Java Airlines™

Group:

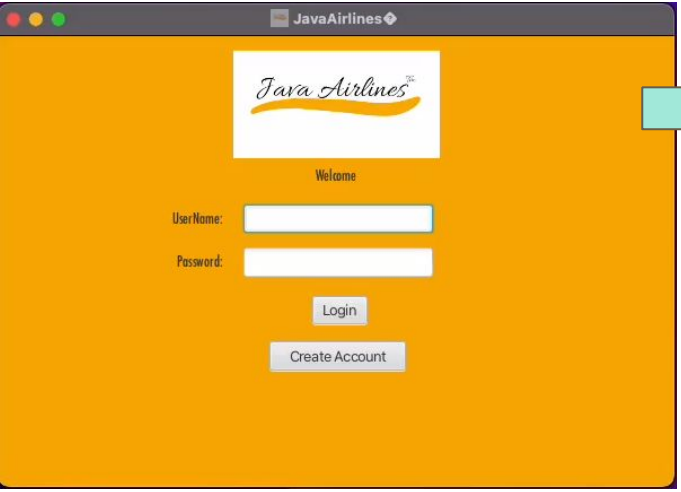
Miguel Datoc
Madeline Luna
Reenu Mohan
Sergio Guido

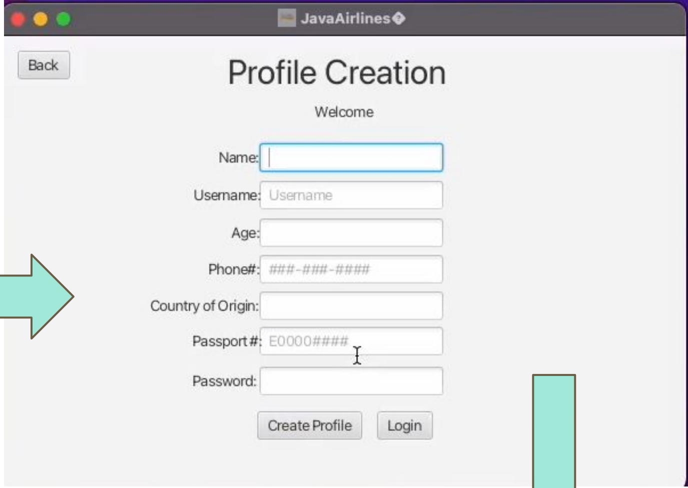
Project Description

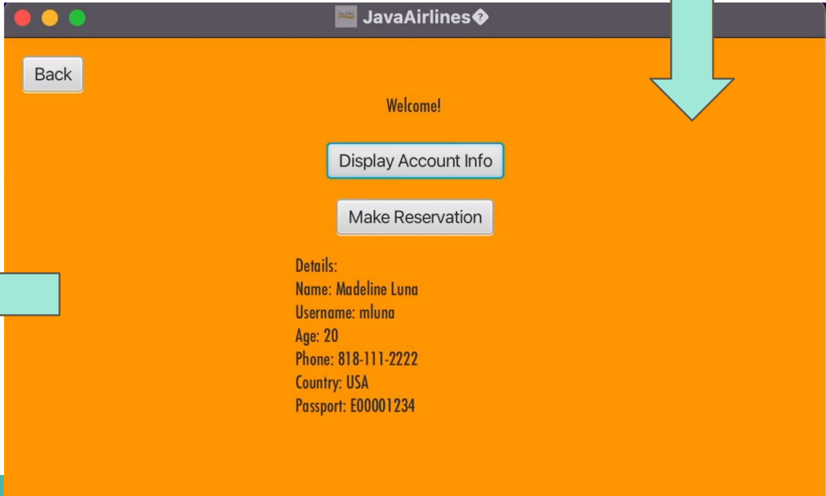
- ➔ Software that performs on-line reservation tasks for an airline company. The project involves a program that handles the reservation process for the company, **Java Airlines™**.
- ➔ Practical in real-life scenarios for online reservations, mimicking popular reservation software/websites such as Expedia, Kayak, or other airline sites.

Program Description

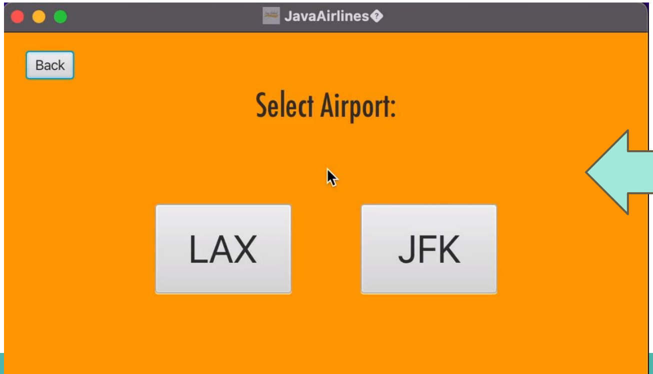
- ➔ Customer functions part 1





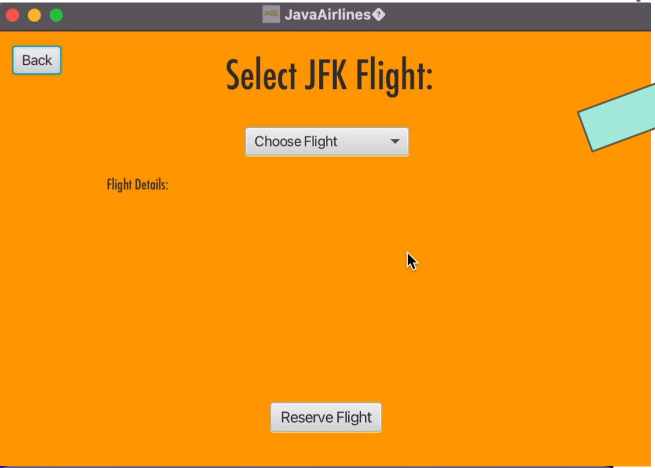


Details:
Name: Madeline Luna
Username: mluna
Age: 20
Phone: 818-111-2222
Country: USA
Passport: E00001234



Program Description

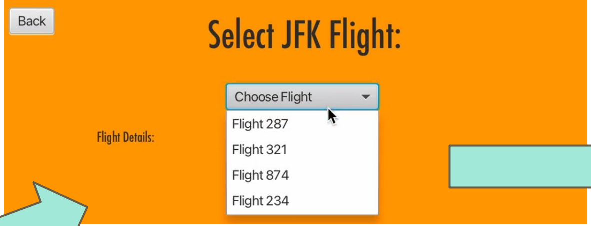
➔ Customer functions part 2



Select JFK Flight:

Choose Flight

Flight Details:

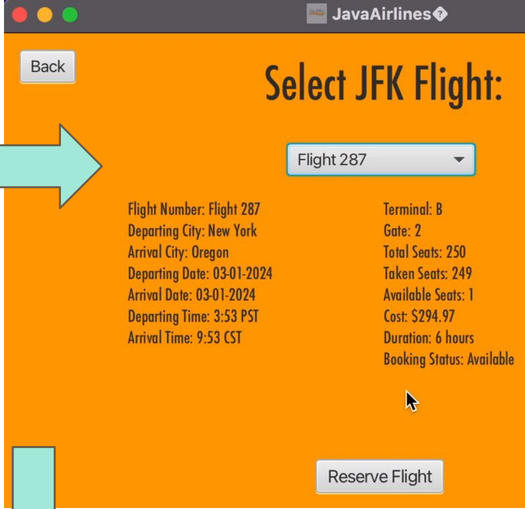


Select JFK Flight:

Choose Flight

Flight Details:

Flight 287
Flight 321
Flight 874
Flight 234

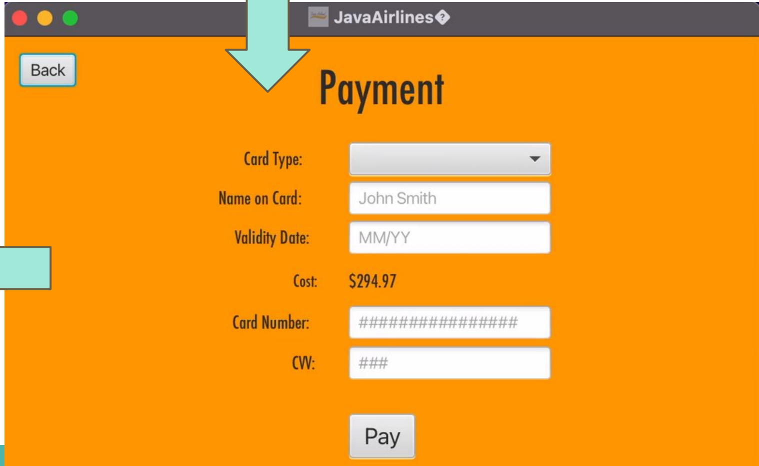


Select JFK Flight:

Flight 287

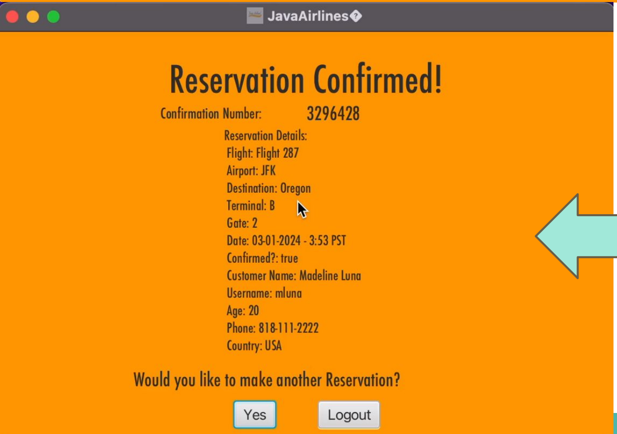
Flight Number: Flight 287
Departing City: New York
Arrival City: Oregon
Departing Date: 03-01-2024
Arrival Date: 03-01-2024
Departing Time: 3:53 PST
Arrival Time: 9:53 CST

Terminal: B
Gate: 2
Total Seats: 250
Taken Seats: 249
Available Seats: 1
Cost: $294.97
Duration: 6 hours
Booking Status: Available



Payment

Card Type:
Name on Card: John Smith
Validity Date: MM/YY
Cost: $294.97
Card Number: ################
CVV: ###



Reservation Confirmed!

