|  |  |  |
| --- | --- | --- |
| **class** | **Responsibility** | **Collaborators** |
| Company | Manage fleet operations and coordinate between passengers and vehicles | Vehicle  Passenger source |
| Vehicle | Transport passengers from pickup to destination locations | Driver  company |
| passenger | Represent a transportation request with pick up/ destination locations | Passenger source  vehicle |
| Passenger source | Generate transportation requests from various sources | Company  passenger |
| driver | Operate vehicle and communicate status update | Vehicle  company |

Question 1 part a

b) package com.mycompany.one.java;

/\*\*

\*

\* @author HP

\*/

import java.util.Random;

public class ONEJava {

public class PassengerSource {

private Company company;

private Random random;

public PassengerSource(Company company) {

this.company = company;

this.random = new Random();

}

public boolean requestPickup() {

// Generate random pickup location (0-100 range)

Location pickup = new Location(

random.nextInt(101),

random.nextInt(101)

);

// Generate random destination location (0-100 range)

Location destination = new Location(

random.nextInt(101),

random.nextInt(101)

);

// Create new passenger

Passenger passenger = new Passenger(pickup, destination);

// Request company to schedule vehicle

return company.schedulePickup(passenger);

}

}

// Supporting classes (would be in separate files)

class Location {

private int x;

private int y;

public Location(int x, int y) {

this.x = x;

this.y = y;

}

// Getters and setters would be here

}

class Passenger {

private Location pickup;

private Location destination;

public Passenger(Location pickup, Location destination) {

this.pickup = pickup;

this.destination = destination;

}

// Getters and setters would be here

}

class Company {

public boolean schedulePickup(Passenger passenger) {

// Implementation would check for available vehicles

// Return true if scheduled, false if no vehicles available

return false; // Placeholder

}

}

public static void main(String[] args) {

System.out.println("Hello World!");

}

}

**C) JUnit Tests for the company class**

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.\*;

public class CompanyTest {

@Test

public void testSchedulePickupWithAvailableVehicle() {

Company company = new Company();

// Add a vehicle to make it available (implementation specific)

// company.addVehicle(new Vehicle());

Passenger passenger = new Passenger(

new Location(10, 20),

new Location(30, 40)

);

boolean result = company.schedulePickup(passenger);

assertTrue(result, "Should return true when vehicle is available");

}

@Test

public void testSchedulePickupWithNoAvailableVehicle() {

Company company = new Company();

// Don't add any vehicles

Passenger passenger = new Passenger(

new Location(5, 15),

new Location(25, 35)

);

boolean result = company.schedulePickup(passenger);

assertFalse(result, "Should return false when no vehicles available");

}

}

d) **How encapsulation is applied:**

1. The internal state (x and y coordinates) is marked as private, hiding implementation details
2. Access to the fields is controlled through public getter and setter methods
3. The setter methods include validation to ensure coordinates stay within the 0-100 range
4. This protects the object's integrity by preventing invalid states
5. The class maintains control over how its data is accessed and modified

This encapsulation allows the Location class to:

* Change its internal representation without affecting other code
* Enforce business rules (like coordinate ranges)
* Maintain consistency of its data
* Hide implementation details from other classes