VANIER COLLEGE

Data Structures and Object Oriented Programming

420-SF2-RE sec.00002

Title:

Airline Reservation System

Submitted by:

Thiago Okamura

Presented to:

Yi Wang

Date of Submission: Sunday, May 11th, 2025

Table of Contents

Project Description:

This project will operate in the context of an airline booking system where customers can search and book airline tickets, as well as view, modify, or cancel existing airline tickets. The airline staff (admin) will be able to handle booking requests, confirmations, keep passenger records, generate transaction reports, and manage flight delays/cancellations. This project aims to simplify manual processing and provide an efficient tool for both airline booking staff and customers.

In my project, I plan to give users the ability to:

- Search for available flights
- Book and cancel tickets
- Modify bookings (e.g., change the date)

The staff will be able to:

- Handle booking requests and confirmations
- Keep passenger records
- Generate transaction reports
- Manage flight delays/cancellations

I plan to use 2 hierarchies in my project. The first being the User class. Both customer and admin classes will extend from the user class. The second hierarchy is the flight class. Both domesticFlight class and InternationalFlight class will extend from the flight class.

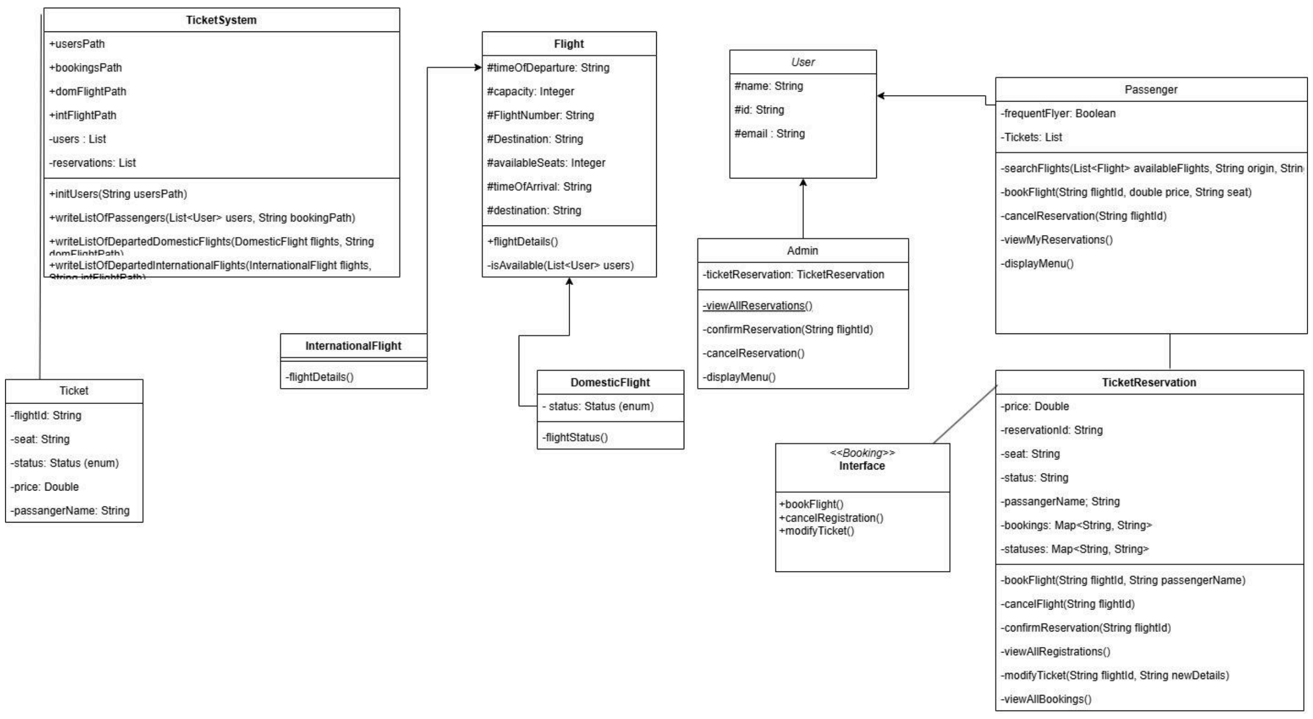
I will need a Booking interface that will be implemented by the TicketReservation class, for the booking, cancelling and modifying tickets methods.