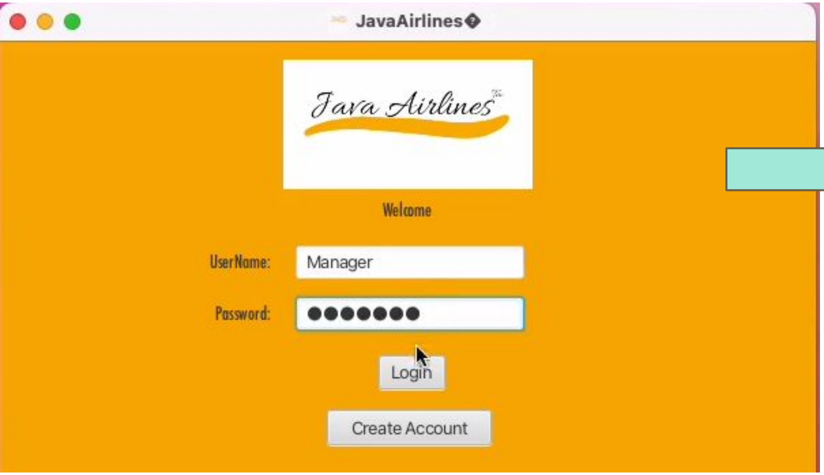
Confirmation Number: 3296428

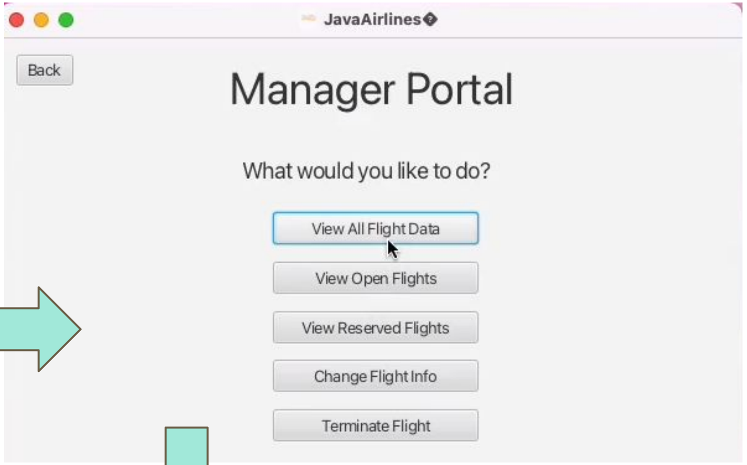
Reservation Details:
Flight: Flight 287
Airport: JFK
Destination: Oregon
Terminal: B
Gate: 2
Date: 03-01-2024 - 3:53 PST
Confirmed?: true
Customer Name: Madeline Luna
Username: mluna
Age: 20
Phone: 818-111-2222
Country: USA

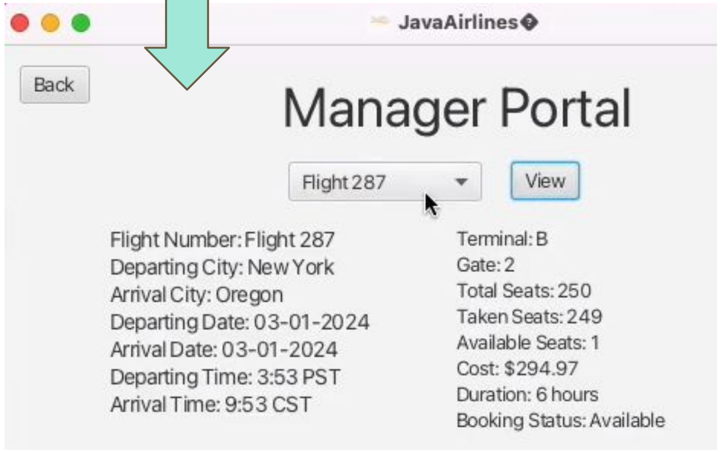
Would you like to make another Reservation?

Program Description

➔ Manager functions



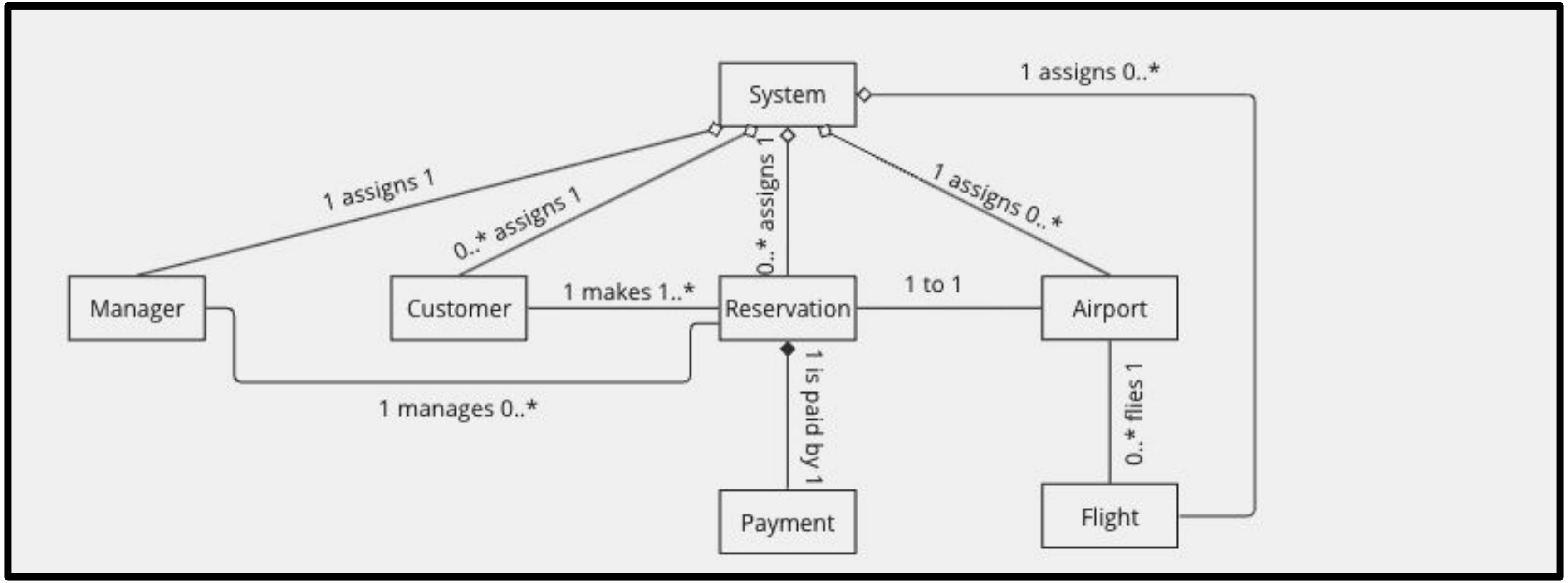




Flight Number: Flight 287
Departing City: New York
Arrival City: Oregon
Departing Date: 03-01-2024
Arrival Date: 03-01-2024
Departing Time: 3:53 PST
Arrival Time: 9:53 CST

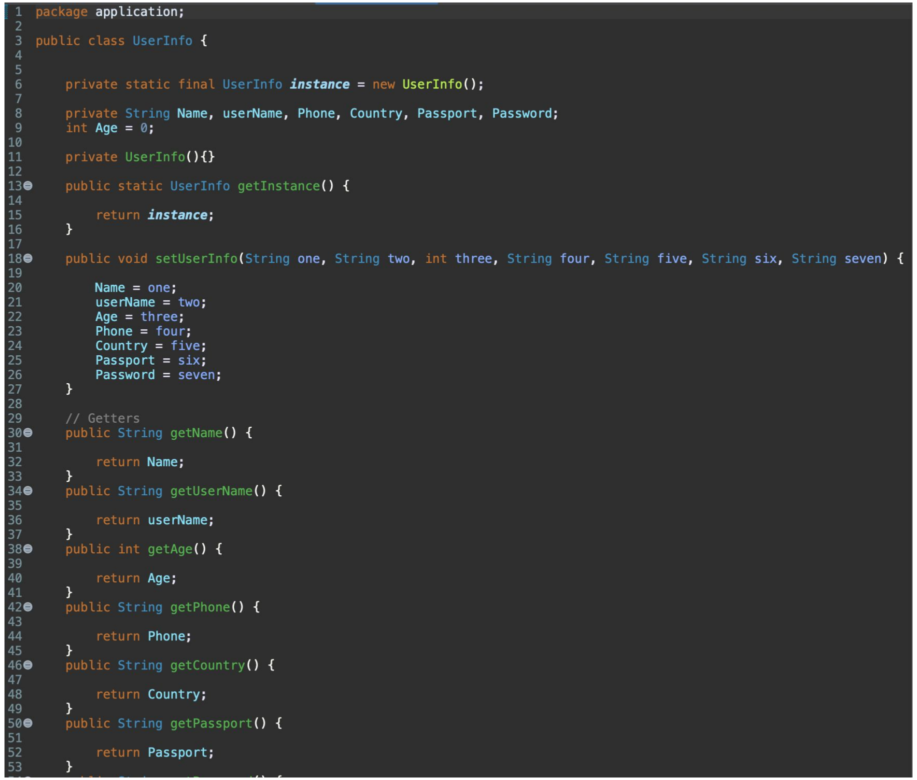
Terminal: B
Gate: 2
Total Seats: 250
Taken Seats: 249
Available Seats: 1
Cost: $294.97
Duration: 6 hours
Booking Status: Available

