**OOP ASSIGNMENT 3**

**Code:**

**#include <iostream>**

**#include <fstream>**

**#include <string>**

**#include <stdexcept>**

**using namespace std;**

**const int MAX\_USERS = 100;**

**const int MAX\_VEHICLES = 50;**

**const int MAX\_BOOKINGS = 200;**

**const int MAX\_SEATS = 52;**

**const int MAX\_ROUTES = 20;**

**const int MAX\_TRANSPORTERS = 2;**

**const int MAX\_DRIVERS = 50;**

**class Vehicle;**

**class User {**

**protected:**

**string name;**

**int id;**

**bool hasPaid;**

**public:**

**User(int id, const string& name)**

**: id(id), name(name), hasPaid(false) {}**

**virtual ~User () {}**

**virtual void makePayment() {**

**hasPaid = true;**

**}**

**virtual double getFare(bool isAC) const = 0;**

**bool paymentDone() const { return hasPaid; }**

**string getName() const { return name; }**

**int getID() const { return id; }**

**virtual string getType() const = 0;**

**};**

**class Student : public User {**

**public:**

**Student(const string& n, int i) : User(i, n) {}**

**double getFare(bool isAC) const override {**

**return 3800 + (isAC ? 2000 : 0);**

**}**

**string getType() const override {**

**return "Student";**

**}**

**};**

**class Faculty : public User {**

**public:**

**Faculty(const string& n, int i) : User(i, n) {}**

**double getFare(bool isAC) const override {**

**return 5000 + (isAC ? 2000 : 0);**

**}**

**string getType() const override {**

**return "Faculty";**

**}**

**};**

**class Driver {**

**private:**

**string name;**

**string licenseNo;**

**public:**

**Driver(const string& n = "", const string& lic = "") : name(n), licenseNo(lic) {}**

**void display() const {**

**cout << "Driver: " << name << ", License: " << licenseNo << endl;**

**}**

**};**

**class Route {**

**private:**

**string start;**

**string end;**

**int distance;**

**public:**

**Route(const string& start = "", const string& end = "", int dist = 0)**

**: start(start), end(end), distance(dist) {}**

**bool isLongRoute() const {**

**return distance >= 25;**

**}**

**int getDistance() const { return distance; }**

**void display() const {**

**cout << "Route: " << start << " to " << end**

**<< " (" << distance << " km)";**

**if (isLongRoute()) cout << " [Long Route]";**

**cout << endl;**

**}**

**string getStart() const { return start; }**

**string getEnd() const { return end; }**

**};**

**class Seat {**

**private:**

**bool isBooked;**

**User\* bookedBy;**

**string seatType;**

**public:**

**Seat(const string& type = "Student") : isBooked(false), bookedBy(nullptr), seatType(type) {}**

**bool book(User\* user) {**

**if (isBooked || user->getType() != seatType) return false;**

**bookedBy = user;**

**isBooked = true;**

**return true;**

**}**

**bool isAvailable() const { return !isBooked; }**

**string getSeatType() const { return seatType; }**

**void display(int index) const {**

**cout << "Seat " << index << ": " << seatType**

**<< (isBooked ? " [Booked]" : " [Available]") << endl;**

**}**

**User\* getBookedUser () const { return bookedBy; }**

**};**

**class Vehicle {**

**private:**

**int id;**

**bool isAC;**

**string type;**

**Route\* route;**

**Driver driver;**

**Seat seats[MAX\_SEATS];**

**int seatCount;**

**public:**

**Vehicle(int vID, const string& vType, bool ac, Route\* r, const Driver& d)**

**: id(vID), isAC(ac), type(vType), route(r), driver(d) {**

**seatCount = (vType == "Bus") ? 