# UNIFIED MODELING LANGUAGE


## Project Members

Ahmed Raza(230958)

Ghulam Qadir(230970)

Muqeem Ahmed(230980)

# Description Of Project Airplane Management System:

The Program Airline Management System tells where passengers can book, modify, and cancel reservations for flights, crews are assigned to flights with various roles, and flights can be scheduled, cancelled, or delayed. The system incorporates pricing strategies through a **`PricingEngine`**, which calculates flight prices based on factors such as seasonality, events, and demand. The program utilizes inheritance to define different flight states and polymorphism to enable flexible pricing calculations. It also includes a file handler to read and update data from a file, enhancing the system's modularity and extensibility. Overall, the program encapsulates various entities and functionalities of an airline operation, providing a comprehensive simulation of airline management.

**Program Overview:**

The provided C++ program simulates an airline management system with classes representing various entities such as passengers, crew members, flights, pricing engines, file handlers, and more. Let's break down the key attributes and functions of these classes, along with the use of inheritance and polymorphism:

1. **Class Overview:**
    - **Passenger:** Represents a passenger with attributes such as name, reservations, loyalty eligibility, and functions to book, cancel, and modify reservations.
    - **Crew:** Represents crew members with attributes like name, role, current state (available or training), compliance with regulations, and functions for assigning flights and resolving schedule conflicts.
    - **Flight:** Represents a flight with attributes including route, passengers, crew, price, current state (scheduled, cancelled, delayed), and functions for updating schedule, changing route, setting price, etc.
    - **Airline:** Manages flights, crews, and passengers. It includes functions to add flights, crews, assign crews to flights, and handle schedule conflicts.
    - **PricingEngine:** Calculates the price of flights based on seasonal factors, event factors, and demand factors. It utilizes polymorphism to allow for different pricing strategies.
    - **FileHandler:** Facilitates reading data from and updating data to a file, such as flight details and pricing factors.
    - **FlightState (Interface) & Concrete States (Scheduled, Cancelled, Delayed):** Represents different states of a flight. Each concrete state overrides the **`updateStatus()`** function to reflect the current status of the flight.
2. **Inheritance:**

- **FlightState Interface:** Concrete state classes (`ScheduledState`, `CancelledState`, `DelayedState`) implement the `FlightState` interface, demonstrating inheritance.
- **Observer Interface:** `PassengerObserver` and `CrewObserver` classes inherit from the `Observer` interface, allowing for generic observer functionality.

3. **Polymorphism:**
   - **FlightState:** Polymorphism is used when the `updateStatus()` function is called on the `currentState` object in the `Flight` class. The actual behavior depends on the concrete state object assigned to `currentState`.
4. **Aggregation:**

   Between **Airline and Flights**
   - The `Airline` class contains a vector of `Flight` pointers (`vector<Flight*> flights`). This represents an aggregation relationship because flights can exist independently of the airline. The airline holds a collection of flights, but the flights themselves are not destroyed when the airline object is destroyed.

   Between **Airline and Crew**
   - Similar to flights, the `Airline` class also contains a vector of `Crew` pointers (`vector<Crew*> crews`). This is another example of aggregation where the airline holds a collection of crew members, but the crew members can exist independently of the airline.

   Between **Passenger and Reservation**
   - There is an aggregation relationship between Passenger and Reservation. A Passenger may have multiple Reservations for different Flights. Reservations exist independently of the Passenger but can be associated with them.

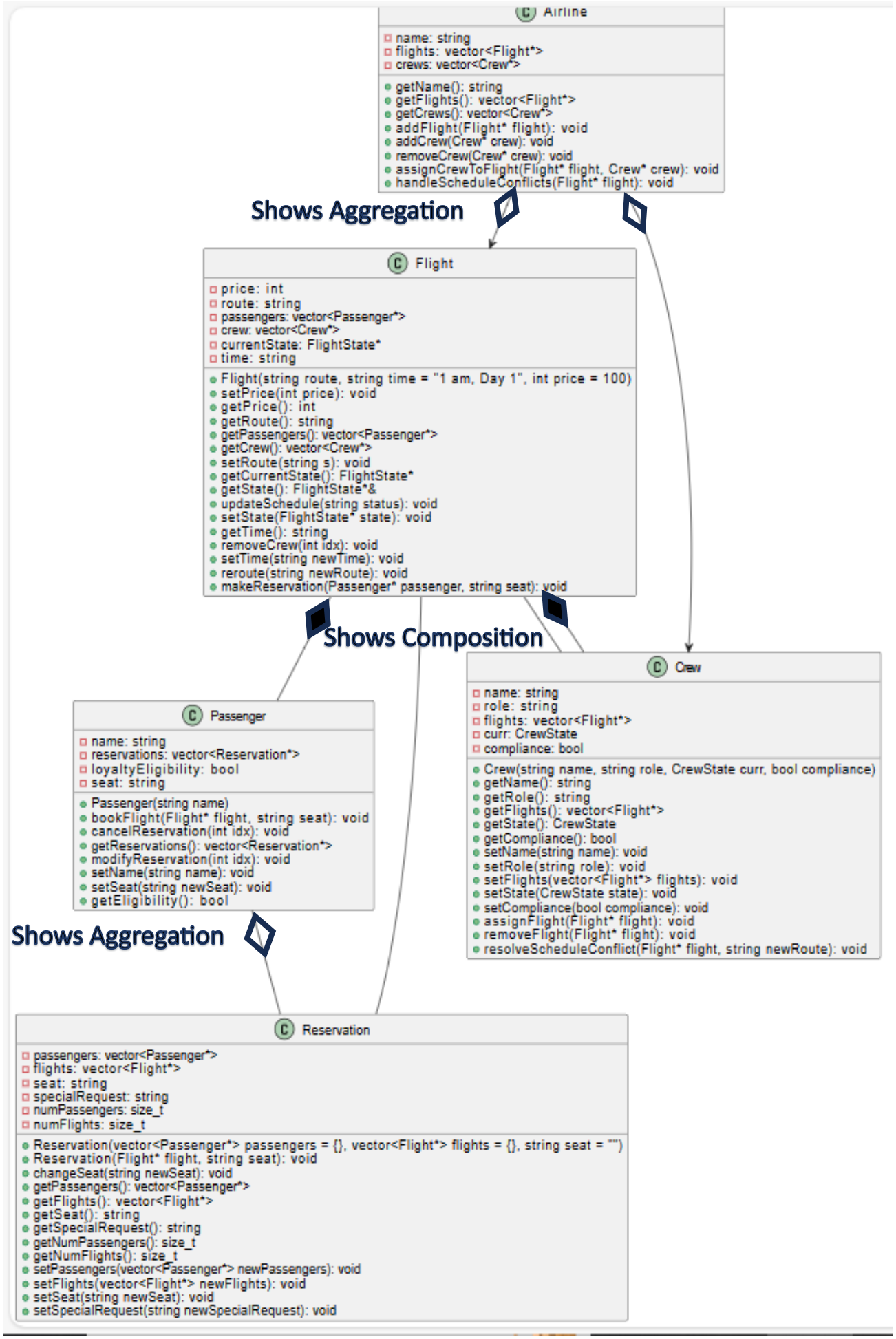
5. **Composition:**

   Between **Flight and Passenger**
   - The `Flight` class contains a vector of `Passenger* (vector<Passenger*> passengers)`. This represents a composition relationship because passengers are strongly tied to a flight. When a flight is destroyed, its passengers are also effectively destroyed or at least no longer associated with that flight. This is because passengers cannot exist without being associated with a flight.

