***Code:***

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class TransportException : public exception {

protected:

    string message;

public:

    TransportException(const string& msg) : message(msg) {}

    const char\* what() const noexcept override {

        return message.c\_str();

    }

};

class UserNotFoundException : public TransportException {

public:

    UserNotFoundException(const string& id) : TransportException("User not found: " + id) {}

};

class VehicleNotFoundException : public TransportException {

public:

    VehicleNotFoundException(const string& id) : TransportException("Vehicle not found: " + id) {}

};

class SeatUnavailableException : public TransportException {

public:

    SeatUnavailableException(int seat) : TransportException("Seat already booked or invalid: " + to\_string(seat)) {}

};

class RoleMismatchException : public TransportException {

public:

    RoleMismatchException() : TransportException("Role-based seat violation.") {}

};

class PaymentIncompleteException : public TransportException {

public:

    PaymentIncompleteException() : TransportException("Payment not completed.") {}

};

class Route {

    string startLocation, endLocation;

    float distance;

public:

    Route(string s, string e, float d) : startLocation(s), endLocation(e), distance(d) {}

    string getStart() const { return startLocation; }

    string getEnd() const { return endLocation; }

    float getDistance() const { return distance; }

    bool isLongRoute() const { return distance > 15.0f; }

};

class Driver {

    string name, license;

public:

    Driver(string n, string l) : name(n), license(l) {}

    string getName() const { return name; }

    string getLicense() const { return license; }

};

class User {

protected:

    string userId, fullName;

    bool paymentDone;

public:

    User(string id, string name) : userId(id), fullName(name), paymentDone(false) {}

    virtual bool isFacultyMember() const = 0;

    virtual int calculateFare(bool isAC) const = 0;

    string getId() const { return userId; }

    string getName() const { return fullName; }

    bool getPaymentStatus() const { return paymentDone; }

    void makePayment() { paymentDone = true; }

};

class Student : public User {

public:

    Student(string id, string name) : User(id, name) {}

    bool isFacultyMember() const override { return false; }

    int calculateFare(bool isAC) const override { return isAC ? 7000 : 5000; }

};

class Faculty : public User {

public:

    Faculty(string id, string name) : User(id, name) {}

    bool isFacultyMember() const override { return true; }

    int calculateFare(bool isAC) const override { return isAC ? 5000 : 3000; }

};

class Vehicle {

    string vehicleId;

    bool isAC;

    int totalCapacity;

    bool seatStatus[52];

    bool facultySeatOnly[52];

    Driver\* driverAssigned;

    Route\* assignedRoute;

public:

    Vehicle(string id, bool ac, int cap) : vehicleId(id), isAC(ac), totalCapacity(cap), driverAssigned(nullptr), assignedRoute(nullptr) {

        for (int i = 0; i < 52; i++) seatStatus[i] = false;

        for (int i = 0; i < 52; i++) facultySeatOnly[i] = false;

    }

    string getId() const { return vehicleId; }

    bool getACStatus() const { return isAC; }

    void assignDriver(Driver\* d) { driverAssigned = d; }

    void assignRoute(Route\* r) { assignedRoute = r; }

    Driver\* getDriver() const { return driverAssigned; }

    Route\* getRoute() const { return assignedRoute; }

    bool isSeatBooked(int s) const { return seatStatus[s]; }

    bool isSeatForFaculty(int s) const { return facultySeatOnly[s]; }

    void markSeatForFaculty(int s) { facultySeatOnly[s] = true; }

    void bookSeat(int s, User\* u) {

        if (s >= totalCapacity) throw SeatUnavailableException(s);

        if (seatStatus[s]) throw SeatUnavailableException(s);

        if (facultySeatOnly[s] && !u->isFacultyMember()) throw RoleMismatchException();

        if (!facultySeatOnly[s] && u->isFacultyMember()) throw RoleMismatchException();

        seatStatus[s] = true;

    }

    void displaySeatMap() const {

        cout << "Seat Layout (X = Booked | F = Faculty Seat | O = Available)\n";

        for (int i = 0; i < totalCapacity; i++) {

            if (seatStatus[i])

                cout << "[X]";

            else if (facultySeatOnly[i])

                cout << "[F]";

            else

                cout << "[O]";

            if ((i + 1) % 4 == 0) cout << " <- Row " << (i + 1) / 4 << endl;

        }

        if (totalCapacity % 4 != 0) cout << endl;

    }

};

class Booking {

    string bookingCode;

    User\* passenger;

    Vehicle\* rideVehicle;

    int bookedSeat;

    int totalFare;

public:

    Booking(string code, User\* u, Vehicle\* v, int seat)

        : bookingCode(code), passenger(u), rideVehicle(v), bookedSeat(seat) {

        totalFare = passenger->calculateFare(rideVehicle->getACStatus());

