# Ride-Sharing Application Design Problem

Zadid Bin Azad

September 13, 2024

*Design Patterns Lab-01*

## Scenario: Ride-Sharing Application

You are tasked with designing a ride-sharing platform where riders can book trips, drivers can accept requests, and payments are processed. The system should handle different types of rides (e.g., Carpooling, Luxury, Bike Rides), support various payment methods, and allow ratings for drivers and riders. There will be a robust notification system to keep both riders and drivers informed about the status of trips.

## Requirements

1. **Booking a Ride**:
    - A rider can request a ride by specifying the pickup and drop-off locations.
    - The system finds available drivers based on proximity and the type of ride requested.
    - The system calculates the estimated fare based on the distance, time of day, and type of ride.
    - Once a ride is confirmed, the rider and driver receive notifications.
2. **Types of Rides**:
    - **Carpool**: Riders share the vehicle with others, and the fare is split.
    - **Luxury Ride**: High-end vehicles at a premium price.
    - **Bike Ride**: Single-rider bike service for quick and affordable travel.
3. **Payments**:
    - The platform supports multiple payment methods (e.g., CreditCard, PayPal, Digital Wallet).
    - Riders can save preferred payment methods and choose one during booking.
    - The fare should be automatically deducted after the trip is completed.
4. **Driver and Rider Rating**:
    - After a trip, both the rider and the driver can rate each other and leave feedback.
    - Ratings should impact the availability of drivers for future rides.
5. **Notifications**:
    - Notifications must be sent for events like ride request confirmation, driver arrival, trip start, and completion.

- Notifications can be sent via SMS, Email, or In-App messaging, based on the user's preference.

6. **Admin Panel**:
   - An admin should have the ability to manage drivers and riders, view trip history, and handle disputes.

## Entities

The key entities involved in the system and their interactions are as follows:

- **Rider**:
  - **Attributes**: `id`, `name`, `location`, `rating`, `preferredPaymentMethod`, etc.
  - **Behaviors**: `requestRide()`, `rateDriver()`, `makePayment()`.
  - **Relations**: Can book a `Trip`, interacts with `PaymentMethod` and `Driver`.
- **Driver**:
  - **Attributes**: `id`, `name`, `vehicleType`, `location`, `rating`, `availability`.
  - **Behaviors**: `acceptRide()`, `rateRider()`, `updateLocation()`, `startTrip()`.
  - **Relations**: Accepts a `Trip` and interacts with the `Rider`.
- **Trip**:
  - **Attributes**: `id`, `pickupLocation`, `dropOffLocation`, `rideType`, `status`, `fare`, `distance`.
  - **Behaviors**: `calculateFare()`, `assignDriver()`, `completeTrip()`.
  - **Relations**: Linked to both `Rider` and `Driver`, and utilizes the `NotificationService`.
- **RideType**:
  - Each ride type may have its own rules for fare calculation, capacity, and matching drivers.
- **PaymentMethod**:
  - **Behaviors**: `processPayment()`.
  - Each payment method has its own implementation of how the payment is processed.
- **NotificationService**:
  - **Behaviors**: `sendNotification()`.
  - Allows sending different types of notifications for ride events (start, complete, etc.).
- **Admin**:
  - **Attributes**: `id`, `name`, `role`.
  - **Behaviors**: `manageDriver()`, `manageRider()`, `viewTripHistory()`, `handleDispute()`.
  - Interacts with `Trip` and `User` data.

## Interactions and Complexity

1. **Requesting and Completing a Ride**:
    - The `Rider` interacts with the system by calling `requestRide()`. The system matches the rider with a nearby `Driver` based on location and availability.
    - The ride's type (`RideType` like `Carpool`, `LuxuryRide`) is used to filter drivers and calculate the fare.
    - Once a driver accepts the trip (`acceptRide()`), the trip proceeds. Notifications are sent at each stage (e.g., `driverAssigned()`, `tripStarted()`, `tripCompleted()`).
    - Upon trip completion, the `Rider` pays via the selected `PaymentMethod` interface.
    - Both the `Rider` and `Driver` provide ratings that will affect their future availability.
2. **Handling Payments**:
    - Payment processing is abstracted using the `PaymentMethod` interface. When the trip is completed, the `Trip` object calls `processPayment()` on the selected payment method.
    - Using dependency injection, the system allows for new payment methods to be added easily without changing the `Trip` class.
3. **Rating System**:
    - After each trip, the `Rider` and `Driver` rate each other. The system stores this feedback and adjusts the ratings, impacting future ride requests and acceptances.
    - The `Driver` and `Rider` classes have methods `rateDriver()` and `rateRider()` which store the ratings.
4. **Admin Operations**:
    - Admins interact with the system through the admin panel. They have the ability to deactivate or manage drivers, view trip histories, and handle disputes between riders and drivers.
    - They also have access to a dashboard where they can view metrics on rides, payments, and user ratings.
5. **Notification System**:
    - The `NotificationService` interface allows for different notification types (SMS, Email, In-App) to be sent to both riders and drivers.
    - Each event (ride request, driver arrival, trip start, trip complete) triggers a notification, and users can specify their preferred notification method.