Project Outline: Flight Booking System

1. Project Overview:

- The project implements a Flight Booking System with a graphical user interface (GUI) using Java Swing.
- It includes classes for managing flight schedules, bookings, and user interactions.

2. Class Structure:

Part 1: MainMenuProject and MainMenuHandler

MainMenuProject:

- Displays the main menu of the application.
- Allows the user to choose between viewing flight schedules and exiting the program.

MainMenuHandler:

- Handles user interactions in the main menu, directing to appropriate functionalities.

Part 2: FlightSchedule and FlightScheduleHandler

FlightSchedule:

- Represents a flight schedule with details such as departure and destination, airline, date, time, ticket price, stops, and flight ID.
- Implements Serializable for object serialization.
- Provides methods for setting and getting attributes, and validation.

FlightScheduleHandler:

- Manages the creation, serialization, and deserialization of FlightSchedule objects.
- Provides methods for reading and writing flight schedules to a file.

Part 3: FlightScheduleGui and Booking

FlightScheduleGui:

- Extends JFrame and displays flight schedules using Swing components.

- Reads flight schedules from a file and allows users to book a flight.

Booking:

- Represents a booking with details like contact information, traveler details, ticket class, booking number, and associated flight.
- Implements Serializable for object serialization.
- Provides methods for setting and getting attributes and writing bookings to a file.

BookingGui:

- Extends JFrame and provides a GUI for users to enter booking details, select ticket class, and confirm the booking.

Part 4: MainPage

MainPage:

- The main class with the main method serving as the entry point for the application.
- Initializes sample flight schedules, writes them to a file, and demonstrates reading from the file.
- Starts the main menu GUI to begin user interactions.

3. Project Functionality:

- Users are presented with a main menu offering options to view flight schedules or exit program.
- Flight schedules are displayed in a GUI where users can choose a flight to book.
- The booking process involves entering contact details, traveler information, selecting a ticket class, and confirming the booking.
- Bookings are stored in a file for future reference.

4. Overall Flow:

- The program starts with the main menu, guiding users to view schedules or exit.
- Users can view available flight schedules, choose a flight, and proceed to book it.
- The booking process involves providing necessary details and confirming the booking.
- Flight schedules and bookings are stored persistently in separate files.

5. Key Features:

- Object serialization for storing and retrieving flight schedules and bookings.
- Graphical User Interface (GUI) for user-friendly interactions.
- File I/O for reading and writing data.
- Flight schedules include details like departure, destination, airline, date, time, price, and stops.
- Bookings include contact details, traveler information, ticket class, and associated flight details.

Work Distribution

The work was distributed based on page wise modules.

Main Page (Muhammad Nouman)

This part of the code sets up the registration page and related functionality. The system uses object serialization to store user data in a binary file. The UserDatabase class manages user-related operations. The MainPage class is the entry point for the system, displaying the main interface with options for registration, login, guest access, and admin login.

Main Menu Page (Muhammad Abdullah)

The code has 'About Us' section which displays information about the airline's owner, CEO, general manager, and country head. The 'Booking History' section, available for signed-in users, exhibits past booking records. The 'File a Complaint' functionality allows users to submit complaints by providing their name, email, and complaint details. Validations are in place to ensure proper input, and the complaints are serialized and stored in a binary file named "complaints.bin."

Admin Page (Abdullah Rizwan)

The system allows administrators to manage flight schedules, bookings, user complaints, and user information. The main graphical user interface (GUI) is implemented in the FlightManagementSystemAdminGUI class, featuring a background image and a set of buttons for various functionalities. Users can add, delete, and edit flight schedules, view complaints, bookings, flights, and user details. The code includes several auxiliary classes such as AddFlightFrame, DeleteFlightFrame, and EditFlightFrame, which are separate frames for adding, deleting, and editing flight schedules, respectively. The application utilizes object serialization for storing and retrieving flight schedules, complaints, bookings, and user information persistently.

Booking Flights (Hamza Saif) (Leader)

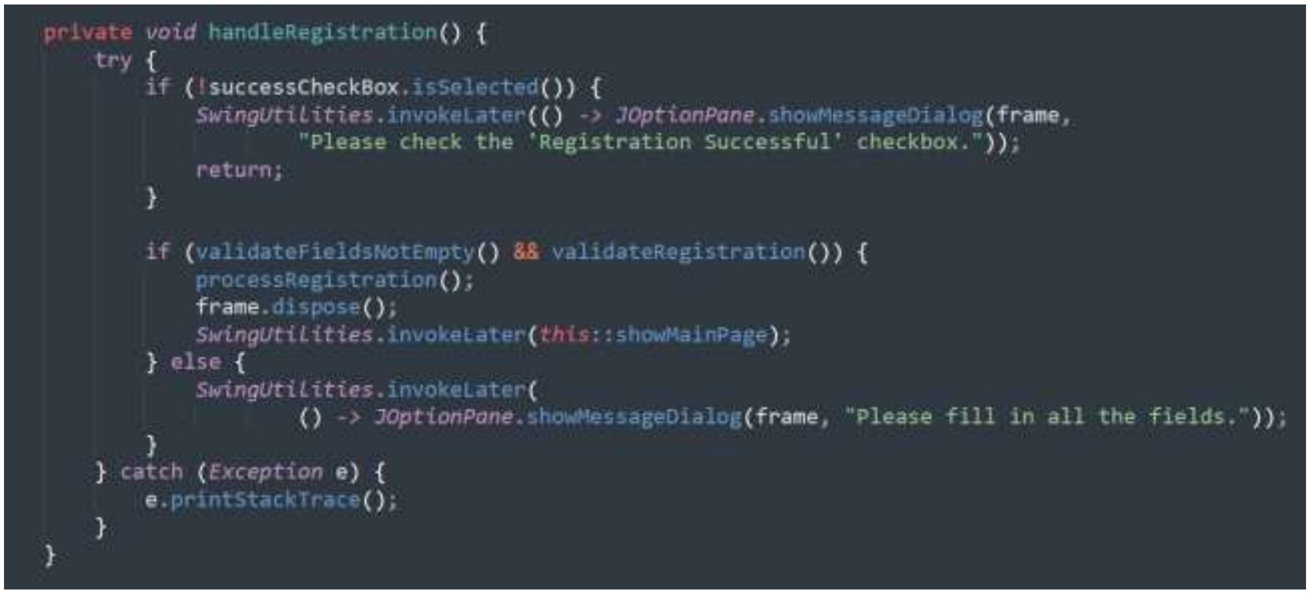
The code offers functionalities such as viewing flight schedules, Booking Flights based on schedule, checking booking history, and returning to the main page. The system considers users as either guests or signed-in users. Booking History is displayed to only to the signed in users. Guest users can’t view booking history.

Compilation of everyone’s part was done by Hamza.

Source Code

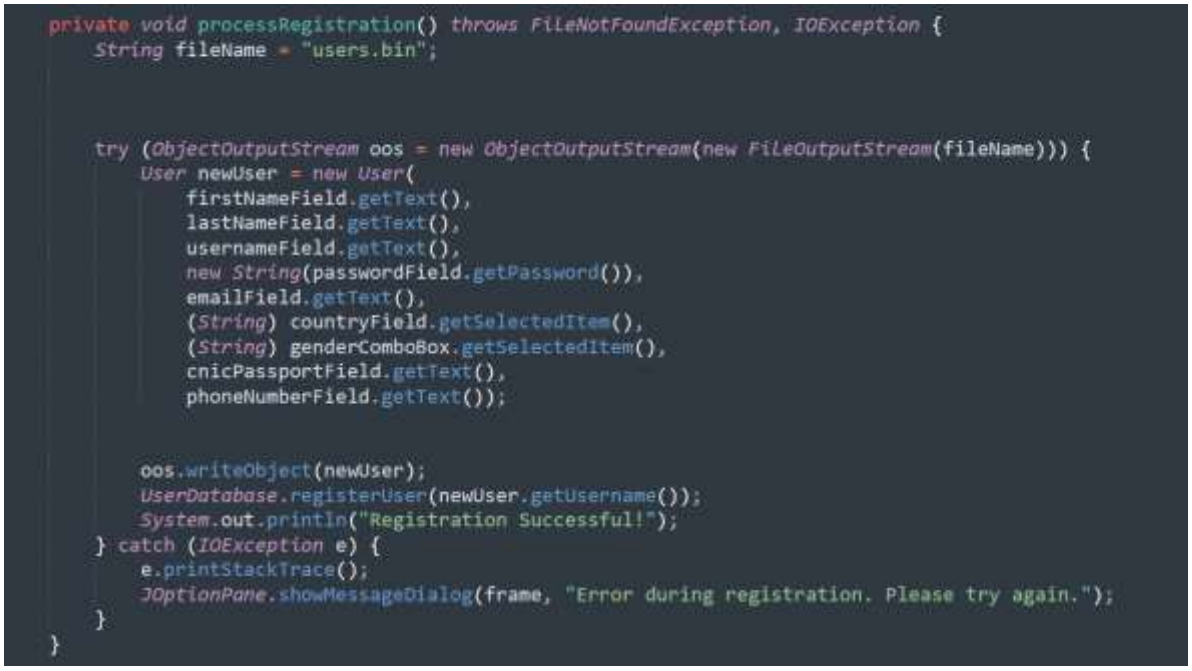
- ## MainPage Classes

  - **class RegisterPage**



```
private void handleRegistration() {
    try {
        if (!successCheckBox.isSelected()) {
            SwingUtilities.invokeLater(() -> JOptionPane.showMessageDialog(frame,
                    "Please check the 'Registration Successful' checkbox."));
            return;
        }

        if (validateFieldsNotEmpty() && validateRegistration()) {
            processRegistration();
            frame.dispose();
            SwingUtilities.invokeLater(this::showMainPage);
        } else {
            SwingUtilities.invokeLater(
                    () -> JOptionPane.showMessageDialog(frame, "Please fill in all the fields."));
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

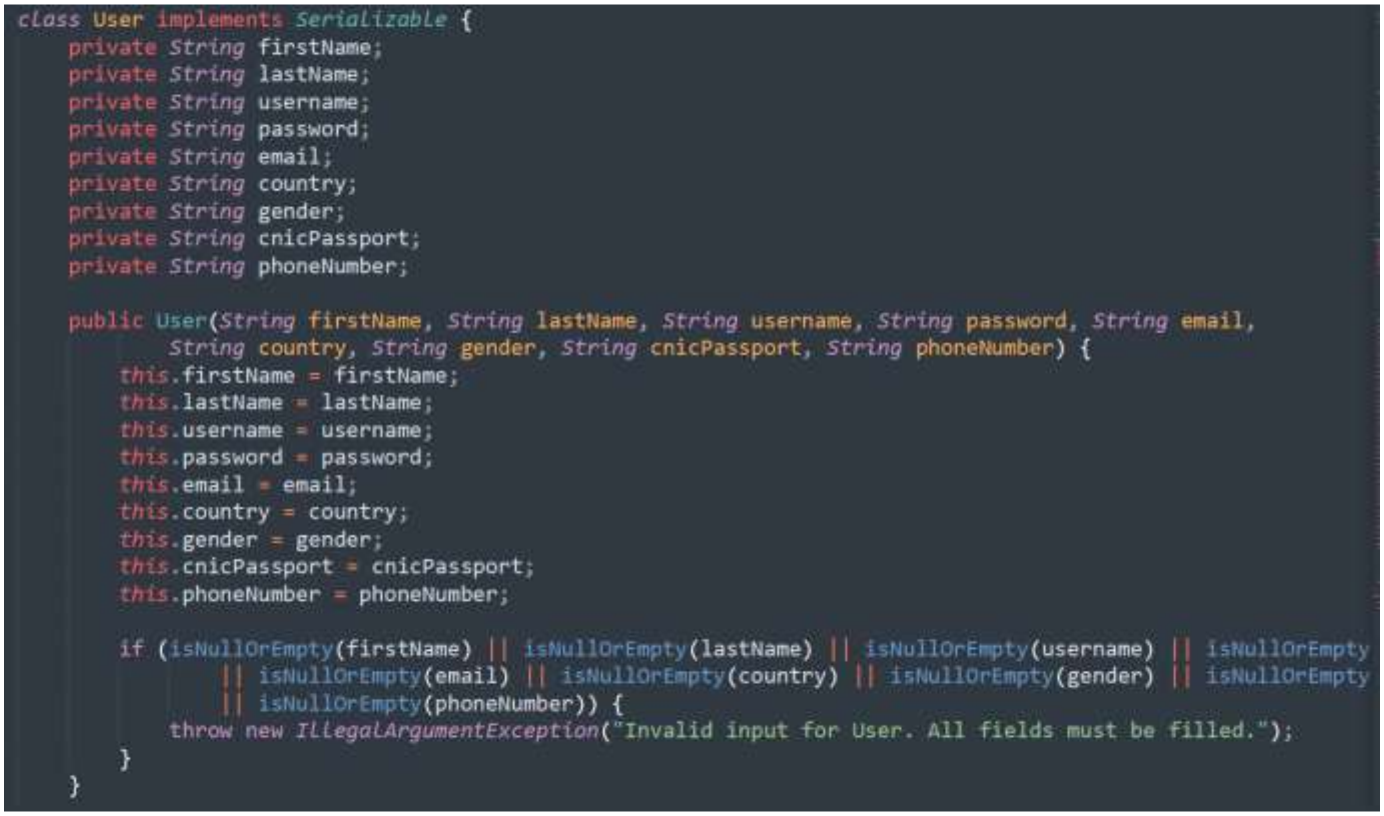
The handleRegistration method in the RegisterPage class checks if the **'Registration Successful'** checkbox is selected. If not, it prompts the user to select the checkbox. It then validates mandatory fields and initiates the registration process. Upon successful registration, it closes the current frame and navigates to the main page. Exception handling is included for robustness.