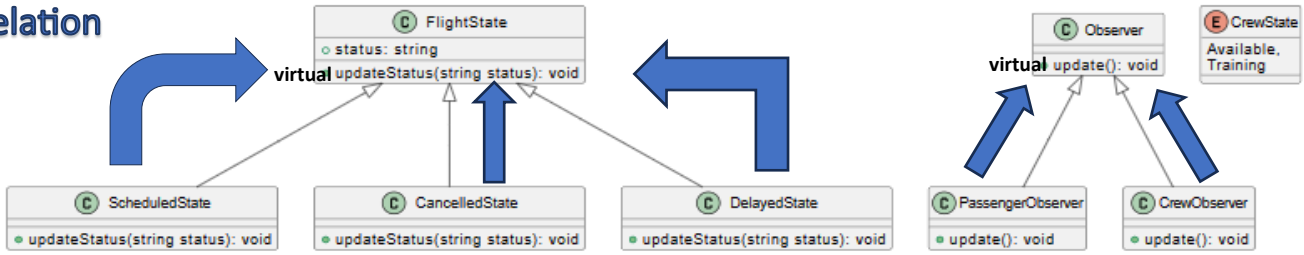
   Between **Flight and Crew**
   - Similarly, the `Flight` class contains a vector of `Crew*` (`vector<Crew*> crew`). This is another example of composition where the crew members are strongly associated with a flight. Crew members are typically assigned to a specific flight and are not interchangeable between flights, indicating a strong relationship

**Understanding Report:**

- **Inheritance Usage:** Inheritance is utilized in the program to establish an "is-a" relationship between base and derived classes. For example, **`ScheduledState`**, **`CancelledState`**, and **`DelayedState`** classes inherit from the **`FlightState`** interface, ensuring that they implement the required functionality to update the flight status.
- **Polymorphism Usage:** Polymorphism enables objects of different derived classes to be treated through a common interface. For instance, the **`PricingEngine`** class can calculate prices using different pricing strategies interchangeably, allowing for flexibility and extensibility in the pricing system.
- **Class Responsibilities:** Each class encapsulates specific responsibilities, such as managing passengers, flights, crews, pricing, and file operations. This promotes modularity, making the system easier to understand, maintain, and extend.
- **File Handling:** The **`FileHandler`** class encapsulates file-related operations, demonstrating the use of the facade design pattern to provide a simplified interface for reading and updating data.
- **Observer Pattern:** The program incorporates the observer pattern through the **`Observer`** interface and concrete observer classes (**`PassengerObserver`**, **`CrewObserver`**). This allows for loosely coupled communication between observers and subjects, enhancing flexibility and scalability.


Airline
name: string
flights: vector<Flight*>
crews: vector<Crew*>
getName(): string
getFlights(): vector<Flight*>
getCrews(): vector<Crew*>
addFlight(Flight* flight): void
addCrew(Crew* crew): void
removeCrew(Crew* crew): void
assignCrewToFlight(Flight* flight, Crew* crew): void
handleScheduleConflicts(Flight* flight): void
Shows Aggregation
Flight
price: int
route: string
passengers: vector<Passenger*>
crew: vector<Crew*>
currentState: FlightState*
time: string
Flight(string route, string time = "1 am, Day 1", int price = 100)
setPrice(int price): void
getPrice(): int
getRoute(): string
getPassengers(): vector<Passenger*>
getCrew(): vector<Crew*>
setRoute(string s): void
getCurrentState(): FlightState*
getState(): FlightState*&
updateSchedule(string status): void
setState(FlightState* state): void
getTime(): string
removeCrew(int idx): void
setTime(string newTime): void
reroute(string newRoute): void
makeReservation(Passenger* passenger, string seat): void
Shows Composition
Crew
name: string
role: string
flights: vector<Flight*>
curr: CrewState
compliance: bool
Crew(string name, string role, CrewState curr, bool compliance)
getName(): string
getRole(): string
getFlights(): vector<Flight*>
getState(): CrewState
getCompliance(): bool
setName(string name): void
setRole(string role): void
setFlights(vector<Flight*> flights): void
setState(CrewState state): void
setCompliance(bool compliance): void
assignFlight(Flight* flight): void
removeFlight(Flight* flight): void
resolveScheduleConflict(Flight* flight, string newRoute): void
Passenger
name: string
reservations: vector<Reservation*>
loyaltyEligibility: bool
seat: string
Passenger(string name)
bookFlight(Flight* flight, string seat): void
cancelReservation(int idx): void
getReservations(): vector<Reservation*>
modifyReservation(int idx): void
setName(string name): void
setSeat(string newSeat): void
getEligibility(): bool
Shows Aggregation
Reservation
passengers: vector<Passenger*>
flights: vector<Flight*>
seat: string
specialRequest: string
numPassengers: size_t
numFlights: size_t
Reservation(vector<Passenger*> passengers = {}, vector<Flight*> flights = {}, string seat = "")
Reservation(Flight* flight, string seat): void
changeSeat(string newSeat): void
getPassengers(): vector<Passenger*>
getFlights(): vector<Flight*>
getSeat(): string
getSpecialRequest(): string
getNumPassengers(): size_t
getNumFlights(): size_t
setPassengers(vector<Passenger*> newPassengers): void
setFlights(vector<Flight*> newFlights): void
setSeat(string newSeat): void
setSpecialRequest(string newSpecialRequest): void

## These arrows show Polymorphism Relation



# SOURCE CODE

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <vector>
#include <string>
#include <sstream>

using namespace std;

// Forward declarations
class Passenger;
class FlightState;
class Flight;
class Crew;
class Airline;
class PricingEngine;
class FileHandler;
class ScheduledState;
class CancelledState;
class DelayedState;
class Observer;
class PassengerObserver;
class CrewObserver;
```

```
class Reservation;

// Enum for crew states (Available or Training)
enum class CrewState {
    Available,
    Training
};

// Enum for crew roles
enum class CrewRole {
    Pilot,
    CoPilot,
    FlightAttendant
};

// Enum for seat types
enum class SeatType {
    Economy,
    Business,
    FirstClass
};

// Function to convert SeatType enum to string
string convertSeatTypeToString(SeatType seatType) {
    switch (seatType) {
    case SeatType::Economy:
        return "Economy";
    case SeatType::Business:
        return "Business";
```

```
    case SeatType::FirstClass:
        return "FirstClass";
    }
    return "";
}
```

```
// PricingEngine class

//calculate the final price of ticket according to provided factors like seasonal factor, event factor and demand factor
class PricingEngine {
private:
    //PricingStrategy* strategy;
    double seasonalFactor;
    double eventFactor;
    double demandFactor;

public:

    PricingEngine(double sf = 1,double ef = 1 , double df = 1) : seasonalFactor(sf),eventFactor(ef),demandFactor(df){}
        /*If seasonalFactor is non-zero, basePrice is multiplied by seasonalFactor; otherwise, basePrice remains unchanged.
If eventFactor is non-zero, the intermediate result is further multiplied by eventFactor.
If demandFactor is non-zero, the result is finally multiplied by demandFactor.
This ensures that the final price is adjusted according to the provided factors.*/

    double calculatePrice(double basePrice) {
```

```
        double ans = (seasonalFactor ? basePrice * seasonalFactor : basePrice);
        if (eventFactor) ans *= eventFactor;
        if (demandFactor) ans *= demandFactor;
        return ans;
    }
};

// FileHandler class (Facade for handling file operations)
class FileHandler {
private:
    string filePath;

public:
    FileHandler(string filePath) : filePath(filePath) {}

    void readData(Airline& airline);

    void updateData(string route, double basePrice, double demandFactor, double
seasonalFactor, double eventFactor);
};

// FlightState interface
class FlightState {
public:
    string status;
    virtual void updateStatus(string status) = 0;
};

// ScheduledState concrete state
class ScheduledState : public FlightState {
```

```
public:
    //string status;
    void updateStatus(string status) override {
        // Implement updating flight status to Scheduled
        this->status = "Scheduled";
    }
};

// CancelledState concrete state
class CancelledState : public FlightState {
public:
    //string status;
    void updateStatus(string status) override {
        // Implement updating flight status to Cancelled
        this->status = "Cancelled";
    }
};

// DelayedState concrete state
class DelayedState : public FlightState {
public:
    //string status;
    void updateStatus(string status) override {
        // Implement updating flight status to delayed
        this->status = "Delayed";
    }
};

// Observer interface
```

```
class Observer {
public:
    virtual void update() = 0;
};

// PassengerObserver concrete observer
class PassengerObserver : public Observer {
public:
    void update() override {
        // No Implementation Required
    }
};

// CrewObserver concrete observer
class CrewObserver : public Observer {
public:
    void update() override {
        // No Implementation Required
    }
};