Runtime-polimorphism will be used in the bookFlight, cancelFlight, and modifyTicket methods for overriding from the Booking interface.

The TextIO will be implemented in the TicketSystem class in order to read a file containing user information and to write all the booking confirmations into a CSV file.

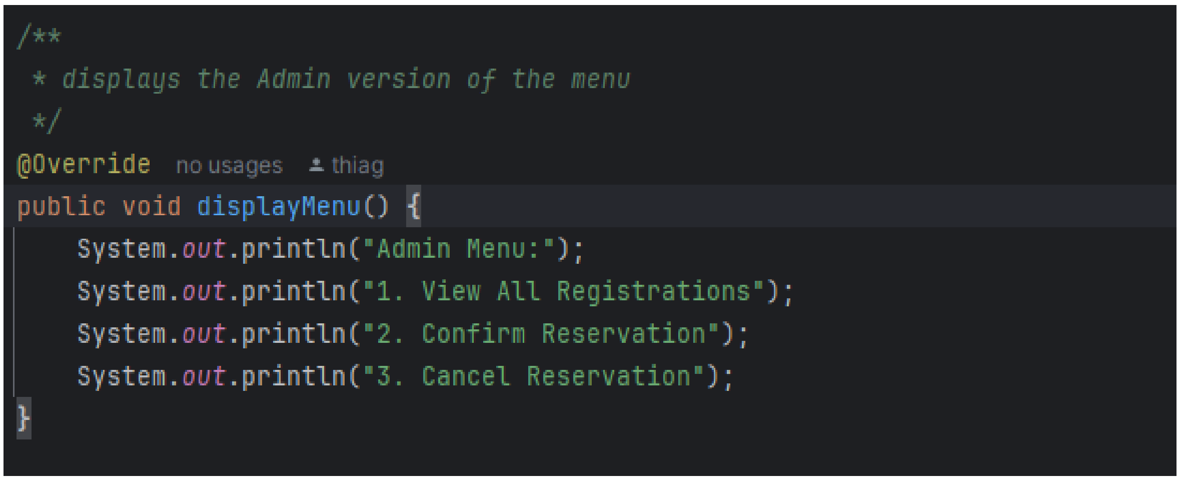
The comparable will be implemented in the Flight class to allow sorting based on departure time. The comparator will be implemented in the Passenger class to allow sorting based on name or passenger id.

I plan to implement the User, Admin, and Passenger, and the ticketReservation classes with methods implemented.

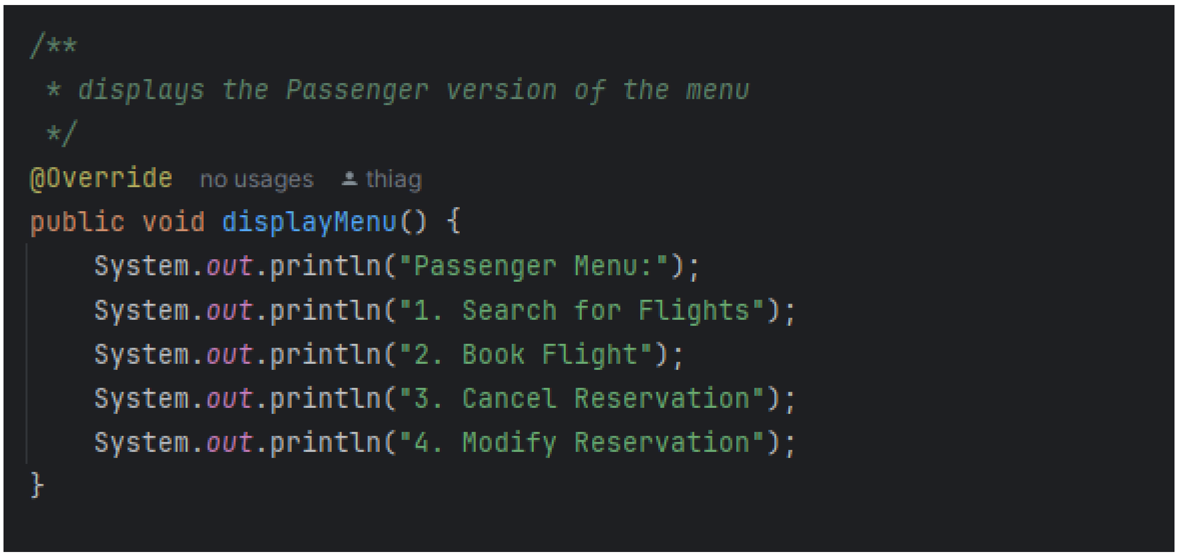


Project Features:

My project allows for a user to search, reserve, cancel and book airline tickets through the use of an interface implemented in the “User” class, and overridden in the “Passenger” class. In addition, it also allows for staff to manage passenger tickets through the use of the same interface implemented in the “User” class, which is also overridden in the “Admin” class.

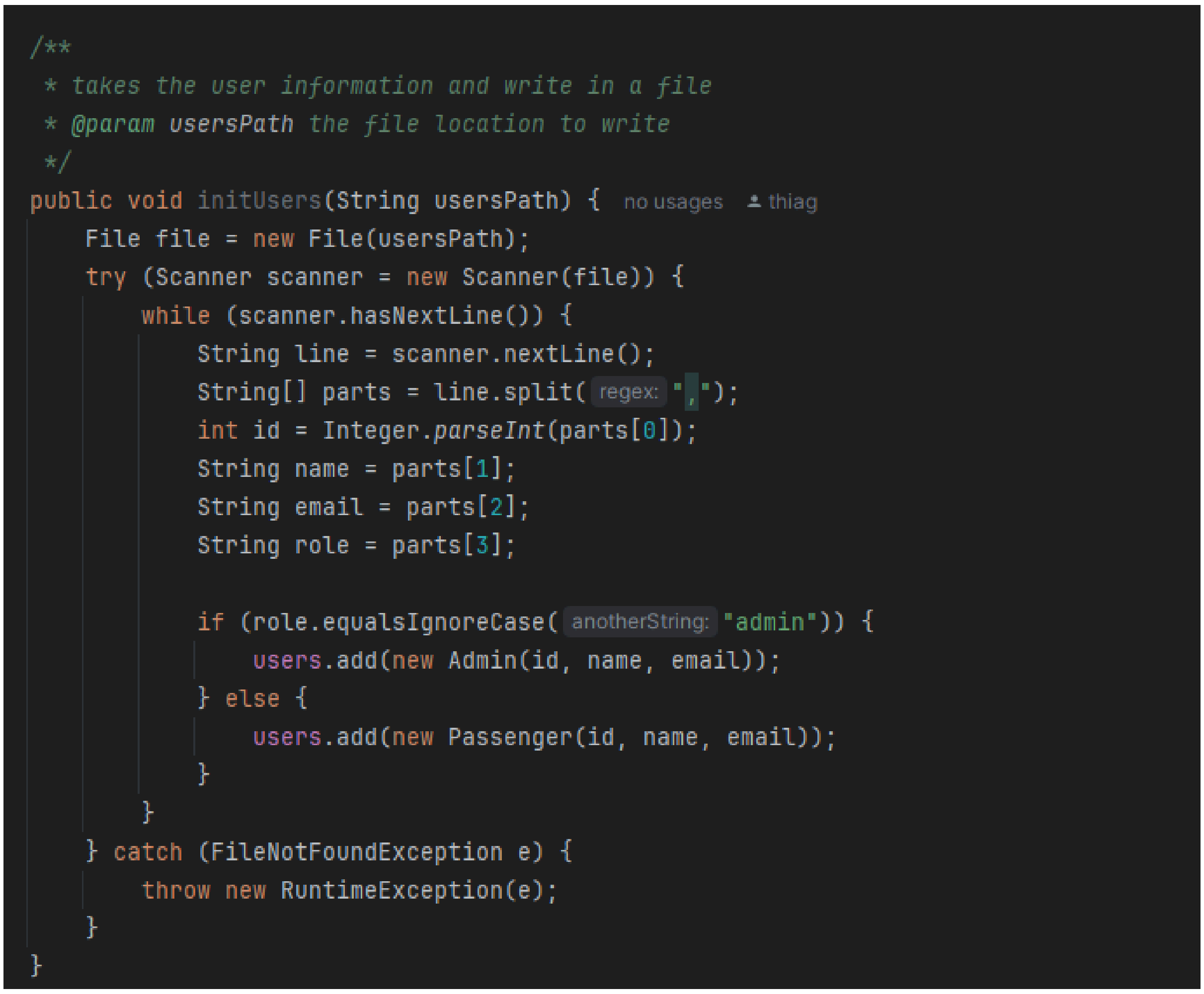


```java
/**
 * displays the Admin version of the menu
 */
@Override  no usages  thiag
public void displayMenu() {
    System.out.println("Admin Menu:");
    System.out.println("1. View All Registrations");
    System.out.println("2. Confirm Reservation");
    System.out.println("3. Cancel Reservation");
}
```



