#include <iostream>

#include <string>

#include <vector>

using namespace std;

class TransportError : public exception

{

protected:

    string message;

public:

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

    const char \*what() const noexcept override

    {

        return message.c\_str();

    }

};

class PassengerNotFoundError : public TransportError

{

public:

    PassengerNotFoundError(const string &passengerId)

        : TransportError("Passenger not found: " + passengerId) {}

};

class BusNotFoundError : public TransportError

{

public:

    BusNotFoundError(const string &busId)

        : TransportError("Bus not found: " + busId) {}

};

class SeatOccupiedError : public TransportError

{

public:

    SeatOccupiedError(int seatNumber)

        : TransportError("Seat already booked or invalid: " + to\_string(seatNumber)) {}

};

class RoleRestrictedError : public TransportError

{

public:

    RoleRestrictedError()

        : TransportError("Role-based seat restriction violated.") {}

};

class PaymentPendingError : public TransportError

{

public:

    PaymentPendingError()

        : TransportError("Payment has not been completed.") {}

};

class TravelRoute

{

    string departure, destination;

    float distanceInKilometers;

public:

    TravelRoute(string from, string to, float km)

        : departure(from), destination(to), distanceInKilometers(km) {}

    string getDeparture() const { return departure; }

    string getDestination() const { return destination; }

    float getDistance() const { return distanceInKilometers; }

    bool isLongDistance() const { return distanceInKilometers > 20.0f; }

};

class Chauffeur

{

    string name, drivingLicense;

public:

    Chauffeur(string driverName, string license)

        : name(driverName), drivingLicense(license) {}

    string getName() const { return name; }

    string getLicense() const { return drivingLicense; }

};

class Traveler

{

protected:

    string travelerId, fullName;

    bool hasPaid;

public:

    Traveler(string id, string name)

        : travelerId(id), fullName(name), hasPaid(false) {}

    virtual bool isInstructor() const = 0;

    virtual int calculateTicketPrice(bool hasAirConditioning) const = 0;

    string getId() const { return travelerId; }

    string getName() const { return fullName; }

    bool hasCompletedPayment() const { return hasPaid; }

    void markPaymentAsComplete() { hasPaid = true; }

};

class Pupil : public Traveler

{

public:

    Pupil(string id, string name) : Traveler(id, name) {}

    bool isInstructor() const override { return false; }

    int calculateTicketPrice(bool hasAirConditioning) const override

    {

        return hasAirConditioning ? 6500 : 4500;

    }

};

class Professor : public Traveler

{

public:

    Professor(string id, string name) : Traveler(id, name) {}

    bool isInstructor() const override { return true; }

    int calculateTicketPrice(bool hasAirConditioning) const override

    {

        return hasAirConditioning ? 4500 : 2500;

    }

};

class Bus

{

    string busNumber;

    bool hasAirConditioning;

    int totalSeats;

    bool seatBookedStatus[52];

    bool reservedForFaculty[52];

    Chauffeur \*assignedChauffeur;

    TravelRoute \*currentRoute;

public:

    Bus(string id, bool ac, int seats)

        : busNumber(id), hasAirConditioning(ac), totalSeats(seats),

          assignedChauffeur(nullptr), currentRoute(nullptr)

    {

        for (int i = 0; i < 52; ++i)

        {

            seatBookedStatus[i] = false;

            reservedForFaculty[i] = false;

        }

    }

    string getBusId() const { return busNumber; }

    bool hasAC() const { return hasAirConditioning; }

    void assignChauffeur(Chauffeur \*chauffeur) { assignedChauffeur = chauffeur; }

    void assignRoute(TravelRoute \*route) { currentRoute = route; }

    Chauffeur \*getDriver() const { return assignedChauffeur; }

    TravelRoute \*getRoute() const { return currentRoute; }

    bool isSeatBooked(int seatNumber) const { return seatBookedStatus[seatNumber]; }

    bool isFacultySeat(int seatNumber) const { return reservedForFaculty[seatNumber]; }

    void designateSeatForFaculty(int seatNumber) { reservedForFaculty[seatNumber] = true; }

    void bookSeat(int seatNumber, Traveler \*traveler)

    {

        if (seatNumber >= totalSeats)

            throw SeatOccupiedError(seatNumber);

        if (seatBookedStatus[seatNumber])

            throw SeatOccupiedError(seatNumber);

        if (reservedForFaculty[seatNumber] && !traveler->isInstructor())

            throw RoleRestrictedError();

        if (!reservedForFaculty[seatNumber] && traveler->isInstructor())

            throw RoleRestrictedError();

        seatBookedStatus[seatNumber] = true;

    }

    void displaySeatingArrangement() const

    {

        cout << "Bus Seating 1 = Occupied | F = Faculty Only | 0 = Available)\n";

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

        {

            if (seatBookedStatus[i])

                cout << "[1]";

            else if (reservedForFaculty[i])

                cout << "[F]";

            else

                cout << "[0]";

            if ((i + 1) % 4 == 0)

                cout << " <- Row " << (i + 1) / 4 << endl;

