OOP-Based Vehicle Rent System - Design Format

1. Abstract Class / Interface - Vehicle (Abstraction)

```
abstract class Vehicle {
    private string vehicleNumber;
    private string fuelType;     // Petrol, Diesel, CNG
    private string vehicleType;  // Normal, Sport

    public string GetVehicleNumber() {
        return vehicleNumber;
    }

    public void SetVehicleNumber(string value) {
        vehicleNumber = value;
    }

    public string GetFuelType() {
        return fuelType;
    }

    public void SetFuelType(string value) {
        fuelType = value;
    }

    public string GetVehicleType() {
        return vehicleType;
```

```
    }

    public void SetVehicleType(string value) {
        vehicleType = value;
    }

    public abstract double CalculateRent(int hours);
}
```

2. Inheritance - TwoWheeler and FourWheeler Classes

```
class TwoWheeler : Vehicle {
    public override double CalculateRent(int hours) {
        double rate = 0;
        if (GetFuelType() == "Petrol" && GetVehicleType() == "Normal")
            rate = 50;
        else if (GetFuelType() == "Petrol" && GetVehicleType() == "Sport")
            rate = 100;
        return rate * hours;
    }
}

class FourWheeler : Vehicle {
    public override double CalculateRent(int hours) {
        double rate = 0;
```

```
        if (GetFuelType() == "Diesel" && GetVehicleType() == "Normal")
            rate = 120;
        else if (GetFuelType() == "CNG" && GetVehicleType() == "SUV")
            rate = 150;
        return rate * hours;
    }
}
```

3. Encapsulation - Data Hiding and Access Control

Encapsulation is implemented in Vehicle class:

- Fields are private: vehicleNumber, fuelType, vehicleType
- Public getters and setters control the access

Example:

```
private string vehicleNumber;
public string GetVehicleNumber() { return vehicleNumber; }
public void SetVehicleNumber(string value) { vehicleNumber = value; }
```

4. Polymorphism - Runtime Method Overriding

- Vehicle class has an abstract method: CalculateRent()
- TwoWheeler and FourWheeler provide different implementations of CalculateRent()

Polymorphic usage:

```
List<Vehicle> vehicles = new List<Vehicle>();
```

```
vehicles.Add(new TwoWheeler(...));

vehicles.Add(new FourWheeler(...));


foreach (Vehicle v in vehicles) {

    double rent = v.CalculateRent(4);

    Console.WriteLine("Rent: " + rent);

}
```

5. RentManager - Booking & Management Class

```
class RentManager {

    private List<Vehicle> availableVehicles = new List<Vehicle>();


    public void AddVehicle(Vehicle v) {

        availableVehicles.Add(v);

    }


    public double RentVehicle(string vehicleNumber, int hours) {

        foreach (Vehicle v in availableVehicles) {

            if (v.GetVehicleNumber() == vehicleNumber) {

                return v.CalculateRent(hours);

            }

        }

        return 0;

    }
```

```
}
```

6. Main Program - Example Usage

```
class Program {
    static void Main(string[] args) {
        RentManager manager = new RentManager();

        Vehicle v1 = new TwoWheeler();
        v1.SetVehicleNumber("TW123");
        v1.SetFuelType("Petrol");
        v1.SetVehicleType("Normal");

        Vehicle v2 = new FourWheeler();
        v2.SetVehicleNumber("FW456");
        v2.SetFuelType("Diesel");
        v2.SetVehicleType("Normal");

        manager.AddVehicle(v1);
        manager.AddVehicle(v2);

        double rent = manager.RentVehicle("TW123", 4);
        Console.WriteLine("Total Rent: " + rent);
    }
}
```

OOP-Based Vehicle Rent System - Design Format

7. Summary - OOP Concepts Used

1. Abstraction - Vehicle class is abstract with common structure.

2. Inheritance - TwoWheeler and FourWheeler inherit from Vehicle.

3. Encapsulation - Private fields with getters/setters in Vehicle.

4. Polymorphism - Method overriding in CalculateRent().

5. Composition - RentManager contains list of Vehicles.

6. Real-life Mapping - Classes and methods mimic real rental system.