```
private void processRegistration() throws FileNotFoundException, IOException {
    String fileName = "users.bin";


    try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(fileName))) {
        User newUser = new User(
                firstNameField.getText(),
                lastNameField.getText(),
                usernameField.getText(),
                new String(passwordField.getPassword()),
                emailField.getText(),
                (String) countryField.getSelectedItem(),
                (String) genderComboBox.getSelectedItem(),
                cnicPassportField.getText(),
                phoneNumberField.getText());


        oos.writeObject(newUser);
        UserDatabase.registerUser(newUser.getUsername());
        System.out.println("Registration Successful!");
    } catch (IOException e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(frame, "Error during registration. Please try again.");
    }
}
```

The processRegistration method in the **RegisterPage** class writes user registration data to a file named "users.bin" using an ObjectOutputStream. It creates a new User object with information from the registration fields. The method then writes this user object to the file, registers the username in the UserDatabase, and prints a success message to the console.

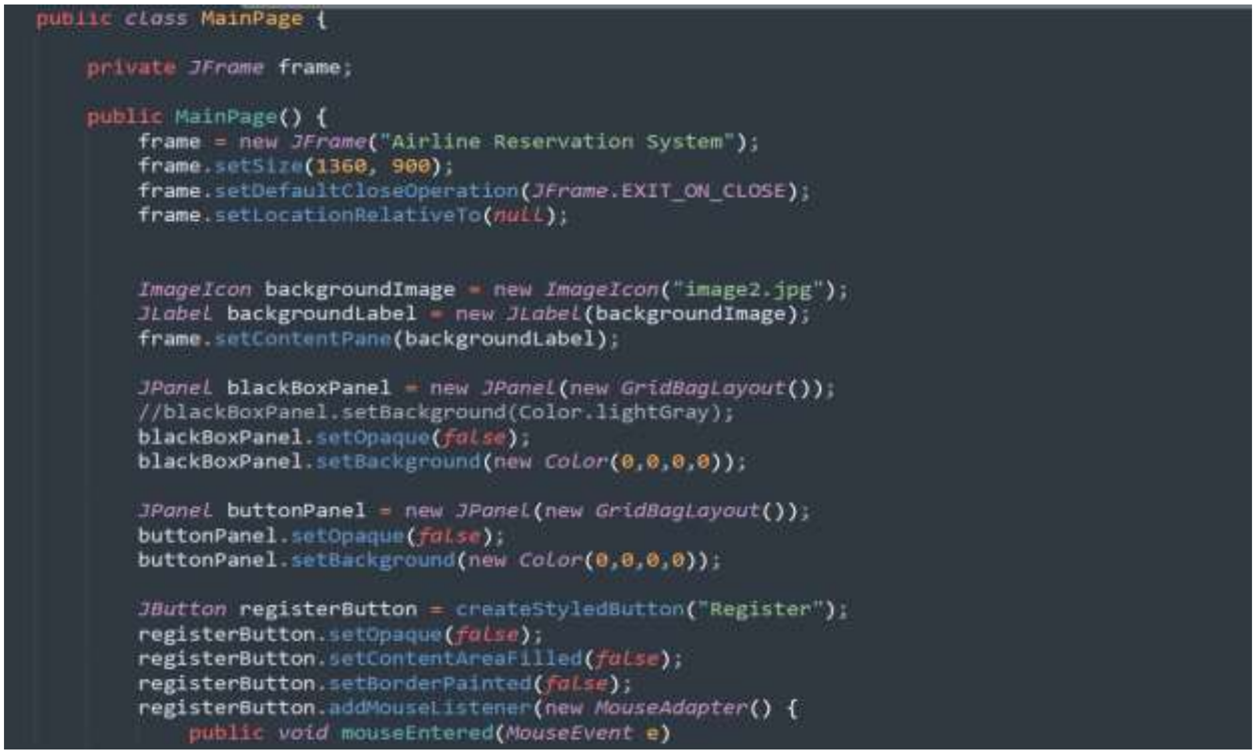
- **Class User**



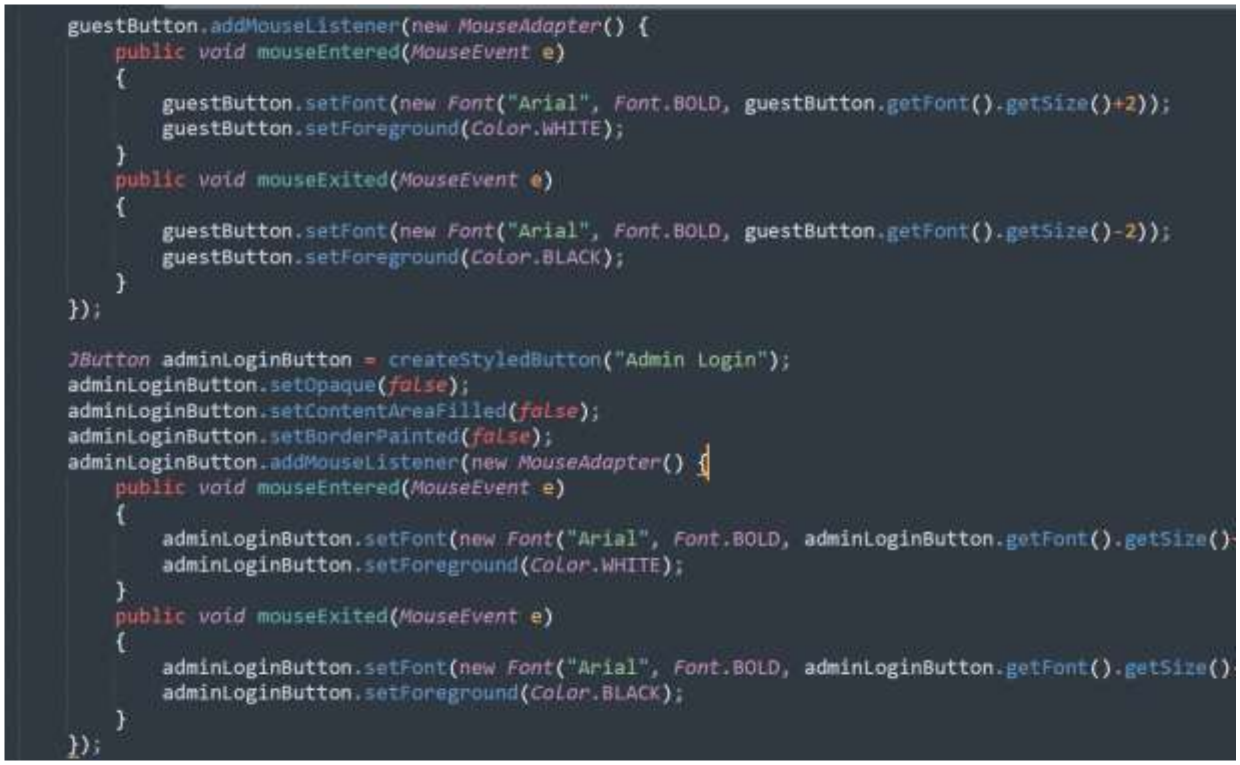
```java
class User implements Serializable {
    private String firstName;
    private String lastName;
    private String username;
    private String password;
    private String email;
    private String country;
    private String gender;
    private String cnicPassport;
    private String phoneNumber;

    public User(String firstName, String lastName, String username, String password, String email,
            String country, String gender, String cnicPassport, String phoneNumber) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.username = username;
        this.password = password;
        this.email = email;
        this.country = country;
        this.gender = gender;
        this.cnicPassport = cnicPassport;
        this.phoneNumber = phoneNumber;

        if (isNullOrEmpty(firstName) || isNullOrEmpty(lastName) || isNullOrEmpty(username) || isNullOrEmpty
                || isNullOrEmpty(email) || isNullOrEmpty(country) || isNullOrEmpty(gender) || isNullOrEmpty
                || isNullOrEmpty(phoneNumber)) {
            throw new IllegalArgumentException("Invalid input for User. All fields must be filled.");
        }
    }
```

The `User` class represents a serializable user object in Java with attributes such as first name, last name, username, password, email, country, gender, CNIC/passport number, and phone number. Its constructor initializes these attributes and includes input validation, throwing an `**IllegalArgumentException**` if any essential field is null or empty. This ensures that a valid and complete user object is created during instantiation.

- **Class MainPage**



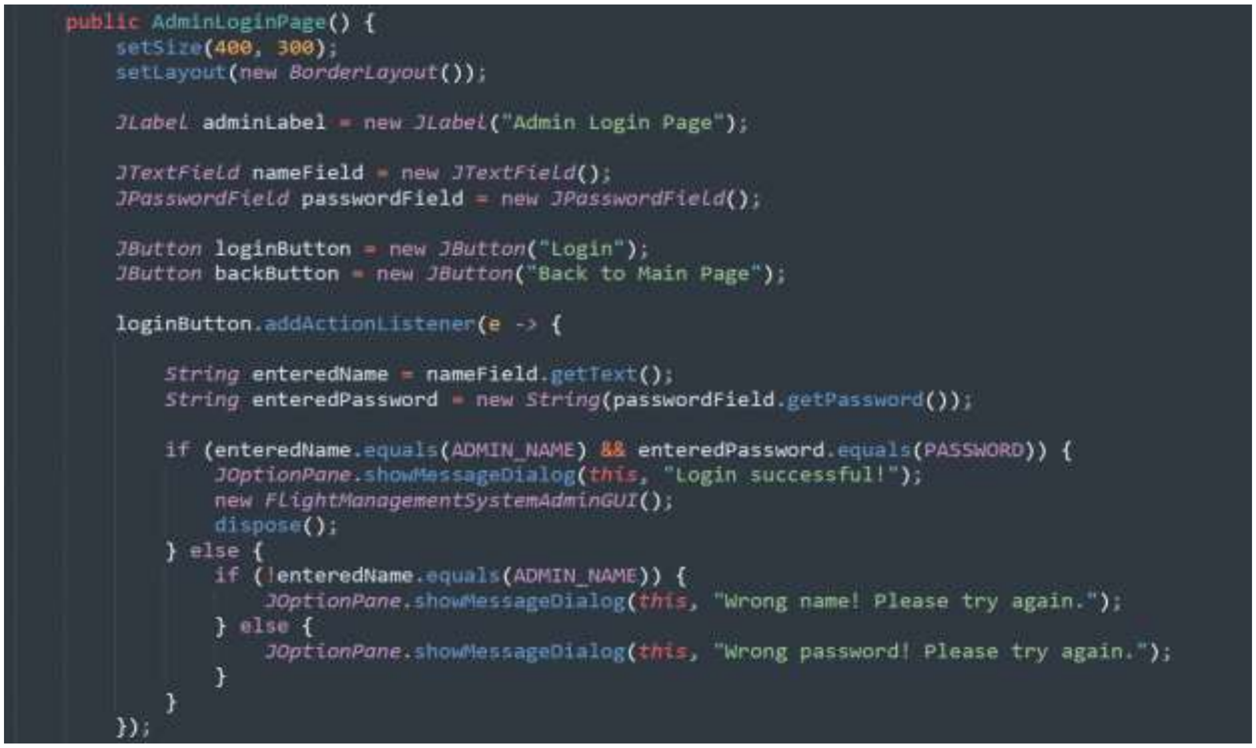
```java
public class MainPage {

    private JFrame frame;

    public MainPage() {
        frame = new JFrame("Airline Reservation System");
        frame.setSize(1360, 900);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);


        ImageIcon backgroundImage = new ImageIcon("image2.jpg");
        JLabel backgroundLabel = new JLabel(backgroundImage);
        frame.setContentPane(backgroundLabel);

        JPanel blackBoxPanel = new JPanel(new GridBagLayout());
        //blackBoxPanel.setBackground(Color.lightGray);
        blackBoxPanel.setOpaque(false);
        blackBoxPanel.setBackground(new Color(0,0,0,0));

        JPanel buttonPanel = new JPanel(new GridBagLayout());
        buttonPanel.setOpaque(false);
        buttonPanel.setBackground(new Color(0,0,0,0));

        JButton registerButton = createStyledButton("Register");
        registerButton.setOpaque(false);
        registerButton.setContentAreaFilled(false);
        registerButton.setBorderPainted(false);
        registerButton.addMouseListener(new MouseAdapter() {
            public void mouseEntered(MouseEvent e)
```



```java
        guestButton.addMouseListener(new MouseAdapter() {
            public void mouseEntered(MouseEvent e)
            {
                guestButton.setFont(new Font("Arial", Font.BOLD, guestButton.getFont().getSize()+2));
                guestButton.setForeground(Color.WHITE);
            }
            public void mouseExited(MouseEvent e)
            {
                guestButton.setFont(new Font("Arial", Font.BOLD, guestButton.getFont().getSize()-2));
                guestButton.setForeground(Color.BLACK);
            }
        });

        JButton adminLoginButton = createStyledButton("Admin Login");
        adminLoginButton.setOpaque(false);
        adminLoginButton.setContentAreaFilled(false);
        adminLoginButton.setBorderPainted(false);
        adminLoginButton.addMouseListener(new MouseAdapter() {
            public void mouseEntered(MouseEvent e)
            {
                adminLoginButton.setFont(new Font("Arial", Font.BOLD, adminLoginButton.getFont().getSize()
                adminLoginButton.setForeground(Color.WHITE);
            }
            public void mouseExited(MouseEvent e)
            {
                adminLoginButton.setFont(new Font("Arial", Font.BOLD, adminLoginButton.getFont().getSize()
                adminLoginButton.setForeground(Color.BLACK);
            }
        });
```

The `**MainPage**` class sets up a JFrame for an Airline Reservation System with styled buttons like "Register," "Login," "Continue as Guest," and "Admin Login." The buttons have dynamic font and color changes on hover. Their actions include opening corresponding pages for registration, login, guest mode, and admin login. The class serves as the main user interface for accessing different functionalities in the system.

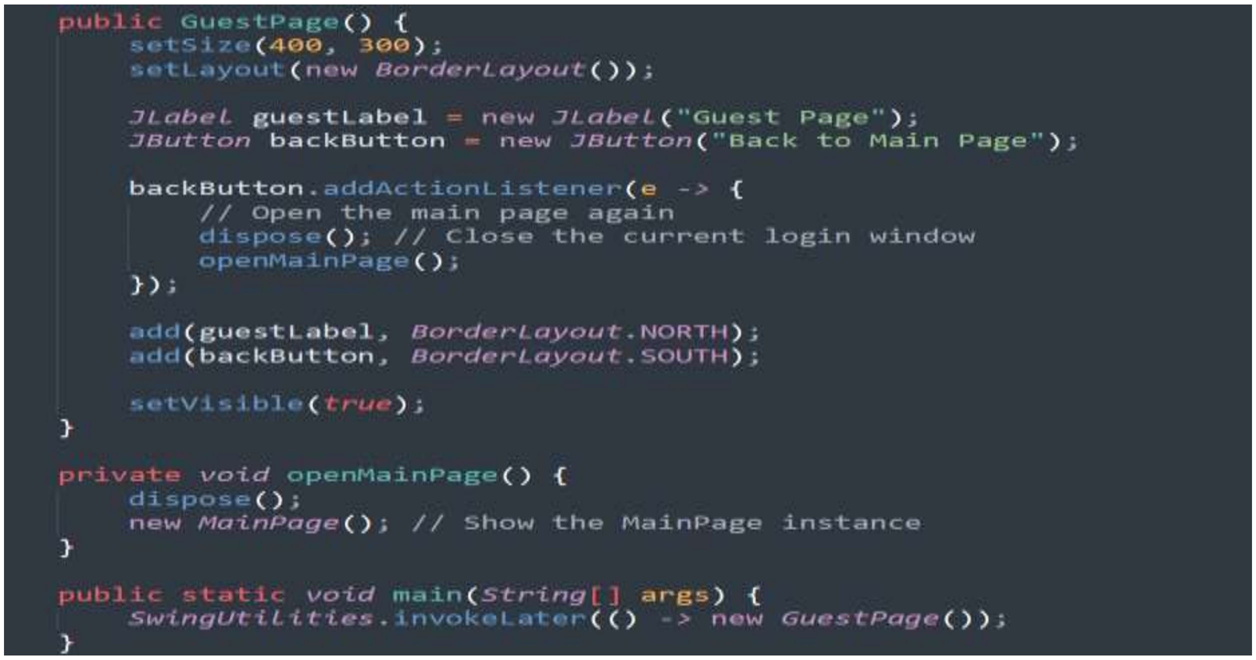
- **Class AdminLoginPage**



```java
public AdminLoginPage() {
    setSize(400, 300);
    setLayout(new BorderLayout());

    JLabel adminLabel = new JLabel("Admin Login Page");

    JTextField nameField = new JTextField();
    JPasswordField passwordField = new JPasswordField();

