

NUCES Airline Flight System

NUCES Airline Flight System (NAFS) is a console app for a newly established airline. The goal of NAFS is to provide a functional flight reservation system for the airline, which operates in five major cities of Pakistan: Islamabad, Lahore, Quetta, Peshawar, and Karachi. Each of these cities has two airports, one located in the North and the other in the South.

To begin with, NAFS has a fleet of 10 airplanes stationed in each city, with a maximum of 5 planes allowed to land at each airport simultaneously. The airline has also established a network of flights connecting 25 countries around the world.

On an annual basis, NAFS serves approximately 50,000 passengers, and its operations include ten local flights and five international flights departing from each airport daily, as per their designated schedules. The seating capacity of each airplane in NAFS consists of 50 seats in economy class and 10 seats in business class.

I designed and implemented a console application that facilitates the flight reservation process for NAFS. The application will allow passengers to search for available flights, make reservations, select their preferred class, and manage their bookings.

The application will need to integrate various functionalities, including a database to store flight information, passenger details, and reservation records.


The application has implemented necessary checks and validations to ensure the system's integrity and reliability. This includes verifying seat availability, enforcing maximum plane limits at each airport.

I implemented Following classes and I have Explained OOP concept used in each class.

1. 'User' class:

This class represents a user and has protected attributes such as 'name', 'username', and 'password'.

It encapsulates the user's information and provides a base class for more specific user types.




```
1 class User {
2     protected:
3         string name;
4         string username;
5         string password;
6 }
```

2. 'Passport' class:

This class represents a passport and has protected attributes such as 'id', 'country', and 'visa'.

It encapsulates the passport information of a passenger.



```
1 class Passport {
2     protected:
3         long int id;
4         string country;
5         bool visa;
```


3. 'Passenger' class (inherits from 'User'):

This class represents a passenger and inherits from the 'User' class.

It adds additional attributes like 'cnic' (national identification number) and 'passport' (an instance of the 'Passport' class).

It also has a static member variable 'num' to keep track of the number of passenger objects.

The 'Passenger' class has a composition relationship with the 'Passport' class, as it includes an instance of the 'Passport' class as a member.




```
1 class Passanger : public User {
2     private:
3         long long int cnic;
4         Passport passport;
5     public:
6         static int num;
```

4. 'Admin' class (inherits from 'User'):

This class represents an administrator and inherits from the 'User' class.

It adds an additional attribute called 'role'.

It also has a static member variable 'num' to keep track of the number of admin objects.



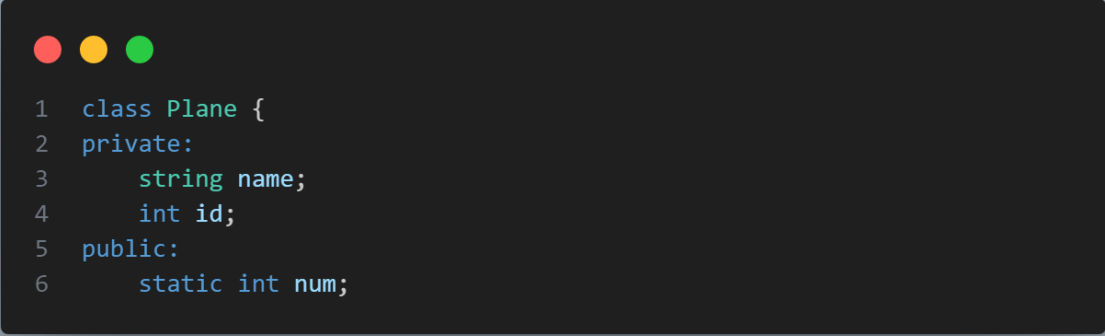
```
1 class Admin : public User {
2     private:
3         string role;
4     public:
5         static int num;
```

5. 'Plane' class:

This class represents a plane and has private attributes such as 'name' and 'id'.

It encapsulates the plane's information.

