**Vehicle Class**

The Vehicle class is the base class for all vehicle types. It contains common attributes like vehicleId, model, baseRentalRate, and isAvailable. It also defines the abstract methods calculateRentalCost() and isAvailableForRental(), which are implemented by subclasses.

* **Fields**: vehicleId, model, baseRentalRate, and isAvailable
* **Methods**:
  + calculateRentalCost(int days): Abstract method to calculate the rental cost.
  + isAvailableForRental(): Abstract method to check if the vehicle is available for rental.

**Car, Motorcycle, and Truck Classes**

These classes inherit from Vehicle and implement the abstract methods. They also introduce unique properties specific to each vehicle type.

* **Car**: Includes a field hasGPS to represent whether the car includes GPS. The rental cost is adjusted based on whether GPS is included.
* **Motorcycle**: Includes a field isHelmetIncluded to indicate if the rental includes a helmet.
* **Truck**: Includes a field loadCapacity to represent the truck's load capacity, affecting rental pricing.

**Rentable Interface**

The Rentable interface defines methods rent(Customer customer, int days) and returnVehicle() that must be implemented by any class that can be rented. This interface allows polymorphism, as vehicles of different types (cars, motorcycles, trucks) can be treated in the same way when renting or returning.

**Customer Class**

Represents customers who rent vehicles. The class keeps track of the rental history (rentalHistory) and allows the addition of new rental transactions. The addRentalTransaction() method stores a RentalTransaction for each vehicle rental.

**RentalAgency Class**

Manages the fleet of vehicles. The RentalAgency class provides functionality to add vehicles to the fleet, check for available vehicles, and process rentals. It simulates the role of a rental agency in the real world.

**RentalTransaction Class**

Represents a rental transaction between a customer and a vehicle. The class stores the vehicle rented, the rental days, and the total cost of the rental.

**4. Future Enhancements and Extensions**

* **Loyalty Program**: Implement a loyalty program for customers to offer discounts based on rental history.
* **Custom Exceptions**: Introduce exceptions like VehicleNotAvailableException or InvalidRentalPeriodException to handle specific rental errors.
* **Rating System**: Add a rating system for customers and vehicles to enable feedback and improve services.

**Conclusion**

This **Vehicle Rental Management System** applies key OOP principles to provide a modular, extensible, and maintainable solution for managing vehicle rentals

**Vehicle Rental Management System - Project Documentation**

**Overview**

The **Vehicle Rental Management System** is designed to manage vehicle rentals, supporting different vehicle types such as cars, motorcycles, and trucks. The system handles rental calculations, availability checks, and customer management, demonstrating core Object-Oriented Programming (OOP) principles: **Encapsulation**, **Inheritance**, **Polymorphism**, **Abstraction**, and **Composition**.

**Key Components**

1. **Vehicle (Abstract Class)**: The base class for all vehicle types. It defines common fields (vehicleId, model, baseRentalRate, isAvailable) and abstract methods (calculateRentalCost(), isAvailableForRental()) that must be implemented by subclasses.
2. **Car, Motorcycle, Truck (Concrete Classes)**: These subclasses inherit from Vehicle and implement specific behaviors for each vehicle type, including how rental costs are calculated and how availability is checked.
3. **Rentable (Interface)**: The Rentable interface defines the contract for renting vehicles, with methods like rent() and returnVehicle(). Each vehicle type implements these methods to handle rentals.
4. **Customer**: Represents customers who rent vehicles, maintaining their rental history and the vehicles they currently have rented.
5. **RentalAgency**: Manages the fleet of vehicles and handles rental transactions. It can check availability and process rentals.
6. **RentalTransaction**: Records details about each rental, including the vehicle rented, rental days, and total cost.

**Core OOP Principles Applied**

* **Encapsulation**: Critical fields in classes like Vehicle are private, and controlled access is provided through getter and setter methods. For instance, the baseRentalRate field has a setter that ensures no negative rental rates can be set.
* **Inheritance**: The Car, Motorcycle, and Truck classes inherit from the Vehicle class, sharing common functionality (like rental cost calculation) while allowing specific behavior for each vehicle type.
* **Polymorphism**: The Rentable interface allows different vehicle types to be treated in a unified way, such as using the rent() and returnVehicle() methods polymorphically for any vehicle type.
* **Abstraction**: The Vehicle class is abstract, providing a simplified interface for interacting with vehicles while hiding the specific details of each vehicle's rental behavior in its subclasses.
* **Composition**: The RentalAgency class is composed of Vehicle objects, and the Customer class contains a list of RentalTransaction objects to record each rental event.

**How It Works**

1. **Vehicles** are instantiated with a unique ID, model, base rental rate, and availability status.
2. **Customers** rent vehicles by calling the rent() method, specifying rental days. The system calculates the cost based on the vehicle type and rental duration.
3. **Rental transactions** are recorded, and the customer's rental history is updated.
4. **Vehicle availability** is checked before a rental, and each rental transaction updates the vehicle's availability.

**Conclusion**

This system efficiently models the process of renting vehicles, leveraging OOP principles to ensure flexibility, reusability, and maintainability. It allows easy expansion for future features such as loyalty programs, ratings, or more vehicle types.