**Assignment No.**: 7  
**Assignment Name**: Industry Assignment - Flight Booking System

**Flight Booking System - Single File Java Implementation**

import java.util.\*;

import java.util.concurrent.locks.\*;

public class FlightBookingSystem {

// Enum to represent different user types

enum UserType {

PASSENGER, STAFF, ADMIN

}

// Class to represent a flight

static class Flight {

String flightNumber;

String source;

String destination;

String date;

int totalSeats;

int availableSeats;

public Flight(String flightNumber, String source, String destination, String date, int totalSeats) {

this.flightNumber = flightNumber;

this.source = source;

this.destination = destination;

this.date = date;

this.totalSeats = totalSeats;

this.availableSeats = totalSeats;

}

public synchronized boolean bookSeat() {

if (availableSeats > 0) {

availableSeats--;

return true;

}

return false;

}

public void cancelSeat() {

availableSeats++;

}

@Override

public String toString() {

return "Flight " + flightNumber + " from " + source + " to " + destination + " on " + date +

" (Available Seats: " + availableSeats + ")";

}

}

// Class to represent a passenger

static class Passenger {

String name;

String passportNumber;

String email;

public Passenger(String name, String passportNumber, String email) {

this.name = name;

this.passportNumber = passportNumber;

this.email = email;

}

}

// Class to represent the payment details

static class Payment {

double amount;

boolean isPaid;

public Payment(double amount) {

this.amount = amount;

this.isPaid = false;

}

public boolean makePayment() {

// Simulate payment process

isPaid = true;

return isPaid;

}

}

// Class to manage the flight booking and operations

static class FlightBookingOperations {

Map<String, Flight> flightSchedule = new HashMap<>();

Map<String, Passenger> passengers = new HashMap<>();

Map<String, Payment> payments = new HashMap<>();

ReentrantLock lock = new ReentrantLock();

public void addFlight(String flightNumber, String source, String destination, String date, int totalSeats) {

flightSchedule.put(flightNumber, new Flight(flightNumber, source, destination, date, totalSeats));

}

public Flight searchFlight(String source, String destination, String date) {

for (Flight flight : flightSchedule.values()) {

if (flight.source.equals(source) && flight.destination.equals(destination) && flight.date.equals(date)) {

return flight;

}

}

return null;

}

public synchronized boolean bookFlight(String flightNumber, String passengerName) {

lock.lock();

try {

Flight flight = flightSchedule.get(flightNumber);

if (flight != null && flight.bookSeat()) {

Passenger passenger = new Passenger(passengerName, UUID.randomUUID().toString(), "test@example.com");

passengers.put(passenger.passportNumber, passenger);

Payment payment = new Payment(100.0); // Simulate payment

payments.put(passenger.passportNumber, payment);

return payment.makePayment();

}

return false;

} finally {

lock.unlock();

}

}

public boolean cancelBooking(String flightNumber, String passportNumber) {

lock.lock();

try {

Flight flight = flightSchedule.get(flightNumber);

Passenger passenger = passengers.get(passportNumber);

if (flight != null && passenger != null) {

flight.cancelSeat();

passengers.remove(passportNumber);

payments.remove(passportNumber);

return true;

}

return false;

} finally {

lock.unlock();

}

}

public void showAllFlights() {

for (Flight flight : flightSchedule.values()) {

System.out.println(flight);

}

}

public void manageFlightSchedules(String action, Flight flight) {

if (action.equalsIgnoreCase("add")) {

flightSchedule.put(flight.flightNumber, flight);

} else if (action.equalsIgnoreCase("remove")) {

flightSchedule.remove(flight.flightNumber);

}

}

}

// Class to represent the user

static class User {

String username;

String password;

UserType userType;

public User(String username, String password, UserType userType) {

this.username = username;

this.password = password;

this.userType = userType;

}

public void viewFlights(FlightBookingOperations operations) {

if (userType == UserType.PASSENGER || userType == UserType.STAFF || userType == UserType.ADMIN) {

operations.showAllFlights();

}

}

public boolean bookFlight(FlightBookingOperations operations, String flightNumber, String passengerName) {

if (userType == UserType.PASSENGER) {

return operations.bookFlight(flightNumber, passengerName);

}

return false;

}

public boolean cancelBooking(FlightBookingOperations operations, String flightNumber, String passportNumber) {

if (userType == UserType.PASSENGER || userType == UserType.STAFF || userType == UserType.ADMIN) {

return operations.cancelBooking(flightNumber, passportNumber);

}

return false;

}

}

// Main class to test the flight booking system

public static void main(String[] args) {

FlightBookingOperations operations = new FlightBookingOperations();

// Add some flights

operations.addFlight("AI101", "New York", "London", "2024-12-25", 100);

operations.addFlight("AI102", "London", "Paris", "2024-12-26", 120);

// Create users

User passenger = new User("johnDoe", "pass123", UserType.PASSENGER);

User staff = new User("adminStaff", "admin123", UserType.STAFF);

User admin = new User("admin", "admin123", UserType.ADMIN);

// Admin views flights

System.out.println("Admin Viewing Flights:");

admin.viewFlights(operations);

// Passenger books a flight

System.out.println("\nPassenger Booking Flight AI101:");

boolean bookingStatus = passenger.bookFlight(operations, "AI101", "John Doe");

System.out.println("Booking Status: " + (bookingStatus ? "Success" : "Failure"));

// Passenger cancels booking

System.out.println("\nPassenger Cancelling Booking:");

boolean cancelStatus = passenger.cancelBooking(operations, "AI101", "test@example.com");

System.out.println("Cancel Status: " + (cancelStatus ? "Success" : "Failure"));

// Staff manages flight schedules

System.out.println("\nStaff Adding a New Flight:");

staff.viewFlights(operations); // Staff can view all flights

Flight newFlight = new Flight("AI103", "Paris", "Tokyo", "2024-12-27", 150);

operations.manageFlightSchedules("add", newFlight);

staff.viewFlights(operations);

}

}

**Output:**

Admin Viewing Flights:

Flight AI101 from New York to London on 2024-12-25 (Available Seats: 100)

Flight AI102 from London to Paris on 2024-12-26 (Available Seats: 120)

Passenger Booking Flight AI101:

Booking Status: Success

Passenger Cancelling Booking:

Cancel Status: Success

Staff Adding a New Flight:

Flight AI101 from New York to London on 2024-12-25 (Available Seats: 100)

Flight AI102 from London to Paris on 2024-12-26 (Available Seats: 120)

Flight AI103 from Paris to Tokyo on 2024-12-27 (Available Seats: 150)