



Gina Cody School of Engineering and Computer Science

Department of Electrical and Computer Engineering

COEN 6312- Model Driven Software Engineering

Winter 2022

Group 6: DELIVERABLE 4

Airline Reservation System

Submitted To:

Dr. Wahab Hamou-Lhadj

Submitted By:

Nag Prudhvi Lanka (40164553)

Sai Srija Boddu (40165172)

Bhavani Shankar Reddy Annu (40168191)

Shivani Palle (40161491)

Venu Babu Podila (40159998)

Table of Contents

| | |
|---|-----------|
| <i>List of Figures</i> | 3 |
| <i>Problem Statement</i> | 4 |
| <i>Domain Analysis</i> | 5 |
| Functional Requirements: | 5 |
| Non-Functional Requirements: | 5 |
| User Stories: | 6 |
| Use Case Diagram: | 7 |
| <i>Design of the System</i> | 8 |
| Class Diagram: | 8 |
| OCL Constraints: | 10 |
| State Machine Diagram: | 12 |
| Introduction: | 12 |
| AdminDetails: | 14 |
| Passenger: | 15 |
| FlightInfo: | 17 |
| <i>Screenshots of the Output</i> | 18 |
| <i>Comments on OCL Constraints</i> | 19 |
| <i>Testing of the System</i> | 20 |
| Test cases: | 20 |
| <i>References</i> | 20 |

List of Figures

| | |
|---|----|
| Figure 1: Use Case Diagram | 7 |
| Figure 2: Class Diagram | 9 |
| Figure 3: State Diagram for AdminDetails Class | 14 |
| Figure 4: State Diagram for Passenger Class..... | 16 |
| Figure 5: State Diagram for FlightInfo Class | 17 |
| Figure 6: Screen capture from code (Passenger Class)..... | 18 |
| Figure 7: Screen capture from console window | 18 |
| Figure 8: Screen capture of the output..... | 19 |

Problem Statement

The Airline Reservation System is a web-based online ticket reservation site that store and retrieve information related to the air travel. The system allows the customers to search for the availability, fares, helps to reserve a ticket, and pay online without any difficulty. A reliable system is developed to provide the customer with an updated information on price, flight timings, availability of seats and online transaction history. It also allows the end user to search for specific flights from any source to destination, keep customer records and itinerary details. The main advantage of this system is convenience, which means you can plan your trip online anytime in a day or night.

The target or potential customers for an airline reservation system are the people in need to travel without standing in a queue outside an airline office. Customers on the go can even make reservations on their mobile phones, tablets, or any portable devices with internet access. This Airline Reservation System includes an admin and a customer, where admin manages the information related to the booking. Customer/passenger is provided with a username and password upon registration to access the system.

Maven is used as the project management tool to build our Airline Reservation System. It is based on the POM (Project Object Model) concept, which is used to build a project, support better development practices and documentation. By keeping the source code separately and using test case conventions to execute tests helps in best development practices.

Domain Analysis

Functional Requirements:

- Flight availability can be checked by unregistered user, this is the only feature that an unregistered user can act upon
- Users need to sign in to reserve or cancel the flight ticket
- Unregistered users need to sign up to perform actions of bookings
- When making a reservation, users must choose the airport's source and destination cities
- User can book multiple tickets with limiting the total bookings to four
- Users can also check the status of the reservation that they have made
- Only the user who made reservation can cancel the specific booking
- Notifications will be displayed after every task that user performs

Non-Functional Requirements:

- Availability: The System shall be available to the user 24x7
- Security: Sensitive information such as user's login credentials and payment details should be secured by the system. Security of user's data is one of the requirements in our system
- Performance: The system response time and processing time for application loading, refreshing, functions and calculations shall be as minimum as possible
- Reliability: The system shall run without any failure. Even if there is any system failure it shall be recovered quickly
- Capacity: The system shall be robust and fault tolerant

User Stories:

| S. No | User Stories |
|-------|---|
| 1 | As a User, I want to login into my account |
| 2 | As a User, I want to create an account for logging in |
| 3 | As a User, I want to get an option to make a reservation |
| 4 | As a User, I want to select the airlines for my flight journey |
| 5 | As a User, I want to specify the kind of travel, such as one-way or round trip. |
| 6 | As a User, I want to book a ticket on a particular date |
| 7 | As a User, I want to access my trip history |
| 8 | As a User, I want to select the type of class of the flight for that particular date |
| 9 | As a User, I want to select the type of card for payment |
| 10 | As a User, I want to modify my booking details at the time of booking or even after the booking is done |
| 11 | As a User, I want to check the status of my flight |
| 12 | As a User, I want to cancel the reservation of flight |

Use Case Diagram:

The figure below depicts the use case diagram of the proposed system.