52 : 32;**

**for (int i = 0; i < seatCount; i++) {**

**seats[i] = (i < seatCount / 2) ? Seat("Student") : Seat("Faculty");**

**}**

**}**

**void display() const {**

**cout << "\nVehicle ID: " << id << " (" << type << ") "**

**<< (isAC ? "[AC]" : "[Non-AC]") << endl;**

**driver.display();**

**route->display();**

**cout << "Seats Status:\n";**

**for (int i = 0; i < seatCount; ++i) {**

**seats[i].display(i + 1);**

**}**

**}**

**bool bookSeat(User\* user) {**

**for (int i = 0; i < seatCount; ++i) {**

**if (seats[i].isAvailable() && seats[i].getSeatType() == user->getType()) {**

**return seats[i].book(user);**

**}**

**}**

**return false;**

**}**

**int getID() const { return id; }**

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

**bool isACVehicle() const { return isAC; }**

**};**

**class Bus : public Vehicle {**

**public:**

**Bus(int vID, bool ac, Route\* r, const Driver& d)**

**: Vehicle(vID, "Bus", ac, r, d) {}**

**};**

**class Coaster : public Vehicle {**

**public:**

**Coaster(int vID, bool ac, Route\* r, const Driver& d)**

**: Vehicle(vID, "Coaster", ac, r, d) {}**

**};**

**class BookingException : public exception {**

**private:**

**string message;**

**public:**

**BookingException(const string& msg) : message(msg) {}**

**const char\* what() const noexcept override {**

**return message.c\_str();**

**}**

**};**

**class Transporter {**

**private:**

**string name;**

**Vehicle\* vehicles[MAX\_VEHICLES];**

**int vehicleCount;**

**public:**

**Transporter(const string& tName = "") : name(tName), vehicleCount(0) {}**

**bool addVehicle(Vehicle\* v) {**

**if (vehicleCount < MAX\_VEHICLES) {**

**vehicles[vehicleCount++] = v;**

**return true;**

**}**

**return false;**

**}**

**Vehicle\* getVehicleByID(int id) {**

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

**if (vehicles[i]->getID() == id)**

**return vehicles[i];**

**}**

**return nullptr;**

**}**

**void displayVehicles() const {**

**cout << "\n--- Vehicles under " << name << " ---\n";**

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

**vehicles[i]->display();**

**}**

**string getName() const { return name; }**

**};**

**class Booking {**

**private:**

**User\* user;**

**Vehicle\* vehicle;**

**Route\* route;**

**public:**

**Booking(User\* u, Vehicle\* v)**

**: user(u), vehicle(v), route(v->getRoute()) {}**

**void confirm() {**

**if (!user->paymentDone())**

**throw BookingException("Payment not completed!");**

**if (!vehicle->bookSeat(user))**

**throw BookingException("No available seat for this user type!");**

**cout << "Booking confirmed for " << user->getName() << " on vehicle ID " << vehicle->getID() << endl;**

**}**

**void display() const {**

**cout << "\n--- Booking Summary ---\n";**

**cout << ":User  " << user->getName() << "\n";**

**vehicle->getRoute()->display();**

**}**

**};**

**class TransportSystem {**

**private:**

**User\* users[MAX\_USERS];**

**Driver\* drivers[MAX\_DRIVERS];**

**Route\* routes[MAX\_ROUTES];**

**Transporter\* transporters[MAX\_TRANSPORTERS];**

**Booking\* bookings[MAX\_BOOKINGS];**

**int userCount, driverCount, routeCount, bookingCount;**

**public:**

**TransportSystem() : userCount(0), driverCount(0), routeCount(0), bookingCount(0) {**

**transporters[0] = new Transporter("Nadeem Transporter");**

**transporters[1] = new Transporter("Zulfiqar Transporter");**

**}**

**~ TransportSystem() {**

**for (int i = 0; i < userCount; ++i) delete users[i];**

**for (int i = 0; i < driverCount; ++i) delete drivers[i];**

**for (int i = 