Public Transportation System: Design and Analysis Report

Tran Le Dung (MSSV: 24110084)

September 14, 2025

1 Object-Oriented Analysis (OOA)

The Object-Oriented Analysis (OOA) for the Public Transportation System (PTS) focuses on modeling a system to manage vehicles, stations, routes, passengers, and schedules. The key entities identified are:

- **Vehicle**: Represents transportation modes (e.g., buses, metro, trains, bikes) with attributes like route, capacity, booked seats, and ticket price. Behaviors include calculating travel time, displaying information, and booking tickets.
- **Station**: Represents physical stops with a name and location.
- **Route**: Defines a path between a start and end station.
- **Passenger**: Represents individuals booking tickets, identified by their name.
- **Schedule**: Links vehicles to stations and times for operational planning.

The relationships are:

- Vehicles operate on routes and have schedules at stations.
- Passengers book tickets for specific vehicles.
- Routes connect stations, forming the network for vehicle operations.

The system supports functionalities like adding vehicles, managing stations, booking tickets, and generating revenue reports, ensuring scalability and maintainability.

2 Class Design and Inheritance

The class design leverages object-oriented principles, particularly inheritance, to model different vehicle types while maintaining shared functionality. The hierarchy is structured as follows:

- **Vehicle (Base Class)**: An abstract base class encapsulating common attributes (`route`, `capacity`, `bookedSeats`, `ticketPrice`) and methods (`calculateTravelTime`, `displayInfo`, `displayTickets`, `bookTicket`). The `calculateTravelTime` and `displayInfo` methods are virtual to allow customization in derived classes.
- **Derived Classes (ExpressBus, Metro, Train, Bike)**: Each inherits from `Vehicle` to represent specific transportation modes. `ExpressBus` overrides `calculateTravelTime`

to account for variable speed, while all derived classes override displayInfo to provide type-specific output formatting.

Inheritance is used to:

- Promote code reuse by sharing common functionality (e.g., ticket booking logic) in the Vehicle class.
- Enable polymorphism, allowing different vehicle types to be managed uniformly through base class pointers in the vehicles vector.
- Facilitate extensibility, as new vehicle types can be added by creating new derived classes without modifying existing code.

The Station, Route, Passenger, and Schedule classes are standalone, focusing on their specific roles without inheritance, as they do not share a hierarchical relationship.

3 Code Walkthrough

The C++ code implements the PTS with a modular structure. Key components include:

- **Vehicle Class Hierarchy**: The Vehicle base class defines shared attributes and virtual methods. For example, bookTicket checks capacity before incrementing bookedSeats. Derived classes like ExpressBus customize calculateTravelTime using a speed attribute.

```cpp
class ExpressBus : public Vehicle {
    double speed;
public:
    ExpressBus(string r, int c, double s, double price)
        : Vehicle(r, c, price), speed(s) {}
    double calculateTravelTime(double distance) override {
        return distance / speed;
    }
};
```

- **Global Data**: Vectors store pointers to Vehicle objects and instances of Station, Route, Passenger, and Schedule, enabling dynamic management of system entities.
- **Menu System**: The main function provides a menu-driven interface with sub-menus (vehicleMenu, passengerMenu, etc.) for user interaction. Each menu handles specific operations like adding vehicles or booking tickets.

```cpp
void vehicleMenu() {
    int choice;
    do {
        system("cls");
        cout << "\n===== Vehicle Menu =====\n";
        cout << "1. Add Vehicle\n";
        // ... menu options ...
        cin >> choice;
        // Handle choices
    } while (choice != 0);
}
```

- **Revenue Report**: The `revenueReport` function aggregates ticket sales across all vehicles, demonstrating the system's financial tracking capability.

4 Test Results

The system was tested with the initial data from `initData`, which populates vehicles, stations, routes, and schedules. Sample outputs include:

- **Vehicle List Output**:

  ```
  === Vehicle List ===
  [ExpressBus] Route: RouteA | Speed: 60 | Capacity: 2 | Booked: 0 | Ti
  [Metro] Route: RouteB | Capacity: 100 | Booked: 0 | Ticket: $1.5
  [Train] Route: RouteC | Capacity: 200 | Booked: 0 | Ticket: $10
  [Bike] Route: RouteD | Capacity: 1 | Booked: 0 | Ticket: $0.5
  ```

  This demonstrates the polymorphic `displayInfo` method, showing type-specific details for each vehicle.

- **Ticket Booking Output**:

  ```
  Enter passenger name: Alice
  Choose vehicle index:
  0. [ExpressBus] Route: RouteA | Speed: 60 | Capacity: 2 | Booked: 0 |
  1. [Metro] Route: RouteB | Capacity: 100 | Booked: 0 | Ticket: $1.5
  2. [Train] Route: RouteC | Capacity: 200 | Booked: 0 | Ticket: $10
  3. [Bike] Route: RouteD | Capacity: 1 | Booked: 0 | Ticket: $0.5
  0
   Ticket booked successfully for route RouteA
  ```

  This shows successful ticket booking, updating `bookedSeats` for the selected vehicle.

- **Revenue Report Output**:

  ```
  === Revenue Report ===
  [Tickets] Route: RouteA | Sold: 1 | Remaining: 1 | Capacity: 2 | Reve
  [Tickets] Route: RouteB | Sold: 0 | Remaining: 100 | Capacity: 100 |
  [Tickets] Route: RouteC | Sold: 0 | Remaining: 200 | Capacity: 200 |
  [Tickets] Route: RouteD | Sold: 0 | Remaining: 1 | Capacity: 1 | Reve
  Total Revenue: $5
  ```

  This confirms the system accurately tracks ticket sales and calculates total revenue.

These outputs validate the system's functionality in managing vehicles, booking tickets, and reporting revenue.

5 LLM Usage

I used an LLM (Grok, created by xAI) to brainstorm the inheritance hierarchy for the vehicle classes. The prompt was: *"Suggest inheritance hierarchies for vehicles in a public transportation system, including potential attributes and methods."* The LLM suggested a base `Vehicle` class with derived classes like `Bus`, `Train`, and `Bike`, along with attributes like capacity and ticket price. I customized this by adding ExpressBus and Metro, incorporating a speed attribute for ExpressBus, and defining specific methods like calculateTravelTime. The LLM also provided ideas for menu-driven interfaces, which inspired the modular menu system. The actual C++ code was written independently to meet the project's requirements.

A LLM Prompt and Response

Prompt: Suggest inheritance hierarchies for vehicles in a public transportation system, including potential attributes and methods.

Response: A suitable hierarchy could include a base Vehicle class with attributes like route, capacity, and ticketPrice, and methods like bookTicket and displayInfo. Derived classes such as Bus, Train, and Bike could inherit from Vehicle, with specific attributes (e.g., speed for Bus) and overridden methods for customized behavior. A menu-driven system could manage user interactions for adding vehicles and booking tickets.