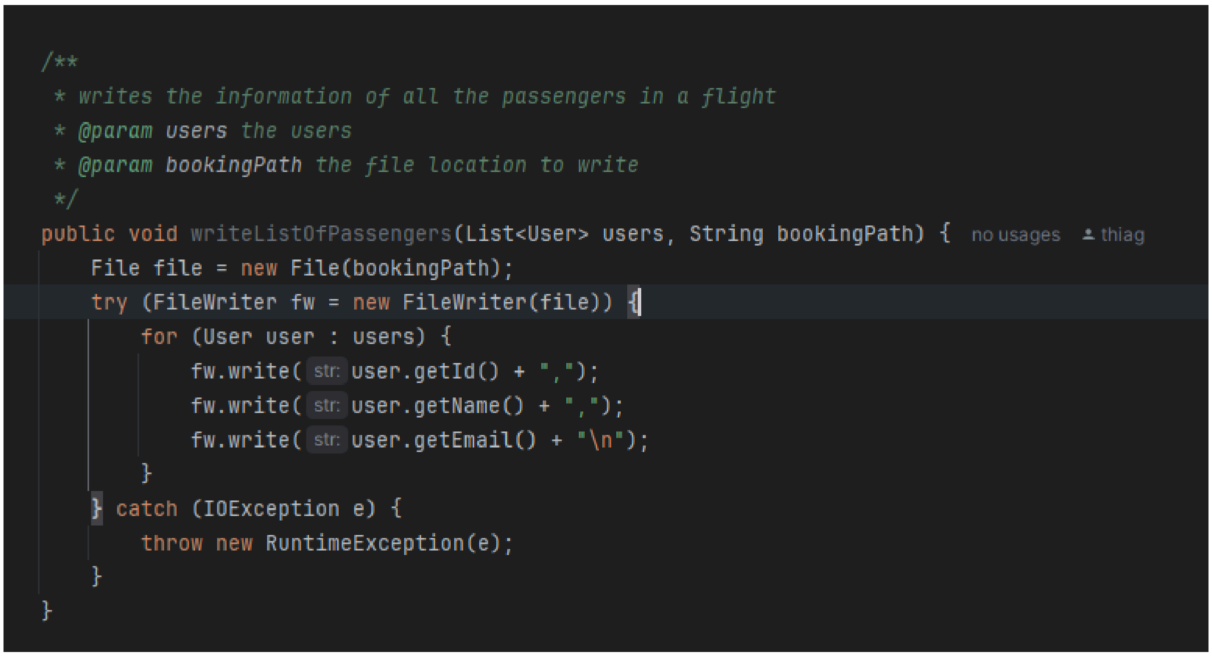
```java
/**
 * displays the Passenger version of the menu
 */
@Override  no usages  thiag
public void displayMenu() {
    System.out.println("Passenger Menu:");
    System.out.println("1. Search for Flights");
    System.out.println("2. Book Flight");
    System.out.println("3. Cancel Reservation");
    System.out.println("4. Modify Reservation");
}
```

I also decided to implement a way to receive user information through textIO using a scanner. Also, to create a list of passengers in a flight and to write two lists of departed flights, domestic flights and international flights, all using file writer to csv files.