        }

        if (totalSeats % 4 != 0)

            cout << endl;

    }

};

class Reservation

{

    string reservationNumber;

    Traveler \*passenger;

    Bus \*vehicle;

    int seatNumber;

    int ticketPrice;

public:

    Reservation(string code, Traveler \*traveler, Bus \*bus, int seat)

        : reservationNumber(code), passenger(traveler), vehicle(bus), seatNumber(seat)

    {

        ticketPrice = passenger->calculateTicketPrice(vehicle->hasAC());

    }

    void displayDetails() const

    {

        cout << "Reservation No. : " << reservationNumber << "\n";

        cout << "Passenger Name  : " << passenger->getName()

             << " (" << passenger->getId() << ")\n";

        cout << "Role            : " << (passenger->isInstructor() ? "Professor" : "Pupil") << "\n";

        cout << "Bus ID          : " << vehicle->getBusId() << "\n";

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

        cout << "Ticket Price    : " << ticketPrice << " PKR\n";

    }

};

class TransportCompany

{

    string companyName;

    vector<Chauffeur \*> chauffeurs;

    vector<Bus \*> buses;

    vector<TravelRoute \*> routes;

public:

    TransportCompany(string name) : companyName(name) {}

    string getName() const { return companyName; }

    void addChauffeur(Chauffeur \*chauffeur)

    {

        chauffeurs.push\_back(chauffeur);

    }

    void addBus(Bus \*bus)

    {

        buses.push\_back(bus);

    }

    void addRoute(TravelRoute \*route)

    {

        routes.push\_back(route);

    }

    Bus \*findBusById(string busId)

    {

        for (Bus \*bus : buses)

        {

            if (bus->getBusId() == busId)

            {

                return bus;

            }

        }

        return nullptr;

    }

};

class TransportManagementSystem

{

    vector<Traveler \*> travelers;

    vector<Reservation \*> reservations;

    vector<TransportCompany \*> companies;

public:

    TransportManagementSystem() {}

    void registerTraveler(Traveler \*traveler)

    {

        travelers.push\_back(traveler);

    }

    Traveler \*findTravelerById(string travelerId)

    {

        for (Traveler \*traveler : travelers)

        {

            if (traveler->getId() == travelerId)

            {

                return traveler;

            }

        }

        throw PassengerNotFoundError(travelerId);

    }

    void addCompany(TransportCompany \*company)

    {

        companies.push\_back(company);

    }

    TransportCompany \*findCompanyByName(string companyName)

    {

        for (TransportCompany \*company : companies)

        {

            if (company->getName() == companyName)

            {

                return company;

            }

        }

        return nullptr;

    }

    bool makeReservation(string travelerId, string busId, int seatNumber)

    {

        Traveler \*traveler = findTravelerById(travelerId);

        if (!traveler->hasCompletedPayment())

        {

            throw PaymentPendingError();

        }

        for (TransportCompany \*company : companies)

        {

            Bus \*bus = company->findBusById(busId);

            if (bus != nullptr)

            {

                bus->bookSeat(seatNumber, traveler);

                reservations.push\_back(new Reservation("RES" + to\_string(reservations.size() + 1), traveler, bus, seatNumber));

                return true;

            }

        }

        throw BusNotFoundError(busId);

    }

    void displayAllReservations() const

    {

        for (const auto &reservation : reservations)

        {

            reservation->displayDetails();

        }

    }

};

int main()

{

    TransportManagementSystem transportSystem;

    try

    {

        transportSystem.registerTraveler(new Pupil("S25-001", "Ali Khan"));

        transportSystem.registerTraveler(new Professor("F25-007", "Dr. Aisha"));

        TransportCompany \*swiftWheels = new TransportCompany("Swift Wheels");

        transportSystem.addCompany(swiftWheels);

        Chauffeur \*driverA = new Chauffeur("Kamran", "D4567");

        swiftWheels->addChauffeur(driverA);

        TravelRoute \*routeA = new TravelRoute("Lahore", "Islamabad", 370.5f);

        swiftWheels->addRoute(routeA);

        Bus \*busA = new Bus("LHR-ISB1", true, 45);

        busA->assignChauffeur(driverA);

        busA->assignRoute(routeA);

        busA->designateSeatForFaculty(0);

        busA->designateSeatForFaculty(2);

        swiftWheels->addBus(busA);

        Traveler \*student1 = transportSystem.findTravelerById("S25-001");

        student1->markPaymentAsComplete();

        Traveler \*faculty1 = transportSystem.findTravelerById("F25-007");

        faculty1->markPaymentAsComplete();

        transportSystem.makeReservation("S25-001", "LHR-ISB1", 5);

        transportSystem.makeReservation("F25-007", "LHR-ISB1", 2);

        busA->displaySeatingArrangement();

        transportSystem.displayAllReservations();

    }

    catch (TransportError &e)

    {

        cerr << "Transport Error: " << e.what() << endl;

    }

    return 0;

}

