

Public Transportation Management System

Student Name: Võ Hoàng Tùng

Student ID: 21110811

1. Object-Oriented Analysis (OOA)

Step 1: Identify Objects (Nouns)

- **Schedule**
- **Vehicle**
- **ExpressBus**
- **Station**
- **Passenger**

Step 2: Identify Attributes (Descriptive Nouns)

- **Schedule:** time, action, route
- **Vehicle:** route, capacity, passengers, assignedStations, vehicleSchedules.
- **ExpressBus:** Inherits all Vehicle attributes, expressSpeed.
- **Station:** name, location, schedules.
- **Passenger:** name, id, bookedTickets.

Step 3: Identify Methods (Verbs)

- **Schedule:** displayInfo().
- **Vehicle:** calculateTravelTime(), assignToStation(), addSchedule(), bookSeat(), cancelSeat(), displayInfo().
- **ExpressBus:** calculateTravelTime(), displayInfo().
- **Station:** addSchedule(), displayInfo().
- **Passenger:** bookTicket(), cancelTicket(), displayInfo().

Step 4: Inheritance Relationships

ExpressBus inherits from **Vehicle**, inheriting attributes like route but adding speed and overriding calculateTravelTime() to account for express efficiency

2. Class Design and Inheritance Explanation

2.1. Design

I chose single inheritance with ExpressBus inheriting from Vehicle for several key reasons:

- Logical Relationship: An express bus is fundamentally still a vehicle, just with enhanced capabilities
- Code Reuse: Common functionality (booking, capacity management) doesn't need duplication
- Polymorphism: The same interface can handle different vehicle types seamlessly
- Future Extensibility: Additional vehicle types can easily be added to the hierarchy

2.2. Virtual Function Implementation

Key virtual functions demonstrate polymorphic behavior:

- Vehicle class:

```
// Calculate travel time
virtual double calculateTravelTime(double distance)
{
    return distance / 50.0; // 50 km/h
}
```

- ExpressBus class:

```
double calculateTravelTime(double distance) override
{
    return distance / expressSpeed;
}
```

3. Code Walkthrough

- **Schedule:** This class stores a time, an action ("departure" or "arrival"), and a route. It has simple getters and a displayInfo() function to print the schedule.
- **Vehicle Class:** The Vehicle class is the main base class. It keeps track of the route, capacity, current passengers, assigned stations, and schedules. The constructor sets the route and capacity. The calculateTravelTime() method uses 50 km/h by default. Vehicles can be assigned to stations and given schedules. The

bookSeat() and cancelSeat() methods check capacity when booking or canceling. displayInfo() shows all vehicle data, including assigned stations and schedules.

- **ExpressBus Class:** ExpressBus inherits from Vehicle and adds an expressSpeed value (default 80 km/h). It overrides calculateTravelTime() to use the faster speed. Its displayInfo() method also shows the express speed.
- **Station Management:** Station represents a bus station with a name, location, and up to 10 schedules. addSchedule() will reject new schedules if the limit is reached. displayInfo() shows station details and numbers the schedules.
- **Passenger System:** Passenger stores a name, id, and a list of booked tickets as strings. bookTicket() calls vehicle->bookSeat() and adds the ticket if there is space. cancelTicket() removes a ticket and frees a seat if it exists. displayInfo() shows passenger details and booked tickets.

4. Test Results

4.1. Create object test

```
Created station Ben Thanh Station
Created station Thu Duc Station
Created vehicle for route Route 01
Created vehicle for route Express Route 02

Vehicle Route 01 assigned to station: Ben Thanh Station
Vehicle Express Route 02 assigned to station: Thu Duc Station
```

4.2. Assign vehicle to station and add schedule to vehicle

```
Vehicle Route 01 assigned to station: Ben Thanh Station
Vehicle Express Route 02 assigned to station: Thu Duc Station
Added schedule to vehicle Route 01: 08:00 - Route 01 departure
Added schedule to vehicle Route 01: 09:30 - Route 01 arrival
Added schedule to vehicle Express Route 02: 08:15 - Express Route 02 departure
```

4.3. Inheritance test

```
=== TRAVEL TIME COMPARISON ===
Distance: 100 km
Bus travel time: 2 hours
Express bus travel time: 1.66667 hours
```

This output confirms that ExpressBus correctly overrides the calculateTravelTime() method, using its higher speed (60 km/h) versus the base Vehicle speed (50 km/h).