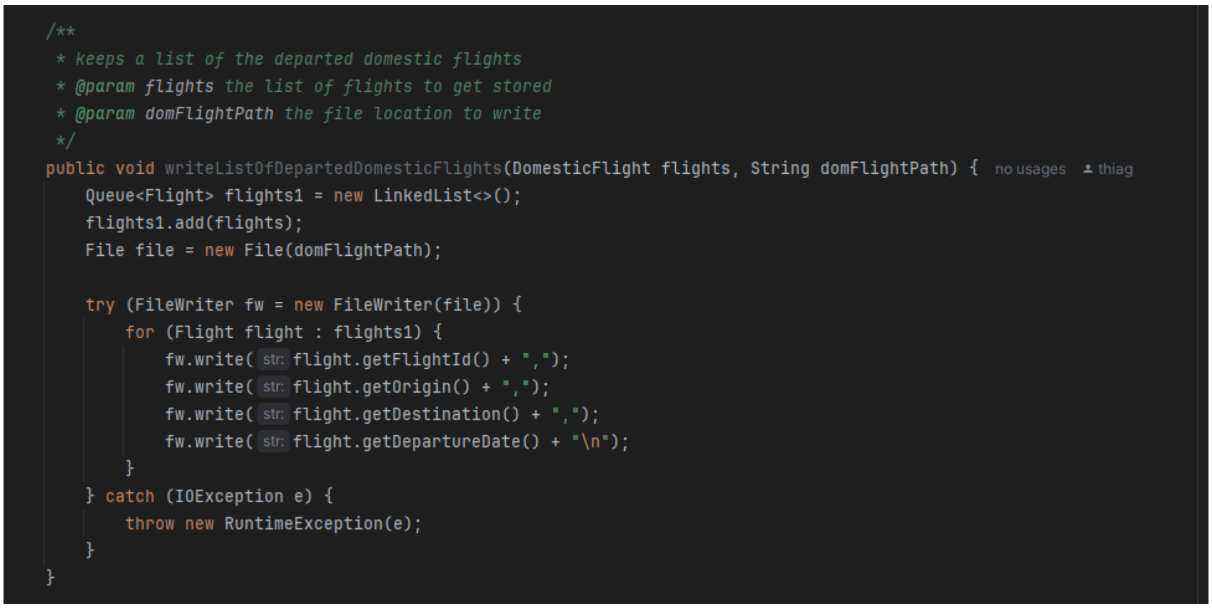


```java
/**
 * takes the user information and write in a file
 * @param usersPath the file location to write
 */
public void initUsers(String usersPath) {  no usages  thiag
    File file = new File(usersPath);
    try (Scanner scanner = new Scanner(file)) {
        while (scanner.hasNextLine()) {
            String line = scanner.nextLine();
            String[] parts = line.split( regex: ",");
            int id = Integer.parseInt(parts[0]);
            String name = parts[1];
            String email = parts[2];
            String role = parts[3];

            if (role.equalsIgnoreCase( anotherString: "admin")) {
                users.add(new Admin(id, name, email));
            } else {
                users.add(new Passenger(id, name, email));
            }
        }
    } catch (FileNotFoundException e) {
        throw new RuntimeException(e);
    }
}
```

