BIRMINGHAM CITY
University

Flight Booking System (Report)

Authors: Bibek Sapkota & Solomon Silwal

Student ID: 23189618 & 23189650

Date: 23 Jun 2024

Module: Object Oriented Programming

Module Code: SUNB S2

Page Count: 17 pages

Table of Contents

Introduction

Flight Booking System

The Flight Booking System is a software solution that streamlines and automates the management of flights, bookings, and customers. This system is designed to make booking and organising flights simple, guaranteeing a smooth experience for clients and raising their level of satisfaction at every turn.

Technologies Used

The Flight Booking System is developed in Java, and object-oriented design techniques are used to efficiently organise and modularize the code. It ensures a visually appealing and interactive user experience by developing a graphical user interface using the JavaFX framework. To confirm the functionality of its developed features, the project also incorporates the JUnit testing framework.

Project Scope

The Flight Booking System will be used to oversee customer reservations, flight operations, and financial transactions for our hypothetical airline. Customers will be able to make reservations, look up flights, and get real-time booking updates with it. To provide a productive and user-friendly platform for all parties concerned, administrators will have access to tools for flight management, pricing modifications, and customer support.

Project Structure and Package Organization

Command Package:

The system includes classes for handling user input and executing commands. These commands correspond to actions that users can perform within the system, such as making travel arrangements or modifying existing ones. It features classes like AddFlightCommand for adding new flights, BookFlightCommand for flight booking, DeleteFlightCommand for removing flights, and CommandParser for interpreting commands.

DataManager Package:

This package is tasked with managing and storing data. It comprises classes that interact with data storage, retrieve information, and update the system's status. Examples include CustomerDataManager and BookingDataManager.

GUIPackage:

The classes in this package manage the graphical user interface (GUI). It controls the presentation layer, which gives users an intuitive and aesthetically pleasing interface through which to interact with the system. MainWindow, CancelBookingPane, and IssueBookingWindow are a few of these classes.

MainPackage:

This component acts as the application's launchpad and often contains the primary class that starts the program's execution. i.e. Main.java

ModelPackage:

This package contains the fundamental data structures and business logic essential to the Flight Booking System. It encompasses classes that represent key entities such as flights, customers, bookings, and FlightBookingSystem.

TestPackage:

JUnit test classes are included in this section to ensure that the system's component parts work as intended. Verifying that every component of the system functions as intended requires testing. FlightTest, BookingTest, and CustomerTest are a few of these tests.

Command Packages

A group of classes that implement the Command design pattern make up the Flight Booking System Project's Command Package. This package is essential for mediating the communication between the application's main business logic and user interface. This package contains commands that represent various operations and features that users can perform within the system.

Command class:

The flight booking system's action organization relies heavily on the `Command` interface. By creating a single `execute` method for all tasks, including listing flights, adding customers, and handling bookings, it ensures consistency and makes system growth easier. By encapsulating command logic and maintaining modularity, this solution effortlessly integrates with the core `FlightBookingSystem}. Additionally, the `HELP_MESSAGE} field improves user interaction by offering clear usage instructions and command references.

Flight Commands

AddFlight:

With the help of the `AddFlight` command, users can quickly add new flights to the system by compiling the necessary information and incorporating it into the flight booking infrastructure. This makes sure that the system manages flight-related data effectively.

ListFlights:

The `ListFlights` command executes within a flight booking system, invoking the `listAllFlights` method of `FlightBookingSystem`. This method retrieves and displays a list of all flights currently registered in the system, providing users with comprehensive visibility of available flight options.

ShowSpecificFlight:

The `ShowFlight` command fetches and presents detailed information about a specific flight identified by its ID in the booking system. It includes key details such as flight number, origin, destination, and departure date. If passengers are booked, it also lists their names, contact details, and booked class, providing users with a clear overview of the flight's details and passenger manifest.

DeleteFlight:

The `DeleteFlight` command is responsible for removing a flight from the system based on its ID. It verifies the existence of the flight and proceeds to delete it, providing a confirmation

message upon successful deletion. This functionality ensures efficient management of flight records within the booking system.