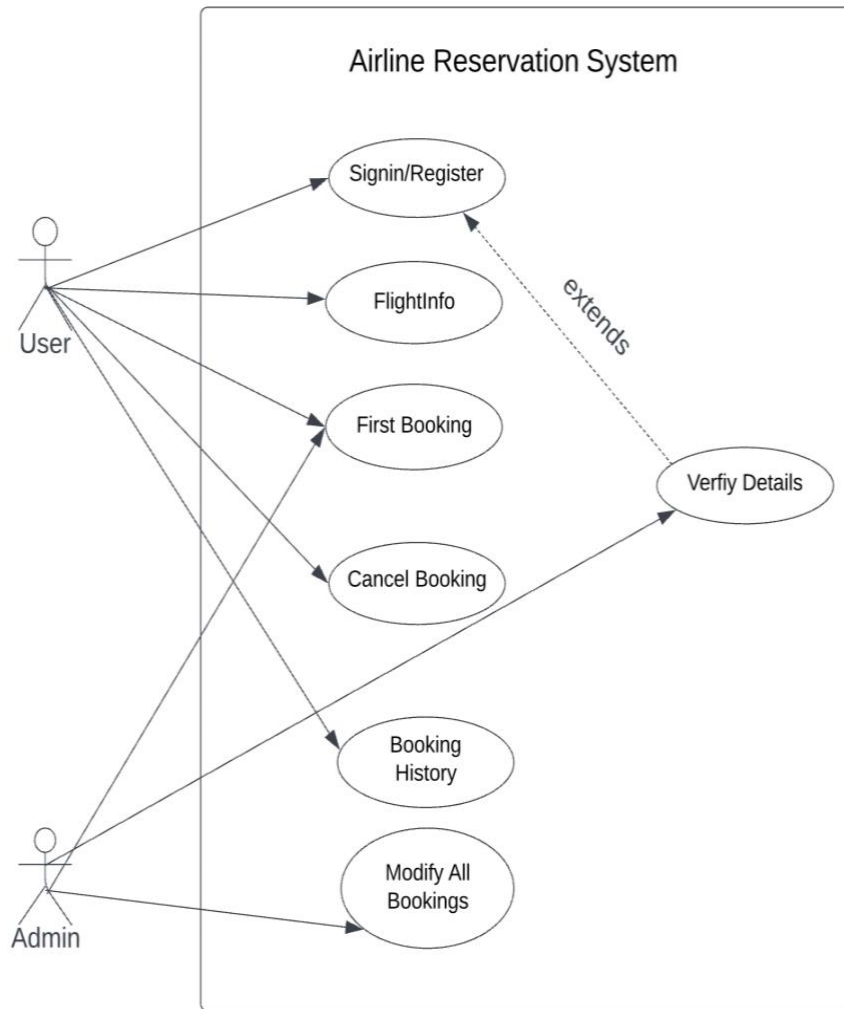


Figure 1: Use Case Diagram

The use case diagram of the Airline Reservation System consists of two actors, user who is the potential customer and the administrator. The user is the primary actor and admin is the secondary actor. The user shall sign-in or register and then view information of available flights. The user shall book the flight tickets and check booking history. If in case, they change their mind the user can cancel the booking. The admin shall verify details of the users. The admin shall also check ticket booking and do modifications if there are any.

Design of the System

Class Diagram:

A class diagram is a representation of the classes, attributes, and associations between classes. It determines the relationships and source code dependency among classes in Unified Modeling Language (UML). In context, a class defines variables in an object, and methods. Class diagrams were useful in Object Oriented Programming (OOP), which is static in nature. As it gives static view of an application. Class diagram is not only used for visualization, description, booking but also for creating executable code of the software application. It is also defined as Structural diagram that shows collection of classes, associations, interfaces, constraints, and collaborations.

In our class diagram, we have six classes. For each class we have specific functions and attributes that are pertaining to a class.

- **Passenger:**
This class gives the details of the passenger or customer along with data types such as passenger name, phone number, etc.,
- **FlightInfo:**
This class contains all the information about the Flight along with their data types such as flight schedule, cost, seat capacity.
- **AdminDetails:**
This is the class for Admin which control all the information for most of the classes and thus it will be synchronized during the development phase accordingly.
- **Run:**
The main () is the starting point to execute a program in Java. The main method of this system is in the Run class.
- **Booking:**
This class is used to make passenger booking.
- **SeatAllocation:**
This class allocates and confirms a seat.

