Public Transportation Station Management System

# Step 1: Object-Oriented Analysis (OOA) Model

The system models a simplified public transportation management system for bus and train stations. Below is the analysis following the OOA 4-step model:

1. Objects (Nouns): Station, Vehicle, ExpressBus, Passenger, Schedule

2. Attributes:

- Station: name, location, type, schedules (list of arrivals/departures)

- Vehicle: route, capacity, status (on-time or delayed)

- ExpressBus: route, capacity, status, speed (higher than normal buses)

- Passenger: name, ID, booked tickets

3. Methods:

- Station: addSchedule(), removeSchedule(), displaySchedules()

- Vehicle: assignToStation(), calculateTravelTime(), displayInfo()

- ExpressBus: calculateTravelTime() [overridden], displayInfo()

- Passenger: bookRide(), cancelRide(), displayInfo()

4. Inheritance: ExpressBus inherits from Vehicle and overrides calculateTravelTime().

# Step 2: Explanation of Class Design

The system was designed around four main classes: Vehicle, ExpressBus, Station, and Passenger. Vehicle serves as a base class with attributes such as route, capacity, and status. ExpressBus inherits from Vehicle, extending it with speed and overriding the calculateTravelTime() method to simulate reduced travel time (20% faster). Station manages schedules and associated vehicles, while Passenger models user interactions through booking and canceling rides. Inheritance ensures code reuse and simplifies extension to new vehicle types in the future.

# Step 3: Code Walkthrough

Key parts of the C++ implementation:

- Vehicle class: Defines route, capacity, and a virtual calculateTravelTime() method.

- ExpressBus class: Inherits from Vehicle and overrides calculateTravelTime() to make travel faster.

- Station class: Stores schedules, provides methods to add/remove schedules, and link vehicles.

- Passenger class: Allows booking and canceling tickets, prevents booking when full.

Encapsulation is applied through private attributes and public methods. Polymorphism is demonstrated by overriding calculateTravelTime() in ExpressBus.

Sample C++ snippet:

class Vehicle {  
protected:  
 std::string route;  
 int capacity;  
 bool status;   
public:  
 Vehicle(std::string r, int c) : route(r), capacity(c), status(true) {}  
 virtual double calculateTravelTime(double distance) {  
 return distance / 40.0; // assume avg speed = 40 km/h  
 }  
};  
  
class ExpressBus : public Vehicle {  
private:  
 double speed;  
public:  
 ExpressBus(std::string r, int c, double s) : Vehicle(r, c), speed(s) {}  
 double calculateTravelTime(double distance) override {  
 return (distance / speed) \* 0.8; // 20% faster  
 }  
};

# Step 4: Test Results

Sample test cases executed in the main function:

- Created a Station object ('Central Station')

- Created a Vehicle and an ExpressBus, assigned them to the station

- Booked multiple passengers until the capacity was full, ensuring overbooking was prevented

- Calculated travel times for Vehicle and ExpressBus to demonstrate polymorphism

Example Console Output:

Station: Central Station (Bus)  
Schedules:  
 - Vehicle Route A, Capacity: 2, Status: On-Time  
Passenger John booked successfully on Route A.  
Passenger Alice booked successfully on Route A.  
Passenger Bob could not book: Vehicle full.  
Travel time by regular bus: 2.5 hours  
Travel time by express bus: 2.0 hours

# Step 5: Use of LLM AI Model

I used ChatGPT to assist in brainstorming the OOA steps and refining the inheritance design. For example, I asked: 'Suggest inheritance hierarchies for vehicles in a transportation system.' I also used the LLM to clarify testing approaches. However, the final implementation, code, and document were written independently.