Booking Commands

AddBooking:

The `AddBooking` command facilitates the addition of a customer's booking for a specific flight. By encapsulating the customer ID, flight ID, booking date, and flight class, it implements the `Command` interface. Once it runs, it checks for things like flight capacity and availability, makes a new reservation, adds it to the system, and makes sure the data is persistent. In order to provide dependable integration into the system's command structure, error handling controls exceptions during data operations.

ListBooking:

The `ListBooking` command retrieves and displays all bookings stored in the Flight Booking System by calling `flightBookingSystem.listAllBookings`. This command provides a concise way to access and view detailed booking information within the system.

UpdateBooking:

The `UpdateBooking` command modifies a booking by verifying and retrieving the customer and flight using their IDs, updating the booking's details if new values are provided, and storing the changes. Any `IOException` results in an error message.

CancelBooking:

The `CancelBooking` command cancels a booking for a customer on a specific flight by verifying their IDs, preventing cancellation if the booking is completed, calculating the cancellation fee, updating records, and providing feedback on success or errors.

Customer Commands

AddCustomer:

The `AddCustomer` command creates and adds a new customer to the system with provided details (name, phone, email), generating a unique customer ID and handling IO exceptions during data storage.

ShowSpecificCustomer:

The `ShowCustomer` command displays a customer's details such as ID, name, phone, and email, along with their bookings, using a specified customer ID. Error handling is included for cases where the customer ID is not found in the system.

DeleteCustomer:

The `DeleteCustomer` command removes a customer from the flight booking system based on a specified customer ID. It checks if the customer exists before deleting and throws an exception if the customer ID is not found. After successful deletion, it prints a message confirming the removal of the customer.

ListCustomer:

The `ListCustomers` command retrieves and displays a list of customers from the flight booking system. It first checks if the list of customers is empty. If it is, it prints "No customers found."; otherwise, it lists all available customers with their details.

DataManger

The `DataManager` interface outlines essential methods (`loadData` and `storeData`) and a constant (`SEPARATOR`) for managing data within the Flight Booking System. It facilitates loading data into the system and storing data from it, while ensuring consistent data formatting and error handling, particularly for file operations and system-specific exceptions.

```
public interface DataManager {

    public static final String SEPARATOR = "::";

    public void loadData(FlightBookingSystem fbs) throws IOException,
FlightBookingSystemException;
    public void storeData(FlightBookingSystem fbs) throws IOException;

}
```

FlightBookingSystemDataManager

The `FlightBookingSystemData` class organizes `DataManager` instances (`FlightDataManager`, `CustomerDataManager`, `BookingDataManager`) to manage data operations within the Flight Booking System. It initializes these managers and provides methods `load()` and `store()` to respectively load data into and store data from a `FlightBookingSystem` instance, ensuring efficient data management throughout the application.

```
 9 public class FlightBookingSystemData {
10
11     private static final List<DataManager> dataManagers = new ArrayList<>();
12
13     static {
14         dataManagers.add(new FlightDataManager());
15          dataManagers.add(new CustomerDataManager());
16          dataManagers.add(new BookingDataManager());
17     }
18     public static FlightBookingSystem load() throws FlightBookingSystemException, IOException {
19
20         FlightBookingSystem fbs = new FlightBookingSystem();
21         for (DataManager dm : dataManagers) {
22             dm.loadData(fbs);
23         }
24         return fbs;
25     }
26     public static void store(FlightBookingSystem fbs) throws IOException {
27
28         for (DataManager dm : dataManagers) {
29             dm.storeData(fbs);
30         }
31     }
32 }
```

FlightDataManager, CustomerDataManager, BookingDataManager

The `FlightDataManager` class implements `DataManager` to handle loading and storing flight data from/to `flights.txt`. It parses flight details during loading, manages exceptions, and formats data for storage while maintaining integrity and formatting conventions (`SEPARATOR` and date format).