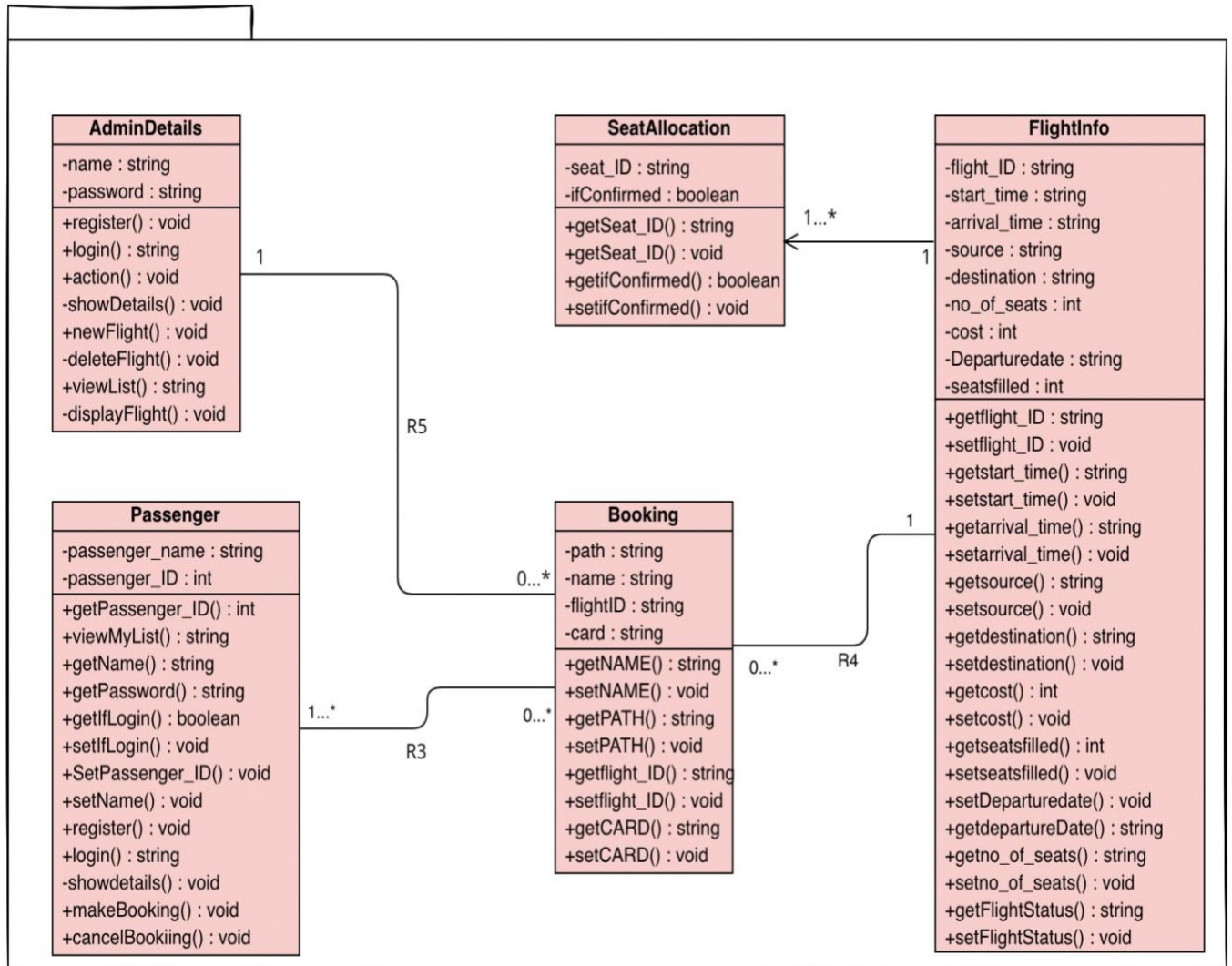


Figure 2: Class Diagram

OCL Constraints:

There are several constraints in our Airline Reservation System which are specified using the Object Constraint Language (OCL). OCL is textual language of typed expressions. It enhances the precision and simplicity of the system. Below are few OCL constraints for each class specified clearly.

- **Passenger:**

- 1. Every passenger registering in the Airline Reservation System should have unique passenger ID.**

context Passenger

inv: allInstances -> forAll(p1,p2:passengers | p1<>p2 implies p1.passenger_ID <>p2.passenger_ID)

- 2. The same passenger cannot have more than one booking for the same flight.**

context Passenger

inv: self.R3.R4 ->

forAll(f1,f2:FlightInfo | f1<>f2 AND f1.flight_ID=f2.flight_ID) implies
(f1.date_of_departure <> f2.date_of_departture AND f1.start_time <> f2.start_time)

- 3. Passenger can only cancel the flight if there exist a booking respective to their name.**

context Passenger::cancelBooking(string passenger_name): void

pre: self.Booking.FlightInfo -> includes(passenger_name)

post: self.Booking.FlightInfo -> excludes(passenger_name)

- 4. If the passenger wants to check any bookings, they can only check the bookings which belong to them.**

context Passenger::viewMyList()

inv: self.Booking.path -> select(self.passenger_name)

- 5. Phone number of Customer/Passenger should be of 10 digits.**

context Passenger

inv: self.phone_number -> size() = 10

- **AdminDetails:**

6. The flights are generated only by the administrators.

context AdminDetails

pre: self.Booking.Flight_Info.FlightStatus = 'notPosted'

post: self.Booking.Flight_Info.FlightStatus = 'Accessible'

- **FlightInfo:**

7. Each airline flight must have unique flight number.

context FlightInfo

inv: allInstances -> forAll(f1,f2: FlightInfo | f1<>f2 implies f1.flight_ID<>f2.flight_ID)

8. Each airline flight can have maximum of 60 seats.

context FlightInfo

inv: self.no_of_seats -> size()<=60

9. The booking is only added if it does not exist previously.

context Passenger::make(Booking:b): void

pre: self.R3 -> excludes(b)

post: self.R3 -> includes(b)

10. The booking can only be cancelled if it exists already.

context Passenger::cancel(Booking:b): void

pre: self.R3 -> includes(b)

post: self.R3 -> excludes(b)

11. Flight departure time should not be after the arrival time.

context FlightInfo

inv: allInstances -> forAll(f:Flight_Info | f.start_time<f.arrival_time)

12. The departure city of airline flight should be different from its arrival city.

context FlightInfo

inv: allInstances -> forAll(f1:Flight_Info | f1.from<>f1.to)

State Machine Diagram:

Introduction:

A State Machine Diagram is used to describe the behaviour of a system with respect to its operation i.e., steps involved from start of an operation to its end. This includes (internal as well as external) events, transitions (from one to another state), with several intermediate states before the end point of the process. This is a modeling diagram, which is used to develop a real-world modeling system.