    JButton loginButton = new JButton("Login");
    JButton backButton = new JButton("Back to Main Page");

    loginButton.addActionListener(e -> {

        String enteredName = nameField.getText();
        String enteredPassword = new String(passwordField.getPassword());

        if (enteredName.equals(ADMIN_NAME) && enteredPassword.equals(PASSWORD)) {
            JOptionPane.showMessageDialog(this, "Login successful!");
            new FlightManagementSystemAdminGUI();
            dispose();
        } else {
            if (!enteredName.equals(ADMIN_NAME)) {
                JOptionPane.showMessageDialog(this, "Wrong name! Please try again.");
            } else {
                JOptionPane.showMessageDialog(this, "Wrong password! Please try again.");
            }
        }
    });
```

The `**AdminLoginPage**` class extends JFrame and provides a simple admin login interface. It includes fields for entering admin name and password, along with "Login" and "**Back to Main Page"** buttons. The `loginButton` action checks entered credentials, displaying a success message and opening a new admin GUI if correct. If incorrect, it prompts the user with specific error messages for name and password mismatches. The "Back to Main Page" button disposes of the current frame and returns to the main page.

- **Class GuestPage**

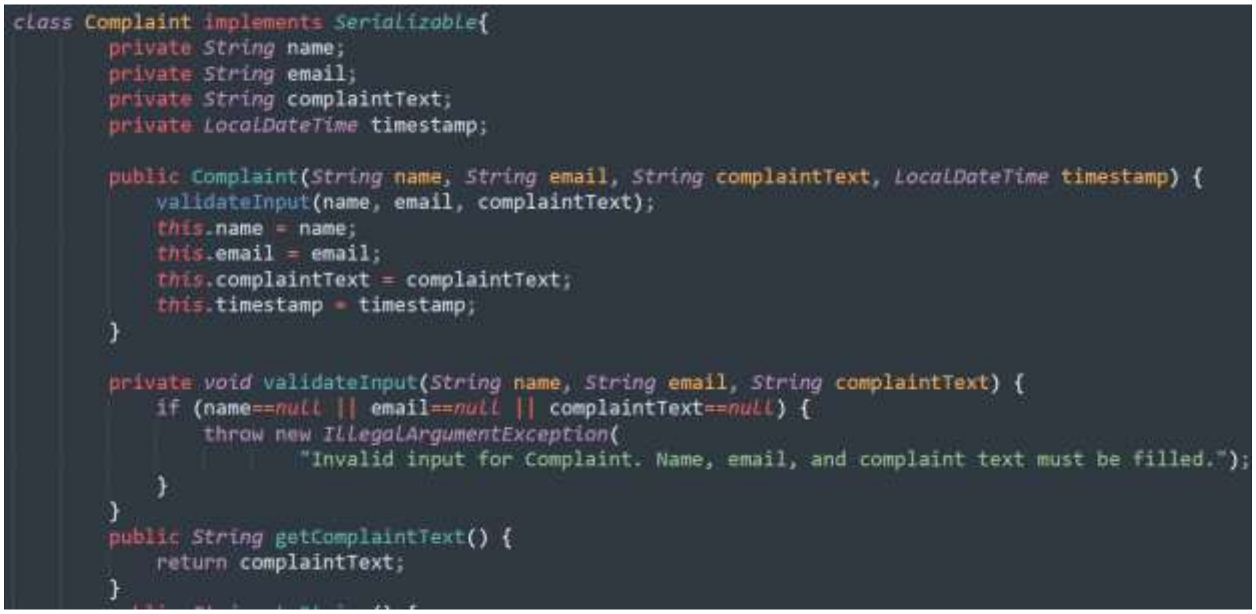


```java
public GuestPage() {
    setSize(400, 300);
    setLayout(new BorderLayout());

    JLabel guestLabel = new JLabel("Guest Page");
    JButton backButton = new JButton("Back to Main Page");

    backButton.addActionListener(e -> {
        // Open the main page again
        dispose(); // Close the current login window
        openMainPage();
    });

    add(guestLabel, BorderLayout.NORTH);
    add(backButton, BorderLayout.SOUTH);

    setVisible(true);
}

private void openMainPage() {
    dispose();
    new MainPage(); // Show the MainPage instance
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> new GuestPage());
}
```

The `**GuestPage**` class extends JFrame, representing a guest interface. It includes a "Guest Page" label and a "Back to Main Page" button. The **`backButton`** action disposes of the current frame and opens the main page. The **`openMainPage`** method facilitates this operation. The class is initialized and displayed upon calling the main method, showcasing the guest interface.

MainMenu Classes

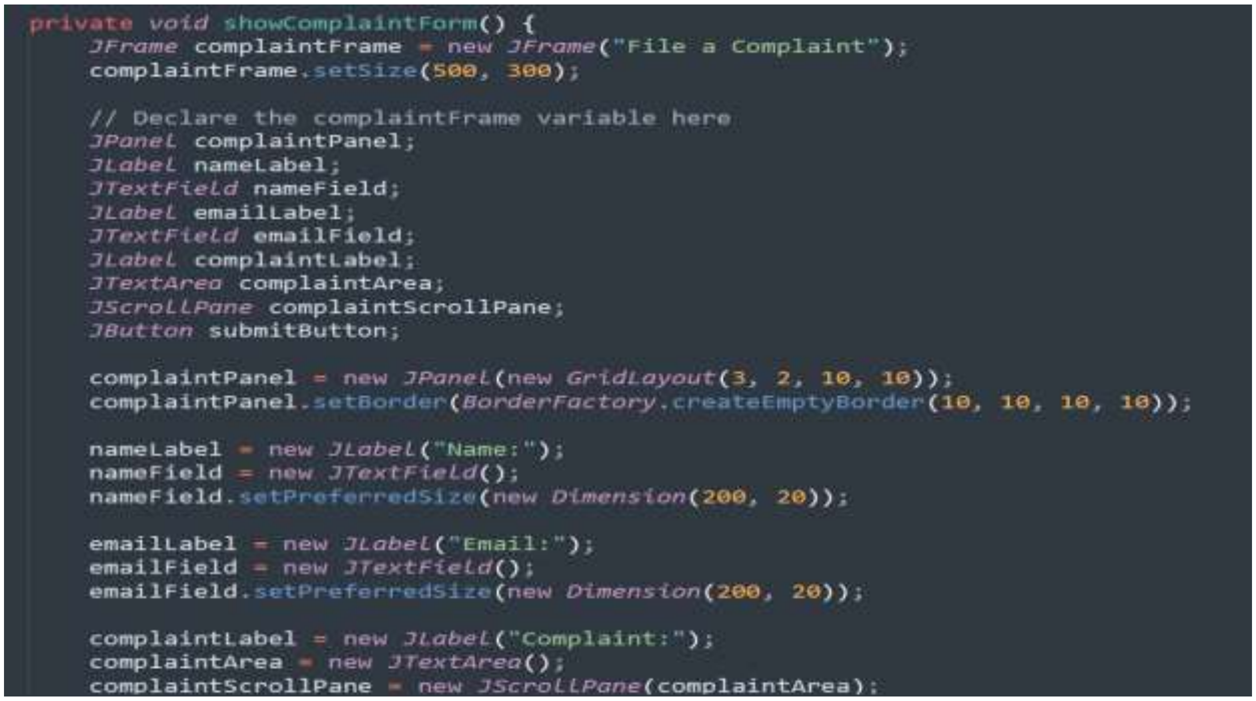
- **Class Complaint**



```
class Complaint implements Serializable{
        private String name;
        private String email;
        private String complaintText;
        private LocalDateTime timestamp;

        public Complaint(String name, String email, String complaintText, LocalDateTime timestamp) {
            validateInput(name, email, complaintText);
            this.name = name;
            this.email = email;
            this.complaintText = complaintText;
            this.timestamp = timestamp;
        }

        private void validateInput(String name, String email, String complaintText) {
            if (name==null || email==null || complaintText==null) {
                throw new IllegalArgumentException(
                        "Invalid input for Complaint. Name, email, and complaint text must be filled.");
            }
        }
        public String getComplaintText() {
            return complaintText;
        }
```

The `Complaint` class implements `Serializable` and represents a complaint entity. It has attributes such as `name`, `email`, **`complaintText`,** and `timestamp`. The constructor initializes these attributes, validating that essential fields (`name`, `email`, and `complaintText`) are not null. The `validateInput` method throws an `IllegalArgumentException` if any of these fields are null. The class provides a **`getComplaintText`** method to retrieve the complaint text and a `toString` method for generating a string representation of the complaint, including its name, email, complaint text, and timestamp.

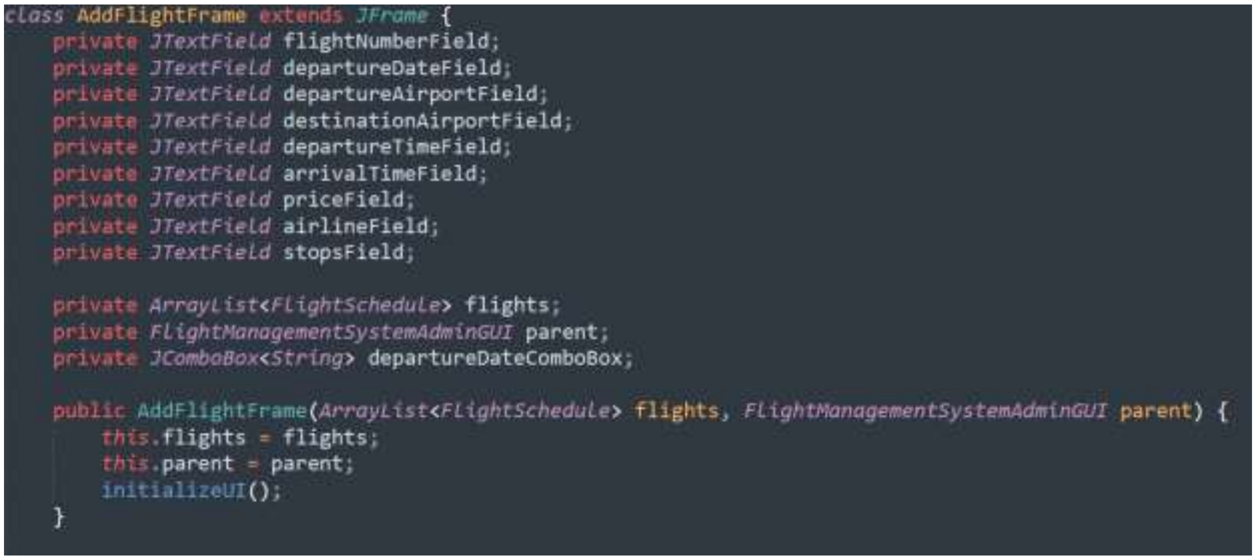
- **Class MainMenuProject**



```
private void showComplaintForm() {
    JFrame complaintFrame = new JFrame("File a Complaint");
    complaintFrame.setSize(500, 300);

    // Declare the complaintFrame variable here
    JPanel complaintPanel;
    JLabel nameLabel;
    JTextField nameField;
    JLabel emailLabel;
    JTextField emailField;
    JLabel complaintLabel;
    JTextArea complaintArea;
    JScrollPane complaintScrollPane;
    JButton submitButton;

    complaintPanel = new JPanel(new GridLayout(3, 2, 10, 10));
    complaintPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

    nameLabel = new JLabel("Name:");
    nameField = new JTextField();
    nameField.setPreferredSize(new Dimension(200, 20));

    emailLabel = new JLabel("Email:");
    emailField = new JTextField();
    emailField.setPreferredSize(new Dimension(200, 20));

    complaintLabel = new JLabel("Complaint:");
    complaintArea = new JTextArea();
    complaintScrollPane = new JScrollPane(complaintArea);
```

The `**MainMenuProject**` class extends JFrame and represents the main menu of an Airline Reservation System. It features a background image, buttons for functionalities like viewing flight schedules, booking history, filing complaints, and an "About Us" section. The class utilizes a grid layout for button placement and styling for hover effects. Button clicks trigger different actions, such as opening specific GUIs or displaying relevant information. The class also includes methods for creating panels, handling label clicks, and displaying an "About Us" panel with detailed information about the airline. Additionally, it provides functionalities to show booking history and file complaints, each with its respective GUI components. The interface is designed with user-friendly features and visual appeal.

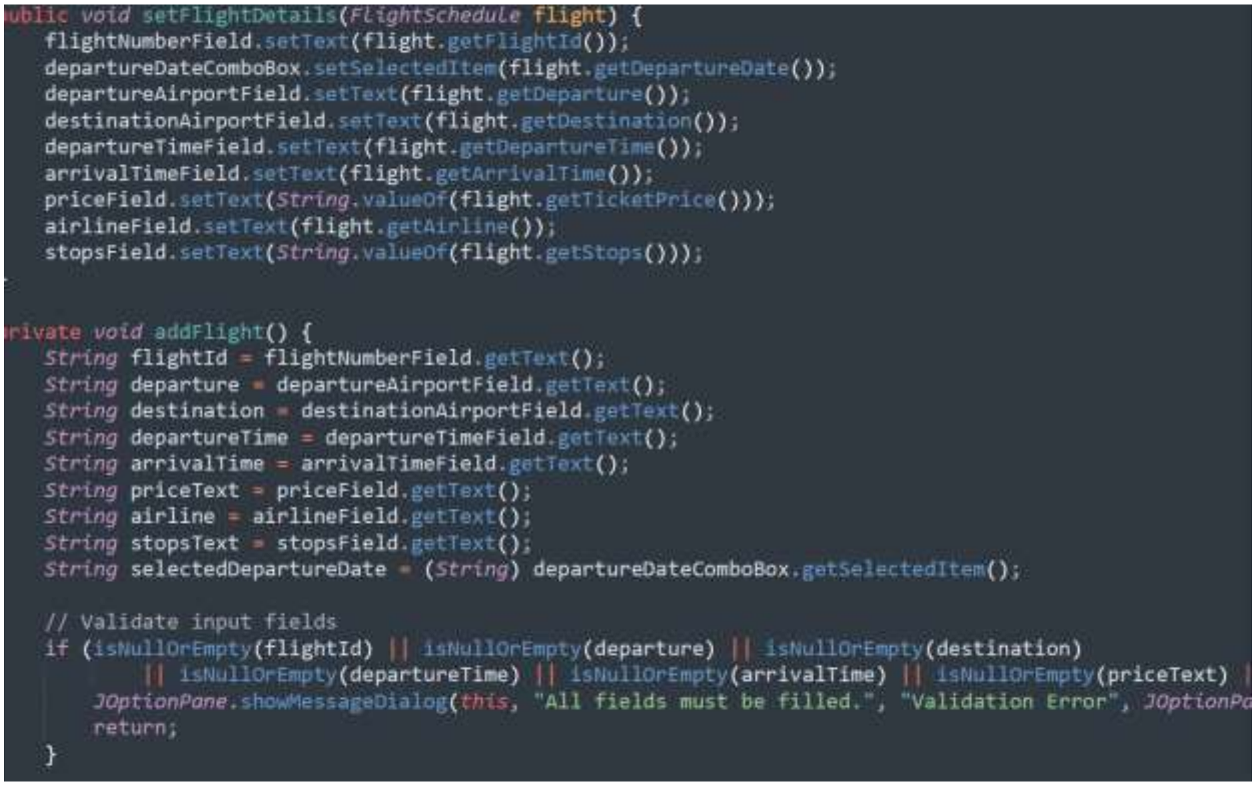
- **Class AddFlightFrame**