// Airline class
class Airline {
private:
    string name;
    vector<Flight*> flights;
    vector<Crew*> crews;
    vector<Passenger*> passengers;
public:
```

```
    Airline(string name) : name(name) {}

    // Getter for name
    string getName() const {
        return name;
    }

    // Getter for flights
    const vector<Flight*>& getFlights() const {
        return flights;
    }

    // Getter for crews
    const vector<Crew*>& getCrews() const {
        return crews;
    }

    // Getter for passengers
    const vector<Passenger*>& getPassengers() const {
        return passengers;
    }

    void addFlight(Flight* flight) {
        flights.push_back(flight);
    }




    void addCrew(Crew* crew) {
```

```
        crews.push_back(crew);
    }

    void removeCrew(Crew* crew) {
        // Implement removal logic
        for (int i = 0; i < crews.size(); i++) {
            if (crews[i] == crew) {
                for (int j = i; j < crews.size() - 1; j++) {
                    crews[i] = crews[i + 1];
                }
                crews.pop_back();
                break;
            }
        }
    }

    void assignCrewToFlight(Flight* flight, Crew* crew);

    void handleScheduleConflicts(Flight* flight) {
        // Handle schedule conflicts for the flight
    }
};

// Flight class with state pattern integration
class Flight {
private:
    int price;
    string route;
    vector<Passenger*> passengers;
```

```
    vector<Crew*> crew;
    FlightState* currentState;
    string time;

public:
    Flight(string route , string time = "1 am ,Day 1",int price = 100) : route(route),
time(time),price(price) {
        currentState = new ScheduledState();
        currentState->status = "Scheduled.\n";
    }

    void setPrice(int price) {
        this->price = price;
    }

    int getPrice() {
        return price;
    }

    // Getter for route
    string getRoute() const {
        return route;
    }

    // Getter for passengers
    const vector<Passenger*>& getPassengers() const {
        return passengers;
    }

    // Getter for crew
```

```
vector<Crew*>& getCrew() {
    return crew;
}

void setRoute(string s) {
    route = s;
}

// Getter for currentState
FlightState* getCurrentState() const {
    return currentState;
}

FlightState*& getState() { return currentState; }

void updateSchedule(string status) {
    currentState->updateStatus(status);
}

void setState(FlightState* state) {
    currentState = state;
    updateSchedule("Cancelled");
}

string getTime() const {
    return time;
}

void removeCrew(int idx) {
```

```
        for (int i = idx; i < crew.size()-1; i++) {
            crew[i] = crew[i + 1];
        }
        crew.pop_back();
    }

    void setTime(string newTime) {
        time = newTime;
        //cout << "\nNew time has been set Successfully.\n";
    }

    void reroute(string newRoute) {

        route = newRoute;
    }

    void makeReservation(Passenger* passenger, string seat) {
        // Logic Already implemented
    }
};



// Passenger class
class Passenger {
private:
    string name;
    vector<Reservation*> reservations;
    bool loyaltyEligibility;
    string seat;
```

```
public:
    Passenger(string name) : name(name), loyaltyEligibility(rand() % 2) {}

    void bookFlight(Flight* flight, string seat) {
        // Implemented
    }

    void cancelReservation(int idx) {
        // Implement cancellation logic
        for (int i = idx; i < reservations.size() - 1; i++) {
            reservations[i] = reservations[i + 1];
        }
        reservations.pop_back();

        cout << "Your Desired Reservation has been successfully removed.\n";
    }

    vector<Reservation*>& getReservations() { return reservations; }

    void modifyReservation(int idx);

    void setName(string name) {
        this->name = name;
    }

    void setSeat(string newSeat) {
        seat = newSeat;
    }
```

```
    bool getEligibility() const {
        return loyaltyEligibility;
    }
};

// Reservation Class
class Reservation {
private:
    vector<Passenger*> passengers;
    vector<Flight*> flights;
    string seat;
    string specialRequest;
    size_t numPassengers;
    size_t numFlights;

public:
    Reservation(vector<Passenger*> passengers = {}, vector<Flight*> flights = {}, string seat =
"")
        : passengers(passengers), flights(flights), seat(seat) {
        numPassengers = passengers.size();
        numFlights = flights.size();
    }

    Reservation(Flight* flight, string seat) : seat(seat) {
        flights.push_back(flight);
    }

    void changeSeat(string newSeat) {
        // Implement special request to change seat
```

```
    seat = newSeat;
}

// Getters
vector<Passenger*>& getPassengers() {
    return passengers;
}

vector<Flight*>& getFlights() {
    return flights;
}

string getSeat() {
    return seat;
}

string getSpecialRequest() {
    return specialRequest;
}

size_t getNumPassengers() {
    return numPassengers;
}

size_t getNumFlights() {
    return numFlights;
}

// Setters
```

```
    void setPassengers(vector<Passenger*> newPassengers) {
        passengers = newPassengers;
        numPassengers = passengers.size();
    }

    void setFlights(vector<Flight*> newFlights) {
        flights = newFlights;
        numFlights = flights.size();
    }

    void setSeat(string newSeat) {
        seat = newSeat;
    }

    void setSpecialRequest(string newSpecialRequest) {
        specialRequest = newSpecialRequest;
    }
};

void Passenger::modifyReservation(int idx) {
    // Implement modification logic
    cout << "You have the option to modify which kind of seat you want to reserve.\n";
    int seatChoice;
    cout << "Please choose the seat type:\n";
    cout << "1. Economy\n";
    cout << "2. Business\n";
    cout << "3. FirstClass\n";
    cin >> seatChoice;
    while (cin.fail() || seatChoice < 1 || seatChoice > 3) {
```

```
        cout << "Please Enter a valid value: \n";
        cin >> seatChoice;
    }
    SeatType chosenSeatType;
    switch (seatChoice) {
    case 1:
        chosenSeatType = SeatType::Economy;
        break;
    case 2:
        chosenSeatType = SeatType::Business;
        break;
    case 3:
        chosenSeatType = SeatType::FirstClass;
        break;
    default:
        cout << "Invalid seat choice\n";
        break;
    }
    seat = convertSeatTypeToString(chosenSeatType);
    reservations[idx]->changeSeat(seat);
    cout << "Your Given Seat Type is Successfully Modified.\n";
}

// Crew class
class Crew {
private:
    string name;
    string role;
    vector<Flight*> flights;
```

```
    CrewState curr; // States of crew (Available or training)
    bool compliance; // Compliance with aviation regulations

public:
    Crew(string name, string role, CrewState curr, bool compliance)
        : name(name), role(role), curr(curr), compliance(compliance) {}

    // Getters
    string getName() const { return name; }
    string getRole() const { return role; }
    vector<Flight*> getFlights() const { return flights; }
    CrewState getState() const { return curr; }
    bool getCompliance() const { return compliance; }

    // Setters
    void setName(string name) { this->name = name; }
    void setRole(string role) { this->role = role; }
    void setFlights(vector<Flight*> flights) { this->flights = flights; }
    void setState(CrewState state) { this->curr = state; }
    void setCompliance(bool compliance) { this->compliance = compliance; }

    void assignFlight(Flight* flight) {
        flights.push_back(flight);
    }

    void removeFlight(Flight* flight) {
        // Implemented removal logic
    }
```

```
    void resolveScheduleConflict(Flight* flight, string newRoute) {
        // Resolved scheduling conflicts by changing the route of a flight
    }
};



// Implementation of FileHandler readData method
void FileHandler::readData(Airline& airline) {
    ifstream file(filePath);
    if (!file.is_open()) {
        cerr << "Error: Unable to open file " << filePath << endl;
        return;
    }

    string line;
    while (getline(file, line)) {
        stringstream ss(line);
        string route, flightState, crewName, crewRole, crewComplianceStr, crewStateStr;
        getline(ss, route, ',');
        getline(ss, flightState, ',');
        getline(ss, crewName, ',');
        getline(ss, crewRole, ',');
        getline(ss, crewComplianceStr, ',');
        getline(ss, crewStateStr);

        bool crewCompliance = (crewComplianceStr == "1");
        CrewState crewState = (crewStateStr == "Available") ? CrewState::Available :
CrewState::Training;