Symbols and Notations used to implement a basic state diagram:

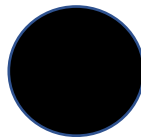
- **STATE:**

- A state is a particular situation in the life cycle of a certain operation executed by a class. It is represented by a rectangular shape with rounded corners with name of state written in it.



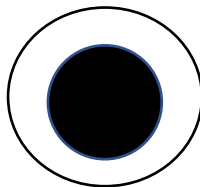
- **Initial State:**

- The initial state of a state diagram is known as the initial pseudo state. It is the start point of the state machine diagram indicated by a solid circle.



- **Final State/ End Point/ Terminator:**

- The final state illustrates the end of a system, which is indicated by a filled circle within a circle.



- **EVENT/ACTION:**

- Instance which generates a transition, which appears on above the transition arrow.

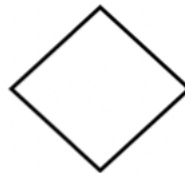
- **TRANSITION:**

- Transition is a part of analysis and design of a system. Transition is determined by an arrow that goes from one state to another. Every transition line depicts the event that causes the transition.



- **Decision-box:**

- The pseudo state represented by a diamond shape, indicates the decision state with dynamic conditions with branched potential results.



AdminDetails:

The administrator has set of features which are explained by the AdminDetails class in the below state diagram. There are set of options for the admin to select in order to perform any actions. These options range from 0 to 4.

1. Initially the admin needs to sign-in. Then these set options will be displayed as a menu.
2. If the admin selects option 1, then he or she shall be able to create a flight if it does not exist already. If the flight which he wants to create with the flight ID already exist, then the admin will not be able to create the flight and asked to enter a new flight ID. If the entered flight ID is new, then the status of the flight is changed.
3. If the admin wants to view all the existing orders, then he selects option 2. The csv file is read to check if there are any existing orders, then those orders will be displayed. If there are no orders, then a message is displayed stating that “there are no flights”.
4. If the admin wants to delete any posted flight, he selects option 3. Then he enters the flight ID which he wants to remove, respective with the csv file is read, if the ID is found then it is removed else it will go to the state ‘view menu’
5. If the admin wants to change the status of the flight, he selects option 4. The posted flights data is fetched from the csv file later the status of the flight is changed to ACCESSIBLE.
6. If the admin wants to exit from the menu, he selects option 0 which brings him to the final state.

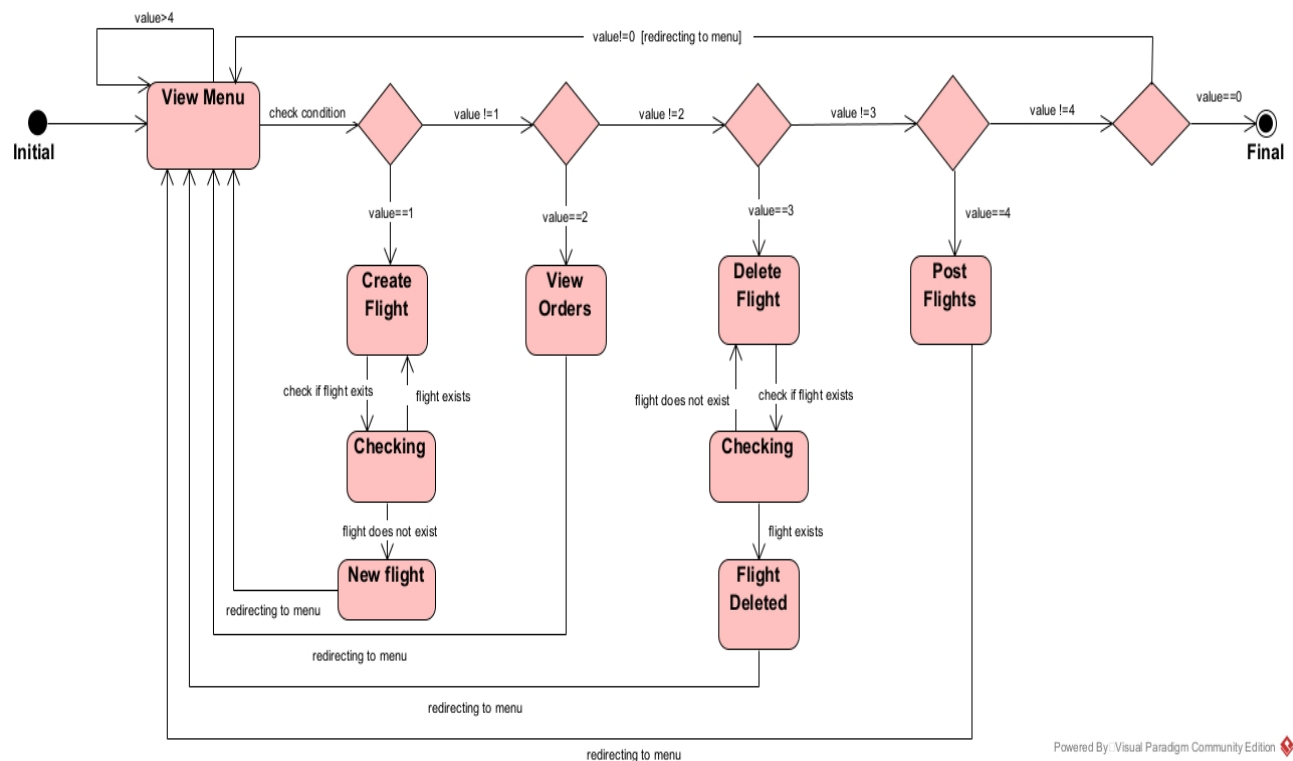


Figure 3: State Diagram for AdminDetails Class

Passenger:

Exactly when Passenger picks Condition 1, he can search for flights depending upon the from and to addresses he supplies. Expecting that he inputs a comparative region as the source and goal, he will be referenced to survey and enter the source and objective regions. Regardless, expecting that the source and goal are interesting, the application checks the flights record to check whether an excursion with the huge source and goal is available or not. There is no such thing as expecting an outing with the necessary source and goal, a notice is shown

Upon logging as a passenger, a menu will be displayed to the user after being triggered by the initial state.

