**Carpooling System - Detailed Document**

**1. Introduction**

A carpooling system helps people share rides by connecting drivers with passengers going in the same direction. This makes commuting cheaper and more eco-friendly.

**2. System Overview**

The system has different parts, including user sign-up, ride management, booking, payments, and tracking. It follows Object-Oriented Programming (OOP) concepts using Java.

**3. System Components**

**3.1 User (Base Class) [Unit: Classes and Objects]**

**Details:**

* **User ID**: A unique number assigned to each user.
* **Name**: Full name of the user.
* **Phone Number**: Contact number.
* **Email**: User’s email address.
* **Location**: User’s current location.

**Actions:**

* **Sign up**: Users create an account.
* **Update Profile**: Users can change their details.

**3.2 Driver (Extends User) [Unit: Inheritance]**

**Extra Details:**

* **Vehicle Info**: Car model and number plate.
* **License Number**: Driver’s legal driving ID.
* **Availability**: Whether the driver is available for rides.
* **Rating**: Score given by passengers.

**Extra Actions:**

* **Offer a Ride**: Allows the driver to post a ride.
* **Change Availability**: Updates ride status.

**3.3 Passenger (Extends User) [Unit: Inheritance]**

**Extra Details:**

* **Preferred Route**: Usual travel path.
* **Payment Method**: Chosen way to pay for rides.

**Extra Actions:**

* **Search for Rides**: Finds available rides.
* **Cancel Ride**: Cancels a booked ride.
* **Make Payment**: Pays the driver.

**3.4 Ride [Unit: Classes and Objects, Encapsulation]**

**Details:**

* **Ride ID**: Unique ride number.
* **Driver**: Assigned driver.
* **Passengers**: People who booked the ride.
* **Route**: Path of the ride.
* **Status**: Whether the ride is Scheduled, Ongoing, or Completed.
* **Fare**: Cost of the ride.

**Actions:**

* **Add Passenger**: Adds a passenger to the ride.
* **Remove Passenger**: Removes a passenger.
* **Calculate Fare**: Computes the ride cost.
* **Update Status**: Changes ride status.

**4. How the System Works**

**4.1 User Registration [Unit: Object and Class Implementation]**

1. User enters name, phone, email, and location.
2. System generates a unique user ID.
3. User is added to the database.

**4.2 Offering a Ride (Driver) [Unit: Inheritance, Polymorphism]**

1. Driver enters ride details (start, destination, available seats, fare).
2. System generates a unique ride ID.
3. Ride is listed as available.
4. Driver’s availability is updated.

**4.3 Finding a Ride (Passenger) [Unit: Collections, Data Structures]**

1. Passenger enters start and destination.
2. System finds matching rides.
3. Matching rides are shown to the passenger.

**4.4 Booking a Ride [Unit: Exception Handling, Interfaces]**

1. Passenger selects a ride.
2. System checks seat availability.
3. If available:
   * Passenger is added to the ride.
   * Seat count is updated.
   * Booking is confirmed.
4. If full, the passenger is notified.

**4.5 Tracking Ride Status [Unit: File Handling, Multithreading]**

1. Ride status updates automatically (Scheduled → Ongoing → Completed).
2. System tracks the driver’s location.

**4.6 Payment & Rating [Unit: Interfaces, Polymorphism]**

1. Passenger pays using cash, UPI, or wallet.
2. After the ride, passengers can rate the driver.

**5. OOP Concepts Used [Unit: OOP Principles]**

* **Encapsulation**: Protects user data.
* **Inheritance**: Passenger and Driver share common user features.
* **Polymorphism**: Different payment methods handled through interfaces.
* **Abstraction**: Hides complex details from users.

**6. Data Storage [Unit: Collections]**

The system uses **ArrayList** and **HashMap** to store users and rides. Future updates may include an **SQL database**.

**7. Future Improvements**

* **AI-based ride suggestions** based on past trips.
* **Automatic fare calculation** using GPS distance.
* **In-app messaging** between passengers and drivers.