4.4. Display vehicle info test

```
=== VEHICLE INFO ===
Route: Route 01
Capacity: 40 people
Current passengers: 0 people
Stations assigned: Ben Thanh Station
Vehicle schedules: 2
    08:00 - Route 01 departure
    09:30 - Route 01 arrival

=== EXPRESS BUS ===
Route: Express Route 02
Capacity: 50
Passengers: 0
Express speed: 60 km/h
Stations assigned: Thu Duc Station
Vehicle schedules: 1
    08:15 - Express Route 02 departure
```

4.5. Add schedule to station and display station info

```
Added schedule: 08:00 - Route 01 departure
Added schedule: 08:15 - Express Route 02 departure
Added schedule: 09:30 - Route 01 arrival

=== STATION INFO ===
Station name: Ben Thanh Station
Location: Ben Thanh
Number of schedules: 3/10
Schedules:
    1. 08:00 - Route 01 departure
    2. 08:15 - Express Route 02 departure
    3. 09:30 - Route 01 arrival

=== STATION INFO ===
Station name: Thu Duc Station
Location: Thu Duc
Number of schedules: 0/10
Schedules:
```

4.6. Booking system test

```
=== BOOKING SYSTEM ===
Vo Hoang Tung booked: Route 01 - 08:00
Nguyen Van A booked: Express Route 02 - 08:15
```

4.7. Capacity limit in vehicle test

```
==== CAPACITY LIMIT TEST ==  
Trying to book many tickets for bus (capacity 40)...  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A booked: Route 01 - 08:00  
Nguyen Van A failed to book: vehicle full!  
Bus is full after 40 tickets!
```

4.8. Cancel ticket test

```
=== CANCEL TICKET TEST ===
Nguyen Van A cancelled: Route 01 - 08:00
Vo Hoang Tung does not have ticket: Express Route 02 - 08:15
```

Passenger Vo Hoang Tung doesn't have ticket at Express Route 02 so system displays a message to inform that cancellation cannot be processed.

```
=== PASSENGER INFO ===
Name: Vo Hoang Tung
ID: P001
Number of tickets booked: 1
    Ticket 1: Route 01 - 08:00

=== PASSENGER INFO ===
Name: Nguyen Van A
ID: P002
Number of tickets booked: 39
    Ticket 1: Express Route 02 - 08:15
    Ticket 2: Route 01 - 08:00
    Ticket 3: Route 01 - 08:00
    Ticket 4: Route 01 - 08:00
    Ticket 5: Route 01 - 08:00
    Ticket 6: Route 01 - 08:00
    Ticket 7: Route 01 - 08:00
    Ticket 8: Route 01 - 08:00
    Ticket 9: Route 01 - 08:00
    Ticket 10: Route 01 - 08:00
    Ticket 11: Route 01 - 08:00
    Ticket 12: Route 01 - 08:00
```

4.9. Schedule limit test

```
=== SCHEDULE LIMIT TEST ===
Added schedule: 10:04 - Route 4 departure
Added schedule: 10:05 - Route 5 departure
Added schedule: 10:06 - Route 6 departure
Added schedule: 10:07 - Route 7 departure
Added schedule: 10:08 - Route 8 departure
Added schedule: 10:09 - Route 9 departure
Added schedule: 10:10 - Route 10 departure
Cannot add - already have 10 schedules!
Cannot add - already have 10 schedules!

=== STATION INFO ===
Station name: Ben Thanh Station
Location: Ben Thanh
Number of schedules: 10/10
Schedules:
    1. 08:00 - Route 01 departure
    2. 08:15 - Express Route 02 departure
    3. 09:30 - Route 01 arrival
    4. 10:04 - Route 4 departure
    5. 10:05 - Route 5 departure
    6. 10:06 - Route 6 departure
    7. 10:07 - Route 7 departure
    8. 10:08 - Route 8 departure
    9. 10:09 - Route 9 departure
    10. 10:10 - Route 10 departure
```