UML Class Diagram



User Info Class

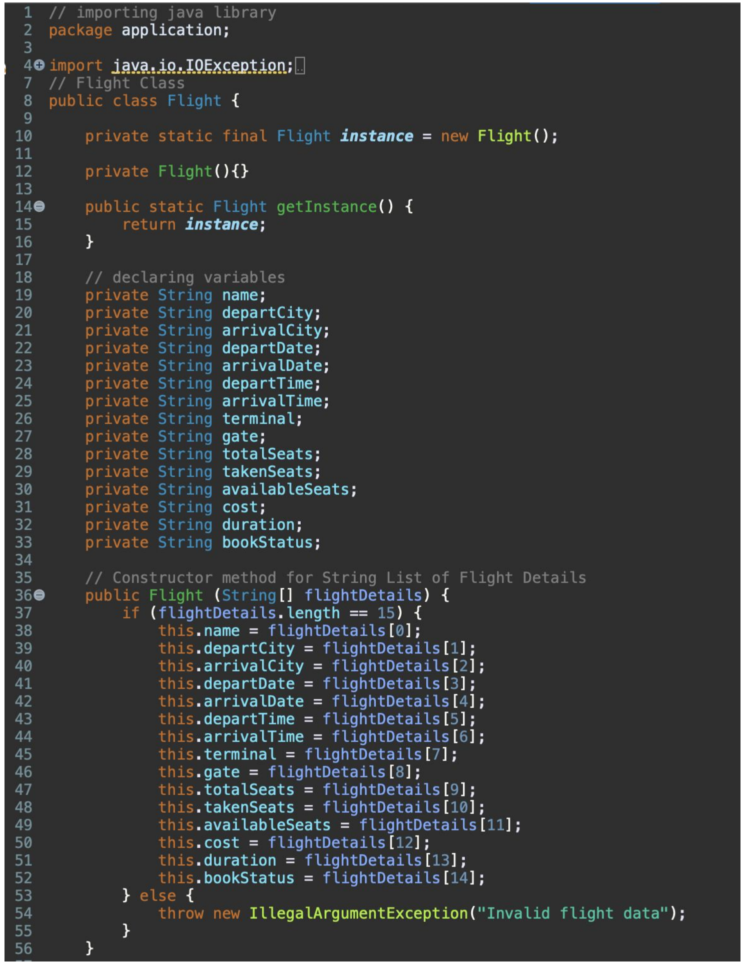
➔ Responsible for creating and holding customer information, including:
- ◆ Name
- ◆ Username
- ◆ Age
- ◆ Phone number
- ◆ Country of origin
- ◆ Passport number
- ◆ Password



```java
package application;

public class UserInfo {


    private static final UserInfo instance = new UserInfo();

    private String Name, userName, Phone, Country, Passport, Password;
    int Age = 0;

    private UserInfo(){}

    public static UserInfo getInstance() {

        return instance;
    }

    public void setUserInfo(String one, String two, int three, String four, String five, String six, String seven) {

        Name = one;
        userName = two;
        Age = three;
        Phone = four;
        Country = five;
        Passport = six;
        Password = seven;
    }

    // Getters
    public String getName() {

        return Name;
    }
    public String getUserName() {

        return userName;
    }
    public int getAge() {

        return Age;
    }
    public String getPhone() {

        return Phone;
    }
    public String getCountry() {

        return Country;
    }
    public String getPassport() {

        return Passport;
    }
```

Flight Class

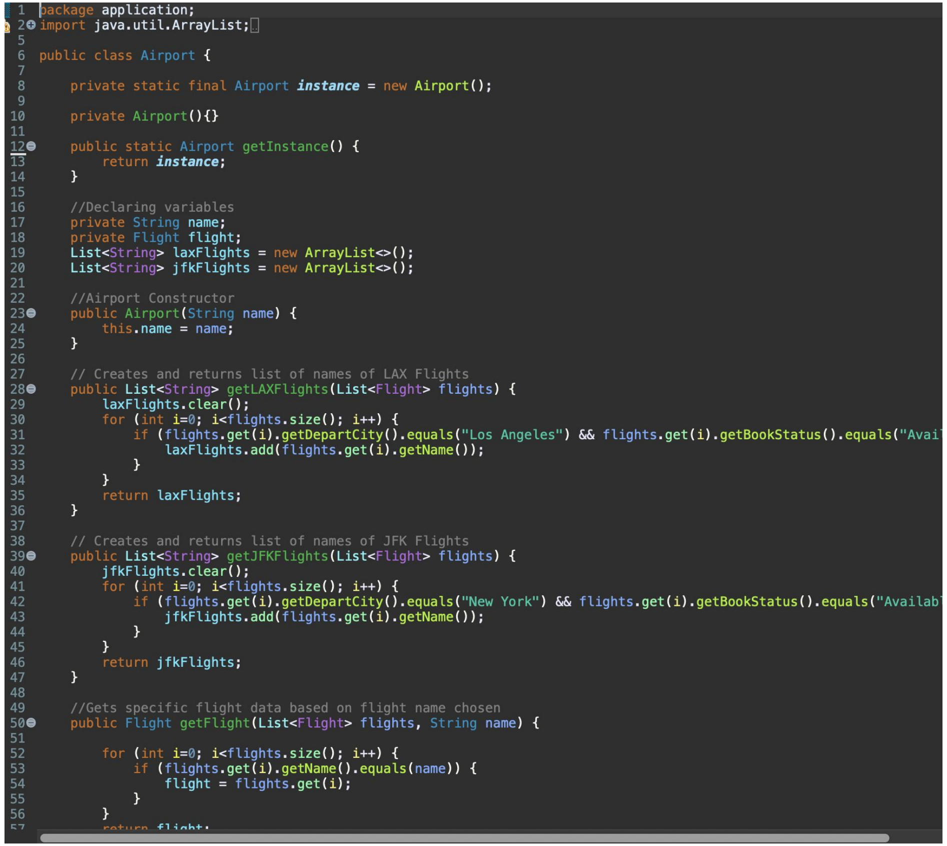
➔ Responsible for initializing and holding a flight and its flight information, including:
  - Flight number
  - Departure City
  - Arrival City
  - Depart Date
  - Arrival Date
  - Depart Time
  - Arrival Time
  - Boarding terminal
  - Boarding gate
  - Seat capacity
  - Number of taken seats
  - Number of available seats
  - Cost
  - Flight duration
  - Booking status



```java
// importing java library
package application;

import java.io.IOException;
// Flight Class
public class Flight {

    private static final Flight instance = new Flight();

    private Flight(){}

    public static Flight getInstance() {
        return instance;
    }

    // declaring variables
    private String name;
    private String departCity;
    private String arrivalCity;
    private String departDate;
    private String arrivalDate;
    private String departTime;
    private String arrivalTime;
    private String terminal;
    private String gate;
    private String totalSeats;
    private String takenSeats;
    private String availableSeats;
    private String cost;
    private String duration;
    private String bookStatus;

    // Constructor method for String List of Flight Details
    public Flight (String[] flightDetails) {
        if (flightDetails.length == 15) {
            this.name = flightDetails[0];
            this.departCity = flightDetails[1];
            this.arrivalCity = flightDetails[2];
            this.departDate = flightDetails[3];
            this.arrivalDate = flightDetails[4];
            this.departTime = flightDetails[5];
            this.arrivalTime = flightDetails[6];
            this.terminal = flightDetails[7];
            this.gate = flightDetails[8];
            this.totalSeats = flightDetails[9];
            this.takenSeats = flightDetails[10];
            this.availableSeats = flightDetails[11];
            this.cost = flightDetails[12];
            this.duration = flightDetails[13];
            this.bookStatus = flightDetails[14];
        } else {
            throw new IllegalArgumentException("Invalid flight data");
        }
    }
```

Airport Class

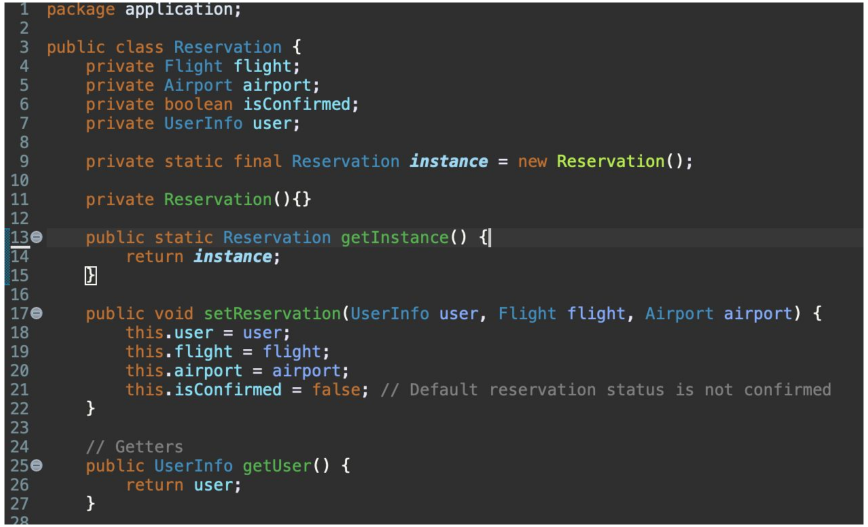
- ➔ Responsible for initializing and creating LAX and JFK airport instance
- ➔ Holds information for each individual airport, including:
  - ◆ Airport name
  - ◆ An array list of flights and their data associated with the airport



