Step 1: Object-Oriented Analysis (OOA)

1. Identify Objects (Nouns):

- **Station**
- **Schedule**
- **Vehicle**
- **ExpressBus (specialized vehicle)**
- **Passenger**
- **Ticket**

2. Identify Attributes:

- **Station: name, location, type, schedules (list of Schedule).**
- **Schedule: vehicle name, start time, end time.**
- **Vehicle: route, capacity, status.**
- **ExpressBus: (inherits Vehicle) + speed.**
- **Passenger: name, id, tickets (list of Ticket).**
- **Ticket: passenger name, route, vehicle type, start time, end time.**

3. Identify Methods (Verbs):

- **Station: addSchedule(), removeSchedule(), displaySchedule().**
- **Schedule: display().**
- **Vehicle: assignToStation(), reduceCapacity(), increaseCapacity(), calculateTravelTime(), displayInfo().**
- **ExpressBus: override calculateTravelTime(), displayInfo2().**
- **Passenger: bookTicket(), cancelTicket(), displaypa(), displaytk().**
- **Ticket: displayticket().**

4. Inheritance:

- **Vehicle → Base class.**
- **ExpressBus → Derived class (adds speed, overrides travel time calculation).**

Step 2: Class Design

Main classes from code:

class Schedule { ... };

class Station { ... };

class Vehicle { ... };

class Expressbus : public Vehicle { ... };

class Ticket { ... };

class Passenger { ... };

Inheritance:

- **Expressbus inherits from Vehicle.**
- **Other classes (Station, Passenger, Ticket, Schedule) are independent and interact through composition (contain objects).**

Step 3: Code Walkthrough

- **Schedule: Represents a timetable entry for a vehicle with start and end times.**
- **Station: Contains multiple schedules. Limited to 10 schedules max. Can display all schedules.**
- **Vehicle: Represents buses or trains with attributes route, capacity, and status. Has a base travel time calculation (distance ÷ 50).**
- **Expressbus: Inherits from Vehicle. Has higher speed and overrides travel time calculation (distance ÷ speed).**
- **Ticket: Stores passenger booking info (who, which route, vehicle, start/end time).**
- **Passenger: Can book tickets if vehicle has capacity. Cancels tickets to free capacity. Stores tickets in a vector.**

Main Function Flow:

1. **Create a station (Central Station) with schedules for Bus-01 and Bus-02.**
2. **Create vehicles: one normal bus (capacity 2) and one express bus (capacity 1, speed 80).**

3. **Display station schedule and vehicle info.**
4. **Calculate travel times with different speeds.**
5. **Passengers (Alice, Bob, Charlie) try booking tickets. Booking fails if capacity is full.**
6. **Cancel a ticket (Alice’s express bus ticket).**
7. **Display passenger information and their tickets.**

Step 4: Test Results

Program Output (based on provided code):

```
=== Station Schedule ===
Station: Central Station
Location: Downtown
Type: Bus
1. Vehicle: Bus-01
Start time: 08:00
End time: 10:00
2. Vehicle: Bus-02
Start time: 10:30
End time: 12:00

=== Vehicle Info ===
-------Vehicle infomation-------
Route: Route A
Capacity: 2
Status: Available
-------Vehicle infomation-------
Route: Route B
```

```
Capacity: 1
Status: Available
Speed: 80

=== Travel Time Test ===
Travel Time: 2 h
Travel Time: 1.25 h

=== Booking Ticket ===
Booked successfully
Booked successfully
Vehicle is full for passenger
Booked successfully
Vehicle is full for passenger

=== Cancel Ticket ===
Ticket cancel successfully

=== Passenger Info ===
Passenger Information
Name: Alice
Id: P001
Passenger Information
Name: Bob
Id: P002
Passenger Information
Name: Charlie
```

```
Id: P003
-----Ticket-----
Name passenger: Alice
Route: Route A
Vehicle type: Bus-01
Start time:  08:00
End time: 10:00
-----Ticket-----
Name passenger: Alice
Route: Route B
Vehicle type: Bus-02
Start time:  10:30
End time: 12:00
-----Ticket-----
Name passenger: Bob
Route: Route A
Vehicle type: Bus-01
Start time:  08:00
End time: 10:00
```

Step 5: Use of LLM AI Model

I used ChatGPT to:

- **Suggest the OOA (objects, attributes, methods).**
- **Explain bugs in the cancelTicket() function.**
- **Propose test cases to check all scenarios.**