```
class AddFlightFrame extends JFrame {
    private JTextField flightNumberField;
    private JTextField departureDateField;
    private JTextField departureAirportField;
    private JTextField destinationAirportField;
    private JTextField departureTimeField;
    private JTextField arrivalTimeField;
    private JTextField priceField;
    private JTextField airlineField;
    private JTextField stopsField;

    private ArrayList<FlightSchedule> flights;
    private FlightManagementSystemAdminGUI parent;
    private JComboBox<String> departureDateComboBox;

    public AddFlightFrame(ArrayList<FlightSchedule> flights, FlightManagementSystemAdminGUI parent) {
        this.flights = flights;
        this.parent = parent;
        initializeUI();
    }
```

The `AddFlightFrame` class extends JFrame and represents a GUI for adding new flight schedules. It includes text fields for flight details and references to existing flights and the parent admin GUI. The constructor initializes these components and sets up the user interface.

- **Set & add Flight Method**

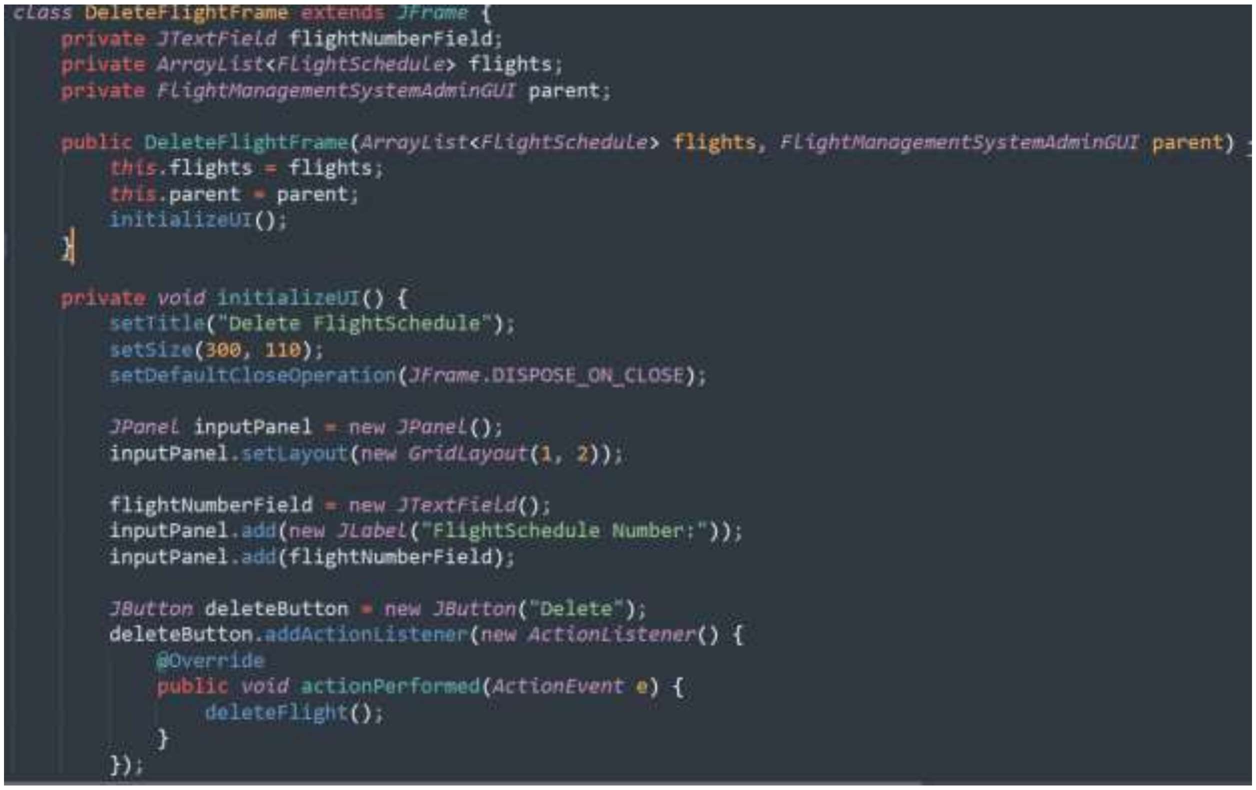


```
ublic void setFlightDetails(FlightSchedule flight) {
   flightNumberField.setText(flight.getFlightId());
   departureDateComboBox.setSelectedItem(flight.getDepartureDate());
   departureAirportField.setText(flight.getDeparture());
   destinationAirportField.setText(flight.getDestination());
   departureTimeField.setText(flight.getDepartureTime());
   arrivalTimeField.setText(flight.getArrivalTime());
   priceField.setText(String.valueOf(flight.getTicketPrice()));
   airlineField.setText(flight.getAirline());
   stopsField.setText(String.valueOf(flight.getStops()));


rivate void addFlight() {
   String flightId = flightNumberField.getText();
   String departure = departureAirportField.getText();
   String destination = destinationAirportField.getText();
   String departureTime = departureTimeField.getText();
   String arrivalTime = arrivalTimeField.getText();
   String priceText = priceField.getText();
   String airline = airlineField.getText();
   String stopsText = stopsField.getText();
   String selectedDepartureDate = (String) departureDateComboBox.getSelectedItem();

   // Validate input fields
   if (isNullOrEmpty(flightId) || isNullOrEmpty(departure) || isNullOrEmpty(destination)
           || isNullOrEmpty(departureTime) || isNullOrEmpty(arrivalTime) || isNullOrEmpty(priceText) |
       JOptionPane.showMessageDialog(this, "All fields must be filled.", "Validation Error", JOptionPa
       return;
   }
```

The `setFlightDetails` method populates the fields of the `AddFlightFrame` with details from a given `FlightSchedule` object. It takes a `FlightSchedule` as a parameter and sets the text of various fields such as flight number, departure date, airports, times, price, airline, and stops based on the provided flight information.

The `addFlight` method is responsible for extracting the user-inputted details from the text fields, converting necessary values, and performing input validation. It checks if any essential field is empty and displays an error message if validation fails. If successful, it retrieves the data entered by the user, such as flight number, departure and destination airports, times, price, airline, stops, and the selected departure date from a combo box. This method is typically invoked when the user intends to add a new flight schedule to the system.

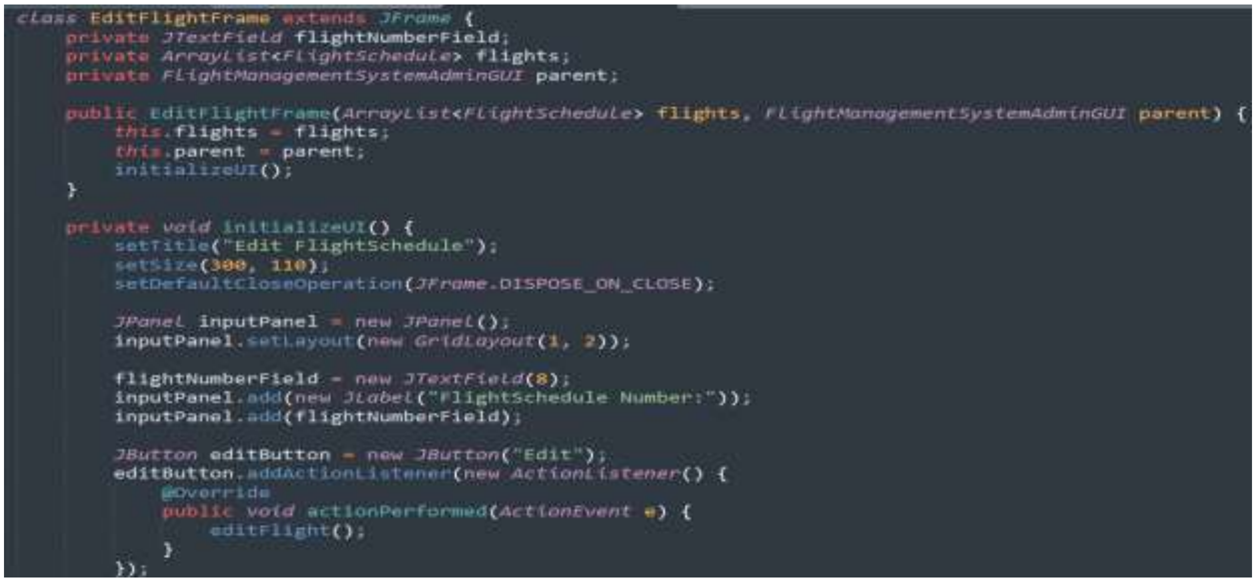
- **Class DeleteFlightFrame**



```java
class DeleteFlightFrame extends JFrame {
    private JTextField flightNumberField;
    private ArrayList<FlightSchedule> flights;
    private FlightManagementSystemAdminGUI parent;

    public DeleteFlightFrame(ArrayList<FlightSchedule> flights, FlightManagementSystemAdminGUI parent)
        this.flights = flights;
        this.parent = parent;
        initializeUI();
    }

    private void initializeUI() {
        setTitle("Delete FlightSchedule");
        setSize(300, 110);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        JPanel inputPanel = new JPanel();
        inputPanel.setLayout(new GridLayout(1, 2));

        flightNumberField = new JTextField();
        inputPanel.add(new JLabel("FlightSchedule Number:"));
        inputPanel.add(flightNumberField);

        JButton deleteButton = new JButton("Delete");
        deleteButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                deleteFlight();
            }
        });
```

The `DeleteFlightFrame` class extends JFrame, providing a GUI for deleting flight schedules. It includes a text field for entering the flight number and a "Delete" button. The constructor initializes the list of flights and the parent admin GUI. The frame dimensions and components are set up in the `initializeUI` method. The "Delete" button is linked to the `**deleteFlight**` method, responsible for handling the deletion process based on the user-inputted flight number.

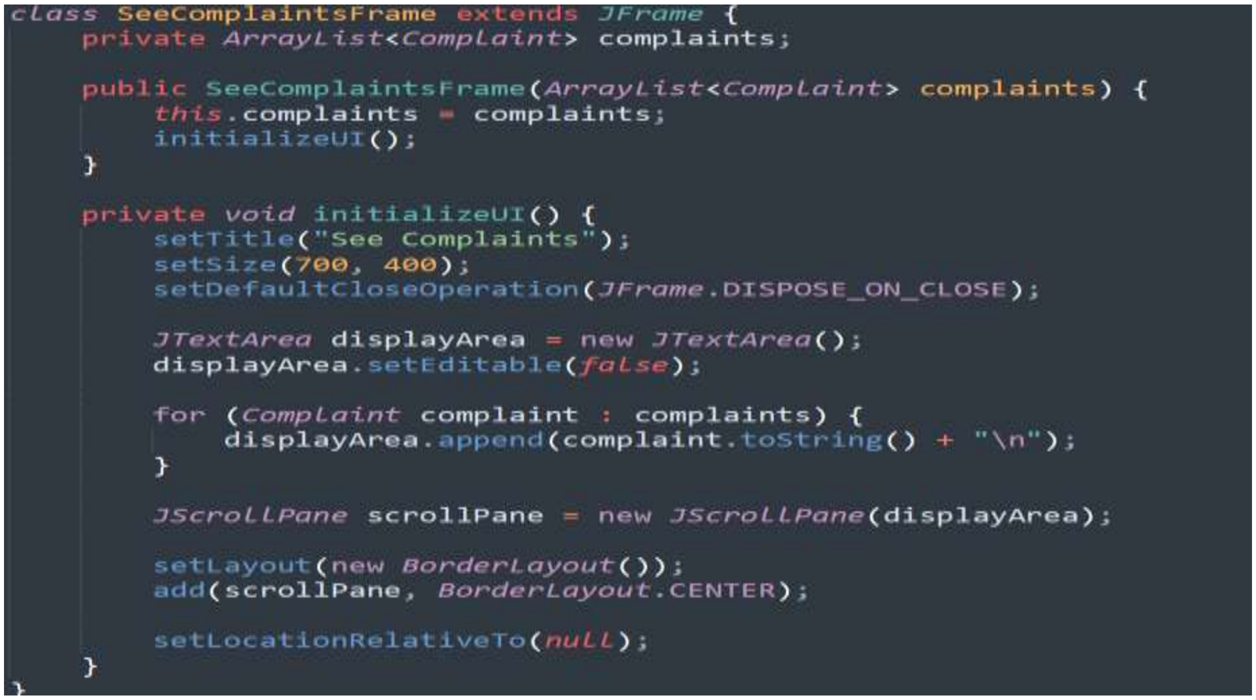
- **Class EditFlight**



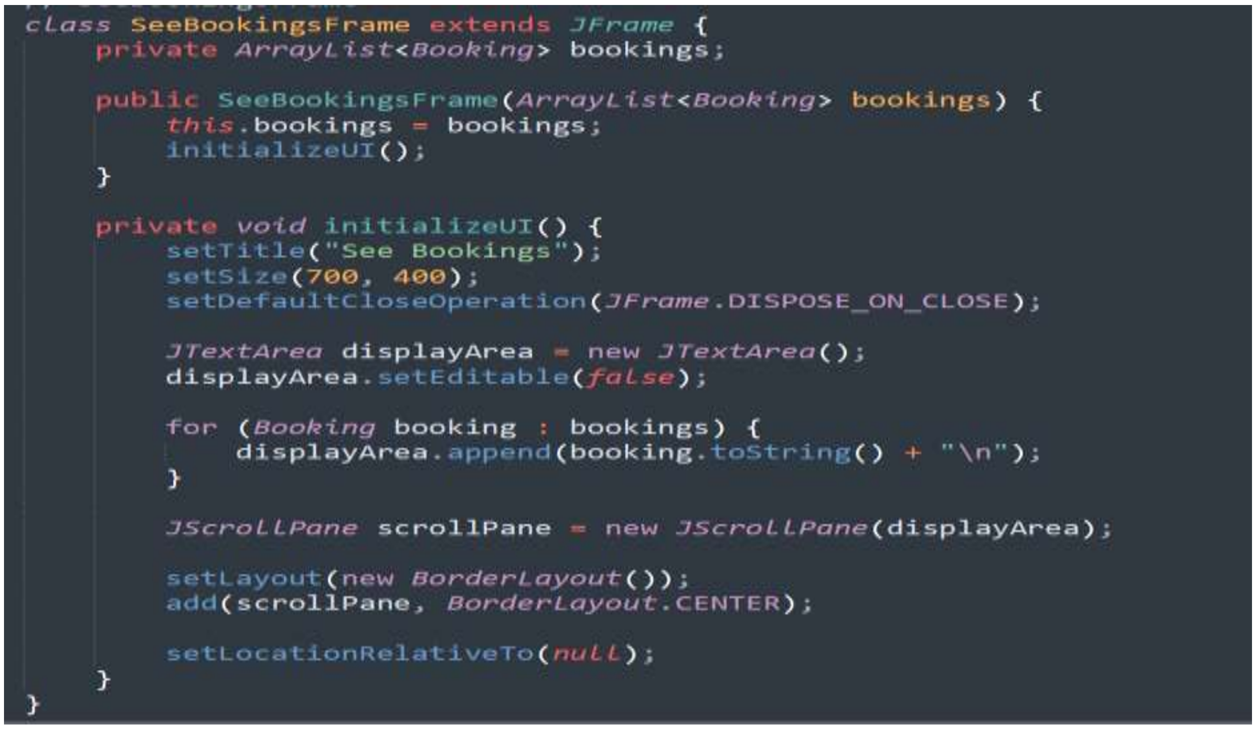
```java
class EditFlightFrame extends JFrame {
    private JTextField flightNumberField;
    private ArrayList<FlightSchedule> flights;
    private FlightManagementSystemAdminGUI parent;

    public EditFlightFrame(ArrayList<FlightSchedule> flights, FlightManagementSystemAdminGUI parent) {
        this.flights = flights;
        this.parent = parent;
        initializeUI();
    }

    private void initializeUI() {
        setTitle("Edit FlightSchedule");
        setSize(300, 110);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        JPanel inputPanel = new JPanel();
        inputPanel.setLayout(new GridLayout(1, 2));

        flightNumberField = new JTextField(8);
        inputPanel.add(new JLabel("FlightSchedule Number:"));
        inputPanel.add(flightNumberField);

