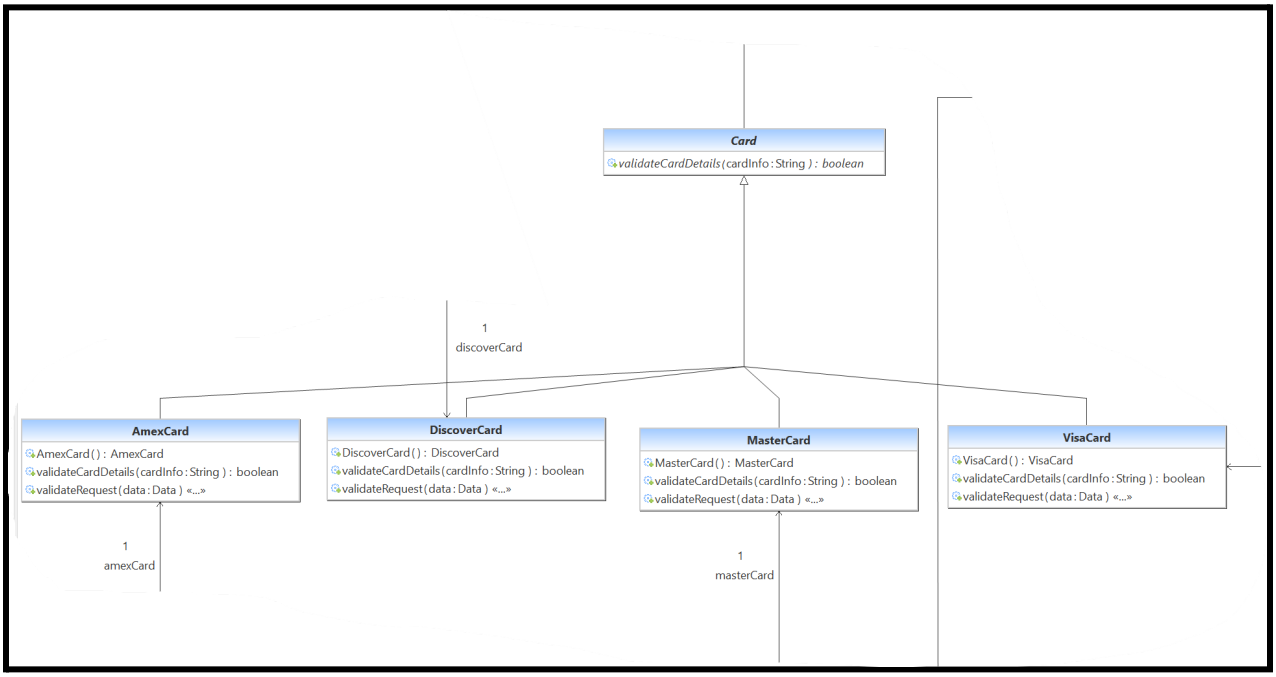
Master card class that extends card

```java
package test;

public class AmexCard extends Card {

    public AmexCard() {
    }

    // Amex Card : Card length should be 15 and second digit of the card should be 4
    // or 7
    public boolean validateCardDetails(String cardInfo) {
        return (cardInfo.length() == 15 && (cardInfo.substring(0, 2) == "4" || cardInfo.substring(0, 2) == "7"));
    }

    // Payment handler : passes the request to next handler if the card is valid
    @Override
    public void validateRequest(Data data) throws Exception {
        if (validateCardDetails(data.paymentCardNumber)) {
            setNextRequest(null);
        } else {
            data.printErrorToFile(data, " : invalid card" + "\n");
            throw new Exception();
        }
    }
}
```