The `CustomerDataManager` class manages customer data loading and storing to/from `customers.txt`, using a defined separator (`SEPARATOR`). It parses customer details during loading and formats them for storage, ensuring data integrity and handling exceptions like `IOException` and `FlightBookingSystemException`.

The `BookingDataManager` class manages booking data for `bookings.txt`, parsing and formatting entries using `SEPARATOR`. It loads bookings into the system, validating associated customers and flights, and stores data back to the file, handling exceptions like `IOException` and `FlightBookingSystemException`.

All data is stored in a text file using "::" to separate booking properties, with methods for parsing and formatting. This ensures correct association with existing customers and flights during data loading, and addresses errors when references are absent.

Code:

```
public class FlightDataManager implements DataManager {

    private final String RESOURCE = "./resources/data/flights.txt";
    private final String SEPARATOR = "::"
  public void loadData(FlightBookingSystem fbs) throws IOException, FlightBookingSystemException {
    try (Scanner sc = new Scanner(new File(RESOURCE))) {
      int line_idx = 1;
      while (sc.hasNextLine()) {
        String line = sc.nextLine();
        String[] properties = line.split(SEPARATOR, -1);
        if (properties.length < 7) {
          throw new FlightBookingSystemException("Insufficient data on line " + line_idx);
        }
        try {
          int id = Integer.parseInt(properties[0]);
          String flightNumber = properties[1];
          String origin = properties[2];
          String destination = properties[3];
          LocalDate departureDate = LocalDate.parse(properties[4]);
          int capacity = Integer.parseInt(properties[5]);
          boolean isDeleted = Boolean.parseBoolean(properties[6]);
          double price = (properties.length >= 8 && !properties[7].isEmpty()) ? Double.parseDouble(properties[7]) : 0.0;

          if (!isDeleted) {
            Flight flight = new Flight(id, flightNumber, origin, destination, departureDate, capacity, isDeleted, price);
            fbs.addFlight(flight);
          }
        } catch (NumberFormatException ex) {
```

```
                throw new FlightBookingSystemException("Unable to parse flight id " + properties[0] + " on
line " + line_idx + "\nError: " + ex);
            }
            line_idx++;
        }
    }
}
```

The FlightDataManger, controls how data is loaded and saved into the system.
BookingDataManager and CustomerDataManger are implemented in a comparable way.

Model Package

The main data structures and business logic of the Flight Booking System are included in the model package. It acts as the central nervous system for all flight, customer, booking, and other entity management. The package is set up to encourage system maintainability, modularity, and clarity.

Flight Class

A flight's unique ID, flight number, origin, destination, departure date, capacity, cost, and whether it has been removed or not are all contained in the Flight Class. A hierarchical enumeration that defines different flying classes is also included.

Property

id: *Unique Id for each flight*

flightNumber: The flight number of the flight.
origin and destination: Origin and destination of flight (airport code).
departure Date: Scheduled date for the flight's departure.
capacity: Maximum passenger capacity of the flight.
price: Base price assigned to the flight.
isDeleted: Status of flight has been deleted.

Dynamic Pricing Calculation:

The dynamic pricing formula offers a dynamic approach to flight pricing for diffrtrnt class(i.e First, Business, Economy) by guaranteeing flexibility to various criteria. This is how the formula is expressed:

Let P(f) denote the dynamic price calculation for a given flight class f.

If the flight has already departed:

$$P(f)=\text{getPrice}(f)$$

If the flight has not departed:

$$P(f)=\text{getPrice}(f)\times\text{daysPriceFactor}\times\text{classPriceFactor}\times\text{bookedSeatsFactor}$$

Where:

- daysPriceFactor=1.0+0.1×(maxDaysForPriceFactor−daysLeft) if daysLeft≤maxDaysForPriceFactors; otherwise daysPriceFactor=1.0
- classPriceFactor=classPrices[f]
- bookedSeatsFactor=1.0+0.1×(getBookedSeats()/getCapacity())

Passenger

The Flight class keeps track of a collection of passengers, which includes individuals who have reserved seats on the flight.