        JButton editButton = new JButton("Edit");
        editButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                editFlight();
            }
        });
```

The **EditFlightFrame** class extends JFrame and provides a GUI for editing flight schedules. It includes a text field for entering the flight number and an "Edit" button. The constructor initializes the list of flights and the parent admin GUI, and the initializeUI method sets up the frame with appropriate components and dimensions.

- **Class SeeComplaints & SeeBookingsFrame**



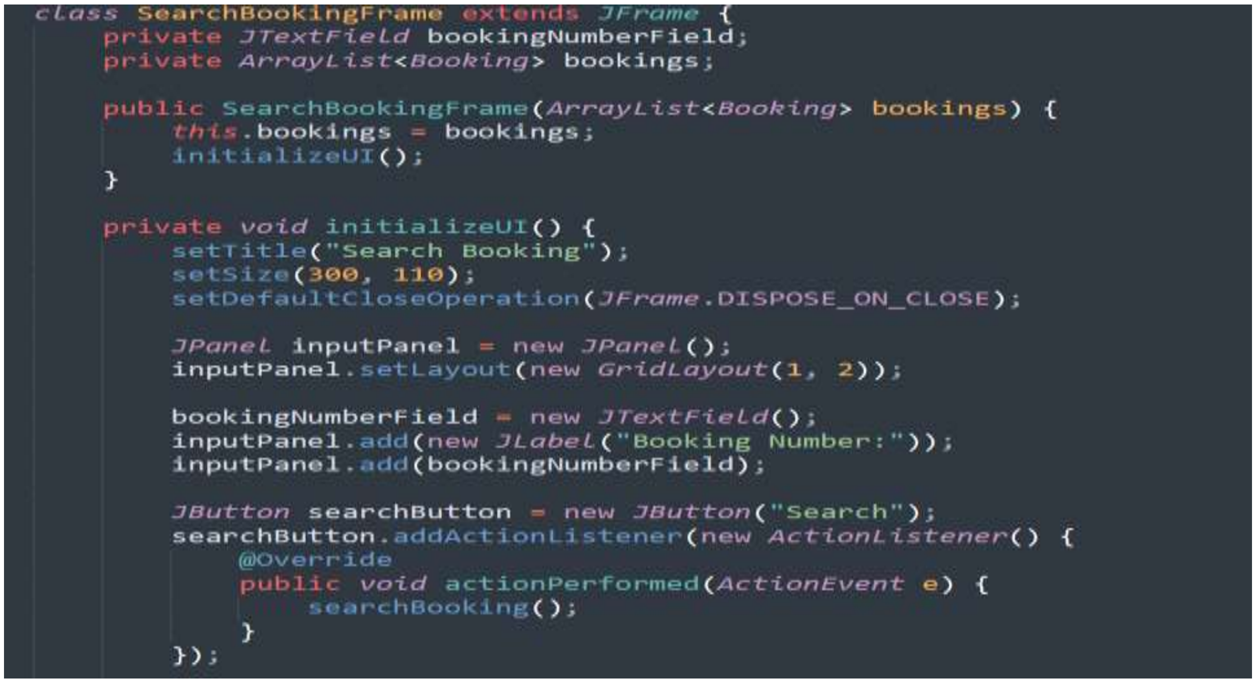
```java
class SeeComplaintsFrame extends JFrame {
    private ArrayList<Complaint> complaints;

    public SeeComplaintsFrame(ArrayList<Complaint> complaints) {
        this.complaints = complaints;
        initializeUI();
    }

    private void initializeUI() {
        setTitle("See Complaints");
        setSize(700, 400);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        JTextArea displayArea = new JTextArea();
        displayArea.setEditable(false);

        for (Complaint complaint : complaints) {
            displayArea.append(complaint.toString() + "\n");
        }

        JScrollPane scrollPane = new JScrollPane(displayArea);

        setLayout(new BorderLayout());
        add(scrollPane, BorderLayout.CENTER);

        setLocationRelativeTo(null);
    }
}
```



```java
class SeeBookingsFrame extends JFrame {
    private ArrayList<Booking> bookings;

    public SeeBookingsFrame(ArrayList<Booking> bookings) {
        this.bookings = bookings;
        initializeUI();
    }

    private void initializeUI() {
        setTitle("See Bookings");
        setSize(700, 400);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        JTextArea displayArea = new JTextArea();
        displayArea.setEditable(false);

        for (Booking booking : bookings) {
            displayArea.append(booking.toString() + "\n");
        }

        JScrollPane scrollPane = new JScrollPane(displayArea);

        setLayout(new BorderLayout());
        add(scrollPane, BorderLayout.CENTER);

        setLocationRelativeTo(null);
    }
}
```

The `**SeeComplaintsFrame**` and `**SeeBookingsFrame**` classes are JFrame components that display a list of complaints and bookings, respectively. They initialize a JFrame with a JTextArea to show the details of each complaint or booking. The information is appended to the JTextArea, and a JScrollPane is used for scrolling if needed. Both frames contribute to the application's functionality by providing a user interface to view complaints and bookings.

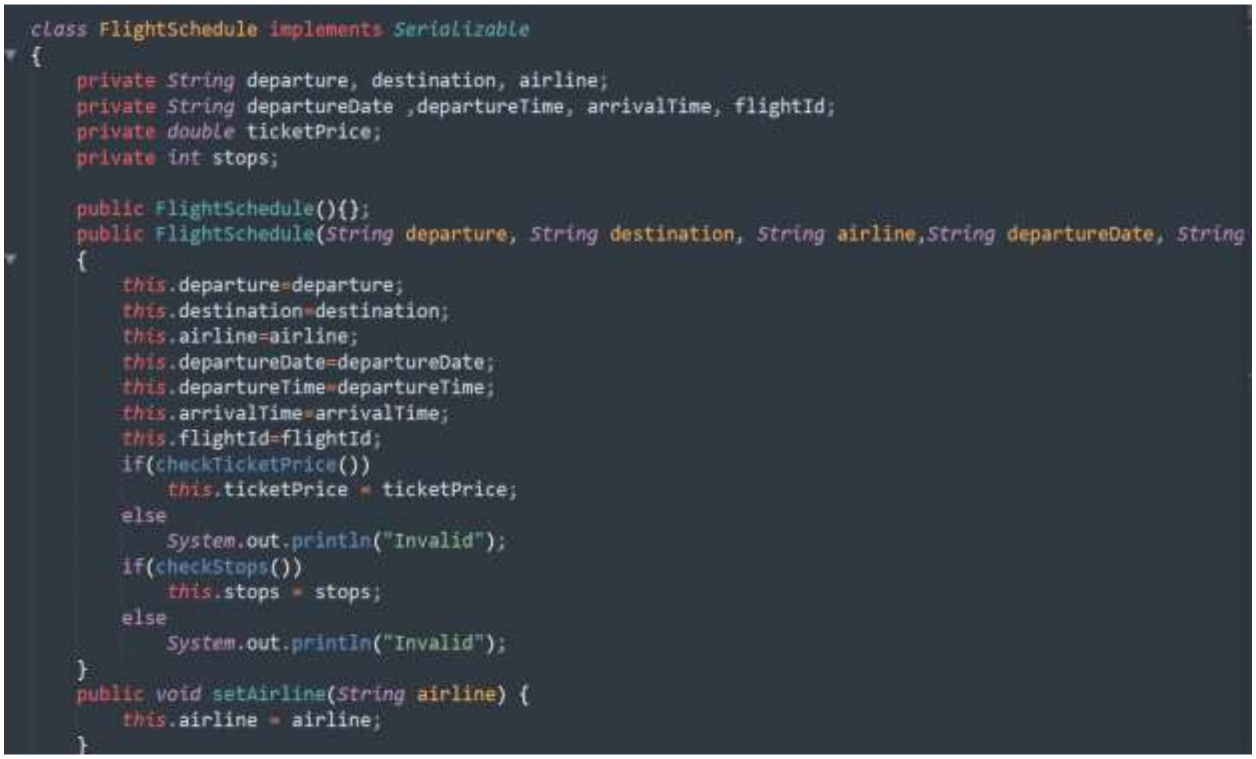
- **Class SearckBookingFrame**



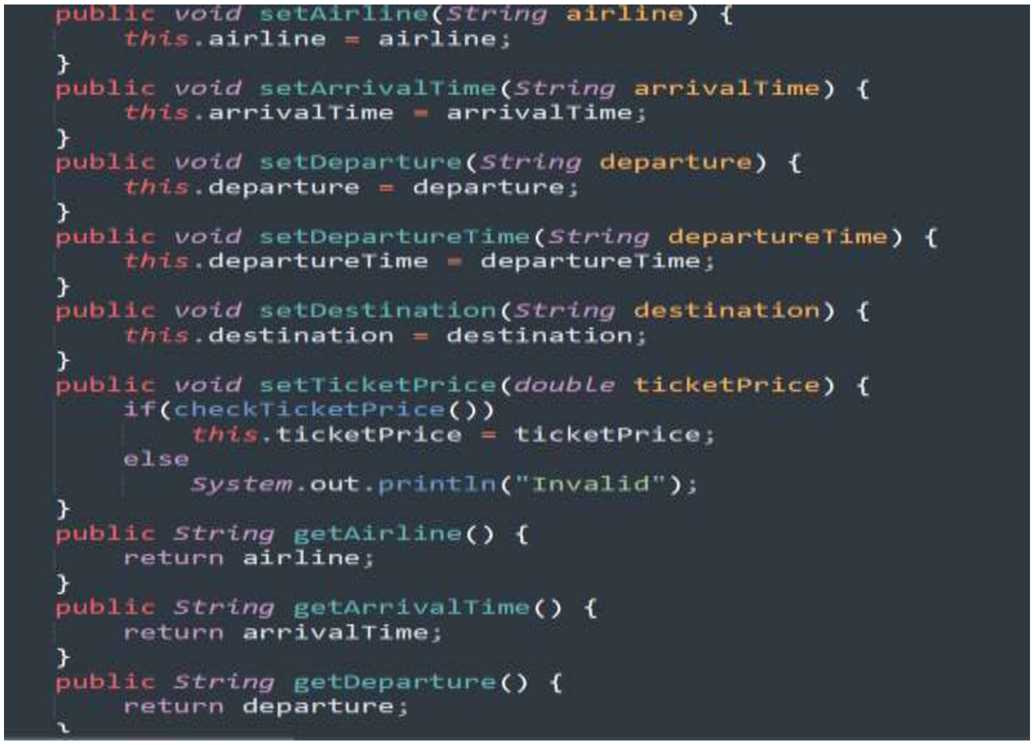
```java
class SearchBookingFrame extends JFrame {
    private JTextField bookingNumberField;
    private ArrayList<Booking> bookings;

    public SearchBookingFrame(ArrayList<Booking> bookings) {
        this.bookings = bookings;
        initializeUI();
    }

    private void initializeUI() {
        setTitle("Search Booking");
        setSize(300, 110);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        JPanel inputPanel = new JPanel();
        inputPanel.setLayout(new GridLayout(1, 2));

        bookingNumberField = new JTextField();
        inputPanel.add(new JLabel("Booking Number:"));
        inputPanel.add(bookingNumberField);

        JButton searchButton = new JButton("Search");
        searchButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                searchBooking();
            }
        });
```

The `SearchBookingFrame` class provides a search functionality for bookings. It includes a JTextField for entering the booking number and a "Search" button. The `searchBooking` method retrieves the booking details based on the user-inputted booking number, displaying them in a JTextArea within a JOptionPane. Both classes contribute to the booking management system in the application.

- **Class FlightSchedule**



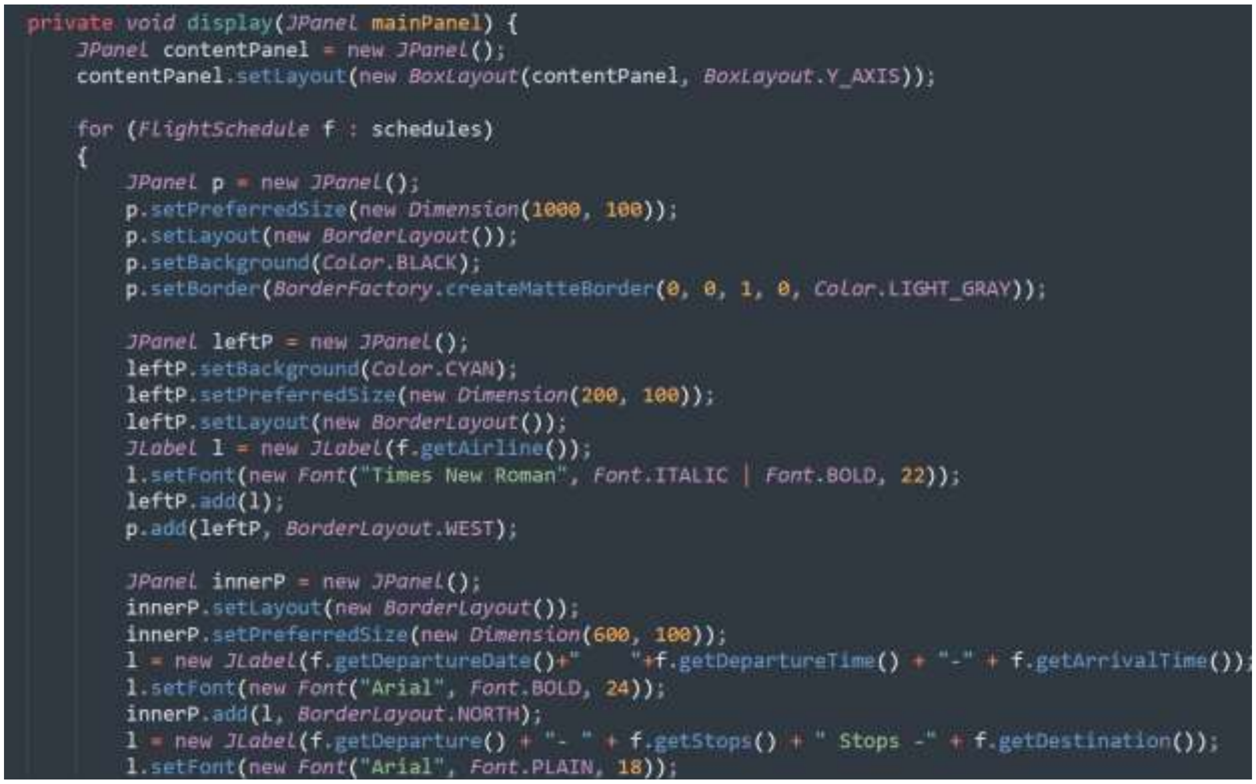
```java
class FlightSchedule implements Serializable
{
    private String departure, destination, airline;
    private String departureDate ,departureTime, arrivalTime, flightId;
    private double ticketPrice;
    private int stops;

