**Vehicle Rental Management System Documentation**

**Overview**

The **Vehicle Rental Management System** is designed to demonstrate core **Object-Oriented Programming (OOP)** principles using Java. It simulates a vehicle rental process where customers can rent vehicles, return them, and manage rental transactions.

**Key OOP Principles Implemented**

1. **Abstraction**: Hides complex details in an abstract class and provides a simplified interface.
2. **Inheritance**: Concrete vehicle classes extend a base vehicle class.
3. **Encapsulation**: All critical data is hidden inside classes with controlled access.
4. **Polymorphism**: Allows different vehicle types to behave differently, even when using the same method.
5. **Composition**: The system is built by combining multiple classes (e.g., vehicles, customers, transactions) that work together.

**Classes Overview**

**1. Vehicle (Abstract Class)**

* **Purpose**: Represents a generic vehicle.
* **Attributes**: vehicleId, model, baseRentalRate, isAvailable
* **Methods**:
  + calculateRentalCost(int days) (abstract): Calculates rental cost for a given number of days.
  + isAvailableForRental() (abstract): Checks if the vehicle is available for rental.

**2. Car, Motorcycle, Truck (Concrete Classes)**

* **Purpose**: Specific types of vehicles (Car, Motorcycle, Truck) that extend Vehicle.
* **Methods**:
  + Each class implements calculateRentalCost() and isAvailableForRental() with its own rules (e.g., a car may have a different rental rate than a motorcycle).

**3. Customer**

* **Purpose**: Represents a customer who rents vehicles.
* **Attributes**: customerId, name, rentedVehicles (list of vehicles rented by the customer).
* **Methods**:
  + rentVehicle(Vehicle vehicle, int days): Allows a customer to rent a vehicle.
  + returnVehicle(Vehicle vehicle): Allows a customer to return a rented vehicle.

**4. RentalAgency**

* **Purpose**: Manages a fleet of vehicles and handles rental transactions.
* **Attributes**: vehicles (list of vehicles available for rent), customers (list of customers).
* **Methods**:
  + addVehicle(Vehicle vehicle): Adds a new vehicle to the agency’s fleet.
  + processRental(Customer customer, Vehicle vehicle, int days): Processes the rental of a vehicle to a customer.

**5. RentalTransaction**

* **Purpose**: Represents a single rental transaction.
* **Attributes**: vehicle, customer, rentalDays.
* **Methods**:
  + calculateTotalCost(): Calculates the total cost of the rental.

**Interfaces**

**Rentable Interface**

* **Methods**:
  + rent(Customer customer, int days): Allows a customer to rent a vehicle.
  + returnVehicle(): Allows a vehicle to be returned.

**Key Features**

* **Loyalty Program (Bonus)**: Customers can earn loyalty points based on rental history for discounts.
* **Exception Handling (Bonus)**: Handles cases where vehicles are unavailable or rental durations are invalid.

**How It Works**

1. **Vehicles** are created by extending the Vehicle class and implementing specific rules for calculating rental costs and checking availability.
2. **Customers** can rent and return vehicles using methods provided in the Customer class.
3. **RentalAgency** manages the fleet and processes rentals and returns.
4. **RentalTransaction** tracks the details of each rental, including cost and duration.

**OOP Principles in Action**

* **Abstraction**: The Vehicle class provides common methods, but the specific rental calculations and availability logic are hidden in subclasses like Car, Truck, and Motorcycle.
* **Inheritance**: Car, Motorcycle, and Truck inherit from the Vehicle class, sharing common behavior but implementing unique logic.
* **Encapsulation**: Vehicle details like vehicleId and baseRentalRate are private and can only be accessed through getter and setter methods, ensuring controlled access.
* **Polymorphism**: The rent() method in Rentable works with any type of vehicle (Car, Truck, Motorcycle), allowing the system to process different vehicle types the same way.
* **Composition**: The system is built by combining multiple classes like Customer, RentalAgency, and RentalTransaction to create a working vehicle rental system.

**Conclusion**

This system uses **Object-Oriented Programming** principles to create a flexible and modular **Vehicle Rental Management System**. It allows customers to rent vehicles, track their rentals, and manage transactions, while also following best practices for OOP design.