        FlightState* currentState;
```

```
        if (flightState == "Scheduled") {
            currentState = new ScheduledState();
        }
        else if (flightState == "Cancelled") {
            currentState = new CancelledState();
        }
        else if (flightState == "Delayed") {
            currentState = new DelayedState();
        }
        else {
            cerr << "Error: Invalid flight state encountered: " << flightState << endl;
            continue;
        }

        Flight* flight = new Flight(route);
        flight->setState(currentState);
        flight->setPrice(100 + rand() % 50);

        Crew* crew = new Crew(crewName, crewRole, crewState, crewCompliance);
        airline.addFlight(flight);
        airline.addCrew(crew);
    }

    file.close();
}

// Implementation of FileHandler updateData method
void FileHandler::updateData(string route, double basePrice, double demandFactor, double
seasonalFactor, double eventFactor) {
    // Implement updating pricing factors in final_draft.csv
```

```
}

void Airline::assignCrewToFlight(Flight* flight, Crew* crew) {
   // Assign crew to the flight
   flight->getCrew().push_back(crew);
   removeCrew(crew);

}

int main() {
        //generate random numbers for time but current time for booking of flight
   srand(time(0));

   cout << " ~*~*~*~*~*~*~*~* Welcome to Airline Reservation System Simulation
*~*~*~*~*~*~*~*~\n";
   string a_name; // Airline name
   cout << "Enter the name for the Airline : ";
   getline(cin, a_name);

   PricingEngine price(1.2, 1, 1.3); // These are the hard coded factors

   // Create airline object
   Airline airline(a_name);
   // Initializing the reservation system
   Reservation reservations;

   // Load data using file handler
   FileHandler fileHandler("final_draft.txt");
   fileHandler.readData(airline);
```

```
    // Menu to choose observer state
    cout << "\nChoose observer state:\n";
    cout << "1. Passenger Observer\n";
    cout << "2. Head Observer\n";
    int observerChoice;
    cin >> observerChoice;
    while (cin.fail() || observerChoice < 1 || observerChoice > 2) {
        cout << "Please Enter a Valid Observer Type: ";
        cin >> observerChoice;
    }
    // Create appropriate observer object based on choice
    Observer* observer = nullptr;
    if (observerChoice == 1) {
        observer = new PassengerObserver();
    }
    else if (observerChoice == 2) {
        observer = new CrewObserver();
        // Setting up time for each flight
        for (int i = 0; i < airline.getFlights().size(); i++) {
            string time = to_string((i + 1) % 12 + 1) + ((i % 24) < 13 ? " pm" : " am") + ", Day " +
to_string(i / 10 + 1);
            airline.getFlights()[i]->setTime(time);
        }
    }
    else {
        cout << "Invalid choice\n";
        return 1;
    }
```

```
bool Flag = 0; // for taking name of the passenger only once
//bool Flag2 = 0; // For Checking if the passenger has any reservations
Passenger newPassenger("");
// Handling choices for functionalities
int choice;
do {
   cout << "\nChoose an option:\n";
   if (observerChoice == 1) {
      cout << "1. Book a flight\n";
      cout << "2. Cancel a reservation\n";
      cout << "3. Modify a reservation\n";
      cout << "4. Exit\n";
   }
   else if (observerChoice == 2) {
      cout << "1. Manage Crews to a flight\n"; // For adding and removing crews
      cout << "2. Handle schedule conflicts\n";
      cout << "3. Update flight status\n";
      cout << "4. Reroute a flight\n";
      cout << "5. Modify Flights \n"; // Adding and Removing
      cout << "6. Exit\n";
   }
   cin >> choice;
   int CrewNum;
   int flightNum;
   switch (choice) {
   case 1: {
      if (observerChoice == 1) {
         // Book a flight
         string passengerName;
```

```
if (!Flag) {
    cout << "Enter passenger name: ";

    getline(cin >> ws, passengerName); // ws for white space
}
int seatChoice;
cout << "Choose seat type:\n";
cout << "1. Economy\n";
cout << "2. Business\n";
cout << "3. FirstClass\n";
cin >> seatChoice;
while (cin.fail() || seatChoice < 1 || seatChoice > 3) {
    cout << "Please Enter a valid value: \n";
    cin >> seatChoice;
}
SeatType chosenSeatType;
switch (seatChoice) {
case 1:
    chosenSeatType = SeatType::Economy;
    break;
case 2:
    chosenSeatType = SeatType::Business;
    break;
case 3:
    chosenSeatType = SeatType::FirstClass;
    break;
default:
    cout << "Invalid seat choice\n";
    continue; // Go back to the main menu
```

```
            }
            string seat = convertSeatTypeToString(chosenSeatType);
            if (!Flag) {
               newPassenger.setName(passengerName);
               Flag = 1;
            }
            cout << "You have the following Avaiable flights : \n";
            for (int i = 0; i < airline.getFlights().size(); i++) {
               string time = to_string((i + 1) % 12 + 1) + ((i % 24) < 13 ? " pm" : " am") + ", Day " +
to_string(i / 10 + 1);
               airline.getFlights()[i]->setTime(time);
               if (airline.getFlights()[i]->getCurrentState()->status == "Scheduled") {
                  cout << " " << i + 1 << ": ~~~ " << airline.getFlights()[i]->getRoute() << ", Time : [
" << airline.getFlights()[i]->getTime() << " ]" << endl;
               }

            }
            int flightNum = 0;
            cout << "Enter the flight you want to take : ";
            cin >> flightNum;
            while (cin.fail() || flightNum < 1 || flightNum > 37 || airline.getFlights()[flightNum -
1]->getCurrentState()->status != "Scheduled") {
               cout << "Please Enter a valid Flight Number : ";
               cin >> flightNum;
            }

            newPassenger.getReservations().push_back(new
Reservation(airline.getFlights()[flightNum - 1], seat));
            // Deleting OLD STATE
            delete airline.getFlights()[flightNum - 1]->getCurrentState();
            // Assigning New State
```

```
            airline.getFlights()[flightNum - 1]->setState(new CancelledState);
            //reservations.getFlights().push_back(new Flight());

            if (seat == "Business") airline.getFlights()[flightNum - 1]-
>setPrice(airline.getFlights()[flightNum - 1]->getPrice() + 40);
            else if(seat == "FirstClass") airline.getFlights()[flightNum - 1]-
>setPrice(airline.getFlights()[flightNum - 1]->getPrice() + 100);
            string loyalityCheck;
            cout << "Are you eligible for the loyality program? \n ";
            cin >> loyalityCheck;
            cout << "\nYou are successfully Registered in the Flight heading " <<
airline.getFlights()[flightNum - 1]->getRoute() << " \n";
            cout << "Regardless of your input , We have decided that you are " <<
(newPassenger.getEligibility() ? "" : "not ") << "eligible for the loyality Program.\n";
            cout << "\n\nConsidering All the Current Factors , Your final price comes out to be :
" << price.calculatePrice(airline.getFlights()[flightNum - 1]->getPrice()) << "$ \n";
        }
        else if (observerChoice == 2) {
            // Assign crew to a flight
            cout << "Choose one of the following : \n";
            cout << "1. Add crew to Flight\n";
            cout << "2. Remove crew from Flight\n";
            int choice = 0;
            cin >> choice;
            while (cin.fail() || choice < 1 || choice > 2) {
                cout << "Enter a Valid Option : ";
                cin >> choice;
            }
            switch (choice) {
            case 1:
                cout << "You have the following Flights Available: \n";
```