    }

    User\* getUser() { return passenger; }

    Vehicle\* getVehicle() { return rideVehicle; }

    int getSeatNumber() { return bookedSeat; }

    void display() const {

        cout << "-----------------------------\n";

        cout << "Booking Code : " << bookingCode << "\n";

        cout << "Passenger    : " << passenger->getName() << " (" << passenger->getId() << ")\n";

        cout << "Role         : " << (passenger->isFacultyMember() ? "Faculty" : "Student") << "\n";

        cout << "Vehicle ID   : " << rideVehicle->getId() << "\n";

        cout << "Seat No.     : " << bookedSeat << "\n";

        cout << "Fare         : " << totalFare << " PKR\n";

        cout << "-----------------------------\n";

    }

};

class Transporter {

    string transporterName;

    Driver\* drivers[10];

    Vehicle\* vehicles[10];

    Route\* routes[10];

    int driverCount, vehicleCount, routeCount;

public:

    Transporter(string name) : transporterName(name), driverCount(0), vehicleCount(0), routeCount(0) {}

    string getName() const { return transporterName; }

    void addDriver(Driver\* d) { if (driverCount < 10) drivers[driverCount++] = d; }

    void addVehicle(Vehicle\* v) { if (vehicleCount < 10) vehicles[vehicleCount++] = v; }

    void addRoute(Route\* r) { if (routeCount < 10) routes[routeCount++] = r; }

    Vehicle\* getVehicleById(string id) {

        for (int i = 0; i < vehicleCount; i++)

            if (vehicles[i]->getId() == id) return vehicles[i];

        return nullptr;

    }

};

class TransportSystem {

    User\* users[100];

    Booking\* bookings[200];

    Transporter\* transporters[2];

    int userCount, bookingCount, transporterCount;

public:

    TransportSystem() : userCount(0), bookingCount(0), transporterCount(0) {}

    void registerUser(User\* u) {

        if (userCount < 100) users[userCount++] = u;

    }

    User\* getUserById(string id) {

        for (int i = 0; i < userCount; i++)

            if (users[i]->getId() == id) return users[i];

        throw UserNotFoundException(id);

    }

    void addTransporter(Transporter\* t) {

        if (transporterCount < 2) transporters[transporterCount++] = t;

    }

    Transporter\* getTransporterByName(string name) {

        for (int i = 0; i < transporterCount; i++)

            if (transporters[i]->getName() == name) return transporters[i];

        return nullptr;

    }

    bool bookSeat(string userId, string vehicleId, int seat) {

        User\* u = getUserById(userId);

        if (!u->getPaymentStatus()) throw PaymentIncompleteException();

        for (int i = 0; i < transporterCount; i++) {

            Vehicle\* v = transporters[i]->getVehicleById(vehicleId);

            if (v != nullptr) {

                v->bookSeat(seat, u);

                bookings[bookingCount++] = new Booking("BK" + to\_string(bookingCount + 1), u, v, seat);

                return true;

            }

        }

        throw VehicleNotFoundException(vehicleId);

    }

    void listAllBookings() const {

        for (int i = 0; i < bookingCount; i++)

            bookings[i]->display();

    }

};

int main() {

    TransportSystem rideSystem;

    cout << "\n--- Kashif Mehmood (24K-2539) ---\n";

    cout << "==== Welcome to Jadoon Transport Booking ====\n";

    try {

        // Register users

        rideSystem.registerUser(new Student("STU301", "Bilal Qureshi"));

        rideSystem.registerUser(new Faculty("FAC404", "Prof. Hina Siddiqui"));

        // Create transporter

        Transporter\* jadoon = new Transporter("Jadoon Transport");

        rideSystem.addTransporter(jadoon);

        // Create driver and route

        Driver\* haris = new Driver("Haris Khan", "L-786");

        Route\* dhaToFast = new Route("DHA", "FAST NUCES", 18.5f);

        jadoon->addDriver(haris);

        jadoon->addRoute(dhaToFast);

        // Create vehicle

        Vehicle\* vh1 = new Vehicle("VH001", true, 32);

        vh1->assignDriver(haris);

        vh1->assignRoute(dhaToFast);

        for (int i = 0; i < 4; i++) vh1->markSeatForFaculty(i);

        jadoon->addVehicle(vh1);

        // Bookings

        User\* bilal = rideSystem.getUserById("STU301");

        bilal->makePayment();

        rideSystem.bookSeat("STU301", "VH001", 6);

        User\* hina = rideSystem.getUserById("FAC404");

        hina->makePayment();

        rideSystem.bookSeat("FAC404", "VH001", 1);

