Code for the Airline Management System

Write the object-oriented code to implement the design of the airline management system problem.

We'll cover the following

We've reviewed different aspects of the airline management system and observed the attributes attached to the problem-solving various UML diagrams. Let's explore the more practical side of things, where we will work on implementing the airline management system using multiple languages. This is usually the last step in an object-oriented design interview process.

We have chosen the following languages to write the skeleton code of the different classes present in the airline management system:

- Java
- C#
- Python
- C++
- JavaScript

The airline management system classes

In this section, we'll provide the skeleton code of the classes designed in the class diagram lesson.

Note: For simplicity, we are not defining getter and setter functions. The reader can assume that all class attributes are private and accessed through their respective public getter methods and modified only through their public methods function.

Constants

The following code provides the definition of the various enums and custom data types being used in the airline management system:

Note: JavaScript does not support enumerations, so we will be using the `Object.freeze()` method as an alternative that freezes an object and prevents further modifications.

```
public class Address {
  private int zipCode;
  private String streetAddress;
  private String city;
  private String state;
  private String country;

}

enum AccountStatus {
  ACTIVE,
  CLOSED,
  BLOCKED
}

enum SeatStatus {
  AVAILABLE,
  RESERVED,
  CHANGE
}

enum SeatType {
  REGULAR,
  WINDOW_SEAT,
  EXTRA_LEG_ROOM
}

enum SeatClass {
  ECONOMY,
  BUSINESS,
  FIRST_CLASS
}
```

Constant definitions

Account and passenger

The **Account** class refers to an account for any user including admin, crew, front desk officer, and a customer. The **Passenger** class represents the passengers in the airline system. The definition of this class is given below:

```
public class Account {
  private AccountStatus status;
  private int accountNo;
  private String password;
  private String username;

  public boolean resetPassword();
}

public class Passenger {
  private String name;
  private String email;
  private String gender;
  private Date dateOfBirth;
  private String passportNumber;
}
```

The Account and Passenger classes

Person

`Person` is an abstract class that represents the various people or actors that can interact with the system. There are four types of persons: `Admin`, `Crew`, `FrontDeskOfficer`, and `Customer`. The implementation of the mentioned classes is shown below:

```
public abstract class Person {
  private String name;
  private Address address;
  private String email;
  private String phone;
  private Account account;
}

public class Admin extends Person {
  public boolean addAircraft(Aircraft aircraft);
  public boolean addFlight(Flight flight);
  public boolean updateFlightStatus(Flight flight);
  public boolean assignCrewToFlight(Flight flight);
  public boolean blockUser(User user);
  public boolean unblockUser(User user);
}

public class Crew extends Person {
  public List<FlightReservation> viewCheckedIn();
}

public class FrontDeskOfficer extends Person {
  public List<Itinerary> searchItinerary();
  public boolean checkInPassenger();
  public boolean cancelReservation();
  public boolean assignSeat();
  public boolean makePayment();
}

public class Customer extends Person {
  ...
}
```

Person and its derived classes

Seat and flight seat

The `Seat` and `FlightSeat` are used to keep track of the customer's seat. Here, `Seat` is the physical seat in the aircraft and `FlightSeat` is the seat assigned to a specific flight instance. The definition of these two classes is given below:

```
public class Seat {
  private String seatNumber;
  private SeatType type;
  private boolean _class;
}

public class FlightSeat extends Seat {
  private double fare;
  private SeatStatus status;
  private String reservationNumber;
}
```

The Seat and FlightSeat classes

Flight and flight instance

The `Flight` and `FlightInstance` classes provide the details of the flight and its instances to the customer. The definition of these classes is given below:

```
public class Flight {
  private String flightNo;
  private int durationMin;
  private String departureTime;
  private Aircraft aircraft;
  private List<FlightInstance> instances;
}

public class FlightInstance {
  private Flight flight;
  private Date departureTime;
  private Date arrival;
  private FlightStatus status;
  private Aircraft aircraft;
  private List<FlightSeat> seats;
}
```

The Flight and FlightInstance classes

Itinerary and flight reservation

The `Itinerary` and `FlightReservation` classes are used to keep track of itineraries and flights reserved by the customers. Both classes are defined below:

```
public class Itinerary {
  private Airport startingAirport;
  private Airport finalAirport;
  private Date creationDate;
  private List<FlightReservation> reservations;
  private List<Passenger> passengers;

  public boolean makeReservation();
  public boolean modifyReservation();
}

public class FlightReservation {
  private String reservationNumber;
  private FlightInstance flight;
  private ReservationStatus status;
  private Date creationDate;

  public static FlightReservation fetchReservationDetails(String reservationNumber);
  public List<Passenger> getPassengers();
}
```

The Itinerary and FlightReservation classes

Payment

The `Payment` class is another abstract class with two child classes: `Cash` and `CreditCard`. This takes the `PaymentStatus` enum to keep track of the payment status. The definition of this class is provided below:

```
public abstract class Payment {
  private int paymentId;
  private double amount;
  private PaymentStatus status;
  private Date timestamp;

  public abstract boolean makePayment();
}

public class Cash extends Payment {
  public boolean makePayment() {
    // functionality
  }
}

public class CreditCard extends Payment {
  private String nameOnCard;
  private String cardNumber;

  public boolean makePayment() {
    // functionality
  }
}
```

Payment and its derived classes

Notification

The `Notification` class is another abstract class responsible for sending notifications with two child classes: `SMSNotification` and `EmailNotification`. The implementation of this class is shown below:

```
public abstract class Notification {
  private int notificationId;
  private Date createdOn;
  private String content;

  public abstract void sendNotification(Account account);
}

class SmsNotification extends Notification {
  public void sendNotification(Account account) {
    // functionality
  }
}

class EmailNotification extends Notification {
  public void sendNotification(Account account) {
    // functionality
  }
}
```

Notification and its derived classes

Search and catalog

The `SearchCatalog` class contains the flight instance information and implements the `Search` interface class to enable the search functionality based on the criteria. Both classes are defined below:

```
public interface Search {
  // Overrides method (does not have a body)
  public List<FlightInstance> searchFlight(Airport source, Airport dest, Date arrival, Date departure);
}

public class SearchCatalog implements Search {
  private HashMap<SearchKey, Airport, Date, Date>, List<FlightInstance> flights;

  public List<FlightInstance> searchFlight(Airport source, Airport dest, Date arrival, Date departure) {
    // functionality
  }
}
```

The Search interface and the SearchCatalog class

Airport, aircraft, and airline

This section contains classes like `Airport`, `Aircraft`, and `Airline` that make up the infrastructure of our airline management system. Here, `Airline` is a Singleton class. The definition of these classes is given below:

```
public class Airport {
  private String name;
  private String city;
  private Address address;
  private List<Flight> flights;
}

public class Aircraft {
  private String name;
  private String model;
  private int numOfSeats;
  private int seatCapacity;
  private List<Seat> seats;
}

public class Airline {
  private String name;
  private String code;
  private List<Flight> flights;
  private List<Aircraft> aircrafts;
  private List<Crew> crew;

  // The Airline is a singleton class that ensures it will have only one active instance at a time
  private static Airline airline = null;

  // Create a static method to access the singleton instance of Airline class
  public static Airline getInstance() {
    if (airline == null) {
      airline = new Airline();
    }
    ...
  }
```

The Airport, Aircraft, and Airline classes

Wrapping up

We've explored the complete design of the airline management system in this chapter. We've looked at how a basic airline management system can be visualized using various UML diagrams and designed using object-oriented principles and design patterns.