```
            for (int i = 0; i < airline.getFlights().size(); i++) {
                //string time = to_string((i + 1) % 12 + 1) + ((i % 24) < 13 ? " pm" : " am") + ", Day " + to_string(i / 10 + 1);
                if (airline.getFlights()[i]->getCurrentState()->status == "Scheduled") {
                    cout << " " << i + 1 << ": ~~~ " << airline.getFlights()[i]->getRoute() << ", Time : [ " << airline.getFlights()[i]->getTime() << " ]" << endl;
                }
                //airline.getFlights()[i]->setTime(time);
            }

            cout << "Enter the Flight number that you want to add the crew to : ";
            cin >> flightNum;
            while (cin.fail() || flightNum < 1 || flightNum > 37 || airline.getFlights()[flightNum - 1]->getCurrentState()->status != "Scheduled") {
                cout << "Please Enter a valid Flight Number : ";
                cin >> flightNum;
            }
            if (airline.getCrews().size()) {
                cout << "You Currently have the following crew : \n";
                for (int i = 0; i < airline.getCrews().size(); i++) {
                    cout << " " << i + 1 << ". " << airline.getCrews()[i]->getName() << " [ " << airline.getCrews()[i]->getRole() << " ] \n";
                }
            }
            else cout << "No , Crew Available.\n";
            cout << "Enter the Crew Number that you want to assign to the this Flight : ";

            cin >> CrewNum;
            while (cin.fail() || CrewNum < 1 || CrewNum > airline.getCrews().size()) {
                cout << "Enter a Valid Crew no. : ";
                cin >> CrewNum;
```

```
            }
            if (airline.getCrews()[CrewNum - 1]->getCompliance()) {
                airline.assignCrewToFlight(airline.getFlights()[flightNum - 1],
airline.getCrews()[CrewNum - 1]);
                cout << "Your Desired Crew is added to the desired Flight.\n";
            }
            else cout << " This Crew's compliance with aviation regulations is subpar.\n So,
Please select another one next time.\n";
            break;
        case 2:
            cout << "You have the following Flights Available: \n";
            for (int i = 0; i < airline.getFlights().size(); i++) {
                //string time = to_string((i + 1) % 12 + 1) + ((i % 24) < 13 ? " pm" : " am") + ",
Day " + to_string(i / 10 + 1);
                if (airline.getFlights()[i]->getCurrentState()->status == "Scheduled") {
                    cout << " " << i + 1 << ": ~~~ " << airline.getFlights()[i]->getRoute() << ", Time
: [ " << airline.getFlights()[i]->getTime() << " ]" << endl;
                }
                //airline.getFlights()[i]->setTime(time);
            }

            cout << "Enter the Flight number that you want to remove the crew from : ";
            cin >> flightNum;
            while (cin.fail() || flightNum < 1 || flightNum > 37 ||
airline.getFlights()[flightNum - 1]->getCurrentState()->status != "Scheduled") {
                cout << "Please Enter a valid Flight Number : ";
                cin >> flightNum;
            }
            if (airline.getFlights()[flightNum-1]->getCrew().size()) {
                cout << "You Currently have the following crew : \n";
                for (int i = 0; i < airline.getFlights()[flightNum - 1]->getCrew().size(); i++) {
```

```
                    cout << " " << i + 1 << ". " << airline.getFlights()[flightNum - 1]->getCrew()[i]->getName() << " [ " << airline.getFlights()[flightNum - 1]->getCrew()[i]->getRole() << " ] \n";
                }
            }
            else {
                cout << "No , Crew Available.\n";
                break;
            }
            cout << "Enter the Crew Number that you want to remove from this flight : ";

            cin >> CrewNum;
            while (cin.fail() || CrewNum < 1 || CrewNum > airline.getFlights()[flightNum - 1]->getCrew().size()) {
                cout << "Enter a Valid Crew no. : ";
                cin >> CrewNum;
            }
            airline.addCrew(airline.getFlights()[flightNum - 1]->getCrew()[CrewNum - 1]);
            airline.getFlights()[flightNum - 1]->removeCrew(CrewNum-1);

            cout << "Your Desired Crew has been successfully removed from the inteneded Flight .\n";
            break;
        default:
            cout << "This line is not supposed to print.\n";
        }
    }
    break;
}
case 2: {
    if (observerChoice == 1) {
```

```
            // Cancel a reservation
            // Implement cancellation logic
            int FlightNum = 0;
            if (newPassenger.getReservations().size()) {
               cout << "Currently , you have the following Flights : \n";
               for (int i = 0; i < newPassenger.getReservations().size(); i++) {
                  cout << " " << i + 1 << ". Heading " << newPassenger.getReservations()[i]-
>getFlights()[i]->getRoute() << "  [ " << newPassenger.getReservations()[i]->getSeat() << " ] ,
Time : " << "[ " << newPassenger.getReservations()[i]->getFlights()[i]->getTime() << " ]" <<
endl;
               }
               cout << "Enter the flight no. that you want to cancel : ";
               cin >> FlightNum;
               while (cin.fail() || FlightNum < 1 || FlightNum >
newPassenger.getReservations().size()) {
                  cout << "Enter a Valid Flight no. : ";
                  cin >> FlightNum;
               }
               if (rand() % 2 == 1) {
                  cout << "Sorry , Your Cancellation Request can't be executed.\n";
                  cout << "Do you want to make a Special Request ? (yes/no)\n";
                  string ans;
                  getline(cin >> ws, ans);
                  while (ans != "yes" && ans != "no") {
                     cout << "Enter a valid Option: ";
                     getline(cin >> ws, ans);
                  }
                  if (ans == "no") {
                     cout << "Sure ! \n";
                     break;
                  }
```

```
                string request;
                cout << "Enter the request you want to give : \n";
                getline(cin >> ws, request);
                newPassenger.getReservations()[FlightNum - 1]->setSpecialRequest(request);
                cout << "\nConsidering Your Special Request , we have made our
decision.\n\n";
            }
            if (rand() % 2 == 1)
                newPassenger.cancelReservation(FlightNum);
            else cout << "You can't Cancel this flight.\n";
        }
        else cout << "You don't have any Reservations.\n";
    }
    else if (observerChoice == 2) {
        // Handle schedule conflicts
        static int DateCnt = 0;
        static int DayCnt = 0;
        cout << "You have the following Flights Available: \n";
        for (int i = 0; i < airline.getFlights().size(); i++) {
            //string time = to_string((i + 1) % 12 + 1) + ((i % 24) < 13 ? " pm" : " am") + ", Day "
+ to_string(i / 10 + 1);
            if (airline.getFlights()[i]->getCurrentState()->status == "Scheduled") {
                cout << " " << i + 1 << ": ~~~ " << airline.getFlights()[i]->getRoute() << ", Time : [
" << airline.getFlights()[i]->getTime() << " ]" << endl;
            }
        }

        cout << "Enter the Flight number that you want to change schedule of the crew to :
";
        cin >> flightNum;
```

```
            while (cin.fail() || flightNum < 1 || flightNum > 37 || airline.getFlights()[flightNum -
1]->getCurrentState()->status != "Scheduled") {
                cout << "Please Enter a valid Flight Number : ";
                cin >> flightNum;
            }
            cout << "The next possible Time that you can set for this flight is : \n";
            if (DateCnt / 24) DayCnt++;
            string newTime =  to_string((1 + DateCnt++) % 12) + (DateCnt % 24 < 13 ? "pm" :
"am")  + ", Day " + to_string(10+DayCnt);
            cout << newTime << endl;
            cout << "Do you want to Modify the previous time (yes/no)? ";
            string timeopt;
            cin >> timeopt;
            while (timeopt != "yes" && timeopt != "no") {
                cout << "Give a valid answer : ";
                cin >> timeopt;
            }
            if (timeopt == "yes") {
                airline.getFlights()[flightNum - 1]->setTime(newTime);
            }
            cout << "Your Request has been fulfilled.\n";
        }
        break;
    }
    case 3: {
        if (observerChoice == 1) {
            // Modify a reservation
            // Implement modification logic
            int FlightNum = 0;
            if (newPassenger.getReservations().size()) {
```