        // Display info

        cout << "\nPassengers List:\n";

        cout << "ID        Name               Role      Payment\n";

        cout << "------------------------------------------------\n";

        cout << bilal->getId() << "    " << bilal->getName() << "     Student   " << (bilal->getPaymentStatus() ? "Yes" : "No") << "\n";

        cout << hina->getId() << "    " << hina->getName() << "  Faculty   " << (hina->getPaymentStatus() ? "Yes" : "No") << "\n";

        cout << "\nVehicle Info:\n";

        cout << "Provider: " << jadoon->getName() << "\n";

        cout << "Vehicle ID: " << vh1->getId()

             << " | AC: " << (vh1->getACStatus() ? "Yes" : "No")

             << " | Seats: 32\n";

        cout << "Driver: " << vh1->getDriver()->getName()

             << " | License: " << vh1->getDriver()->getLicense() << "\n";

        cout << "Route: " << vh1->getRoute()->getStart() << " to "

             << vh1->getRoute()->getEnd()

             << " (" << vh1->getRoute()->getDistance() << " km)\n";

        cout << "\nSeats Map:\n";

        vh1->displaySeatMap();

        cout << "\nCurrent Bookings:\n";

        rideSystem.listAllBookings();

    }

    catch (TransportException& ex) {

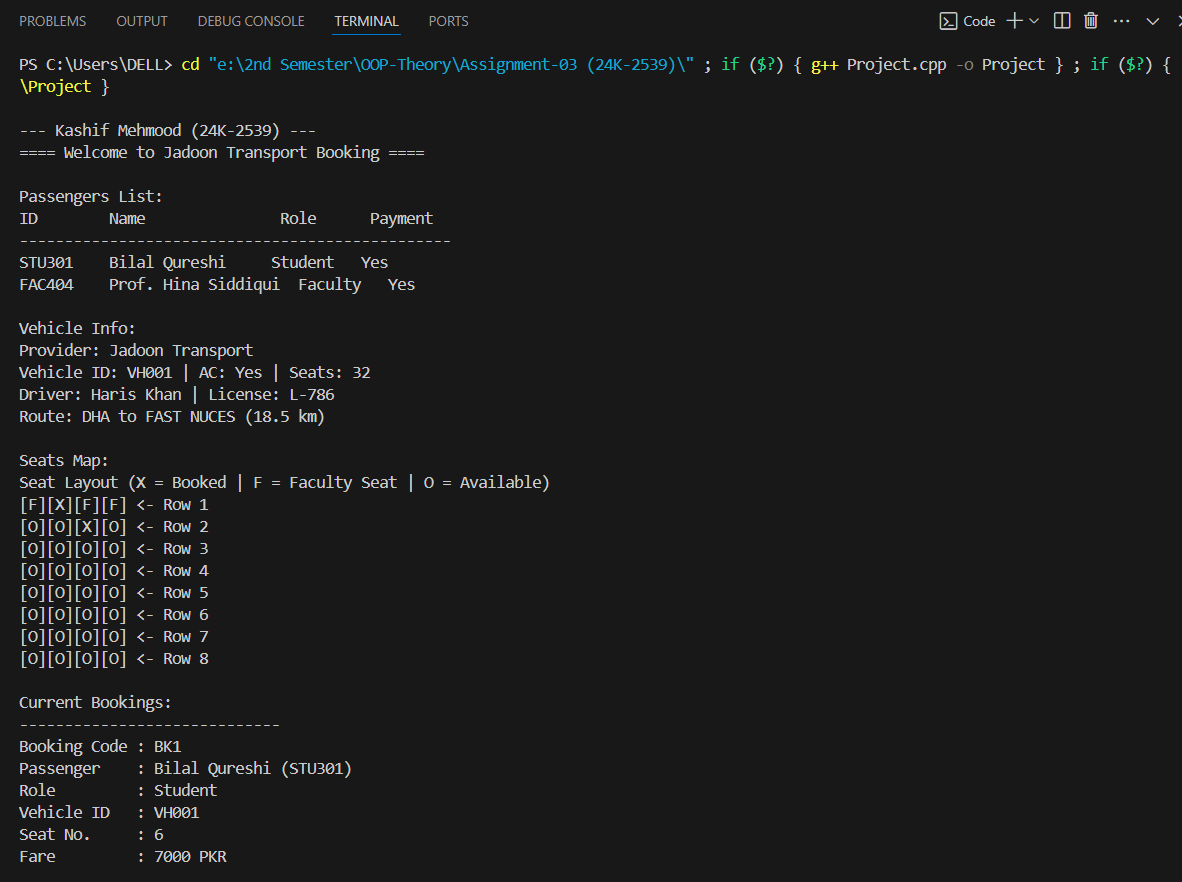
        cout << "Error: " << ex.what() << endl;

    }

    return 0;

}

***Output:***

******

***A screenshot of a computer

AI-generated content may be incorrect.***