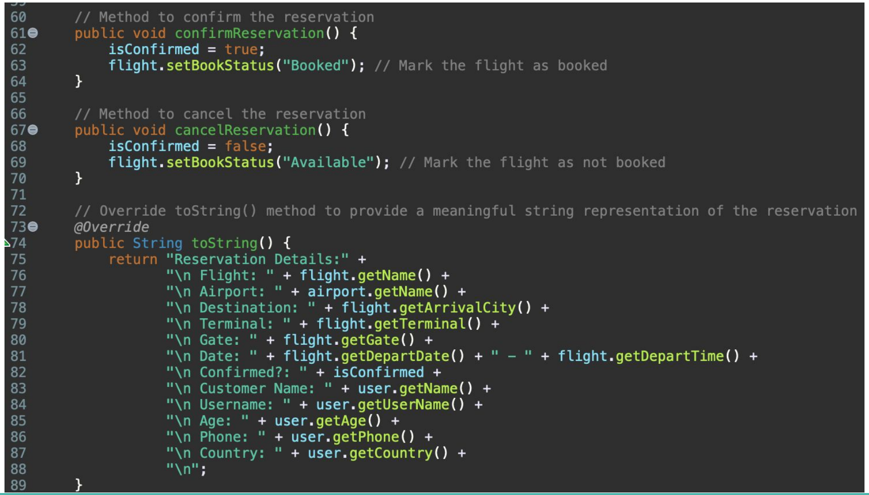
```java
package application;
import java.util.ArrayList;

public class Airport {

    private static final Airport instance = new Airport();

    private Airport(){}

    public static Airport getInstance() {
        return instance;
    }

    //Declaring variables
    private String name;
    private Flight flight;
    List<String> laxFlights = new ArrayList<>();
    List<String> jfkFlights = new ArrayList<>();

    //Airport Constructor
    public Airport(String name) {
        this.name = name;
    }

    // Creates and returns list of names of LAX Flights
    public List<String> getLAXFlights(List<Flight> flights) {
        laxFlights.clear();
        for (int i=0; i<flights.size(); i++) {
            if (flights.get(i).getDepartCity().equals("Los Angeles") && flights.get(i).getBookStatus().equals("Avai
                laxFlights.add(flights.get(i).getName());
            }
        }
        return laxFlights;
    }

    // Creates and returns list of names of JFK Flights
    public List<String> getJFKFlights(List<Flight> flights) {
        jfkFlights.clear();
        for (int i=0; i<flights.size(); i++) {
            if (flights.get(i).getDepartCity().equals("New York") && flights.get(i).getBookStatus().equals("Availab
                jfkFlights.add(flights.get(i).getName());
            }
        }
        return jfkFlights;
    }

    //Gets specific flight data based on flight name chosen
    public Flight getFlight(List<Flight> flights, String name) {

        for (int i=0; i<flights.size(); i++) {
            if (flights.get(i).getName().equals(name)) {
                flight = flights.get(i);
            }
        }
        return flight;
```

Reservation Class

➔ Responsible for creating a reservation and holding its data, including:
- ◆ User info of the customer booking the flight
- ◆ The selected flight
- ◆ The assigned airport
- ◆ Whether the reservation has been confirmed (as determined by the assigned payment)



```java
package application;

public class Reservation {
    private Flight flight;
    private Airport airport;
    private boolean isConfirmed;
    private UserInfo user;

    private static final Reservation instance = new Reservation();

    private Reservation(){}

    public static Reservation getInstance() {
        return instance;
    }

    public void setReservation(UserInfo user, Flight flight, Airport airport) {
        this.user = user;
        this.flight = flight;
        this.airport = airport;
        this.isConfirmed = false; // Default reservation status is not confirmed
    }

    // Getters
    public UserInfo getUser() {
        return user;
    }
```

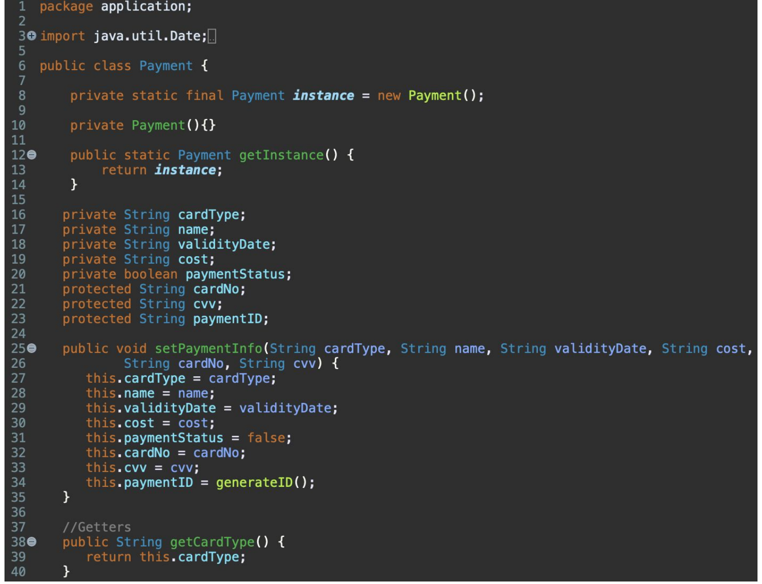


```java
    // Method to confirm the reservation
    public void confirmReservation() {
        isConfirmed = true;
        flight.setBookStatus("Booked"); // Mark the flight as booked
    }

    // Method to cancel the reservation
    public void cancelReservation() {
        isConfirmed = false;
        flight.setBookStatus("Available"); // Mark the flight as not booked
    }

    // Override toString() method to provide a meaningful string representation of the reservation
    @Override
    public String toString() {
        return "Reservation Details:" +
               "\n Flight: " + flight.getName() +
               "\n Airport: " + airport.getName() +
               "\n Destination: " + flight.getArrivalCity() +
               "\n Terminal: " + flight.getTerminal() +
               "\n Gate: " + flight.getGate() +
               "\n Date: " + flight.getDepartDate() + " - " + flight.getDepartTime() +
               "\n Confirmed?: " + isConfirmed +
               "\n Customer Name: " + user.getName() +
               "\n Username: " + user.getUserName() +
               "\n Age: " + user.getAge() +
               "\n Phone: " + user.getPhone() +
               "\n Country: " + user.getCountry() +
               "\n";
    }
```

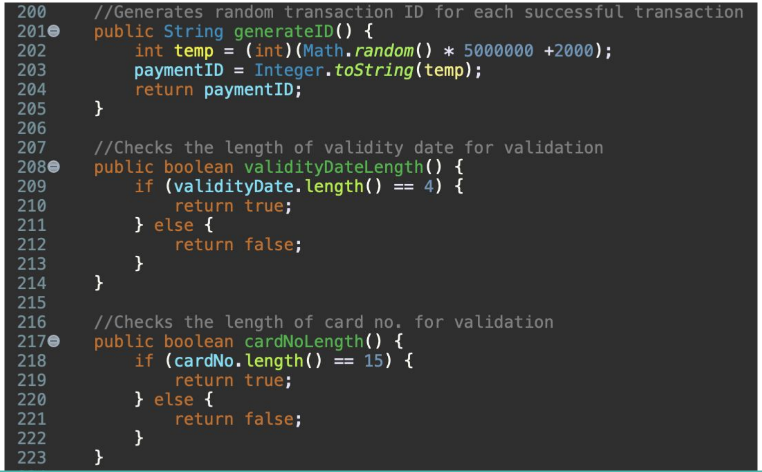
Payment Class

➔ Responsible for making a payment and holding the payment details including:

- Card Type
- Name of the cardholder
- Validity date of the card (also checks length)
- Cost of the flight
- Card no. (also checks length)
- CVV (also checks length)
- Payment ID (Generated at time of successful payment transaction.
- Payment status (Set to true when successful payment transaction.



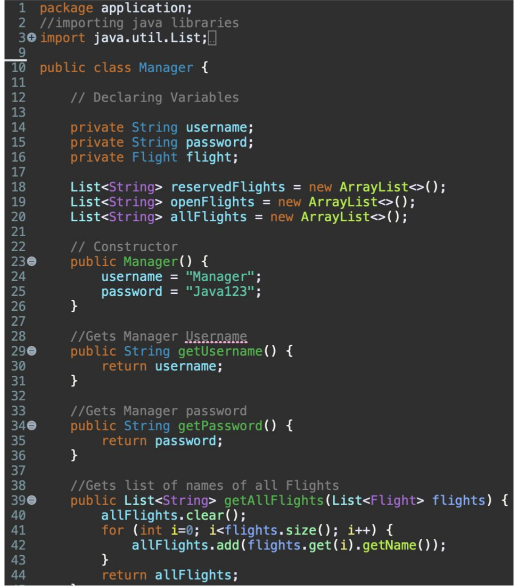
```java
package application;

import java.util.Date;

public class Payment {

    private static final Payment instance = new Payment();

    private Payment(){}

    public static Payment getInstance() {
        return instance;
    }

    private String cardType;
    private String name;
    private String validityDate;
    private String cost;
    private boolean paymentStatus;
    protected String cardNo;
    protected String cvv;
    protected String paymentID;

    public void setPaymentInfo(String cardType, String name, String validityDate, String cost,
            String cardNo, String cvv) {
        this.cardType = cardType;
        this.name = name;
        this.validityDate = validityDate;
        this.cost = cost;
        this.paymentStatus = false;
        this.cardNo = cardNo;
        this.cvv = cvv;
        this.paymentID = generateID();
    }

    //Getters
    public String getCardType() {
        return this.cardType;
    }
```