Passenger Management:

The Flight class provides essential methods for managing passengers, allowing the addition and removal of passengers as well as determining if the flight is fully booked.

Deletion Status and Departure:

The class monitors whether the flight has been marked for deletion and determines its departure status by comparing the scheduled departure date with the current date.

Customer Class

The Customer class in the Flight Booking System is very important since it represents customers and has vital data about them. Information such as the customer ID, name, phone number, email address, booking history, and a flag designating if the customer has been designated for deletion are all stored in this class.

Property

id: Unique Id for each customer.
Name: Name of Customer
PhoneNumber: Number of Customer
Email: Email address of customer

Booking History

The Customer class manages a list of bookings, serving as a complete record of a customer's flight reservations.

Customer Operations:

- Adding and Removing Booking
- Booking Details
- Adding Booking
- Removing Booking

Booking Class

To manage the individual bookings that consumers make for particular flights, the Booking class is essential to the Flight Booking System. Important information such as the flight details, client information, booking date, cancellation status, and flight class are stored. Along with handling these tasks, the class also calculates payments, verifies booking completeness, and updates or cancels reservations.

Property

- **Customer and Flight Information**
- **Cancellation and Deletion**

- **Unique Booking Id**
- **Flight Class**

FlightBookingSystem

The Flight Booking System application's central component, the FlightBookingSystem class, is in charge of managing reservations, flights, and customers. It manages basic functions such as adding, retrieving, and communicating with these entities. Maintaining the current system date and tracking the most recent client and booking IDs are important aspects. It arranges Customer, Flight, and Booking objects using TreeMaps and provides read-only lists for external access.

Main Package

The flight booking system's main package serves as the application's entry point and includes all of the necessary features. Usually, it consists of the main class, which initiates system calls, controls user behavior, and arranges communication between other parts.

Main class:

The Flight Booking System application's entry point, the `Main` class, initializes the system, processes user input, and runs commands. When it exits, the changed data is saved back to a file. It loads the current system data and offers a command-line interface for human input. In an infinite loop, the `main` method accepts commands from the user, parses them into `Command` objects, runs these commands, and deals with any exceptions. It stores the system state and ends the program when it receives the "exit" instruction.

FlightBookingSystemException

Extending from `Exception`, the `FlightBookingSystemException` class acts as a customized exception to report problems or incorrect commands in the flight booking system. It has a constructor that takes a detail message, which the `getMessage()` method can subsequently retrieve.

```
public class FlightBookingSystemException extends Exception {

    public FlightBookingSystemException(String message) {
        super(message);
    }
}
```

Command Parser

The flight booking system's `CommandParser` class performs tasks like adding flights, customers, and bookings by interpreting user commands from the command line interface. To construct equivalent `Command` objects, it reads user inputs through methods like `parseAddFlight`, `parseAddCustomer`, and `parseAddBooking`. Error management guarantees appropriate input error handling and validation, improving the system's functionality and dependability. The flight booking system's basic operations are seamlessly integrated with this modular design, which streamlines command processing.

GUI PACKAGE

The GUI package, which manages the graphical user interface (GUI) features, is an important part of the Flight Booking System application. It includes classes and functions devoted to information presentation and user interaction facilitation.

MainWindow

AFlight Booking Management System's main graphical interface is provided by the `MainWindow` class. It extends {JFrame} and uses Swing elements like buttons, panels, and dialogs to handle user interactions. Adapting dynamically to user activities, the interface makes tasks like viewing flights, managing bookings, and assisting customers easier. It offers a strong user experience for flight management chores by interacting with the backend through `FlightBookingSystem} in a seamless manner. This allows for features like adding flights, modifying reservations, and customizing system settings.