1. In this menu, on the off chance that the passenger chooses condition 0, he leaves the menu and hence comes to the final state.
2. Exactly when Passenger picks Condition 1, he can search for flights depending upon the from and to addresses he supplies. Expecting that he inputs a comparative region as the source and destination, he will be referenced to survey and enter the source and destination regions. Regardless, expecting that the source and destination are interesting, the application checks the flights record to check whether details with the from and to are available or not. There is no such thing as expecting an outing with the necessary source and goal, a notice is shown.
3. Then, based on the Flight ID given by the passenger, there is one more option to save a flight. This is completed by the make Booking command (). To confirm the reservation, the customer inputs the card details for payment. A client whose booking has been confirmed has the option to drop or withdraw a booking, as well as see and question all of his bookings.
4. Whenever the passenger picks condition 3 from the menu, the flight booked by the passenger will be erased. The subtleties of it would be removed as well.
5. One more usefulness that a customer can have as a passenger is that he can question all the appointments made by him. A rundown of all the flight booking done by the client is shown from the passengers.csv file. He can then exit from this choice and arrive at the last state.
6. The client is incited to pick a legitimate option by assuming that he chooses none of the options gave in the menu.

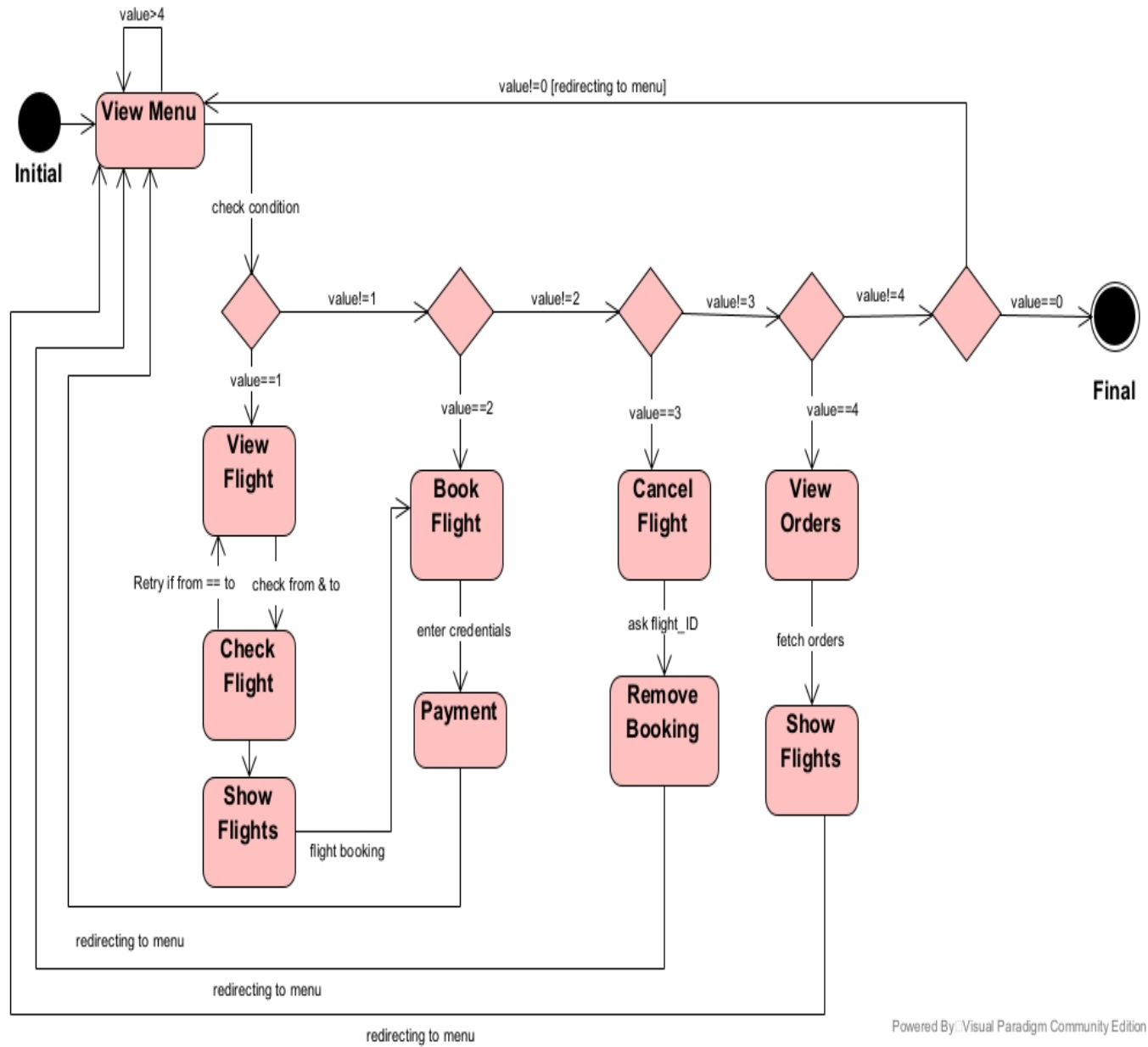


Figure 4: State Diagram for Passenger Class

FlightInfo:

The below state diagram depicts the FlightInfo class of the system. Initially, to search for a flight, set of details needed to be provided within the state. Next, it goes to the verify details state in which the input given is validated in order to show the status of the flight. If the details were verified successfully, it goes to the 'Show Flight Status' state and then to the Final/ end state. If the verification was unsuccessful, then it directly goes to the end state. Also, from the 'get flight details' state if the user wants to cancel the search, it directly quits and goes to an end (final) state.

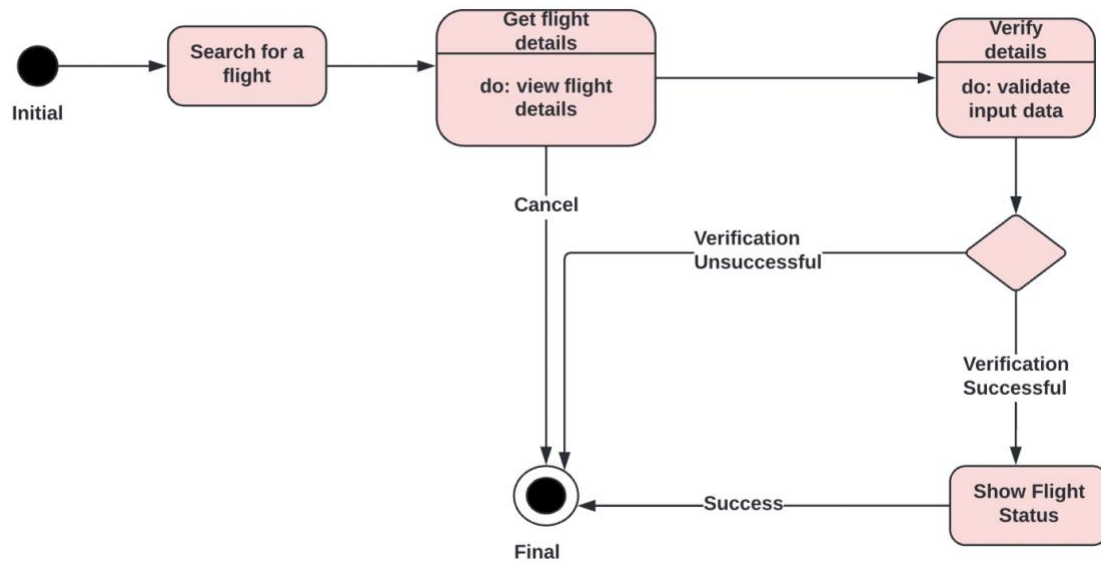
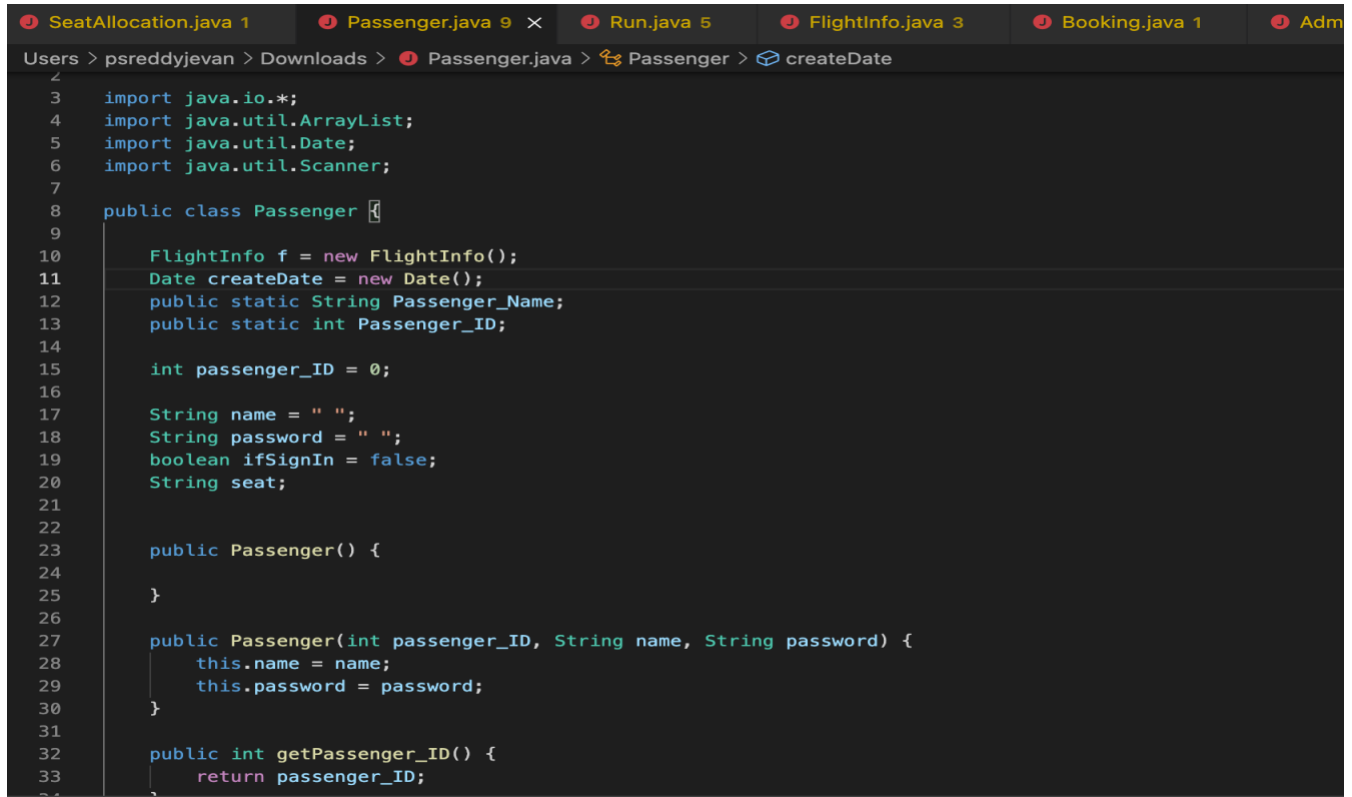