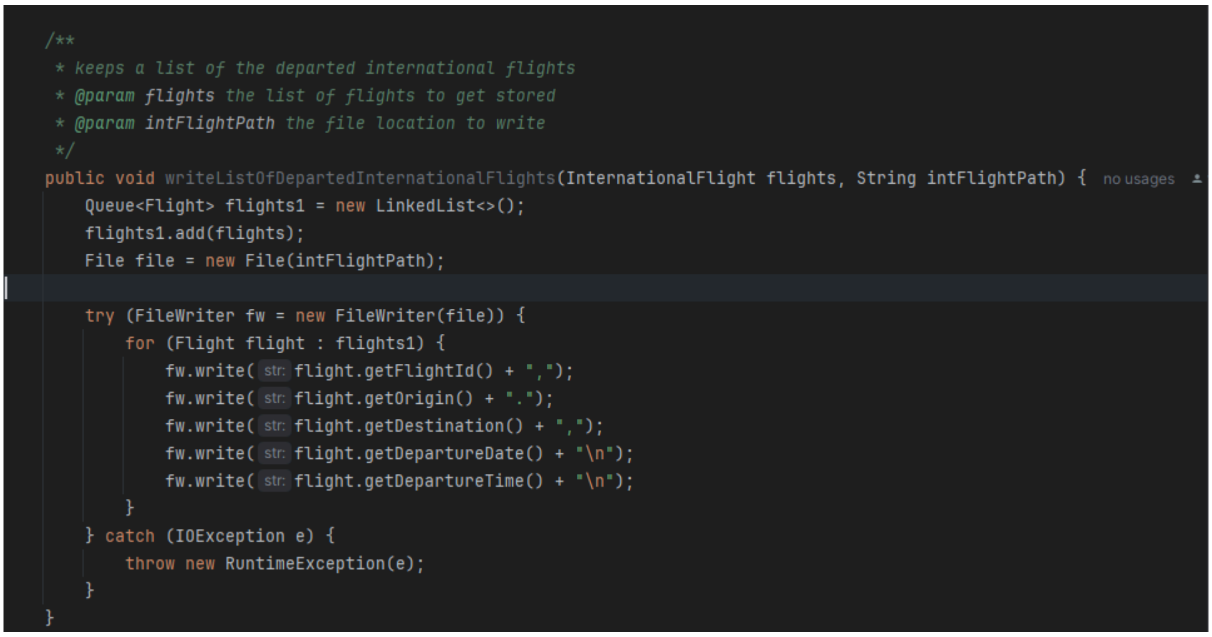


```java
/**
 * writes the information of all the passengers in a flight
 * @param users the users
 * @param bookingPath the file location to write
 */
public void writeListOfPassengers(List<User> users, String bookingPath) {  no usages  thiag
    File file = new File(bookingPath);
    try (FileWriter fw = new FileWriter(file)) {
        for (User user : users) {
            fw.write( str: user.getId() + ",");
            fw.write( str: user.getName() + ",");
            fw.write( str: user.getEmail() + "\n");
        }
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```



```java
/**
 * keeps a list of the departed domestic flights
 * @param flights the list of flights to get stored
 * @param domFlightPath the file location to write
 */
public void writeListOfDepartedDomesticFlights(DomesticFlight flights, String domFlightPath) {  no usages  thiag
    Queue<Flight> flights1 = new LinkedList<>();
    flights1.add(flights);
    File file = new File(domFlightPath);

    try (FileWriter fw = new FileWriter(file)) {
        for (Flight flight : flights1) {
            fw.write( str: flight.getFlightId() + ",");
            fw.write( str: flight.getOrigin() + ",");
            fw.write( str: flight.getDestination() + ",");
            fw.write( str: flight.getDepartureDate() + "\n");
        }
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```



```java
/**
 * keeps a list of the departed international flights
 * @param flights the list of flights to get stored
 * @param intFlightPath the file location to write
 */
public void writeListOfDepartedInternationalFlights(InternationalFlight flights, String intFlightPath) {  no usages
    Queue<Flight> flights1 = new LinkedList<>();
    flights1.add(flights);
    File file = new File(intFlightPath);

    try (FileWriter fw = new FileWriter(file)) {
        for (Flight flight : flights1) {
            fw.write( str: flight.getFlightId() + ",");
            fw.write( str: flight.getOrigin() + ".");
            fw.write( str: flight.getDestination() + ",");
            fw.write( str: flight.getDepartureDate() + "\n");
            fw.write( str: flight.getDepartureTime() + "\n");
        }
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Challenges:

I faced some challenges during the project. First, I had never used Git before and I spent a lot of time trying to understand it and its features. Some other challenges were all the requirements my project was required to cover, such as, comparable and comparator, textIO, interface, etc. At the end, I could not find a way to use an interface in my project, and I ended up not using one. In addition, my initial design did not work, I had a lot of modifications to make in deliverables 1 and 2. This led me to some time loss and frustration. Lastly, I also had trouble with the deadlines. I think they were really fair and I had a very reasonable amount of time to submit each deliverable. The problem was that I had a lot of other assignments and tests to submit/study for.

Learning Outcomes:

During the project, I learned a lot about GitHub, which will be very useful in the future. At the beginning, I knew nothing about it, but now I feel a lot more confident using it. I also learned that the most important and time consuming part of a project is planning and designing it. Not planning properly can lead to a lot of time waste and future problems during coding, so in future projects, I will definitely take more time to make sure I have a good design and plan before diving into coding.