Side panel

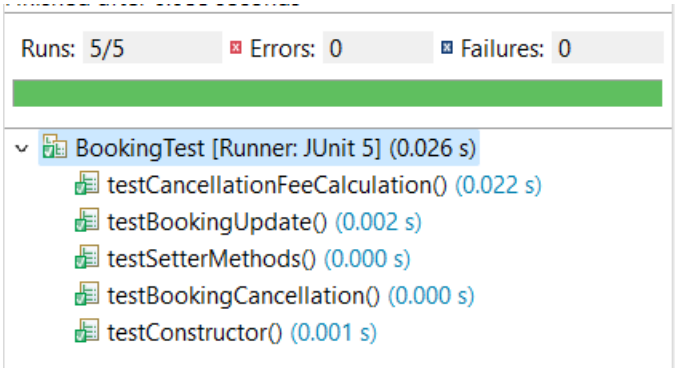
The side panel in `MainWindow` enables navigation through menu buttons like "Home", "Flights", "Bookings", "Customers", and "Settings". Clicking these buttons updates the center panel to display relevant content, facilitating easy management of different aspects of the Flight Booking Management System from a unified interface.

DisplayPanel

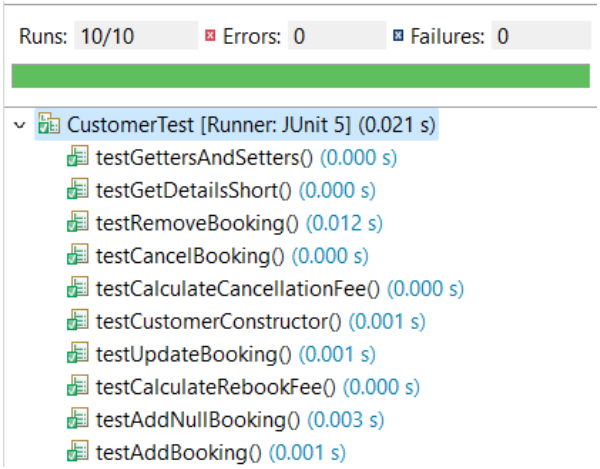
Based on user activities, the display panel in the MainWindow of the Flight Booking Management System dynamically refreshes to show detailed information. It is essential for displaying information such as reservations, flights, and customer details as users go between several menu selections, such as "Bookings" and "Flights".

Testing

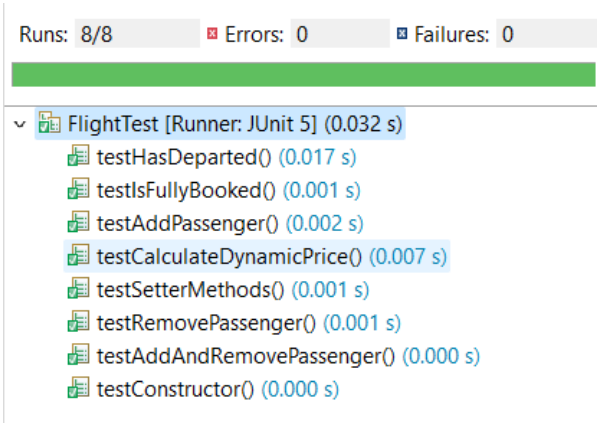
BookingTest:



CustomerTest:



FlightTest:



Outputs

```
Flight Booking System
Enter 'help' to see a list of available commands.
> help
Commands:
        listflights                             print all flights
        listcustomers                           print all customers
        listbookings                            print all bookings
        addflight                               add a new flight
        addcustomer                             add a new customer
        showflight [flight id]                  show flight details
        showcustomer [customer id]              show customer details
        deletecustomer [customer id]                    delete a customer details
        deleteflight [flight id]                delete a Flight
        addbooking [customer id] [flight id]    add a new booking
        cancelbooking [customer id] [flight id] cancel a booking
        editbooking [booking id] [flight id]    update a booking
        loadgui                                 loads the GUI version of the app
        help                                    prints this help message
        exit                                    exits the program
>
```