5. LLM Usage

I used Github Copilot to suggest attributes for each class in the system:

suggest attributes for each class

you are tasked with building a basic Public Transportation Station Management System for a city's bus and train network. The system should handle:

Stations: Each station has a name, location, type (e.g., bus or train), and a list of scheduled arrivals/departures. Stations can add or remove schedules.

Vehicles: Vehicles include route, capacity, and status (e.g., on-time, delayed). Vehicles can be assigned to stations.

Specialized Vehicles: The system distinguishes between regular buses and express buses. Express buses inherit from the general Vehicle class but include additional attributes like speed (higher for express) and fewer stops. They override methods, such as travel time calculation (e.g., express buses take 20% less time on the same route).

Passengers: Passengers have a name, ID, and booked tickets. They can book or cancel rides on vehicles.

Operations: The system should allow creating stations and vehicles, scheduling arrivals/departures, booking passengers, and displaying information about stations, vehicles, and passengers.

This scenario draws from real-world public transportation management, where inheritance can be used to extend base classes (e.g., Vehicle as base, ExpressBus as derived) to handle variations in vehicle types without redundant code.

Here are suggested attributes for each class in your Public Transportation Station Management System:

Station

- name (string)
- location (string)
- type (string: "bus" or "train")
- schedules (list of Schedule objects)

Vehicle

- route (string or Route object)
- capacity (int)
- status (string: "on-time", "delayed", etc.)
- assignedStation (Station object or station name/id)

ExpressBus (inherits from Vehicle)

- speed (float or int, higher than regular bus)
- stops (list of Station objects, fewer than regular bus)
- (inherits route, capacity, status, assignedStation from Vehicle)

Passenger

- name (string)
- id (string or int)
- bookedTickets (list of Schedule or Vehicle objects)

Schedule

- arrivalTime (datetime or string)
- departureTime (datetime or string)
- vehicle (Vehicle object)
- station (Station object)
- status (string: "scheduled", "completed", "cancelled")

These attributes will help you implement the required features and demonstrate object-oriented principles, including inheritance.

I also used ChatGPT to ask whether I should add a **Ticket** class to the system to make the booking system more intuitive.

vậy tôi có cần class ticket không ?

Rất hay! 🧐 Câu hỏi này rất đúng chỗ, vì với phiên bản đơn giản dành cho sinh viên mới học OOP, class **Ticket** có thể được bỏ đi — tùy vào bạn muốn quản lý thông tin vé chi tiết đến đâu.

Khi nào KHÔNG cần **Ticket**

Nếu hệ thống của bạn chỉ cần:

- Biết **Passenger** nào đang ngồi trên xe nào,
- Đếm số lượng hành khách,
- Và hiển thị danh sách vé đơn giản (chỉ cần string mô tả),

👉 Bạn có thể chỉ lưu `vector<string>` trong **Passenger** (chứa thông tin vé) mà không cần hẳn một class **Ticket**.

Điều này giúp code ngắn gọn, dễ hiểu hơn cho sinh viên mới học.

Khi nào NÊN giữ **Ticket**

Bạn cần class **Ticket** nếu muốn:

- Lưu thêm thông tin chi tiết cho từng vé (giá vé, giờ khởi hành, ghế ngồi, mã số vé).
- Quản lý nhiều loại phương tiện khác nhau (bus, tàu, máy bay) mà một vé cần liên kết tới phương tiện tương ứng.
- Xử lý logic nâng cao (đổi vé, hoàn tiền, in vé).