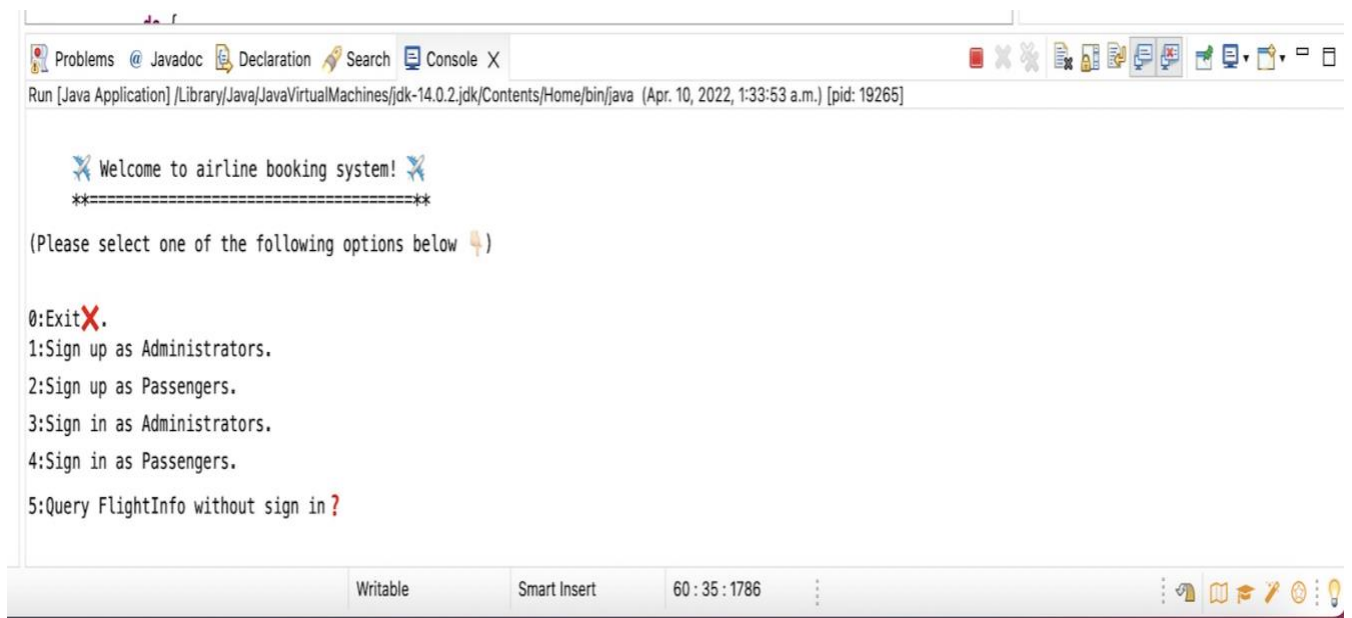
Figure 5: State Diagram for FlightInfo Class

Screenshots of the Output



```
SeatAllocation.java 1 Passenger.java 9 x Run.java 5 FlightInfo.java 3 Booking.java 1 Adm
Users > psreddyjevan > Downloads > Passenger.java > Passenger > createDate
2
3 import java.io.*;
4 import java.util.ArrayList;
5 import java.util.Date;
6 import java.util.Scanner;
7
8 public class Passenger {
9
10     FlightInfo f = new FlightInfo();
11     Date createDate = new Date();
12     public static String Passenger_Name;
13     public static int Passenger_ID;
14
15     int passenger_ID = 0;
16
17     String name = " ";
18     String password = " ";
19     boolean ifSignIn = false;
20     String seat;
21
22
23     public Passenger() {
24
25     }
26
27     public Passenger(int passenger_ID, String name, String password) {
28         this.name = name;
29         this.password = password;
30     }
31
32     public int getPassenger_ID() {
33         return passenger_ID;
34     }
35 }
```