1) ListFlights

```
> listflights
Flight #  Departure Date          Origin            Destination       First Class       Business Class     Economy Class
1         2024-10-23              KTM               QTR               30000.00          18000.00           10000.00
2         2025-06-23              KTM               PKR               300000.00         180000.00          100000.00
3         2024-07-23              QTR               KTM               45000.00          27000.00           15000.00
4         2024-06-22              UK                US                18000.00          10800.00           6000.00
>
```

2) ListCustomers

```
> listcustomers
List of Customers:
Customer ID    Name            Phone          Email
1              Solomon Silwal  9869374141     Solomon@gmail.com
2              Bibek Sapkota   9863027811     bibek@gmail.com
>
```

3) ListBookings

```
> listbookings
Booking ID     Customer                Flight              Booking Date            Flight Class
1              Customer 1 - Solomon SilwalFlight 1 - NPL1234  2024-06-23              ECONOMY_CLASS
3              Customer 2 - Bibek SapkotaFlight 3 - NPL2468  2024-06-23              FIRST_CLASS
>
```

4) Addflight

```
> addflight
Flight Number: A977
Origin: KTM
Destination: UK
Departure Date ("YYYY-MM-DD" format): 2024-06-28
Capacity: 200
Price: 56000
Flight #5 added.
```

5) AddCustomer

```
> addcustomer
Name: Ram Kaji
Phone: 9855655555
Email: kaji123@gmail.com
Customer #3 added.
```

6) ShowFlight

```
> showflight 5
Flight ID: 5
Flight Number: A977
Origin: KTM
Destination: UK
Departure Date: 2024-06-28
No passengers for this flight.
```

7) ShowCustomer

```
> showcustomer 3
Customer ID: 3
Name: Ram Kaji
Phone: 9855655555
Email: kaji123@gmail.com
Bookings:
No bookings for this customer.
```

8) DeleteCustomer

```
> deletecustomer 3
Customer with ID 3 has been deleted successfully.
>
```

9) DeleteFlight

```
> deleteflight 5
Flight with ID 5 has been deleted successfully.
>
```

10) AddBooking

```
> addbooking 1 2
Flight Class (ECONOMY_CLASS/BUSINESS_CLASS/FIRST_CLASS): FIRST_CLASS
Booking for customer Solomon Silwal on flight KPL123 in class FIRST_CLASS added.
>
```