    public FlightSchedule(){};
    public FlightSchedule(String departure, String destination, String airline,String departureDate, String
    {
        this.departure=departure;
        this.destination=destination;
        this.airline=airline;
        this.departureDate=departureDate;
        this.departureTime=departureTime;
        this.arrivalTime=arrivalTime;
        this.flightId=flightId;
        if(checkTicketPrice())
            this.ticketPrice = ticketPrice;
        else
            System.out.println("Invalid");
        if(checkStops())
            this.stops = stops;
        else
            System.out.println("Invalid");
    }
    public void setAirline(String airline) {
        this.airline = airline;
    }
```



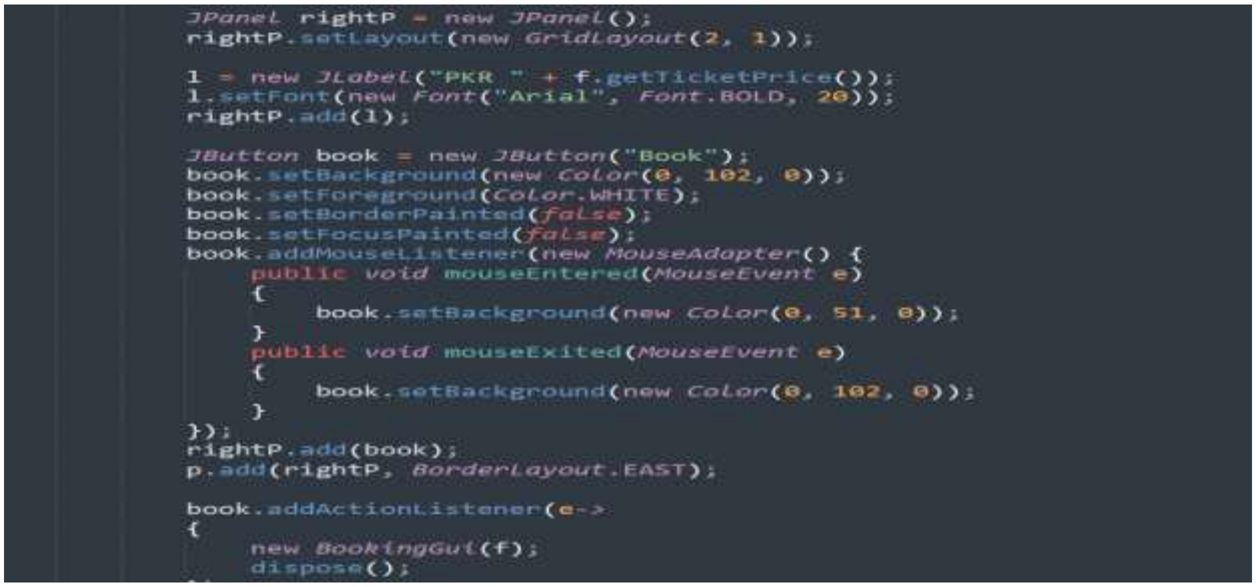
```java
    public void setAirline(String airline) {
        this.airline = airline;
    }
    public void setArrivalTime(String arrivalTime) {
        this.arrivalTime = arrivalTime;
    }
    public void setDeparture(String departure) {
        this.departure = departure;
    }
    public void setDepartureTime(String departureTime) {
        this.departureTime = departureTime;
    }
    public void setDestination(String destination) {
        this.destination = destination;
    }
    public void setTicketPrice(double ticketPrice) {
        if(checkTicketPrice())
            this.ticketPrice = ticketPrice;
        else
            System.out.println("Invalid");
    }
    public String getAirline() {
        return airline;
    }
    public String getArrivalTime() {
        return arrivalTime;
    }
    public String getDeparture() {
        return departure;
    }
```

The `FlightSchedule` class represents flight schedule information in Java. It includes instance variables for details like departure and destination airports, airline, departure date, times, ticket price, stops, and a unique identifier. The constructor initializes these attributes, performing checks for valid ticket prices and non-negative stops. Getter and setter methods provide access to these attributes with validation. Private methods, `checkTicketPrice()` and `checkStops()`, ensure non-negativity. The `readSchedule(ObjectInputStream ois)` method reads a `**FlightSchedule**` object from an `ObjectInputStream` while handling potential exceptions. Overall, the class encapsulates flight details, enforces constraints, and supports object serialization.

- **Display Method**



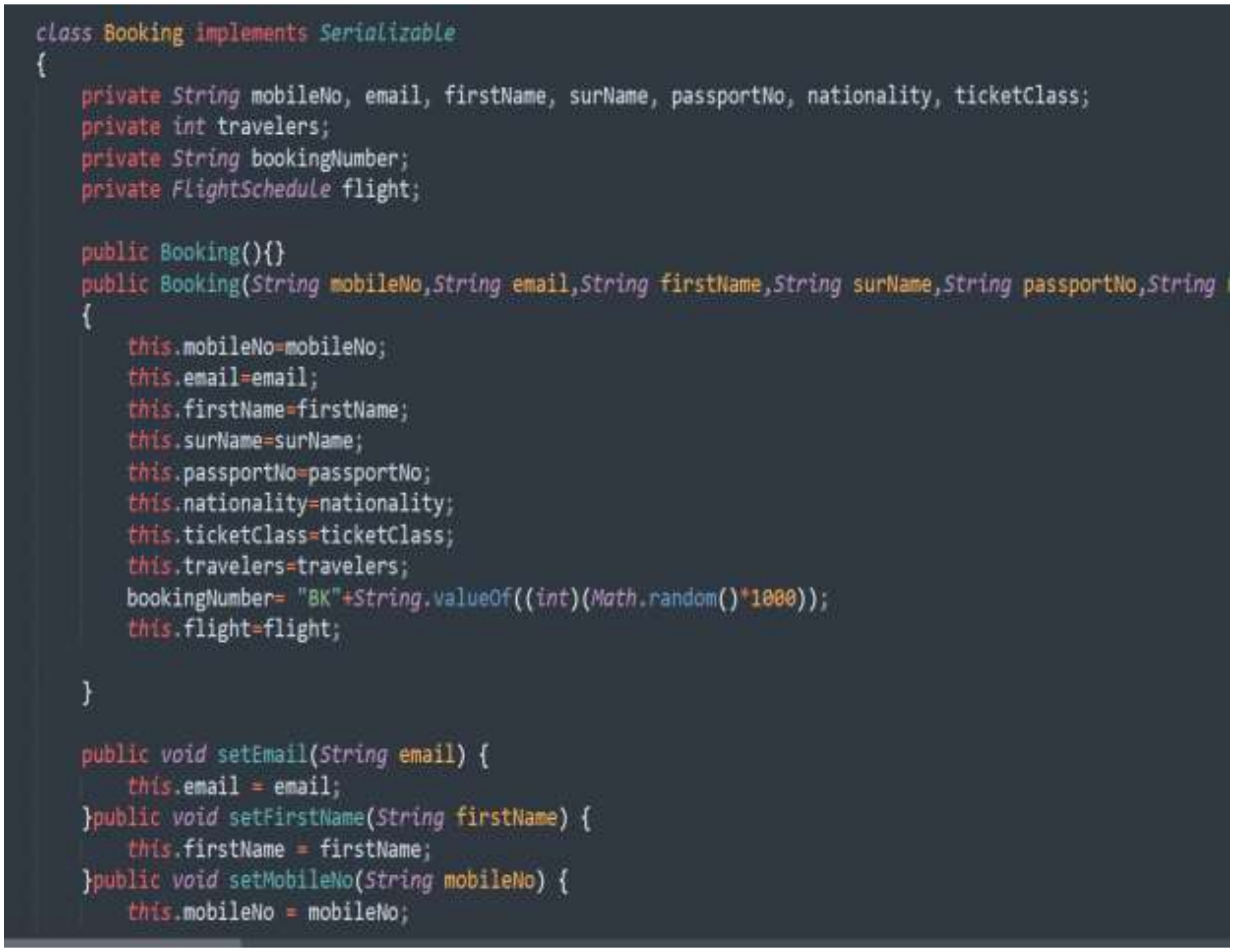
```java
private void display(JPanel mainPanel) {
    JPanel contentPanel = new JPanel();
    contentPanel.setLayout(new BoxLayout(contentPanel, BoxLayout.Y_AXIS));

    for (FlightSchedule f : schedules)
    {
        JPanel p = new JPanel();
        p.setPreferredSize(new Dimension(1000, 100));
        p.setLayout(new BorderLayout());
        p.setBackground(Color.BLACK);
        p.setBorder(BorderFactory.createMatteBorder(0, 0, 1, 0, Color.LIGHT_GRAY));

        JPanel leftP = new JPanel();
        leftP.setBackground(Color.CYAN);
        leftP.setPreferredSize(new Dimension(200, 100));
        leftP.setLayout(new BorderLayout());
        JLabel l = new JLabel(f.getAirline());
        l.setFont(new Font("Times New Roman", Font.ITALIC | Font.BOLD, 22));
        leftP.add(l);
        p.add(leftP, BorderLayout.WEST);

        JPanel innerP = new JPanel();
        innerP.setLayout(new BorderLayout());
        innerP.setPreferredSize(new Dimension(600, 100));
        l = new JLabel(f.getDepartureDate()+"    "+f.getDepartureTime() + "-" + f.getArrivalTime());
        l.setFont(new Font("Arial", Font.BOLD, 24));
        innerP.add(l, BorderLayout.NORTH);
        l = new JLabel(f.getDeparture() + "- " + f.getStops() + " Stops -" + f.getDestination());
        l.setFont(new Font("Arial", Font.PLAIN, 18));
```



```java
            JPanel rightP = new JPanel();
            rightP.setLayout(new GridLayout(2, 1));

            l = new JLabel("PKR " + f.getTicketPrice());
            l.setFont(new Font("Arial", Font.BOLD, 20));
            rightP.add(l);

            JButton book = new JButton("Book");
            book.setBackground(new Color(0, 102, 0));
            book.setForeground(Color.WHITE);
            book.setBorderPainted(false);
            book.setFocusPainted(false);
            book.addMouseListener(new MouseAdapter() {
                public void mouseEntered(MouseEvent e)
                {
                    book.setBackground(new Color(0, 51, 0));
                }
                public void mouseExited(MouseEvent e)
                {
                    book.setBackground(new Color(0, 102, 0));
                }
            });
            rightP.add(book);
            p.add(rightP, BorderLayout.EAST);

            book.addActionListener(e->
            {
                new BookingGui(f);
                dispose();
```

The `display` method constructs a dynamic UI panel to showcase flight schedules. It iterates through a list of `**FlightSchedule**` objects (`schedules`). For each flight schedule, it creates a `JPanel` with distinct sections for airline details, flight times, stops, destination, and ticket price. The panel uses various layouts and styling elements to organize information effectively. The book button, when clicked, triggers the creation of a new `BookingGui` instance for the corresponding flight. The method dynamically adds these flight panels to the main panel (`mainPanel`) within a vertical layout. Overall, it generates a visually appealing and functional display of flight schedules with booking functionality.

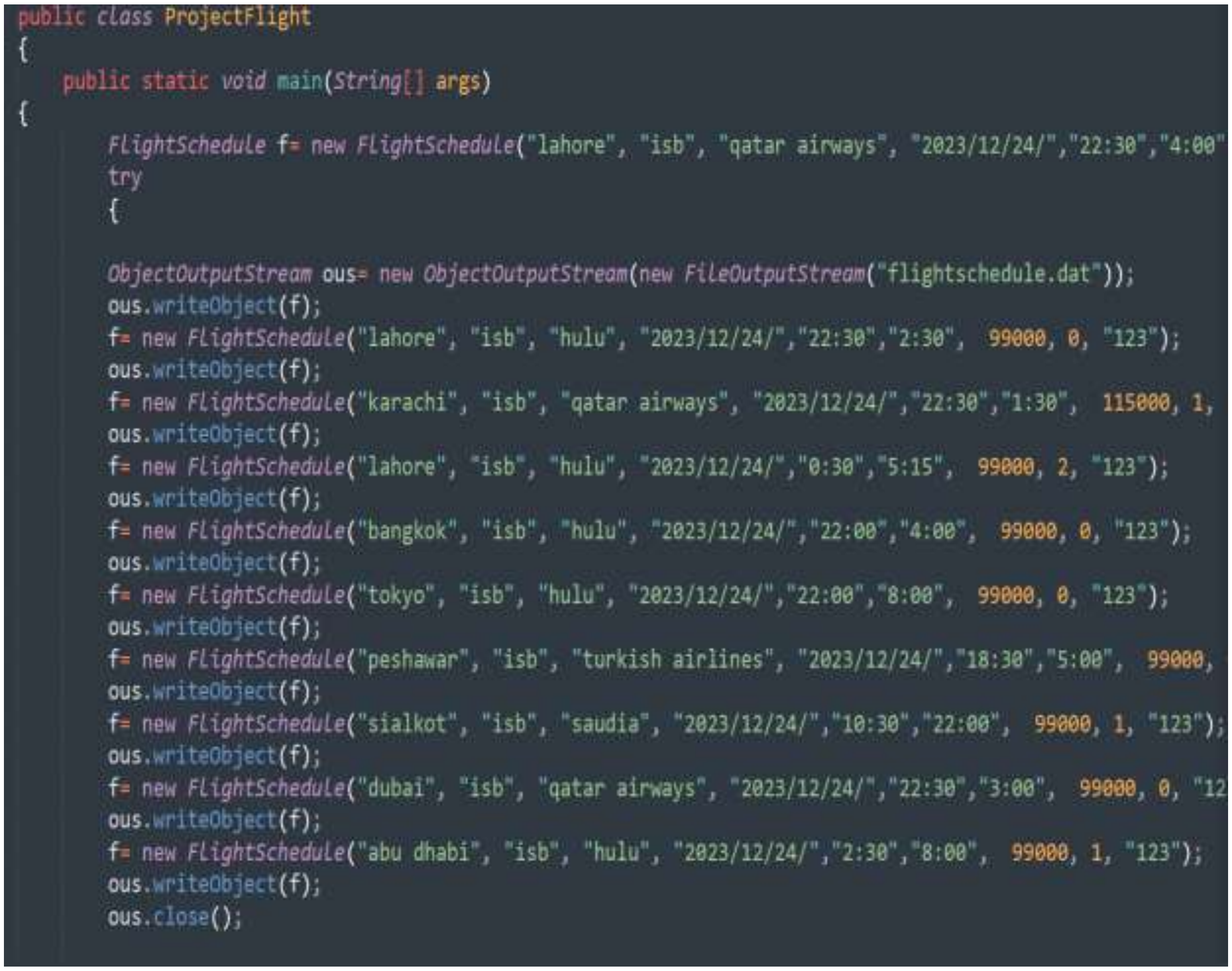
- **Class Booking**