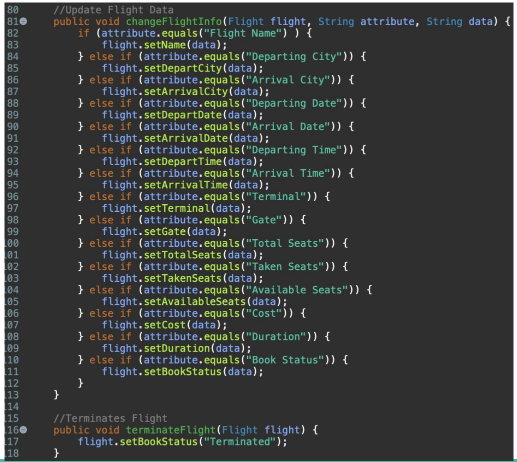
```java
    //Generates random transaction ID for each successful transaction
    public String generateID() {
        int temp = (int)(Math.random() * 5000000 +2000);
        paymentID = Integer.toString(temp);
        return paymentID;
    }

    //Checks the length of validity date for validation
    public boolean validityDateLength() {
        if (validityDate.length() == 4) {
            return true;
        } else {
            return false;
        }
    }

    //Checks the length of card no. for validation
    public boolean cardNoLength() {
        if (cardNo.length() == 15) {
            return true;
        } else {
            return false;
        }
    }
```

Manager Class

➔ Responsible for managing the reservations, and generating the reports of flights including:
  - ◆ List of all, open, and reserved flights (Booking status of flights)
  - ◆ Lists all classes in order to change data on flight selected. (Changes are then forwarded to the text file.)
  - ◆ Able to terminate flights by changing bookStatus → "Terminated"
  - ◆ Has predetermined username and password to access the Manager's control screen.

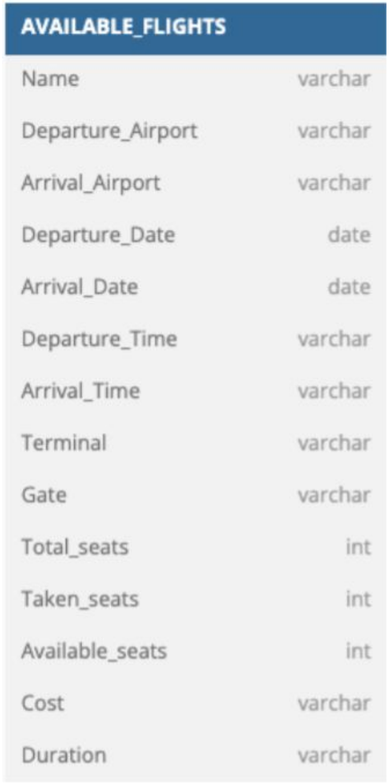


```java
package application;
//importing java libraries
import java.util.List;

public class Manager {

    // Declaring Variables

    private String username;
    private String password;
    private Flight flight;

    List<String> reservedFlights = new ArrayList<>();
    List<String> openFlights = new ArrayList<>();
    List<String> allFlights = new ArrayList<>();

    // Constructor
    public Manager() {
        username = "Manager";
        password = "Java123";
    }

    //Gets Manager Username
    public String getUsername() {
        return username;
    }

    //Gets Manager password
    public String getPassword() {
        return password;
    }

    //Gets list of names of all Flights
    public List<String> getAllFlights(List<Flight> flights) {
        allFlights.clear();
        for (int i=0; i<flights.size(); i++) {
            allFlights.add(flights.get(i).getName());
        }
        return allFlights;
```



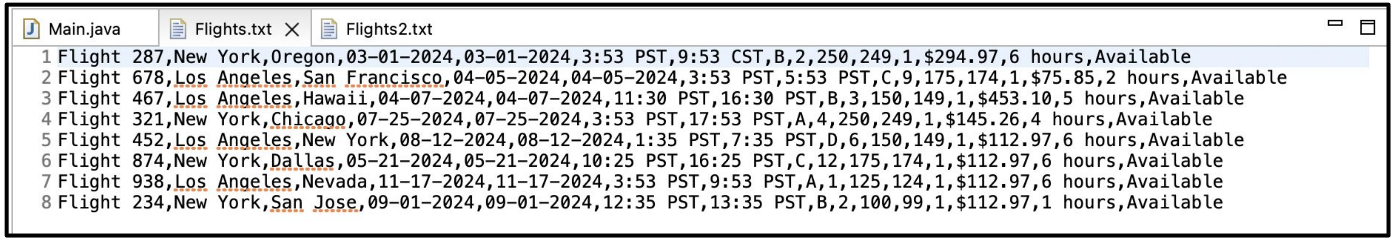
```java
    //Update Flight Data
    public void changeFlightInfo(Flight flight, String attribute, String data) {
        if (attribute.equals("Flight Name") ) {
            flight.setName(data);
        } else if (attribute.equals("Departing City")) {
            flight.setDepartCity(data);
        } else if (attribute.equals("Arrival City")) {
            flight.setArrivalCity(data);
        } else if (attribute.equals("Departing Date")) {
            flight.setDepartDate(data);
        } else if (attribute.equals("Arrival Date")) {
            flight.setArrivalDate(data);
        } else if (attribute.equals("Departing Time")) {
            flight.setDepartTime(data);
        } else if (attribute.equals("Arrival Time")) {
            flight.setArrivalTime(data);
        } else if (attribute.equals("Terminal")) {
            flight.setTerminal(data);
        } else if (attribute.equals("Gate")) {
            flight.setGate(data);
        } else if (attribute.equals("Total Seats")) {
            flight.setTotalSeats(data);
        } else if (attribute.equals("Taken Seats")) {
            flight.setTakenSeats(data);
        } else if (attribute.equals("Available Seats")) {
            flight.setAvailableSeats(data);
        } else if (attribute.equals("Cost")) {
            flight.setCost(data);
        } else if (attribute.equals("Duration")) {
            flight.setDuration(data);
        } else if (attribute.equals("Book Status")) {
            flight.setBookStatus(data);
        }
    }

    //Terminates Flight
    public void terminateFlight(Flight flight) {
        flight.setBookStatus("Terminated");
    }
```

Original Database Schema

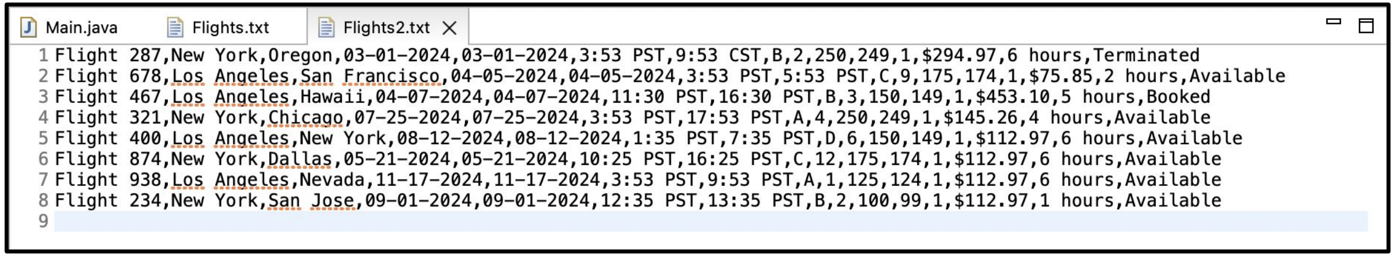


AVAILABLE_FLIGHTS	
Name	varchar
Departure_Airport	varchar
Arrival_Airport	varchar
Departure_Date	date
Arrival_Date	date
Departure_Time	varchar
Arrival_Time	varchar
Terminal	varchar
Gate	varchar
Total_seats	int
Taken_seats	int
Available_seats	int
Cost	varchar
Duration	varchar

- Original database schema for MySQL
- Ended up opting for an output file solution instead