```
            cout << "Currently , you have the following Flights : \n";
            for (int i = 0; i < newPassenger.getReservations().size(); i++) {
               cout << " " << i + 1 << ". Heading " << newPassenger.getReservations()[i]-
>getFlights()[i]->getRoute() << "  [ " << newPassenger.getReservations()[i]->getSeat() << " ] ,
Time : " << "[ " << newPassenger.getReservations()[i]->getFlights()[i]->getTime() << " ]" <<
endl;
            }
            cout << "Enter the flight no. that you want to Modify : ";
            cin >> FlightNum;
            while (cin.fail() || FlightNum < 1 || FlightNum >
newPassenger.getReservations().size()) {
               cout << "Enter a Valid Flight no. : ";
               cin >> FlightNum;
            }
            if (rand() % 2 == 1) {
               cout << "Sorry , Your Flight Modification Request can't be executed.\n";
               cout << "Do you want to make a Special Request ? (yes/no)\n";
               string ans;
               getline(cin >> ws, ans);
               while (ans != "yes" && ans != "no") {
                  cout << "Enter a valid Option: ";
                  getline(cin >> ws, ans);
               }
               if (ans == "no") {
                  cout << "Sure ! \n";
                  break;
               }
               string request;
               cout << "Enter the request you want to give : \n";
               getline(cin >> ws, request);
               newPassenger.getReservations()[FlightNum - 1]->setSpecialRequest(request);
```

```
                cout << "\nConsidering Your Special Request , we have made our
decision.\n\n";
            }
            newPassenger.modifyReservation(FlightNum - 1);
        }
        else cout << "You don't have any Reservations.\n";
    }
    else if (observerChoice == 2) {
        // Update flight status
        cout << "You have the following Flights Available: \n";
        for (int i = 0; i < airline.getFlights().size(); i++) {
            //string time = to_string((i + 1) % 12 + 1) + ((i % 24) < 13 ? " pm" : " am") + ", Day "
+ to_string(i / 10 + 1);
            //if (airline.getFlights()[i]->getCurrentState()->status == "Scheduled") {
            cout << " " << i + 1 << ": ~~~ " << airline.getFlights()[i]->getRoute() << ", Time : [ "
<< airline.getFlights()[i]->getTime() << " ]" << endl;
            //}
        }
        cout << "Select the one that you want to update the state of : \n";
        cin >> flightNum;
        while (cin.fail() || flightNum < 1 || flightNum > airline.getFlights().size()) {
            cout << "Please Enter a valid Flight Number : ";
            cin >> flightNum;
        }
        cout << "The Current State of the given flight is : " << airline.getFlights()[flightNum -
1]->getState()->status << ". \n";
        cout << "Select any of the following Choices that you want the state to update to:
\n";
        cout << "1. Scheduled \n2. Delayed \n3.Cancelled\n";
        int StChoice;
        cin >> StChoice;
```

```
            while (cin.fail() || StChoice < 1 || StChoice > 3) {
                cout << "Enter a valid Choice : ";
                cin >> StChoice;
            }
            switch (StChoice) {
            case 1:
                delete airline.getFlights()[flightNum - 1]->getState();
                airline.getFlights()[flightNum - 1]->getState() = new ScheduledState();
                airline.getFlights()[flightNum - 1]->getState()->updateStatus("Scheduled");
                break;
            case 2:
                delete airline.getFlights()[flightNum - 1]->getState();
                airline.getFlights()[flightNum - 1]->getState() = new DelayedState();
                airline.getFlights()[flightNum - 1]->getState()->updateStatus("Delayed");
                break;
            case 3:
                delete airline.getFlights()[flightNum - 1]->getState();
                airline.getFlights()[flightNum - 1]->getState() = new CancelledState();
                airline.getFlights()[flightNum - 1]->getState()->updateStatus("Cancelled");
                break;
            default:
                cout << "This line is not supposed to print.\n";
            }
            cout << "Your Required State has been set Successfully.\n";
        }
        break;
    }
    case 4: {
        if (observerChoice == 1) {
```

```
            // No action for Passenger Observer
            cout << "Exiting...\n";
            break;
        }
        else if (observerChoice == 2) {
            // Reroute a flight
            cout << "You have the following Flights Available: \n";
            for (int i = 0; i < airline.getFlights().size(); i++) {
                //string time = to_string((i + 1) % 12 + 1) + ((i % 24) < 13 ? " pm" : " am") + ", Day "
+ to_string(i / 10 + 1);
                if (airline.getFlights()[i]->getCurrentState()->status == "Scheduled") {
                    cout << " " << i + 1 << ": ~~~ " << airline.getFlights()[i]->getRoute() << ", Time : [
" << airline.getFlights()[i]->getTime() << " ]" << endl;
                }
            }
            cout << "Enter the Flight number that you want to Reroute : ";
            cin >> flightNum;
            while (cin.fail() || flightNum < 1 || flightNum > 37 || airline.getFlights()[flightNum -
1]->getCurrentState()->status != "Scheduled") {
                cout << "Please Enter a valid Flight Number : ";
                cin >> flightNum;
            }
            cout << "Enter the Route you want it to follow : \n";
            string newRoute;
            cin >> newRoute;
            airline.getFlights()[flightNum - 1]->setRoute(newRoute);
            cout << "Your desired Flight has been rerouted successfully.\n";
        }
        break;
    }
```

```
case 5: {
    // Logic To Add/Remove Flights
    if (observerChoice == 2) {
        cout << "Select one of the following options: \n";
        cout << "1. Add Flight\n"
            << "2. Remove Flight\n";
        int flChoice;
        cin >> flChoice;
        while (cin.fail() || flChoice < 1 || flChoice > 2) {
            cout << "Choose a Valid Option: ";
            cin >> flChoice;
        }
        if (flChoice == 1) {
            cout << "Enter the route for the new Flight.\n";
            string newRoute;
            cin >> newRoute;
            cout << "Now select any of the following times : \n"
                << "1. 10 pm , Day 7\n2. 11 am ,Day 8\n3. 6 pm , Day 8\n4. 9 pm , Day 8\n";
            int timeChoice;
            cin >> timeChoice;
            while (cin.fail() || timeChoice < 1 || timeChoice > 4) {
                cout << "Enter a Valid Option: ";
                cin >> timeChoice;
            }
            string newtime;
            switch (timeChoice) {
            case 1:
                newtime = "10 pm , Day 7";
                break;
```

```
            case 2:
                newtime = "11 am ,Day 8";
                break;
            case 3:
                newtime = "6 pm , Day 8";
                break;
            case 4:
                newtime = "9 pm , Day 8";
                break;
            }
            Flight* newFlight = new Flight(newRoute,newtime);
            airline.addFlight(newFlight);
            cout << "Your Desired Flight has been added successfully.\n";
        }
        else {
            cout << "You have the following Flights Available: \n";
            for (int i = 0; i < airline.getFlights().size(); i++) {
                //string time = to_string((i + 1) % 12 + 1) + ((i % 24) < 13 ? " pm" : " am") + ",
Day " + to_string(i / 10 + 1);
                //if (airline.getFlights()[i]->getCurrentState()->status == "Scheduled") {
                    cout << " " << i + 1 << ": ~~~ " << airline.getFlights()[i]->getRoute() << ", Time
: [ " << airline.getFlights()[i]->getTime() << " ]" << endl;
                //}
            }

            cout << "Enter the Flight number that you want to remove : ";
            cin >> flightNum;
            while (cin.fail() || flightNum < 1 || flightNum > airline.getFlights().size()) {
                cout << "Please Enter a valid Flight Number : ";
                cin >> flightNum;
```

```
                }
            //    airline.removeFlight(airline.getFlights()[flightNum - 1]);
                cout << "Your intended Flight has been removed successfully.\n";
            }
        }
        break;
    }
    case 6: {
        if (observerChoice == 2) {
            cout << "Exiting...\n";
            break;
        }
    }
    default:
        cout << "\nInvalid choice. Please Enter Again: ";
    }
  } while ((choice != 6 && observerChoice == 2) || (choice != 4 && observerChoice == 1));
  delete observer;
  return 0;
}
```

# OUTPUT

## Selecting **Passenger Observer**

### Booking a Flight

C:\Users\gqadi\Documents\GQ SECOND SEMESTER\airport management system\final\New folder (5)\FINAL_DRAFT1.1.exe

```
 ~*~*~*~*~*~*~* Welcome to Airline Reservation System Simulation *~*~*~*~*~*~*~
Enter the name for the Airline : Pakistan Airline(PIA)