It also has a static member variable 'num' to keep track of the number of plane objects.



```
1 class Plane {
2 private:
3     string name;
4     int id;
5 public:
6     static int num;
```

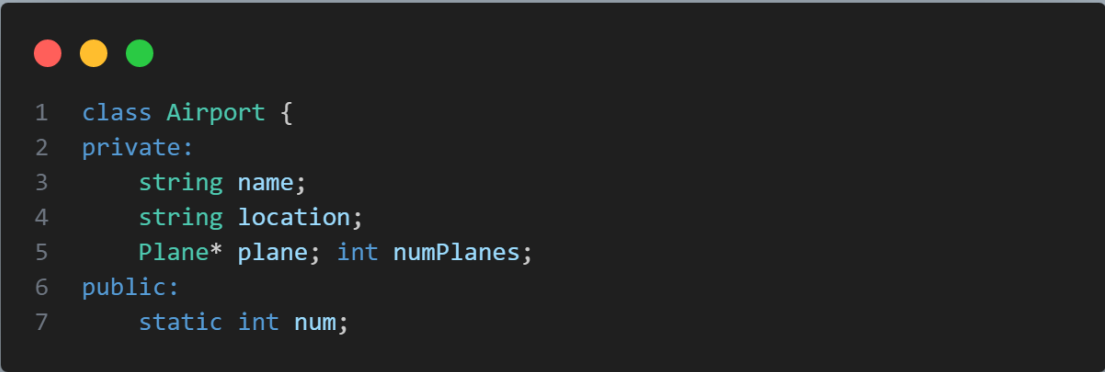
6. 'Airport' class:

This class represents an airport and has private attributes such as 'name', 'location', 'plane', and 'numPlanes'.

It encapsulates the airport's information and includes an aggregation relationship with the 'Plane' class (using a pointer to a 'Plane' object).

It also has a static member variable 'num' to keep track of the number of airport objects.

The 'Airport' class has an aggregation relationship with the 'Plane' class, as it includes a pointer to a 'Plane' object.



```
1 class Airport {
2 private:
3     string name;
4     string location;
5     Plane* plane; int numPlanes;
6 public:
7     static int num;
```

7. 'Ticket' class:

This class represents a ticket and has private attributes such as 'id', 'classType', and 'passenger'.

It encapsulates the ticket's information and includes an aggregation relationship with the 'Passenger' class (using a pointer to a 'Passenger' object).

The 'Ticket' class has an aggregation relationship with the 'Passenger' class, as it includes a pointer to a 'Passenger' object.

```
1 class Ticket {
2 private:
3     int id;
4     char classType; // 'E' or 'B' for economy or business
5     Passenger* passanger;
```

8. 'Date' class:

This class represents a date and has private attributes such as 'day', 'month', 'year', 'hour', and 'minute'.

It encapsulates the date and time information.

```
1 class Date {
2 private:
3     int day;
4     int month;
5     int year;
6
7     int hour;
8     int minute;
```

9. 'Flight' class:

This class represents a flight and has private attributes such as 'id', 'type', 'departureDate', 'duration', 'price', 'source', 'destination', 'plane', 'economyTickets', and 'businessTickets'.