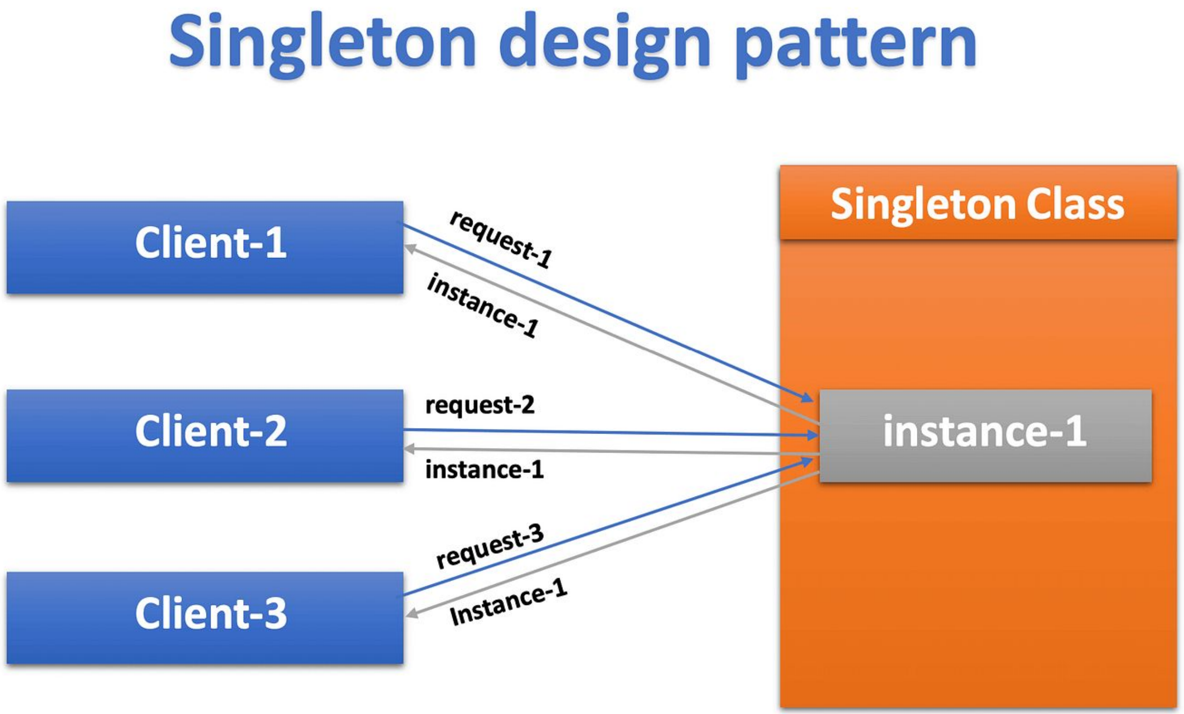


```
Flight 287,New York,Oregon,03-01-2024,03-01-2024,3:53 PST,9:53 CST,B,2,250,249,1,$294.97,6 hours,Available
Flight 678,Los Angeles,San Francisco,04-05-2024,04-05-2024,3:53 PST,5:53 PST,C,9,175,174,1,$75.85,2 hours,Available
Flight 467,Los Angeles,Hawaii,04-07-2024,04-07-2024,11:30 PST,16:30 PST,B,3,150,149,1,$453.10,5 hours,Available
Flight 321,New York,Chicago,07-25-2024,07-25-2024,3:53 PST,17:53 PST,A,4,250,249,1,$145.26,4 hours,Available
Flight 452,Los Angeles,New York,08-12-2024,08-12-2024,1:35 PST,7:35 PST,D,6,150,149,1,$112.97,6 hours,Available
Flight 874,New York,Dallas,05-21-2024,05-21-2024,10:25 PST,16:25 PST,C,12,175,174,1,$112.97,6 hours,Available
Flight 938,Los Angeles,Nevada,11-17-2024,11-17-2024,3:53 PST,9:53 PST,A,1,125,124,1,$112.97,6 hours,Available
Flight 234,New York,San Jose,09-01-2024,09-01-2024,12:35 PST,13:35 PST,B,2,100,99,1,$112.97,1 hours,Available
```

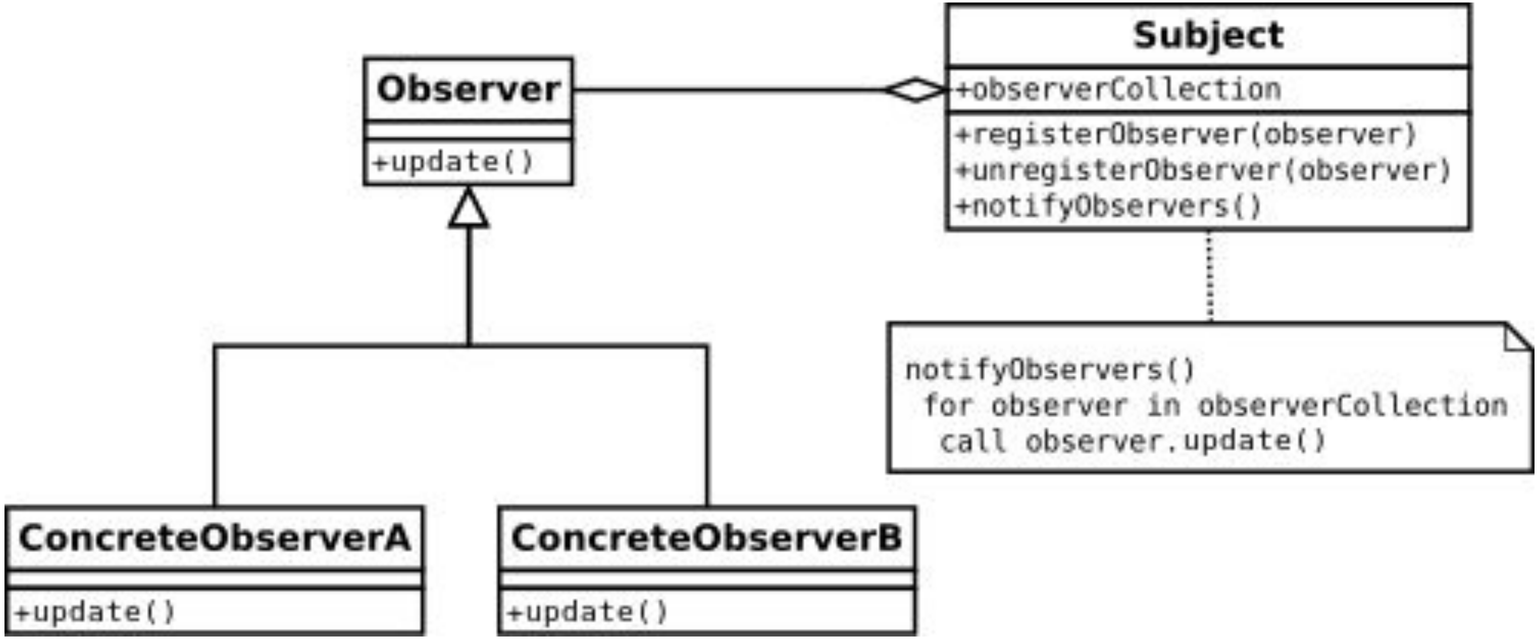


```
Flight 287,New York,Oregon,03-01-2024,03-01-2024,3:53 PST,9:53 CST,B,2,250,249,1,$294.97,6 hours,Terminated
Flight 678,Los Angeles,San Francisco,04-05-2024,04-05-2024,3:53 PST,5:53 PST,C,9,175,174,1,$75.85,2 hours,Available
Flight 467,Los Angeles,Hawaii,04-07-2024,04-07-2024,11:30 PST,16:30 PST,B,3,150,149,1,$453.10,5 hours,Booked
Flight 321,New York,Chicago,07-25-2024,07-25-2024,3:53 PST,17:53 PST,A,4,250,249,1,$145.26,4 hours,Available
Flight 400,Los Angeles,New York,08-12-2024,08-12-2024,1:35 PST,7:35 PST,D,6,150,149,1,$112.97,6 hours,Available
Flight 874,New York,Dallas,05-21-2024,05-21-2024,10:25 PST,16:25 PST,C,12,175,174,1,$112.97,6 hours,Available
Flight 938,Los Angeles,Nevada,11-17-2024,11-17-2024,3:53 PST,9:53 PST,A,1,125,124,1,$112.97,6 hours,Available
Flight 234,New York,San Jose,09-01-2024,09-01-2024,12:35 PST,13:35 PST,B,2,100,99,1,$112.97,1 hours,Available