0; i < routeCount; ++i) delete routes[i];**

**for (int i = 0; i < bookingCount; ++i) delete bookings[i];**

**delete transporters[0];**

**delete transporters[1];**

**}**

**void addUser (User\* u) {**

**if (userCount < MAX\_USERS) users[userCount++] = u;**

**}**

**void addDriver(Driver\* d) {**

**if (driverCount < MAX\_DRIVERS) drivers[driverCount++] = d;**

**}**

**void addRoute(Route\* r) {**

**if (routeCount < MAX\_ROUTES) routes[routeCount++] = r;**

**}**

**void addVehicleToTransporter(Vehicle\* v, int transporterIndex) {**

**transporters[transporterIndex]->addVehicle(v);**

**}**

**User\* getUserByID(int id) {**

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

**if (users[i]->getID() == id)**

**return users[i];**

**throw BookingException("User not found.");**

**}**

**void bookSeat(int userID, int vehicleID, int transporterIndex) {**

**User\* user = getUserByID(userID);**

**Vehicle\* vehicle = transporters[transporterIndex]->getVehicleByID(vehicleID);**

**if (!vehicle) throw BookingException("Vehicle not found.");**

**Booking\* b = new Booking(user, vehicle);**

**try {**

**b->confirm();**

**bookings[bookingCount++] = b;**

**} catch (BookingException& e) {**

**delete b;**

**throw;**

**}**

**}**

**void showAll() {**

**cout << "\n========= All Users =========\n";**

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

**cout << users[i]->getName() << (users[i]->paymentDone() ? " (Paid)" : " (Not Paid)") << endl;**

**cout << "\n========= Routes =========\n";**

**for (int i = 0; i < routeCount; ++i)**

**routes[i]->display();**

**for (int i = 0; i < MAX\_TRANSPORTERS; ++i)**

**transporters[i]->displayVehicles();**

**}**

**void saveUsersToFile(const string& filename) {**

**ofstream out(filename);**

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

**out << users[i]->getID() << "," << users[i]->getName() << "," << users[i]->paymentDone() << "\n";**

**out.close();**

**}**

**void loadUsersFromFile(const string& filename) {**

**ifstream in(filename);**

**int id; string name; bool paid;**

**while (in >> id) {**

**in.ignore(); getline(in, name, ','); in >> paid;**

**User\* user;**

**if (paid)**

**user = new Student(name, id);**

**else**

**user = new Faculty(name, id);**

**addUser (user);**

**}**

**in.close();**

**}**

**};**

**int main() {**

**cout << "Name: Muhammad Omer Siddiqui\nRoll No: 24K-0022\n" << endl;**

**TransportSystem system;**

**Route\* r1 = new Route("FAST - Landhi", "FAST", 45);**

**Route\* r2 = new Route("FAST - Clifton", "FAST", 22);**

**system.addRoute(r1);**

**system.addRoute(r2);**

**Driver\* d1 = new Driver("Akbar", "DL001");**

**Driver\* d2 = new Driver("Imran", "DL002");**

**system.addDriver(d1);**

**system.addDriver(d2);**

**Vehicle\* v1 = new Bus(101, true, r1, \*d1);**

**Vehicle\* v2 = new Coaster(202, false, r2, \*d2);**

**system.addVehicleToTransporter(v1, 0);**

**system.addVehicleToTransporter(v2, 1);**

**User\* u1 = new Student("Ali", 240006);**

**User\* u2 = new Faculty("Prof. Basit", 156);**

**system.addUser (u1);**

**system.addUser (u2);**

**try {**

**u1->makePayment();**

**system.bookSeat(240006, 101, 0);**

**u2->makePayment();**

**system.bookSeat(156, 202, 1);**

**}**

**catch (BookingException& e) {**

**cout << e.what() << endl;**

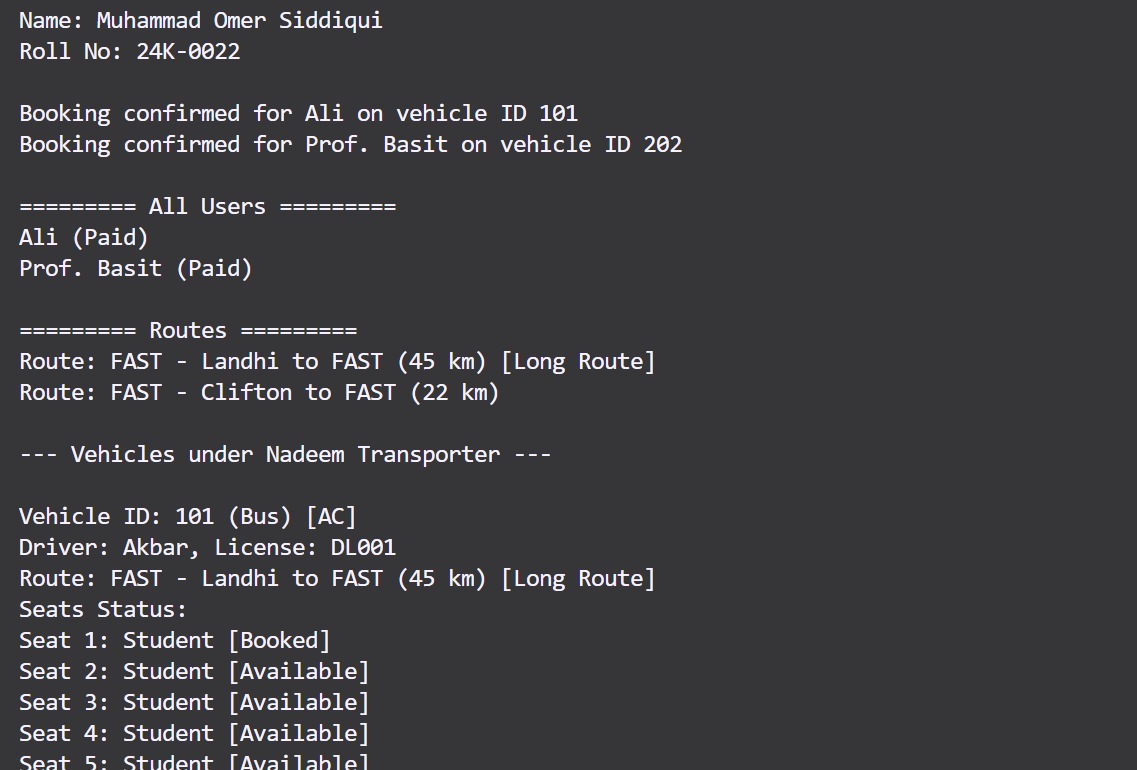
**}**

**system.showAll();**

**system.saveUsersToFile("users.txt");**

**return 0;**

**}**

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.