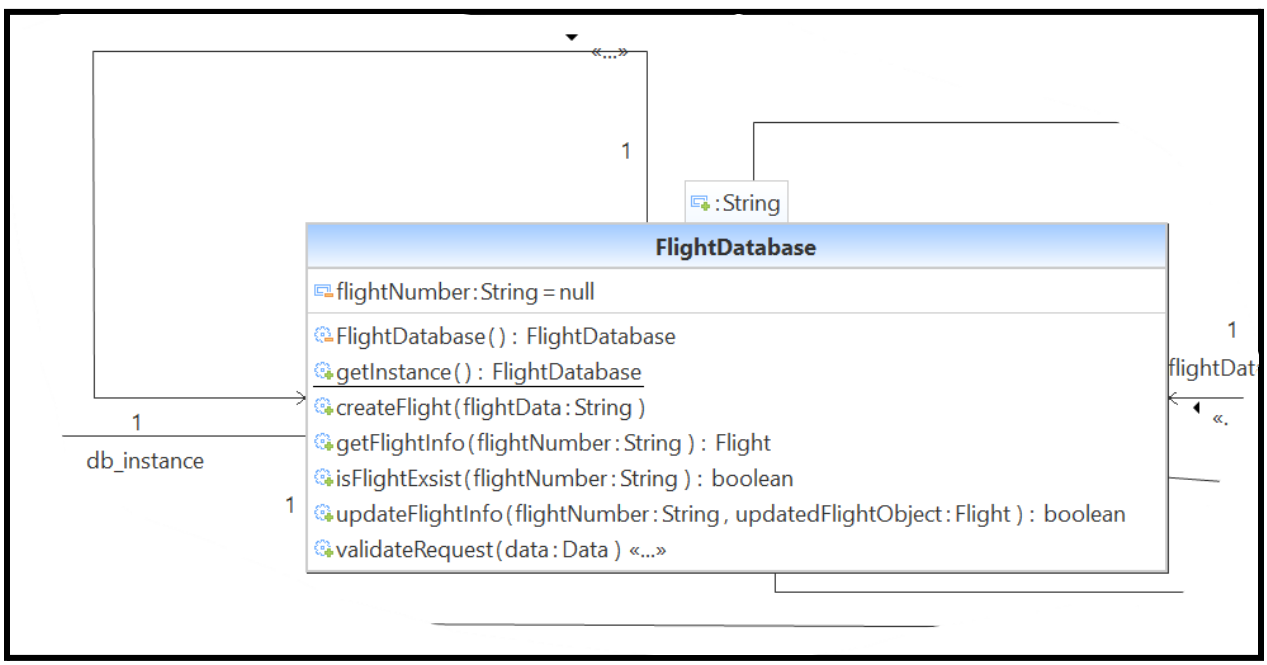
Visa card class that extends card

Class Diagrams for all the patterns used in the Project:

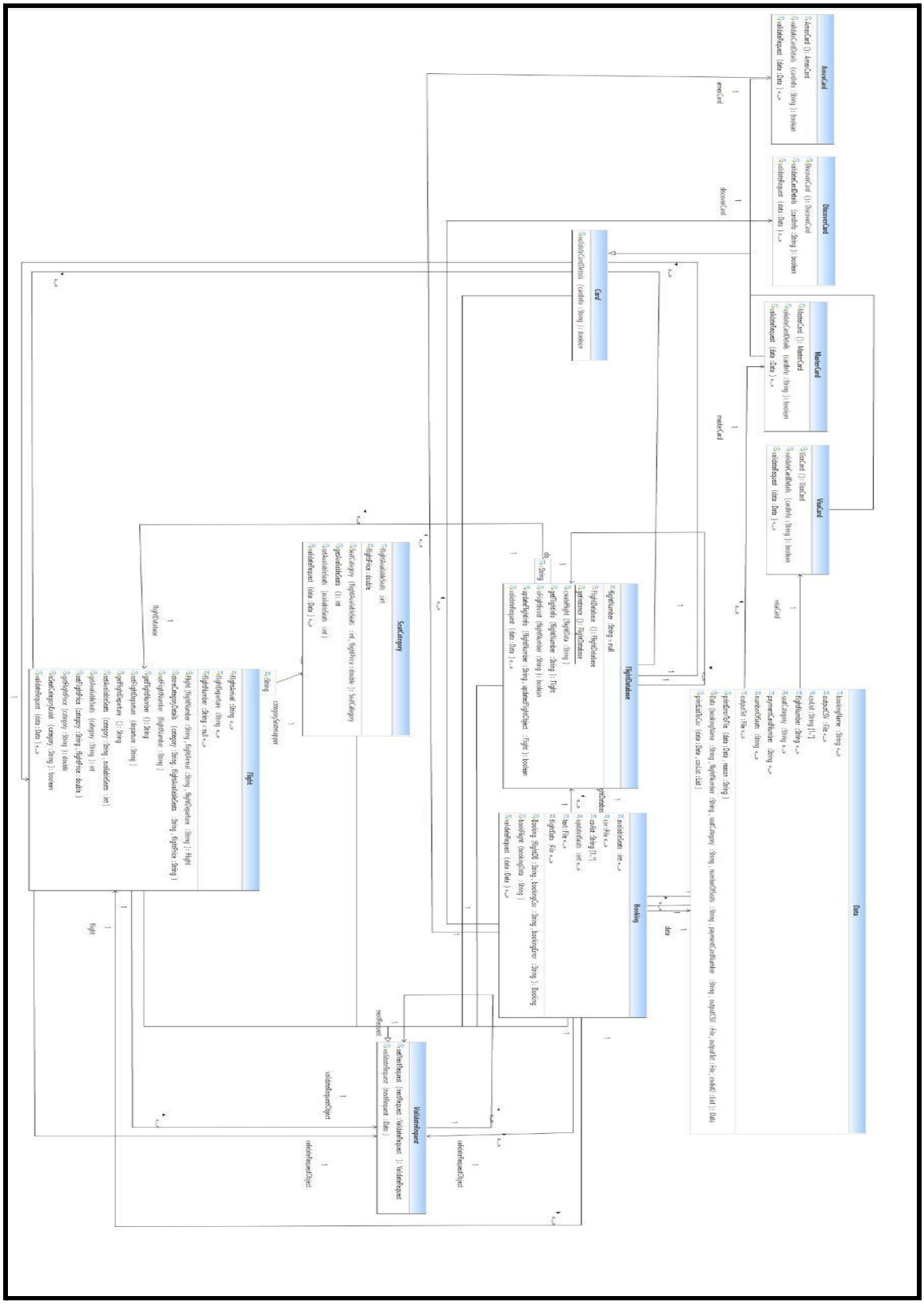
Strategy pattern:



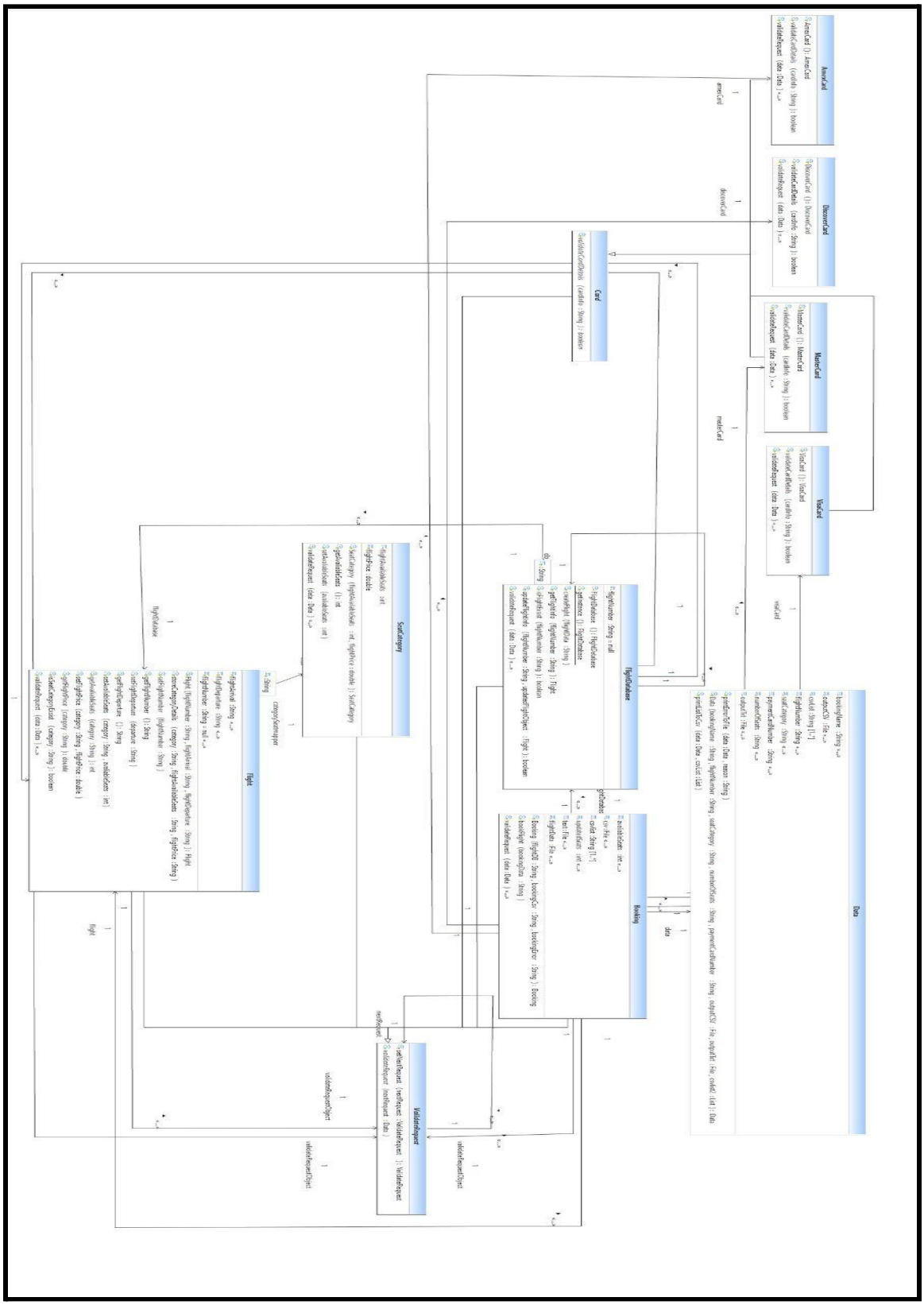
Singleton pattern:



Chain of Responsibility:



Class Diagram for the flight booking system:



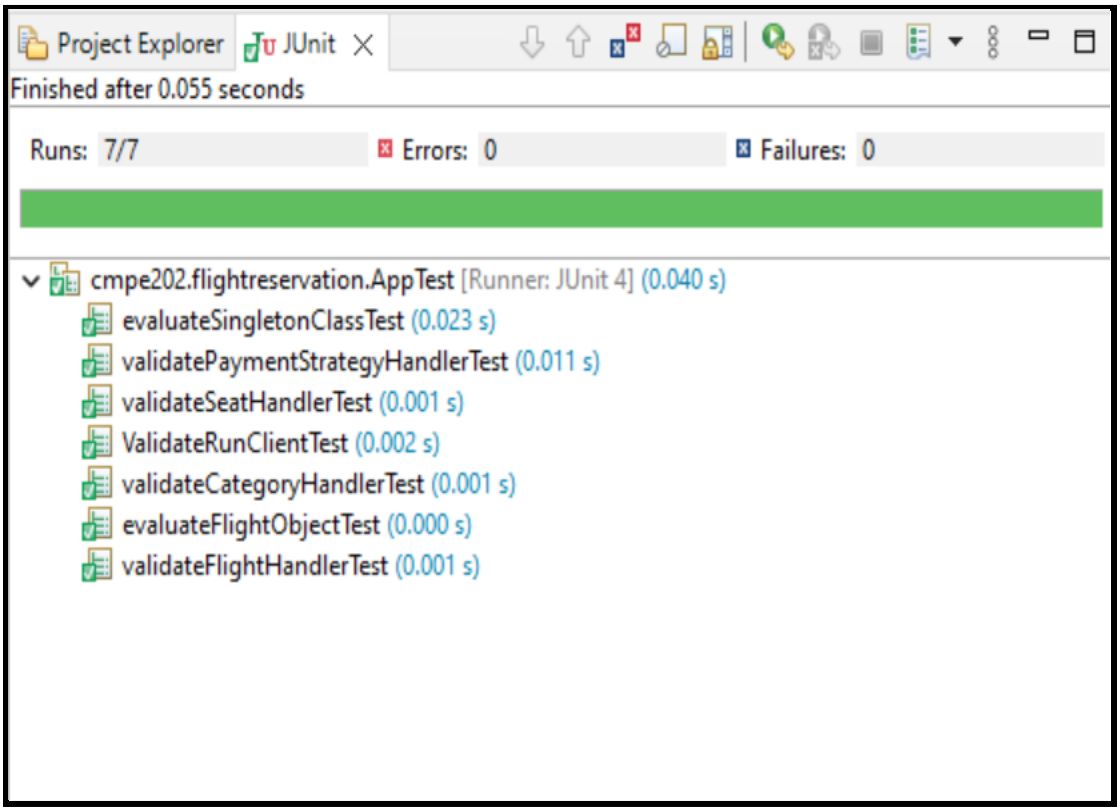
JUnit Test cases and its output:

Test case location : "*flightreservation \ src \ test \ java \ cmpe202 \ flightreservation \ AppTest*"

Resource location: "*flightreservation \ src \ test \ resources*"

Execution of test cases :

1. Run directly from eclipse using run as JUnit test.



2. Using command : `mvn clean test`

```
E:\CMPE-202-IP\flightreservation>mvn clean test
[INFO] Scanning for projects...
[INFO]
[INFO] ---------------------< cmpe202:flightreservation >----------------------
[INFO] Building flightreservation 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ flightreservation ---
[INFO] Deleting E:\CMPE-202-IP\flightreservation\target
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ flightreser
vation ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory E:\CMPE-202-IP\flightreservation\src\m
ain\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ flightreservatio
n ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 12 source files to E:\CMPE-202-IP\flightreservation\target\class
es
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ fli
ghtreservation ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 8 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ flightre
servation ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to E:\CMPE-202-IP\flightreservation\target\test-cl
asses
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ flightreservation -
--
[INFO] Surefire report directory: E:\CMPE-202-IP\flightreservation\target\surefir
e-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running cmpe202.flightreservation.AppTest
Tests run: 7, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.085 sec

Results :

Tests run: 7, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

Reports location: *"flightreservation \ target"*

```
-------------------------------------------------------------------------------
Test set: cmpe202.flightreservation.AppTest
-------------------------------------------------------------------------------
Tests run: 7, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.085 sec
```

Each Test cases explained with outputs :

Test case 1: evaluateSingletonClassTest:
Description: Check no multiple instances of the database gets created.
Input: Nothing
Output: assertion that database doesn't have more than one instance
Screenshot:

```
@Test
public void evaluateSingletonClassTest() {
    FlightDatabase fDatabase = FlightDatabase.getInstance();
    FlightDatabase fDatabase1 = FlightDatabase.getInstance();
    System.out.println("fDatabase : " + fDatabase);
    System.out.println("fDatabase1 : " + fDatabase1);
    assertEquals(fDatabase, fDatabase1);
}
```