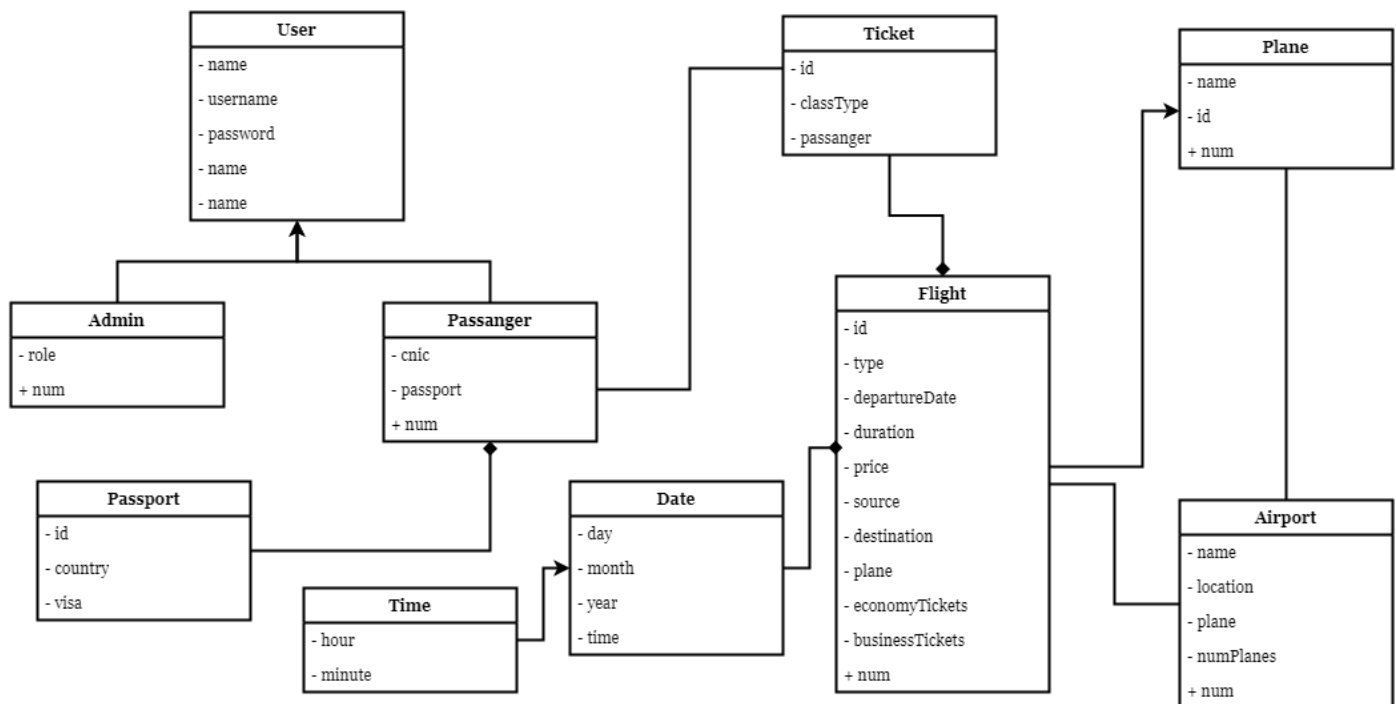
The 'Flight' class has aggregation relationships with the 'Airport' class.

It encapsulates the flight's information and includes aggregation relationships with the 'Airport' class (using pointers to source and destination airports) and the 'Plane' class (using a pointer to a plane object).

It also has static member variables such as 'num', 'econNum', and 'busNum' to keep track of the number of flight objects and the number of economy and business tickets.

```
1  class Flight {
2  private:
3      int id;
4      char type;
5      Date departureDate;
6      Time duration;
7      double price;
8      Airport* source;
9      Airport* destination;
10     Plane* plane;
11     Ticket* economyTickets;
12     Ticket* businessTickets;
13 public:
14     static int num;
15     int econNum;
16     int busNum;
```

Class Diagram:



Application:

The NUCES Airline Flight System (NAFS) console application offers various features to facilitate the flight reservation process and enhance the user experience. Here are the key features of the app:

User Registration and Login:

Users can register an account with their name, username, and password.

Registered users can log in to access their personalized features and make flight reservations.

Flight Search and Selection:

Passengers can search for available flights based on their desired source and destination airports, departure date, and other criteria.

The app provides a list of flights matching the search criteria, displaying flight details such as flight number, departure date, duration, and price.

Flight Reservation:

Passengers can make flight reservations by providing their personal information, such as name, passport details, and contact information.

The app handles the reservation process, including seat assignment, ticket generation, and updating the passenger and flight records.

Admin Panel:

Admin users have access to an administrative panel where they can manage flights, airports, planes, and other system settings.

Admins can add, update, or remove flights, airports, and planes as needed to keep the system up to date.

Passenger Management:

The app maintains passenger records, including personal information, passport details, and reservation history.

Passengers can view and manage their reservations, make changes to their bookings, or cancel flights if necessary.

Overall, the NAFS app combines essential functionalities such as flight search, reservation management, and administration capabilities to provide a comprehensive and user-friendly flight reservation system for the best airline in the world by Fastian.

The End!