Figure 6: Screen capture from code (Passenger Class)



```
Problems @ Javadoc Declaration Search Console X
Run [Java Application] /Library/Java/JavaVirtualMachines/jdk-14.0.2.jdk/Contents/Home/bin/java (Apr. 10, 2022, 1:33:53 a.m.) [pid: 19265]

Welcome to airline booking system!
*****
(Please select one of the following options below )

0:ExitX.
1:Sign up as Administrators.
2:Sign up as Passengers.
3:Sign in as Administrators.
4:Sign in as Passengers.
5:Query FlightInfo without sign in?
```

Figure 7: Screen capture from console window

The screenshot shows an IDE with several tabs: AdminDetails.java, Passenger.java, TicketBookingTe, Booking.java, and Run.java. The main editor displays the Booking.java file with the following code:

```

11 private String card;
12
13 public Booking(String name, String flightID, String card) throws IOException {
14     this.path = "orders.csv";
15     this.name = name;
16     this.flightID = flightID;
17     this.card = card;
18     FileWriter fw = new FileWriter(path, true);
19     BufferedWriter br = new BufferedWriter(fw);
20     PrintWriter pw = new PrintWriter(br);
21
22     pw.println(name+" "+flightID+" "+card);
23     pw.flush();
24     pw.close();
25     System.out.println("Congratulations, you have made the booking! ✅");

```

The Outline pane on the right shows the Booking class structure:

- path : String
- name : String
- flightID : String
- card : String
- Booking(String, String, String)
- getName() : String
- setName(String) : void
- getPath() : String
- setPath(String) : void
- getCARD() : String
- setCARD(String) : void
- getflightID() : String
- setflightID(String) : void

The Console pane at the bottom shows the following output:

```

Run [Java Application] /Library/Java/JavaVirtualMachines/jdk-14.0.2.jdk/Contents/Home/bin/java (Apr. 10, 2022, 1:29:02 a.m.) [pid: 19216]
3:Sign in as Administrators.
4:Sign in as Passengers.
5:Query FlightInfo without sign in ?
3
Enter your username
venu
Enter your password
poddila
Thats the correct username and password!!!

Welcome to the admin menu!!
0:Exit
1:Creat flight
2:View all orders
3>Delete flight
4:Publish flights

```

Figure 8: Screen capture of the output

Comments on OCL Constraints

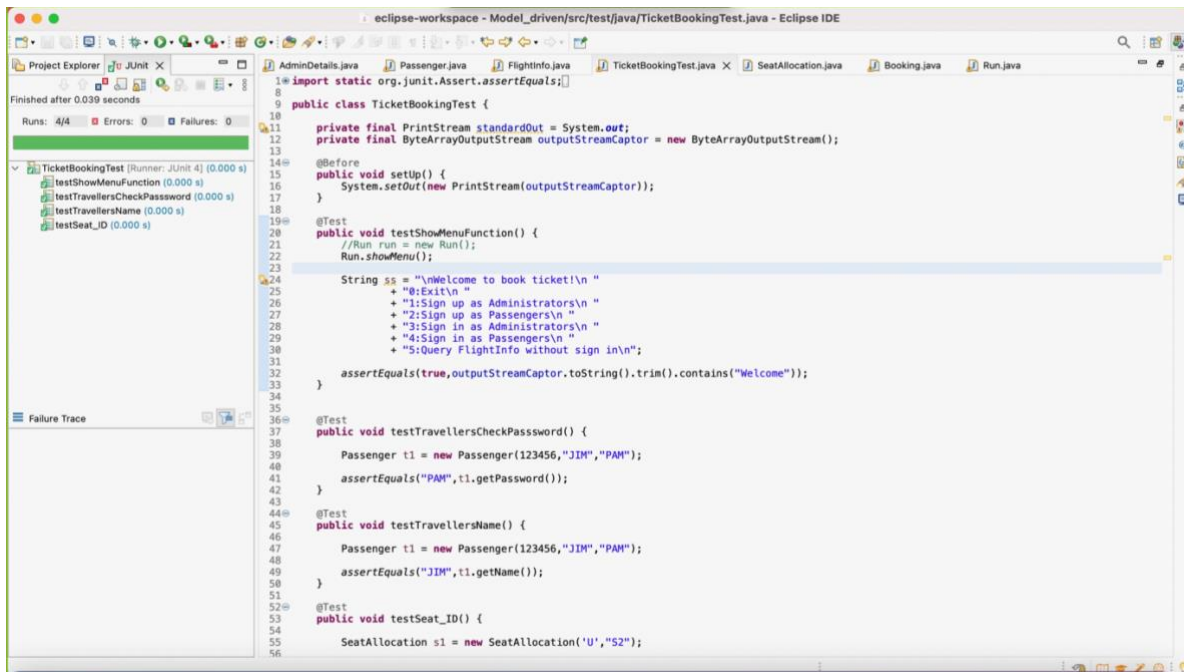
The statements for OCL constraints were commented in the attached source code file of deliverable 3.

Testing of the System

Test cases:

The test cases implemented were attached in the source code file of deliverable 4.

The screen capture of test cases is shown below.



References

- <https://maven.apache.org>
- <https://www.lucidchart.com/pages/uml-class-diagram>
- <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-state-machine-diagram/>
- <https://www.omg.org/spec/OCL/2.0/PDF>
- <https://www.omg.org/spec/OCL/ISO/19507/PDF>
- <https://www.aircanada.com/ca/en/aco/home.html>
- <https://www.softwaretestinghelp.com/junit-tests-examples/>