Choose observer state:
1. Passenger Observer
2. Head Observer
1

Choose an option:
1. Book a flight
2. Cancel a reservation
3. Modify a reservation
4. Exit
1
Enter passenger name: Ghulam Qadir
Choose seat type:
1. Economy
2. Business
3. FirstClass
1
You have the following Avaiable flights :
 1: ~~~ NYC-LON, Time : [ 2 pm, Day 1 ]
 4: ~~~ LAX-JFK, Time : [ 5 pm, Day 1 ]
 6: ~~~ DXB-LHR, Time : [ 7 pm, Day 1 ]
 8: ~~~ LHR-CDG, Time : [ 9 pm, Day 1 ]
 10: ~~~ ICN-HND, Time : [ 11 pm, Day 1 ]
 12: ~~~ AMS-FRA, Time : [ 1 pm, Day 2 ]
 14: ~~~ HKG-BKK, Time : [ 3 am, Day 2 ]
 16: ~~~ LAS-DEN, Time : [ 5 am, Day 2 ]
 18: ~~~ SYD-MEL, Time : [ 7 am, Day 2 ]
 20: ~~~ DFW-MIA, Time : [ 9 am, Day 2 ]
 22: ~~~ MCO-LGA, Time : [ 11 am, Day 3 ]
 24: ~~~ YYZ-YVR, Time : [ 1 am, Day 3 ]
 26: ~~~ HND-PEK, Time : [ 3 pm, Day 3 ]
 28: ~~~ AMS-BCN, Time : [ 5 pm, Day 3 ]
 30: ~~~ HKG-TPE, Time : [ 7 pm, Day 3 ]
 32: ~~~ BKK-KUL, Time : [ 9 pm, Day 4 ]
 34: ~~~ MIA-TPA, Time : [ 11 pm, Day 4 ]
 36: ~~~ MSP-DTW, Time : [ 1 pm, Day 4 ]
Enter the flight you want to take : 1
Are you eligible for the loyality program?
 yes

You are successfully Registered in the Flight heading NYC-LON
Regardless of your input , We have decided that you are eligible for the loyality Program.


Considering All the Current Factors , Your final price comes out to be : 213.72$
```

## Cancelling a Reservation

```
Choose an option:
1. Book a flight
2. Cancel a reservation
3. Modify a reservation
4. Exit
2
Currently , you have the following Flights :
 1. Heading NYC-LON  [ Economy ] , Time : [ 2 pm, Day 1 ]
Enter the flight no. that you want to cancel : 1
Sorry , Your Cancellation Request can't be executed.
Do you want to make a Special Request ? (yes/no)
yes
Enter the request you want to give :
cancel the flight I have emergency

Considering Your Special Request , we have made our decision.

You can't Cancel this flight.

Choose an option:
1. Book a flight
2. Cancel a reservation
3. Modify a reservation
4. Exit
2
Currently , you have the following Flights :
 1. Heading NYC-LON  [ Economy ] , Time : [ 2 pm, Day 1 ]
Enter the flight no. that you want to cancel : 2
Enter a Valid Flight no. : 1
Your Desired Reservation has been successfully removed.
```

## Modifying a Reservation

```
Choose an option:
1. Book a flight
2. Cancel a reservation
3. Modify a reservation
4. Exit
3
Currently , you have the following Flights :
 1. Heading NYC-LON  [ Economy ] , Time : [ 2 pm, Day 1 ]
Enter the flight no. that you want to Modify : 2
Enter a Valid Flight no. : 4
Enter a Valid Flight no. : 1
Sorry , Your Flight Modification Request can't be executed.
Do you want to make a Special Request ? (yes/no)
yes
Enter the request you want to give :
please approve my request

Considering Your Special Request , we have made our decision.

You have the option to modify which kind of seat you want to reserve.
Please choose the seat type:
1. Economy
2. Business
3. FirstClass
3
Your Given Seat Type is Successfully Modified.
```

# Selecting **Head Observer**

## Adding crew to flight

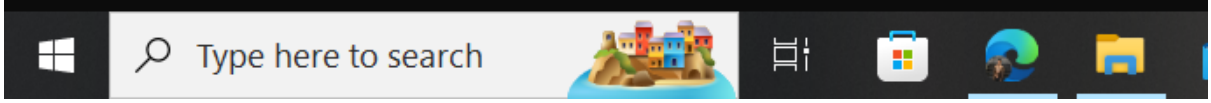
```
Choose observer state:
1. Passenger Observer
2. Head Observer
2

Choose an option:
1. Manage Crews to a flight
2. Handle schedule conflicts
3. Update flight status
4. Reroute a flight
5. Modify Flights
6. Exit
1
Choose one of the following :
1. Add crew to Flight
2. Remove crew from Flight
1
You have the following Flights Available:
 1: ~~~ NYC-LON, Time : [ 2 pm, Day 1 ]
 4: ~~~ LAX-JFK, Time : [ 5 pm, Day 1 ]
 6: ~~~ DXB-LHR, Time : [ 7 pm, Day 1 ]
 8: ~~~ LHR-CDG, Time : [ 9 pm, Day 1 ]
 10: ~~~ ICN-HND, Time : [ 11 pm, Day 1 ]
 12: ~~~ AMS-FRA, Time : [ 1 pm, Day 2 ]
 14: ~~~ HKG-BKK, Time : [ 3 am, Day 2 ]
 16: ~~~ LAS-DEN, Time : [ 5 am, Day 2 ]
 18: ~~~ SYD-MEL, Time : [ 7 am, Day 2 ]
 20: ~~~ DFW-MIA, Time : [ 9 am, Day 2 ]
 22: ~~~ MCO-LGA, Time : [ 11 am, Day 3 ]
 24: ~~~ YYZ-YVR, Time : [ 1 am, Day 3 ]
 26: ~~~ HND-PEK, Time : [ 3 pm, Day 3 ]
 28: ~~~ AMS-BCN, Time : [ 5 pm, Day 3 ]
 30: ~~~ HKG-TPE, Time : [ 7 pm, Day 3 ]
 32: ~~~ BKK-KUL, Time : [ 9 pm, Day 4 ]
 34: ~~~ MIA-TPA, Time : [ 11 pm, Day 4 ]
 36: ~~~ MSP-DTW, Time : [ 1 pm, Day 4 ]
Enter the Flight number that you want to add the crew to : 10
You Currently have the following crew :
 1. John Smith [ Pilot ]
 2. Alice Johnson [ FlightAttendant ]
 3. Michael Brown [ Pilot ]
 4. Emily Davis [ FlightAttendant ]
 5. David Wilson [ Pilot ]
 6. Sarah Martinez [ Pilot ]
 7. Daniel Anderson [ Pilot ]
 8. Laura Taylor [ Pilot ]
 9. James Garcia [ FlightAttendant ]
 10. Emma Rodriguez [ FlightAttendant ]
 11. Liam Lopez [ FlightAttendant ]
 12. Olivia Hernandez [ Pilot ]
 13. Noah Gonzales [ FlightAttendant ]
 14. Ethan Perez [ CoPilot ]
 15. Isabella Carter [ Pilot ]
 16. Mia Collins [ Pilot ]
 17. Aiden Stewart [ FlightAttendant ]
 18. Sophia Morris [ FlightAttendant ]
 19. Logan Evans [ Pilot ]
 20. Amelia Rivera [ FlightAttendant ]
 21. Lucas Hughes [ Pilot ]
 22. Charlotte Flores [ FlightAttendant ]
 23. Jack Gray [ CoPilot ]
 24. Lily Morgan [ Pilot ]
 25. Liam Simmons [ FlightAttendant ]
 26. Samuel Foster [ CoPilot ]
 27. Emily Powell [ FlightAttendant ]
 28. Elizabeth Long [ Pilot ]
 29. Daniel Howard [ Pilot ]
 30. Michael Bailey [ Pilot ]
 31. Emily Bell [ FlightAttendant ]
 32. Ava Murphy [ Pilot ]
 33. William Richardson [ CoPilot ]
 34. Sofia Carter [ FlightAttendant ]
 35. James Foster [ FlightAttendant ]
 36. Charlotte Cox [ CoPilot ]
 37. Ethan Price [ Pilot ]
Enter the Crew Number that you want to assign to the this Flight : 30
Your Desired Crew is added to the desired Flight.
```

## Handling schedule conflict for a flight