```

Design Patterns: Singleton

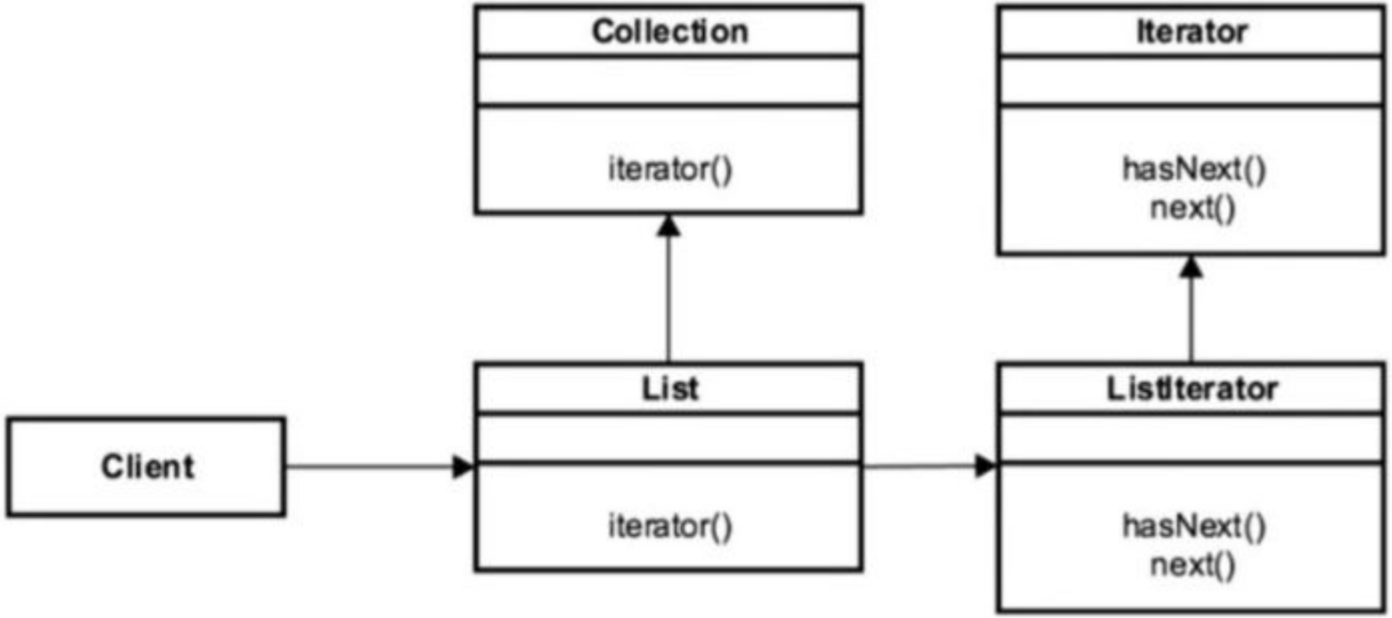


Design Patterns: Observer

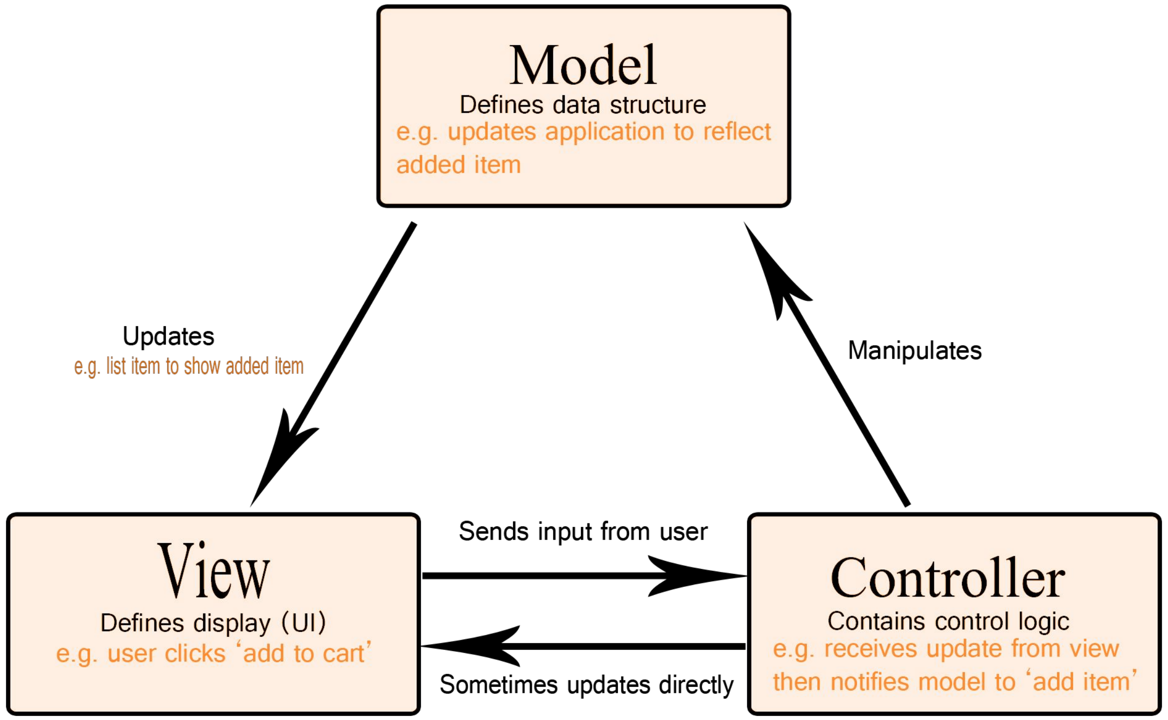


```
notifyObservers()
 for observer in observerCollection
  call observer.update()
```

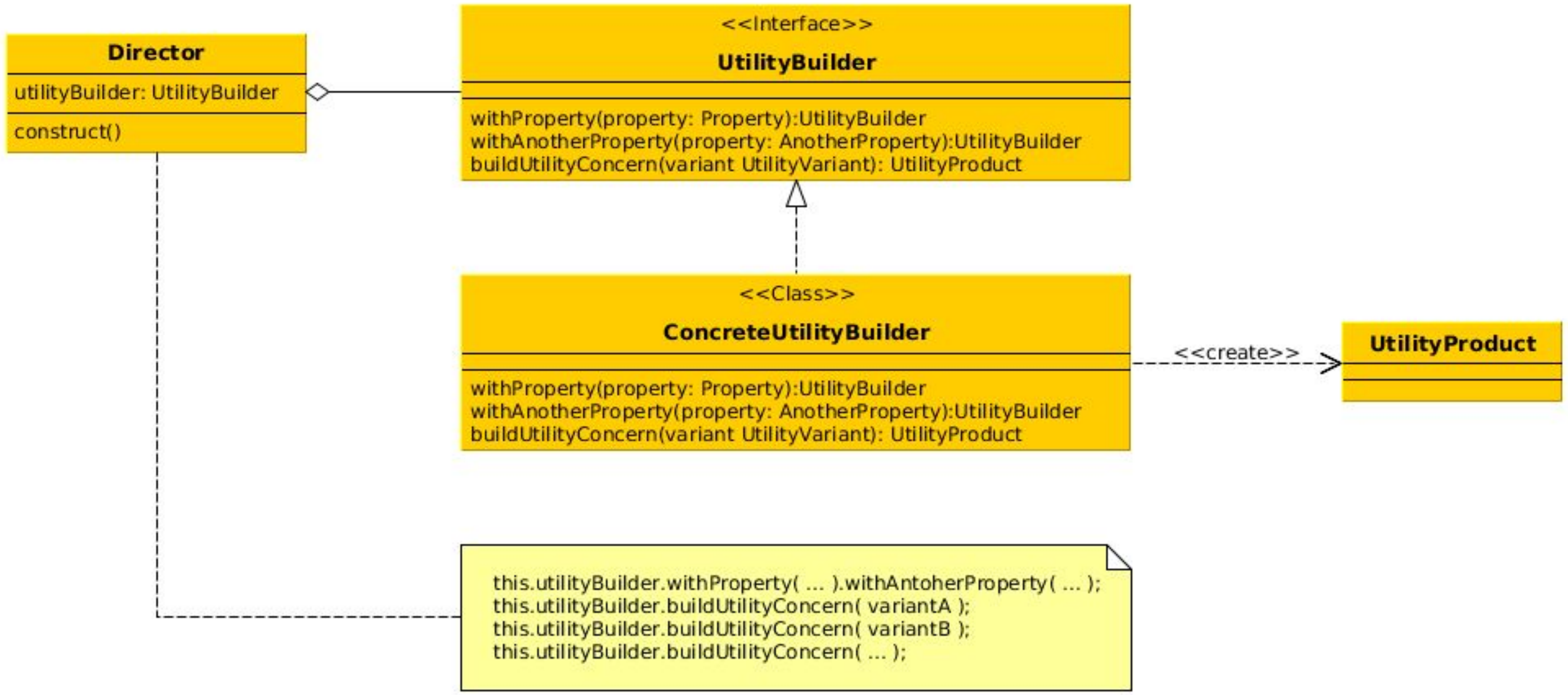
Design Patterns: Iterator



Design Patterns: Model View Controller


Model
Defines data structure
e.g. updates application to reflect added item
Updates
e.g. list item to show added item
Manipulates
View
Defines display (UI)
e.g. user clicks 'add to cart'
Sends input from user
Sometimes updates directly
Controller
Contains control logic
e.g. receives update from view then notifies model to 'add item'

Design Patterns: Utility


Director
utilityBuilder: UtilityBuilder
construct()
<<Interface>>
UtilityBuilder
withProperty(property: Property):UtilityBuilder
withAnotherProperty(property: AnotherProperty):UtilityBuilder
buildUtilityConcern(variant UtilityVariant): UtilityProduct
<<Class>>
ConcreteUtilityBuilder
withProperty(property: Property):UtilityBuilder
withAnotherProperty(property: AnotherProperty):UtilityBuilder
buildUtilityConcern(variant UtilityVariant): UtilityProduct
<<create>>
UtilityProduct

```
this.utilityBuilder.withProperty( ... ).withAntoherProperty( ... );
this.utilityBuilder.buildUtilityConcern( variantA );
this.utilityBuilder.buildUtilityConcern( variantB );
this.utilityBuilder.buildUtilityConcern( ... );
```

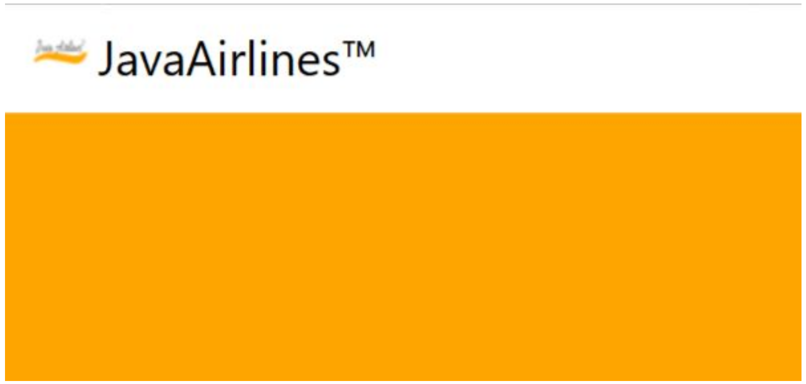
GUI Components

- Controllers- Handles the interaction between the User Interface and actions in the code.

- Cascading Style Sheets(CSS)- Defines the appearance of the GUI, making it it easy to format the color, size, and font of the GUI.

- Icon- Small Graphical image that helps the program stand out.

- Anchorpane- In charge of setting, or "Anchoring" child elements in certain positions in case of window resizing.



GUI Components

- ImageView-

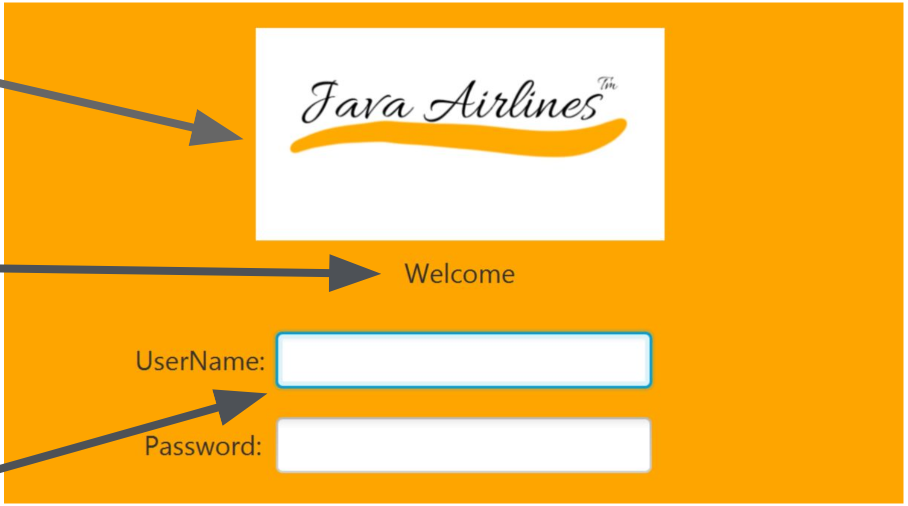
  Component used to display images.