```java
class Booking implements Serializable
{
    private String mobileNo, email, firstName, surName, passportNo, nationality, ticketClass;
    private int travelers;
    private String bookingNumber;
    private FlightSchedule flight;

    public Booking(){}
    public Booking(String mobileNo,String email,String firstName,String surName,String passportNo,String
    {
        this.mobileNo=mobileNo;
        this.email=email;
        this.firstName=firstName;
        this.surName=surName;
        this.passportNo=passportNo;
        this.nationality=nationality;
        this.ticketClass=ticketClass;
        this.travelers=travelers;
        bookingNumber= "BK"+String.valueOf((int)(Math.random()*1000));
        this.flight=flight;

    }

    public void setEmail(String email) {
        this.email = email;
    }public void setFirstName(String firstName) {
        this.firstName = firstName;
    }public void setMobileNo(String mobileNo) {
        this.mobileNo = mobileNo;
```

The `**Booking**` class represents a flight booking and implements the `Serializable` interface for object serialization. It contains attributes such as `mobileNo`, `email`, `firstName`, `surName`, `passportNo`, `nationality`, `ticketClass`, `travelers`, `**bookingNumber**`, and a reference to the associated `**FlightSchedule**`. The class provides constructors for creating a booking and setters for modifying its attributes. The `writeBooking` method serializes a `Booking` object and appends it to a file named "bookings.dat." The `toString` method generates a string representation of the booking, including details like booking number, passenger information, and associated flight information.

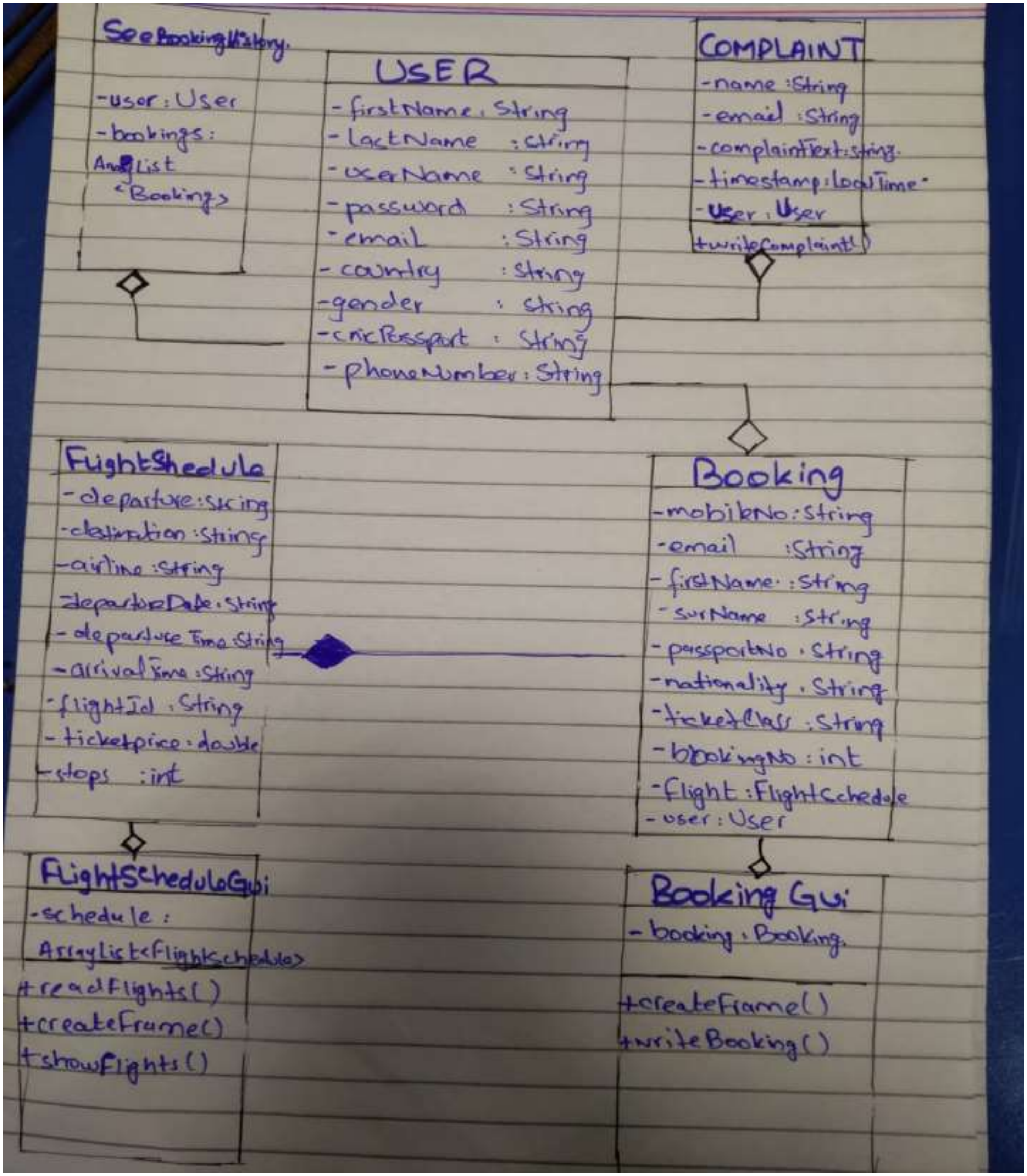
- **Class ProjectFlight**



```
public class ProjectFlight
{
    public static void main(String[] args)
{
        FlightSchedule f= new FlightSchedule("lahore", "isb", "qatar airways", "2023/12/24/","22:30","4:00"
        try
        {