```
Choose an option:
1. Manage Crews to a flight
2. Handle schedule conflicts
3. Update flight status
4. Reroute a flight
5. Modify Flights
6. Exit
2
You have the following Flights Available:
 1: ~~~~ NYC-LON, Time : [ 2 pm, Day 1 ]
 4: ~~~~ LAX-JFK, Time : [ 5 pm, Day 1 ]
 6: ~~~~ DXB-LHR, Time : [ 7 pm, Day 1 ]
 8: ~~~~ LHR-CDG, Time : [ 9 pm, Day 1 ]
 10: ~~~~ ICN-HND, Time : [ 11 pm, Day 1 ]
 12: ~~~~ AMS-FRA, Time : [ 1 pm, Day 2 ]
 14: ~~~~ HKG-BKK, Time : [ 3 am, Day 2 ]
 16: ~~~~ LAS-DEN, Time : [ 5 am, Day 2 ]
 18: ~~~~ SYD-MEL, Time : [ 7 am, Day 2 ]
 20: ~~~~ DFW-MIA, Time : [ 9 am, Day 2 ]
 22: ~~~~ MCO-LGA, Time : [ 11 am, Day 3 ]
 24: ~~~~ YYZ-YVR, Time : [ 1 am, Day 3 ]
 26: ~~~~ HND-PEK, Time : [ 3 pm, Day 3 ]
 28: ~~~~ AMS-BCN, Time : [ 5 pm, Day 3 ]
 30: ~~~~ HKG-TPE, Time : [ 7 pm, Day 3 ]
 32: ~~~~ BKK-KUL, Time : [ 9 pm, Day 4 ]
 34: ~~~~ MIA-TPA, Time : [ 11 pm, Day 4 ]
 36: ~~~~ MSP-DTW, Time : [ 1 pm, Day 4 ]
Enter the Flight number that you want to change schedule of the crew to : 12
The next possible Time that you can set for this flight is :
1pm, Day 10
Do you want to Modify the previous time (yes/no)? yes
Your Request has been fulfilled.
```

## Updating Flight Status

```
Choose an option:
1. Manage Crews to a flight
2. Handle schedule conflicts
3. Update flight status
4. Reroute a flight
5. Modify Flights
6. Exit
3
You have the following Flights Available:
 1: ~~~ NYC-LON, Time : [ 2 pm, Day 1 ]
 2: ~~~ LON-TOK, Time : [ 3 pm, Day 1 ]
 3: ~~~ PAR-IST, Time : [ 4 pm, Day 1 ]
 4: ~~~ LAX-JFK, Time : [ 5 pm, Day 1 ]
 5: ~~~ HKG-SIN, Time : [ 6 pm, Day 1 ]
 6: ~~~ DXB-LHR, Time : [ 7 pm, Day 1 ]
 7: ~~~ SFO-ORD, Time : [ 8 pm, Day 1 ]
 8: ~~~ LHR-CDG, Time : [ 9 pm, Day 1 ]
 9: ~~~ JFK-AMS, Time : [ 10 pm, Day 1 ]
 10: ~~~ ICN-HND, Time : [ 11 pm, Day 1 ]
 11: ~~~ SIN-PEK, Time : [ 12 pm, Day 2 ]
 12: ~~~ AMS-FRA, Time : [ 1pm, Day 10 ]
 13: ~~~ LAX-ORD, Time : [ 2 pm, Day 2 ]
 14: ~~~ HKG-BKK, Time : [ 3 am, Day 2 ]
 15: ~~~ CDG-DXB, Time : [ 4 am, Day 2 ]
 16: ~~~ LAS-DEN, Time : [ 5 am, Day 2 ]
 17: ~~~ LHR-FRA, Time : [ 6 am, Day 2 ]
 18: ~~~ SYD-MEL, Time : [ 7 am, Day 2 ]
 19: ~~~ BOS-SFO, Time : [ 8 am, Day 2 ]
 20: ~~~ DFW-MIA, Time : [ 9 am, Day 2 ]
 21: ~~~ ORD-SEA, Time : [ 10 am, Day 3 ]
 22: ~~~ MCO-LGA, Time : [ 11 am, Day 3 ]
 23: ~~~ LAX-ATL, Time : [ 12 am, Day 3 ]
 24: ~~~ YYZ-YVR, Time : [ 1 am, Day 3 ]
 25: ~~~ DEN-PHX, Time : [ 2 pm, Day 3 ]
 26: ~~~ HND-PEK, Time : [ 3 pm, Day 3 ]
 27: ~~~ SEA-IAH, Time : [ 4 pm, Day 3 ]
 28: ~~~ AMS-BCN, Time : [ 5 pm, Day 3 ]
 29: ~~~ FRA-MAD, Time : [ 6 pm, Day 3 ]
 30: ~~~ HKG-TPE, Time : [ 7 pm, Day 3 ]
 31: ~~~ JFK-MCO, Time : [ 8 pm, Day 4 ]
 32: ~~~ BKK-KUL, Time : [ 9 pm, Day 4 ]
 33: ~~~ MUC-IST, Time : [ 10 pm, Day 4 ]
 34: ~~~ MIA-TPA, Time : [ 11 pm, Day 4 ]
 35: ~~~ HNL-LAX, Time : [ 12 pm, Day 4 ]
 36: ~~~ MSP-DTW, Time : [ 1 pm, Day 4 ]
 37: ~~~ ATL-DFW, Time : [ 2 pm, Day 4 ]
Select the one that you want to update the state of :
33
The Current State of the given flight is : Cancelled.
Select any of the following Choices that you want the state to update to:
1. Scheduled
2. Delayed
3.Cancelled
3
Your Required State has been set Successfully.
```



## Rerouting a Flight

C:\Users\gqadi\Documents\GQ SECOND SEMESTER\airport

```
Choose observer state:
1. Passenger Observer
2. Head Observer
2

Choose an option:
1. Manage Crews to a flight
2. Handle schedule conflicts
3. Update flight status
4. Reroute a flight
5. Modify Flights
6. Exit
4
You have the following Flights Available:
 1: ~~~ NYC-LON, Time : [ 2 pm, Day 1 ]
 4: ~~~ LAX-JFK, Time : [ 5 pm, Day 1 ]
 6: ~~~ DXB-LHR, Time : [ 7 pm, Day 1 ]
 8: ~~~ LHR-CDG, Time : [ 9 pm, Day 1 ]
 10: ~~~ ICN-HND, Time : [ 11 pm, Day 1 ]
 12: ~~~ AMS-FRA, Time : [ 1 pm, Day 2 ]
 14: ~~~ HKG-BKK, Time : [ 3 am, Day 2 ]
 16: ~~~ LAS-DEN, Time : [ 5 am, Day 2 ]
 18: ~~~ SYD-MEL, Time : [ 7 am, Day 2 ]
 20: ~~~ DFW-MIA, Time : [ 9 am, Day 2 ]
 22: ~~~ MCO-LGA, Time : [ 11 am, Day 3 ]
 24: ~~~ YYZ-YVR, Time : [ 1 am, Day 3 ]
 26: ~~~ HND-PEK, Time : [ 3 pm, Day 3 ]
 28: ~~~ AMS-BCN, Time : [ 5 pm, Day 3 ]
 30: ~~~ HKG-TPE, Time : [ 7 pm, Day 3 ]
 32: ~~~ BKK-KUL, Time : [ 9 pm, Day 4 ]
 34: ~~~ MIA-TPA, Time : [ 11 pm, Day 4 ]
 36: ~~~ MSP-DTW, Time : [ 1 pm, Day 4 ]
Enter the Flight number that you want to Reroute : 36
Enter the Route you want it to follow :
4
Your desired Flight has been rerouted successfully.

Choose an option:
1. Manage Crews to a flight
2. Handle schedule conflicts
3. Update flight status
4. Reroute a flight
5. Modify Flights
6. Exit
5
Select one of the following options:
1. Add Flight
2. Remove Flight
1
Enter the route for the new Flight.
32
Now select any of the following times :
1. 10 pm , Day 7
2. 11 am ,Day 8
3. 6 pm , Day 8
4. 9 pm , Day 8
2
Your Desired Flight has been added successfully.
```

**Modifying Flights**