```
Properties  Problems  Console ×
<terminated> AppTest [JUnit] C:\Users\pooja\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe  (May 3, 2022, 6:24
fDatabase : test.FlightDatabase@4fb64261
fDatabase1 : test.FlightDatabase@4fb64261
```

Test case 2: evaluateFlightObjectTest

Description: Evaluate flight object.

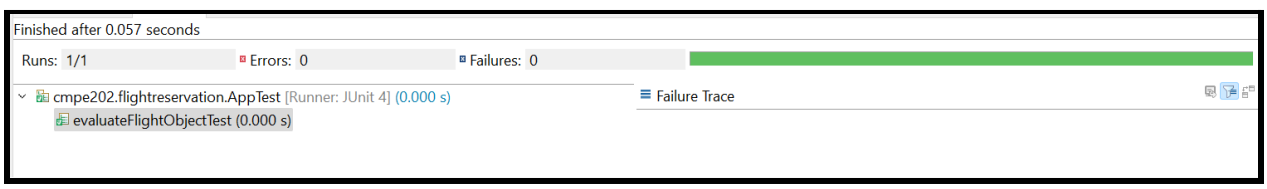
Input: Informations such as flight number, seat available, category and price

Output: Check if object is created with appropriate values

Screenshot:

```
@Test
public void evaluateFlightObjectTest() {
    Flight flight = new Flight("123", "A", "B");
    flight.storeCategoryDetails("Economy", "5", "500.00");

    String flightnum = flight.getFlightNumber();
    assertEquals("123", flightnum);

    int seats = flight.getAvaliableSeats("Economy");
    assertEquals(5, seats);

    flight.setAvaliableSeats("Economy", 20);
    seats = flight.getAvaliableSeats("Economy");
    assertEquals(20, seats);
}
```

Output:



Test case 3: validateFlightHandlerTest

Description: Flight handler validating booking request.

Input: Booking information, Flight database information, Booking confirmation file, Error file

Output: For invalid flight number, Error message will be recorded in Error file.

Screenshot:

```java
@Test
public void validateFlightHandlerTest() throws IOException {
    try {
        Booking booking = new Booking(flightDatabase, bookingConfirmation, errorFile);
        booking.bookFlight(invalidFlight);

        scanner.useDelimiter("\n");

        assertTrue(scanner.hasNext());
        assertEquals("Please enter correct booking details for Sierra1 : invalid flight number", scanner.next());
    } finally {
        scanner.close();
    }
}
```

Input:

```
BookingName, flightNumber, seatCategory, numberOfSeats, paymentCardNumber
Sierra1,XA234,Business,1,6011234578124345
```

Output:

AppTest.java | error.txt

```
Please enter correct booking details for Sierra1 : invalid flight number

```

Test case 4: validateCategoryHandlerTest

Description: Category handler validating booking request category type.

Input: Booking information, Flight database information, Booking confirmation file, Error file

Output: For invalid flight category, Error message will be recorded in the Error file.

Screenshot:

```
@Test
public void validateCategoryHandlerTest() throws IOException {
    try {
        Booking booking = new Booking(flightDatabase, bookingConfirmation, errorFile);
        booking.bookFlight(invalidCategory);

        scanner.useDelimiter("\n");

        assertTrue(scanner.hasNext());
        assertEquals("Please enter correct booking details for Sierra1 : invalid seat category", scanner.next());
    } finally {
        scanner.close();
    }
}
```

Input:

```
BookingName, flightNumber, seatCategory, numberOfSeats, paymentCardNumber
Sierra1,BY110,Class-A,1,6011234578124345
```

Output:

AppTest.java error.txt

```
1 Please enter correct booking details for Sierra1 : invalid seat category
2
```

Test case 5: validateSeatHandlerTest

Description: Seat handler validates seat availability.

Input: Booking information, Flight database information, Booking confirmation file, Error file

Output: For invalid seat number or if no seats available, Error message will be recorded in the Error file.

Screenshot:

```java
@Test
public void validateSeatHandlerTest() throws IOException {
    try {
        Booking booking = new Booking(flightDatabase, bookingConfirmation, errorFile);
        booking.bookFlight(invalidSeatNumber);

        scanner.useDelimiter("\n");

        assertTrue(scanner.hasNext());
        assertEquals(
                "Please enter correct booking details for Sierra1 : Seats not avaliable or seat request invalid",
                scanner.next());
        assertEquals(
                "Please enter correct booking details for Sierra2 : Seats not avaliable or seat request invalid",
                scanner.next());
    } finally {
        scanner.close();
    }
}
```

Input:

```
BookingName, flightNumber, seatCategory, numberOfSeats, paymentCardNumber
Sierra1,BY110,Business,A,6011234578124345
Sierra2,BY110,Business,3,6011234578124345
```

Output:

AppTest.java | error.txt

```
Please enter correct booking details for Sierra1 : Seats not avaliable or seat request invalid
Please enter correct booking details for Sierra2 : Seats not avaliable or seat request invalid
```

Test case 6: validatePaymentStrategyHandler

Description: Payment handlers validates if card is valid or not

Input: Booking information, Flight database information, Booking confirmation file, Error file.

Output: For invalid payment card number, Error message will be recorded in the Error file.

Screenshot:

```
@Test
public void validatePaymentStrategyHandlerTest() throws IOException {
    try {
        Booking booking = new Booking(flightDatabase, bookingConfirmation, errorFile);
        booking.bookFlight(invalidPayment);

        scanner.useDelimiter("\n");

        assertTrue(scanner.hasNext());
        assertEquals("Please enter correct booking details for Sierra1 : invalid card", scanner.next());
        assertEquals("Please enter correct booking details for Sierra2 : invalid card", scanner.next());
        assertEquals("Please enter correct booking details for Sierra3 : invalid card", scanner.next());
        assertEquals("Please enter correct booking details for Sierra4 : invalid card", scanner.next());
        assertEquals("Please enter correct booking details for Sierra5 : invalid card", scanner.next());
        assertEquals("Please enter correct booking details for Sierra6 : invalid card", scanner.next());
    } finally {
        scanner.close();
    }

}
```

Input :

```
BookingName, flightNumber, seatCategory, numberOfSeats, paymentCardNumber
Sierra1,BY110,Business,1,1234561323130
Sierra2,BY110,Business,1,5410000000000000
Sierra3,BY110,Business,1,6011000000000000
Sierra4,BY110,Business,1,40110000000000000
Sierra5,BY110,Business,1,4010000000
Sierra6,BY110,Business,1,340110000000000000
```

Output:

AppTest.java | error.txt

```
Please enter correct booking details for Sierra1 : invalid card
Please enter correct booking details for Sierra2 : invalid card
Please enter correct booking details for Sierra3 : invalid card
Please enter correct booking details for Sierra4 : invalid card
Please enter correct booking details for Sierra5 : invalid card
Please enter correct booking details for Sierra6 : invalid card
```

Test case 7: validateRunClientTest

Description: Positive end to end application flow.

Input: Booking information, Flight database information, Booking confirmation file, Error file.

Output: Complete Application flow which should record booking confirmation into the booking confirmation file.

Screenshot:

```
@Test
public void validateRunClientTest() throws IOException {
    Scanner scanner = new Scanner(new File(bookingConfirmation));
    try {
        Booking booking = new Booking(flightDatabase, bookingConfirmation, errorFile);
        booking.bookFlight(validBooking);

        scanner.useDelimiter("\n");

        assertTrue(scanner.hasNext());
        assertEquals("Anna1,CA453,Economy,1,300.0", scanner.next());
        assertEquals("Anna2,CA453,Premium Economy,1,500.0", scanner.next());
        assertEquals("Anna3,CA453,Business,1,2000.0", scanner.next());
    } finally {
        scanner.close();
        new FileWriter(bookingConfirmation, false).close();
    }
}
```

Input:

```
BookingName, flightNumber, seatCategory, numberOfSeats, paymentCardNumber
Anna1,CA453,Economy,1,4120000000000
Anna2,BY110,Premium Economy,1,4120000000000
Anna3,BY110,Business,1,4120000000000
```

Output:

bookingSuccess - Notepad

File Edit View

```
Anna1,CA453,Economy,1,300.0
Anna2,CA453,Premium Economy,1,500.0
Anna3,CA453,Business,1,2000.0
```