

Ride Sharing System - OOP Assignment 02

Total Marks: 60

Problem Statement

Design and implement a Ride Sharing System (like Uber or Pathao). The system must allow riders and drivers to be registered, create rides, calculate fares using different pricing strategies, process payments, and notify users about ride status updates. Implement using OOP principles and design patterns.

System Requirements

1. User Management

- Support two user types: Riders and Drivers
- Common properties: ID, Name, Phone
- Rider additional: Wallet balance
- Driver additional: Vehicle, Availability status
- Both user types should display their information and identify their role

2. Vehicle Management

- Support three vehicle types: Bike (\$2), CNG (\$3), Car (\$5)
- Use Factory Pattern to create vehicles

3. Driver Management

- Maintain central registry using Singleton Pattern
- Support: Register drivers, view all drivers, filter by vehicle type

4. Ride Management & Pricing

- Create rides with rider, driver, and distance
- Strategy Pattern for pricing:
 - Standard: \$0.50/km
 - Rush Hour: \$1.00/km
 - Midnight: \$0.75/km
- Ride properties: ID, Rider, Driver, Distance, Status
- Status flow: Requested → Accepted → In Progress → Completed

5. Payment Processing

- Support bKash and Credit Card
- Use Adapter Pattern for bKash integration
- Common IPaymentProcessor interface

6. Notification System

- Use Observer Pattern for notifications
- Notify riders (SMS) and drivers (App) on status changes

Design Patterns Implementation

1. Factory Pattern (10 marks)

Create VehicleFactory to instantiate different vehicle types.

Requirements:

- IVehicle interface with GetVehicleType() and GetBaseFare() methods
- Bike, CNG, Car classes implementing IVehicle
- VehicleFactory with CreateVehicle(string type) method

2. Inheritance & Polymorphism (10 marks)

Build user hierarchy with abstract base class.

Requirements:

- Abstract User class with: Id, Name, Phone, abstract DisplayInfo(), abstract GetRole()
- Rider class: adds WalletBalance
- Driver class: adds Vehicle, IsAvailable

3. Singleton Pattern (8 marks)

Implement RideManager ensuring a single instance.

Requirements:

- Private constructor
- Private static instance
- Public static GetInstance() method
- Methods: RegisterDriver(), GetAllDrivers(), GetAvailableDrivers(string type)

4. Strategy Pattern (8 marks)

Dynamic pricing based on conditions.

Requirements:

- IPricingStrategy interface with CalculateFare(distance, baseFare) and GetStrategyName()
- StandardPricing, RushHourPricing, MidnightPricing classes
- Ride class with SetPricingStrategy() and CalculateFare() methods

5. Adapter Pattern (8 marks)

Integrate external bKash payment gateway.

Requirements:

- IPaymentProcessor interface with Pay(paymentInfo, amount) and GetPaymentMethod()
- Use provided BkashPaymentGateway class (see reference code below)
- Implement: BkashPaymentAdapter and CreditCardProcessor

6. Observer Pattern (6 marks)

Notify users on ride status changes.

Requirements:

- IRideObserver interface with Update(rideId, status)
- RiderNotifier and DriverNotifier classes
- Ride class: maintain observers list, AddObserver(), SetStatus() that notifies all observers

Project Structure

Organize code into separate files and namespaces:

```
RideSharing/  
├── Program.cs  
├── Users/ (User.cs, Rider.cs, Driver.cs)  
├── Vehicles/ (IVehicle.cs, Bike.cs, CNG.cs, Car.cs,  
VehicleFactory.cs)  
├── Management/ (RideManager.cs)  
├── Rides/ (Ride.cs)  
├── Pricing/ (IPricingStrategy.cs, StandardPricing.cs,  
RushHourPricing.cs, MidnightPricing.cs)  
├── Payments/ (IPaymentProcessor.cs, BkashPaymentGateway.cs,  
BkashPaymentAdapter.cs, CreditCardProcessor.cs)  
└── Observers/ (IRideObserver.cs, RiderNotifier.cs,  
DriverNotifier.cs)
```

Example namespace usage:

```
namespace RideSharing.Users{    public abstract class User { /* ... */ }}
```

Console Application

Create a menu-driven console application demonstrating all features:

- Register drivers and riders
- Create rides with different vehicle types
- Switch pricing strategies and calculate fares
- Process payments (bKash & Credit Card)
- Update ride status and display notifications

Evaluation Criteria

Criteria	Marks
Factory Pattern	10
Inheritance & Polymorphism	10
Singleton Pattern	8
Strategy Pattern	8
Adapter Pattern	8
Observer Pattern	6
Code Quality & Organization	5
README.md with: - How to run the application - Design patterns used and their locations	5
Total	60

Reference Code

BkashPaymentGateway (Provided - You cannot modify this code 🔒):

```
public class BkashPaymentGateway{
    public string SendMoney(string phoneNumber, double amount
    {
        Console.WriteLine($"bKash: Sending ${amount} to {phoneNumber}");
        return "TXN" + new Random().Next(1000, 9999);
    }

    public bool CheckStatus(string transactionId)
    {
        Console.WriteLine($"bKash: {transactionId} successful");
        return true;
    }
}
```

Good luck! 🚀