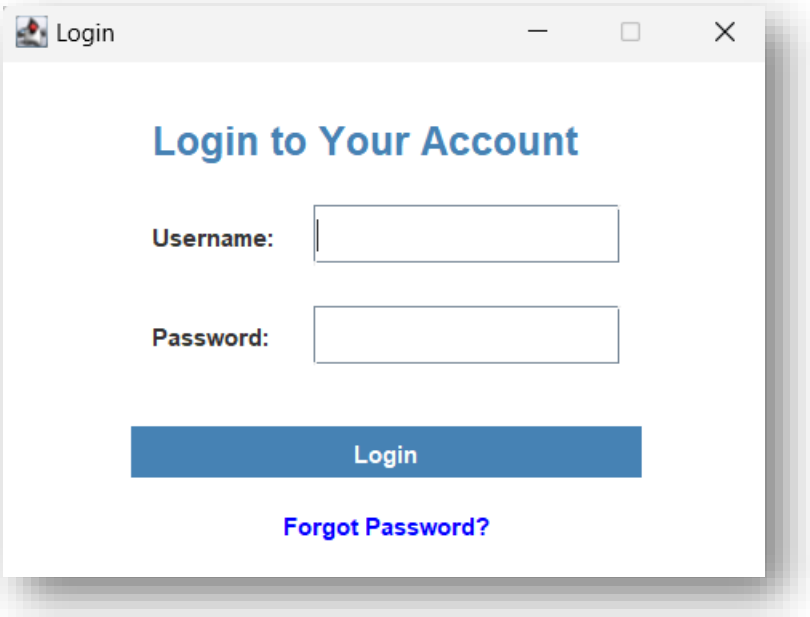
11) CancelBooking

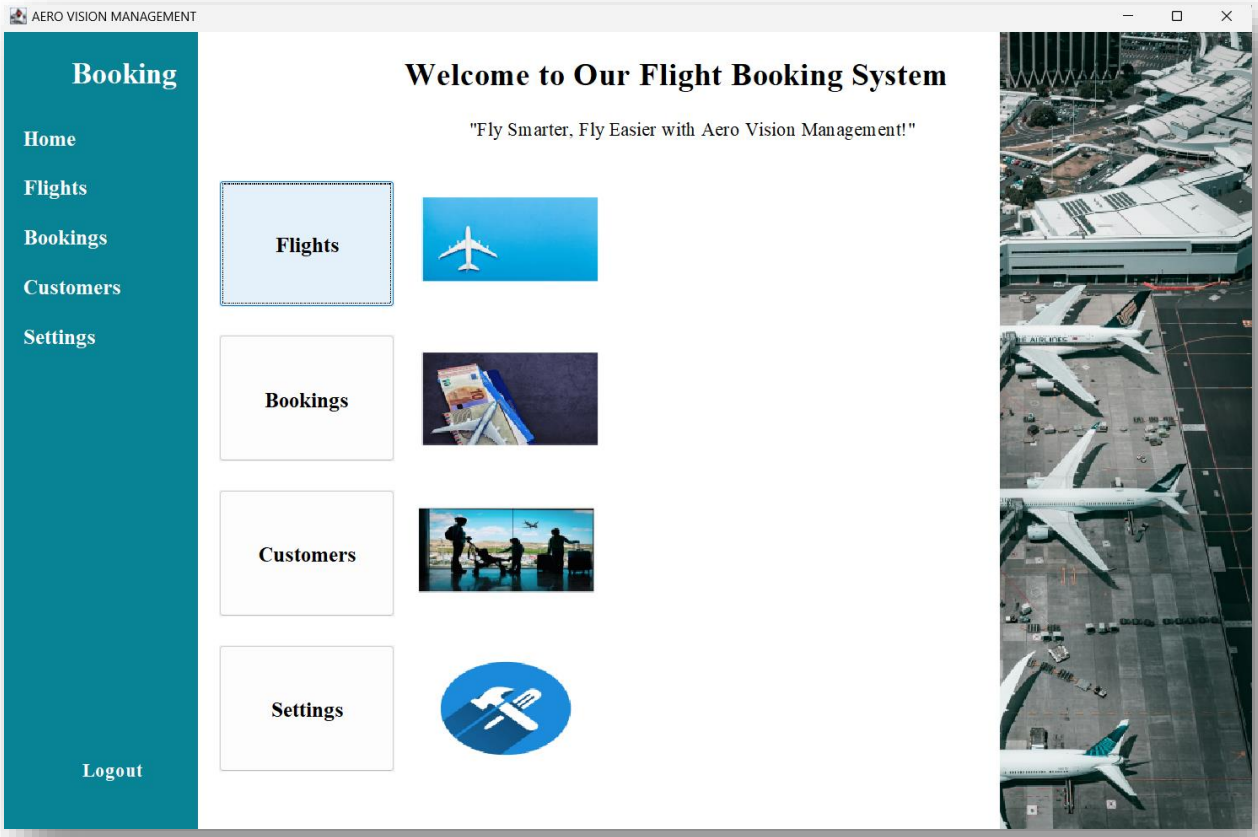
```
> cancelbooking 1 2
Booking for customer Solomon Silwal on flight KPL123 canceled.
>
```

12) EditBooking

```
> editbooking 1 1
Departure Date ("YYYY-MM-DD" format): 2024-06-23
Enter flight class (FIRST_CLASS, BUSINESS_CLASS, ECONOMY_CLASS): FIRST_CLASS
Booking for customer Solomon Silwal on flight NPL1234 updated.
>
```

13) LoadGui




AERO VISION MANAGEMENT
Booking
Home
Flights
Bookings
Customers
Settings
Logout
Welcome to Our Flight Booking System
"Fly Smarter, Fly Easier with Aero Vision Management!"
Flights
Bookings
Customers
Settings