        ObjectOutputStream ous= new ObjectOutputStream(new FileOutputStream("flightschedule.dat"));
        ous.writeObject(f);
        f= new FlightSchedule("lahore", "isb", "hulu", "2023/12/24/","22:30","2:30",  99000, 0, "123");
        ous.writeObject(f);
        f= new FlightSchedule("karachi", "isb", "qatar airways", "2023/12/24/","22:30","1:30",  115000, 1,
        ous.writeObject(f);
        f= new FlightSchedule("lahore", "isb", "hulu", "2023/12/24/","0:30","5:15",  99000, 2, "123");
        ous.writeObject(f);
        f= new FlightSchedule("bangkok", "isb", "hulu", "2023/12/24/","22:00","4:00",  99000, 0, "123");
        ous.writeObject(f);
        f= new FlightSchedule("tokyo", "isb", "hulu", "2023/12/24/","22:00","8:00",  99000, 0, "123");
        ous.writeObject(f);
        f= new FlightSchedule("peshawar", "isb", "turkish airlines", "2023/12/24/","18:30","5:00",  99000,
        ous.writeObject(f);
        f= new FlightSchedule("sialkot", "isb", "saudia", "2023/12/24/","10:30","22:00",  99000, 1, "123");
        ous.writeObject(f);
        f= new FlightSchedule("dubai", "isb", "qatar airways", "2023/12/24/","22:30","3:00",  99000, 0, "12
        ous.writeObject(f);
        f= new FlightSchedule("abu dhabi", "isb", "hulu", "2023/12/24/","2:30","8:00",  99000, 1, "123");
        ous.writeObject(f);
        ous.close();
```

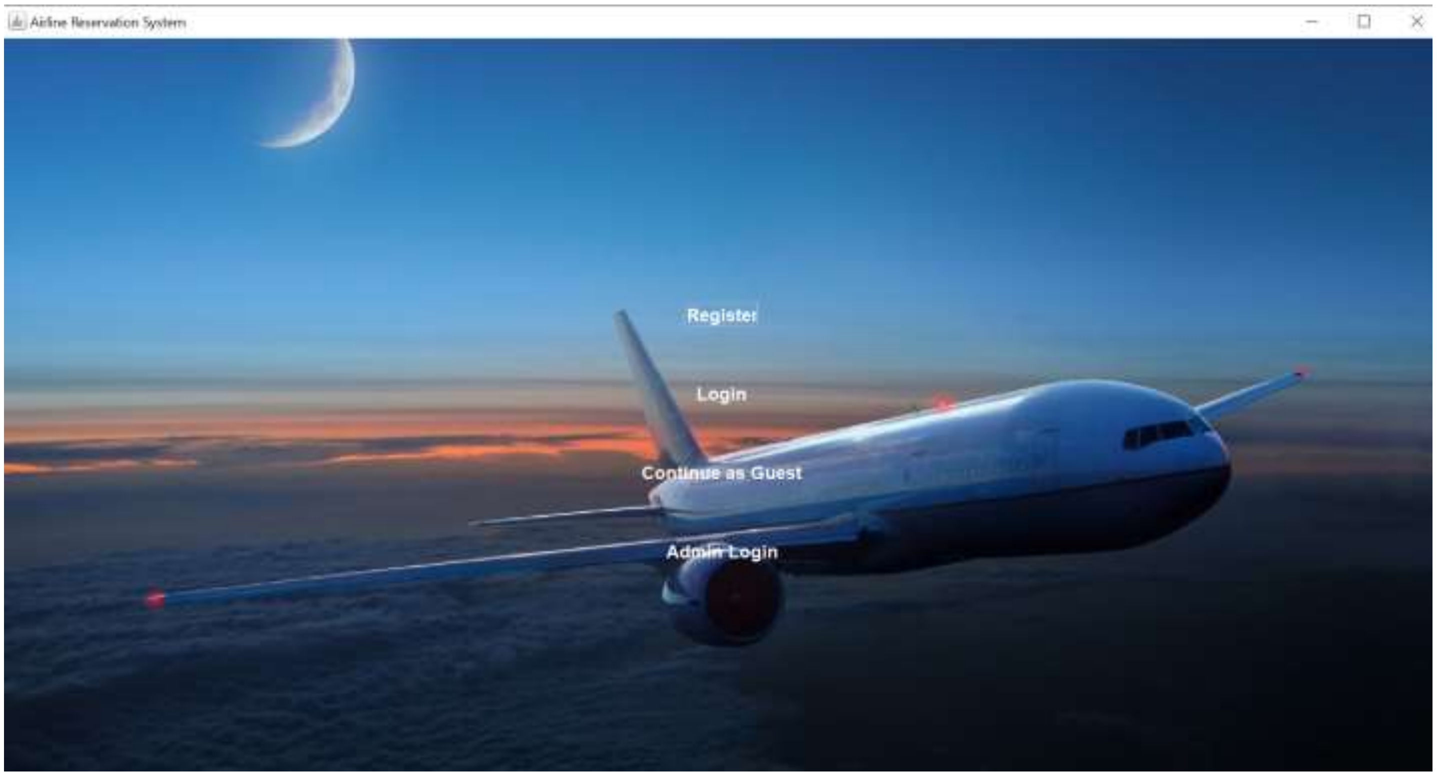
The `**ProjectFlight**` class serves as the main entry point for the flight reservation system. In the `main` method, it initializes several `FlightSchedule` objects representing different flight schedules and writes them to a file named "flightschedule.dat" using `**ObjectOutputStream**`. Subsequently, it reads and displays the details of the first two flight schedules from the file using `**ObjectInputStream**`. Finally, it creates an instance of the `**MainMenuProject**` class, initiating the graphical user interface for the flight reservation system. This class demonstrates the serialization and deserialization of flight schedules and the integration of the main menu into the project.

UML Class Diagram

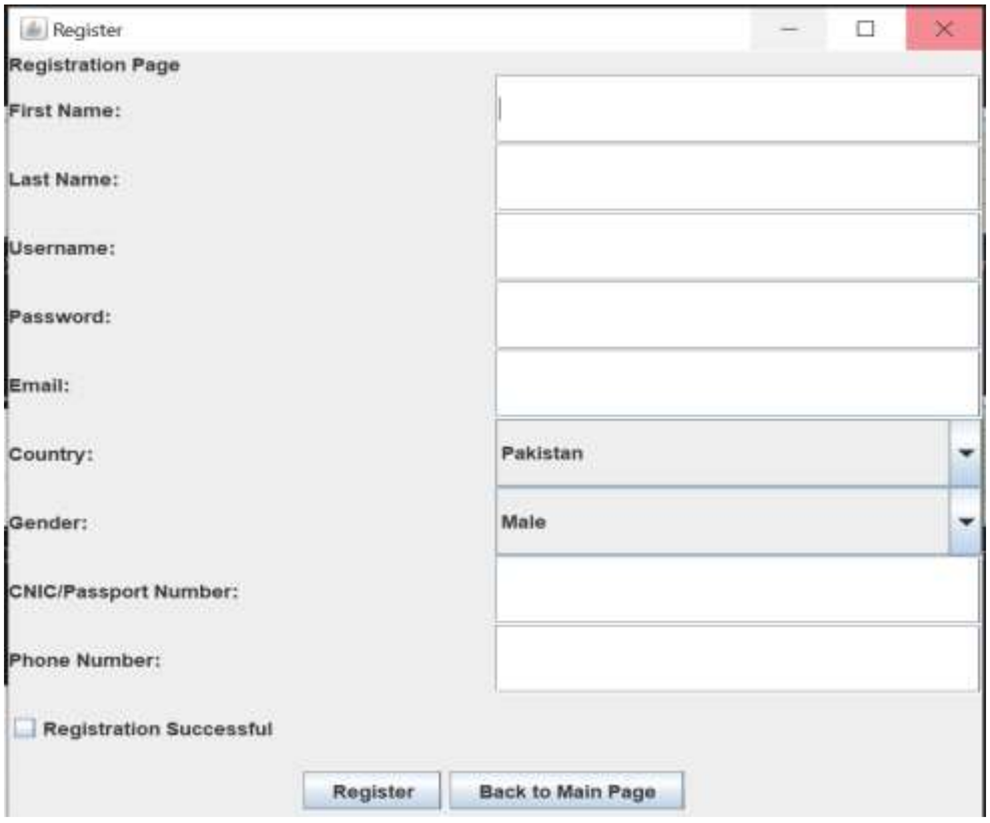


OUTPUT

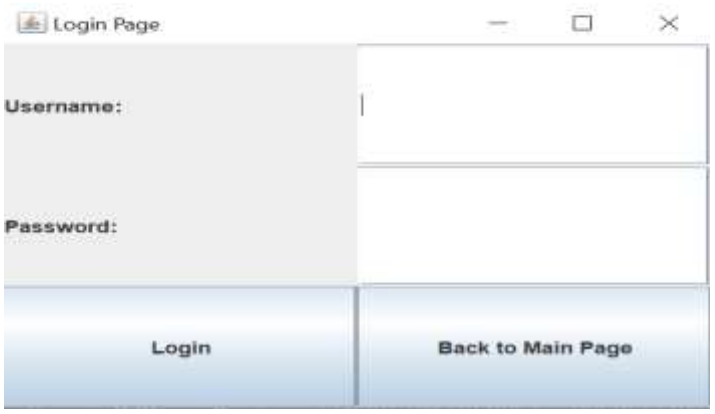
Main Page:



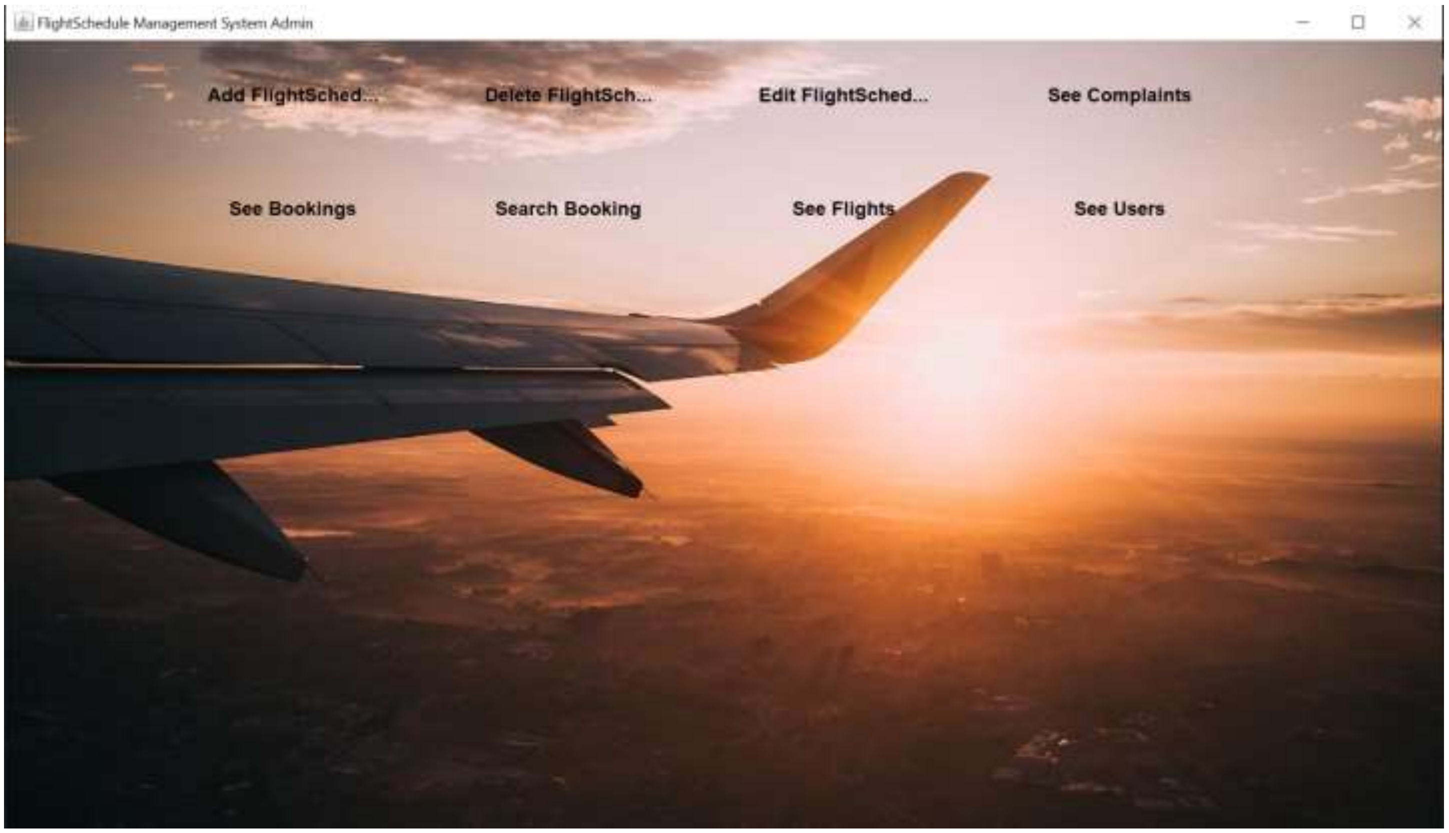
User Registration Window:



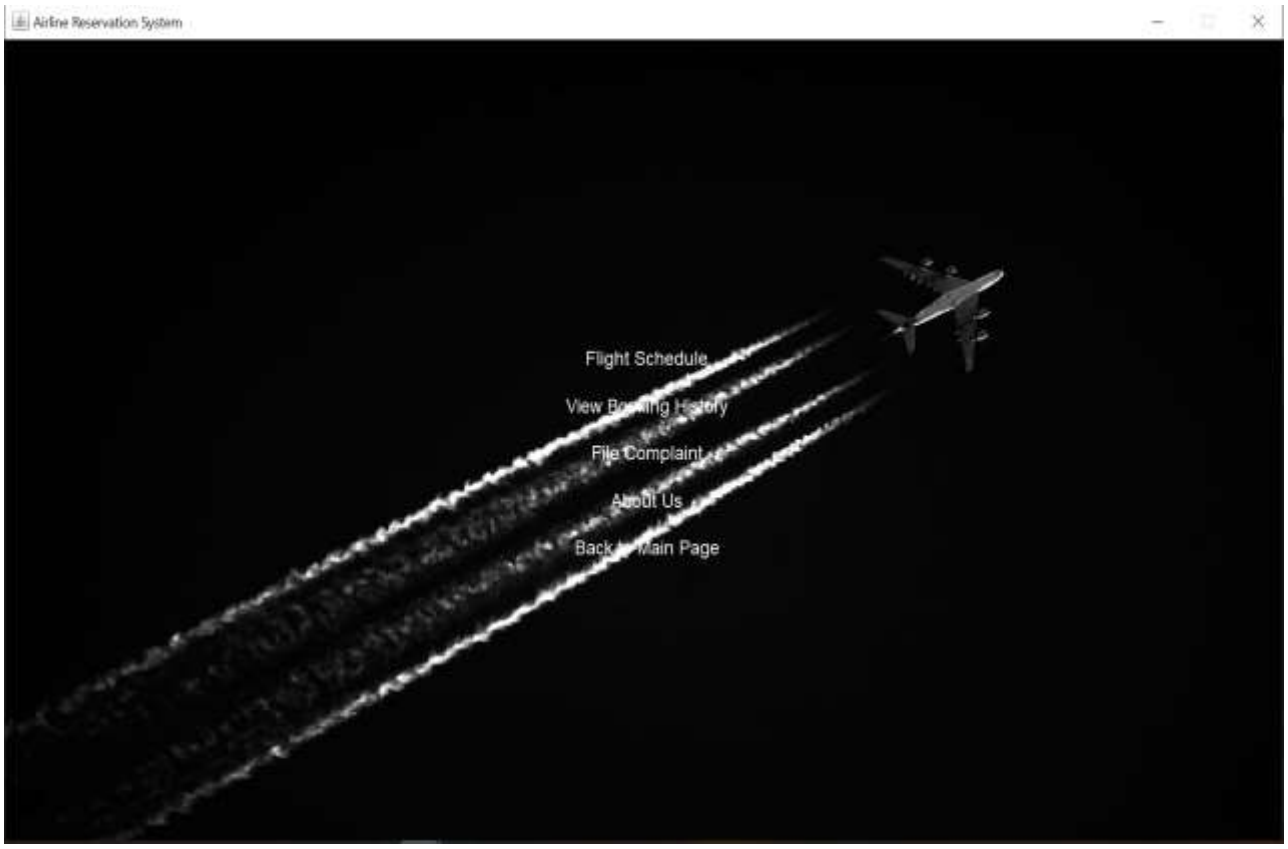
Admin Login Window:



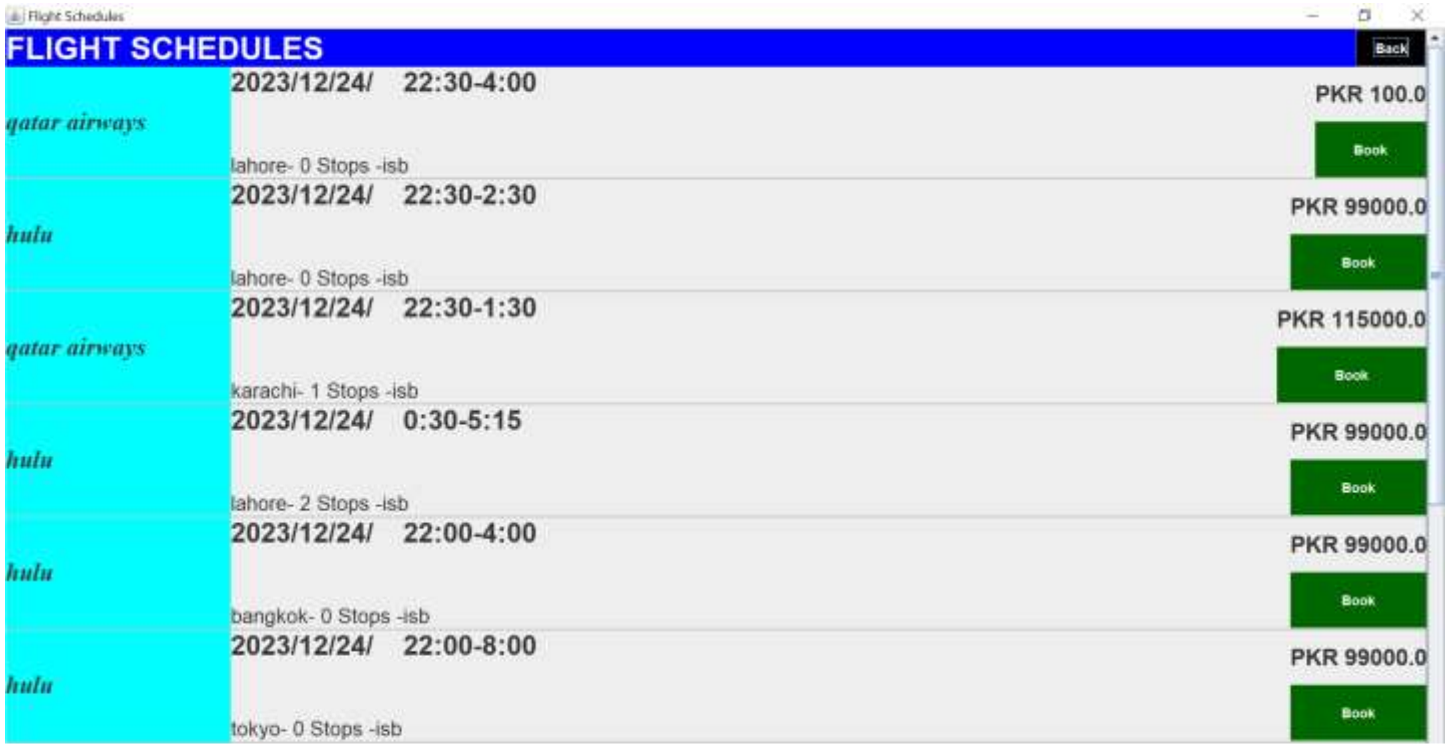
Admin Page:



User Menu:



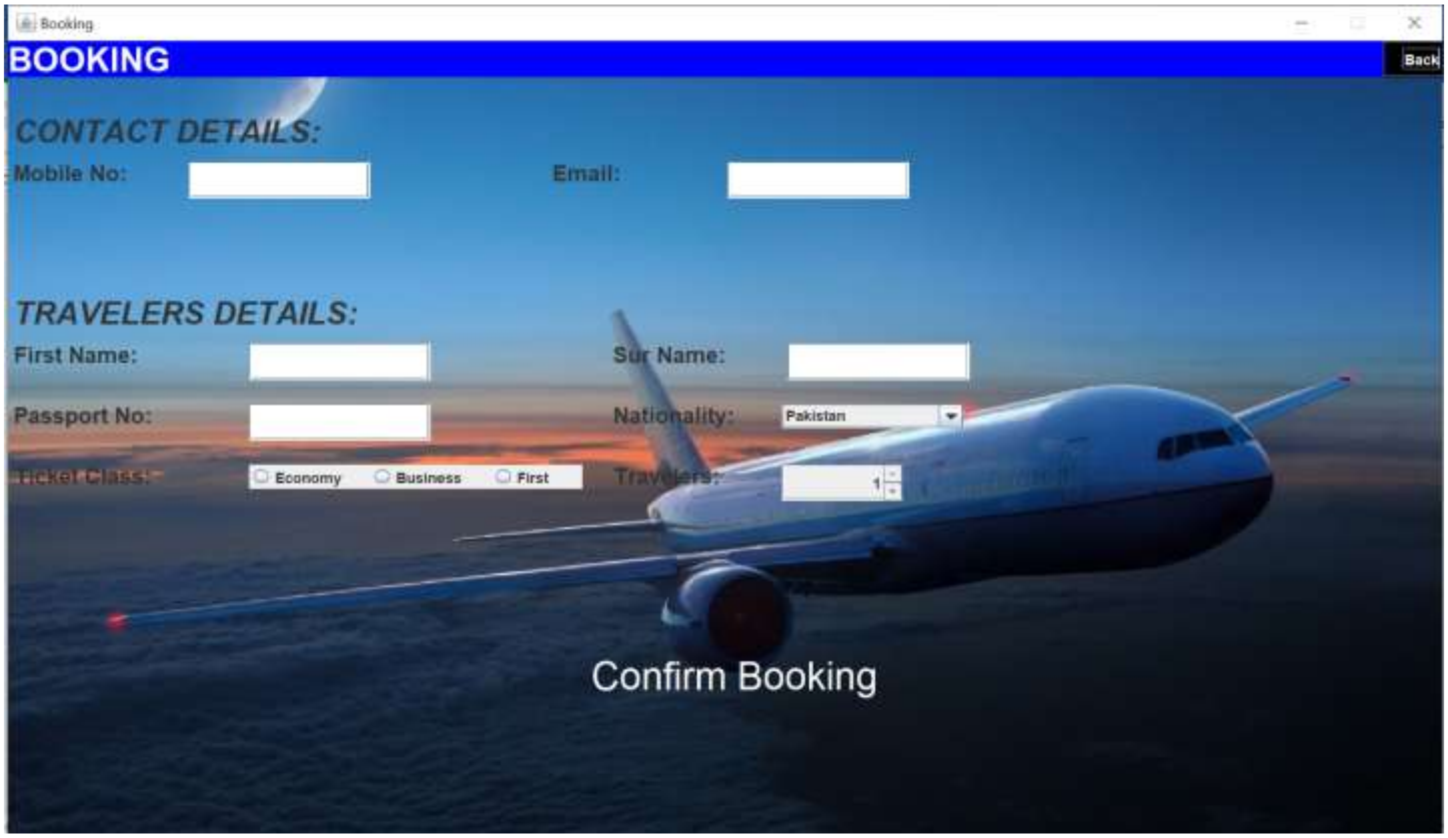
Flight Schedules Page:



FLIGHT SCHEDULES

qatar airways	2023/12/24/ 22:30-4:00 lahore- 0 Stops -isb	PKR 100.0
hulu	2023/12/24/ 22:30-2:30 lahore- 0 Stops -isb	PKR 99000.0
qatar airways	2023/12/24/ 22:30-1:30 karachi- 1 Stops -isb	PKR 115000.0
hulu	2023/12/24/ 0:30-5:15 lahore- 2 Stops -isb	PKR 99000.0
hulu	2023/12/24/ 22:00-4:00 bangkok- 0 Stops -isb	PKR 99000.0
hulu	2023/12/24/ 22:00-8:00 tokyo- 0 Stops -isb	PKR 99000.0

Booking Window:


Booking
BOOKING
Back
CONTACT DETAILS:
Mobile No:
Email:
TRAVELERS DETAILS:
First Name:
Sur Name:
Passport No:
Nationality:
Pakistan
Ticket Class:
Economy
Business
First
Travelers:
1
Confirm Booking