- Labels-

  Static text used to display information.

- TextField-

  Component used to get user input.

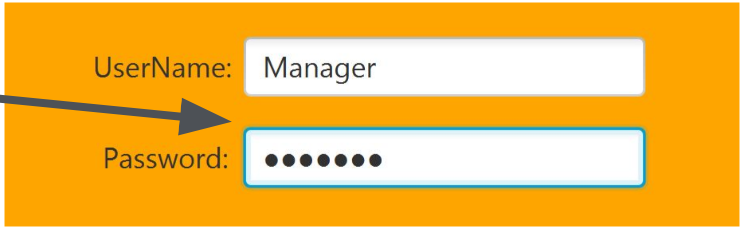


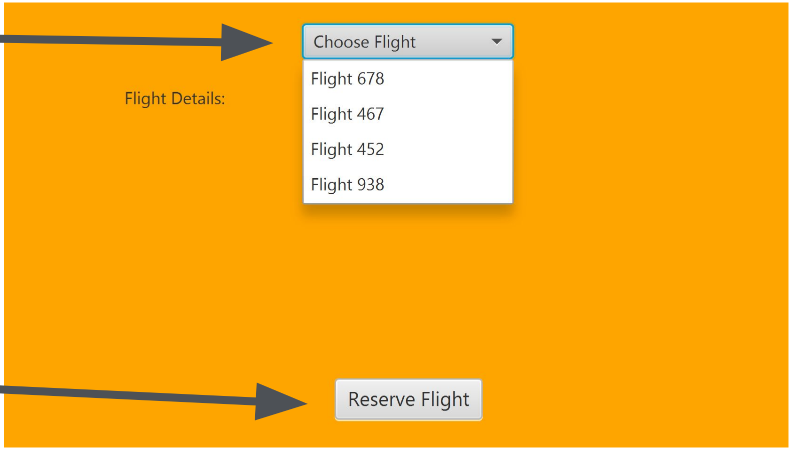
GUI Components

- PasswordField- A type of TextField that hides sensitive user input.

- ComboBox- GUI component that works as a drop-down list where users can select from a predetermined set of items.

- Buttons- Interactive components that have set actions when clicked.

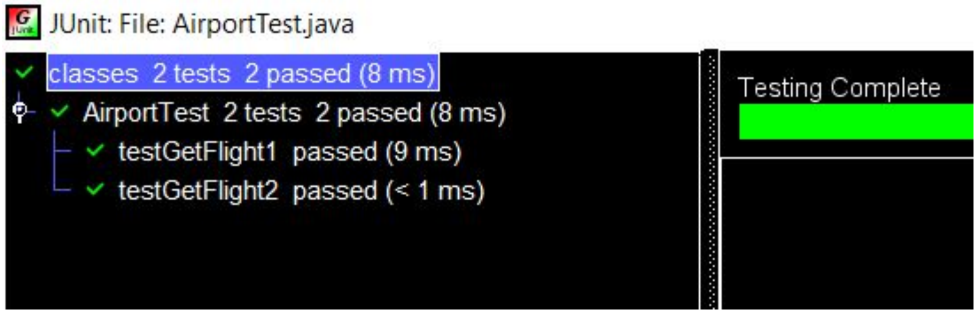




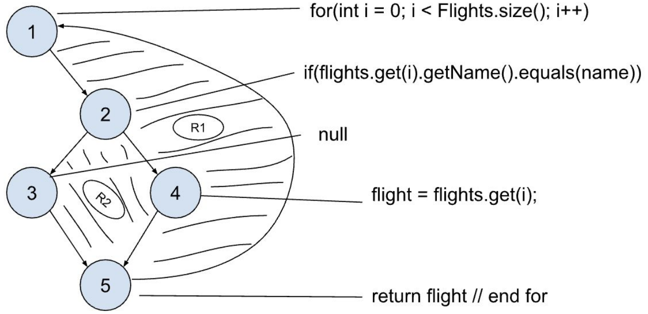
Implementation

Implementation (Testing)

- Test case:
  - Case 1: Flight1
  - Case 2: Flight2
- Expected Value:
  - Case1: "Flight 1", "Los Angeles", "New York", "2024-05-01", "2024-05-02", "10:00", "12:00", "Terminal 1", "Gate A", "100", "50", "50", "$200", "2 hours", "Available"
  - Case 2: Null
- Test Results:



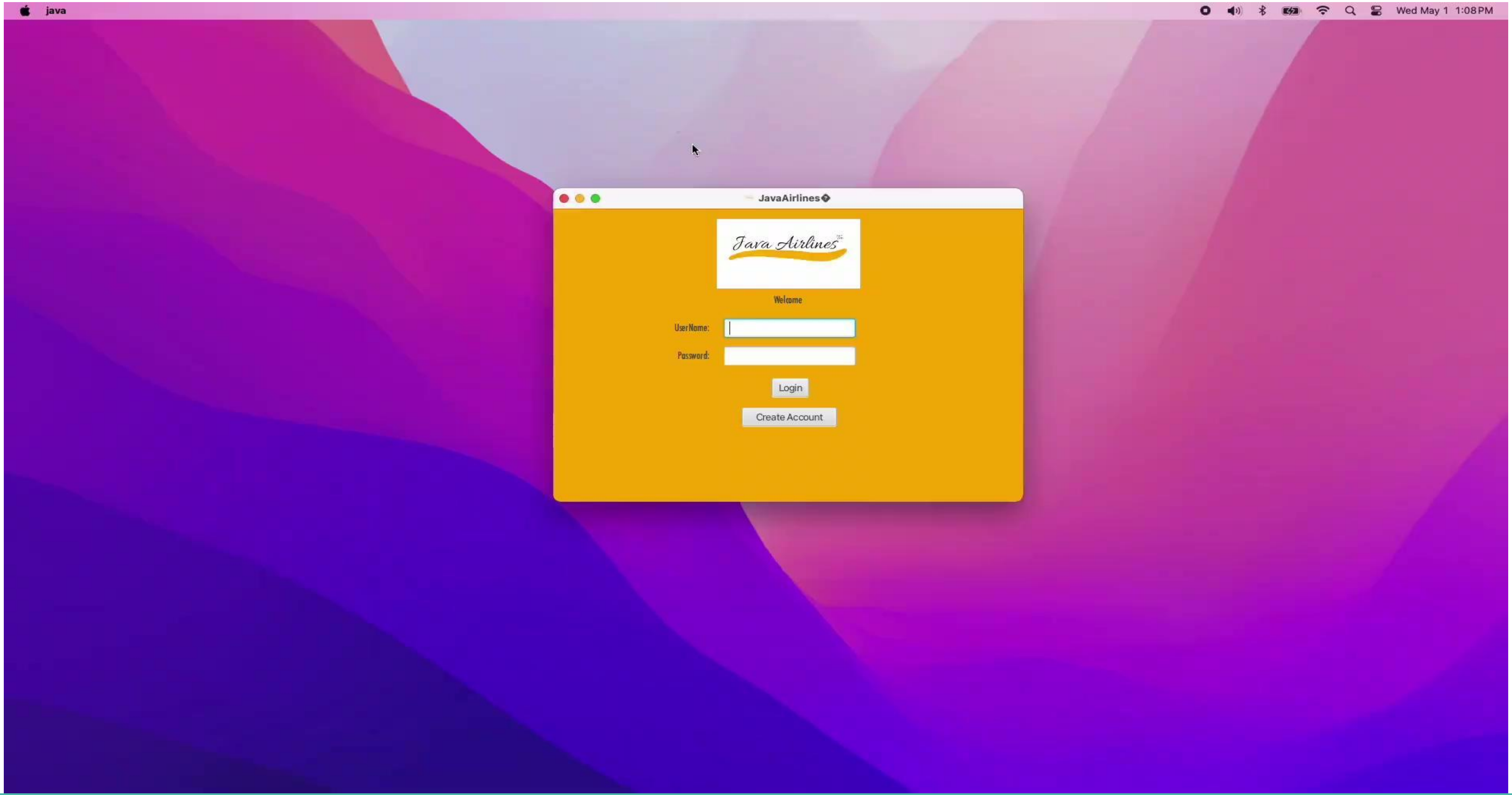
```
JUnit: File: AirportTest.java
✓ classes  2 tests  2 passed (8 ms)
  ✓ AirportTest  2 tests  2 passed (8 ms)
    ✓ testGetFlight1  passed (9 ms)
    ✓ testGetFlight2  passed (< 1 ms)
Testing Complete
```


1
2
3
4
5
R1
R2
for(int i = 0; i < Flights.size(); i++)
if(flights.get(i).getName().equals(name))
null
flight = flights.get(i);
return flight // end for

Independent paths:
Path 1: 1 - 2 - 4 - 5
Path 2: 1 - 2 - 3 - 5

- Conclusion: **Passed**

Demonstration



Conclusion

- Accomplishments
  - Fully functional GUI that mimics a site with basic reservation functionalities
  - Program properly implements original object oriented design into the GUI
  - Writes changes to a text file where all reservations could be found with all data.
- Challenges
  - Different IDEs, different issues come up
  - Different device operating systems (Mac vs Windows), some things like file outputting was limited
  - Implementing JavaFx for the GUI
  - Unfamiliar with Databases or using text files to read data.
- Future Updates
  - Adding a concrete database
  - Have everyone on the same IDE

Questions?