**OOP Principles and Implementation**

1. **Abstraction**:
   * **Abstract Vehicle Class**: Defines methods calculateRentalCost(int days) and isAvailableForRental(), implemented by subclasses.
   * **Rentable Interface**: Provides rent() and returnVehicle() methods for vehicle renting behavior, overridden by each vehicle class.
2. **Inheritance**:
   * **Vehicle Class**: Parent class with common fields like vehicleId, model, baseRentalRate, and isAvailable.
   * **Car, Motorcycle, Truck**: Concrete subclasses inherit Vehicle and implement specific rental cost calculations and availability checks.
3. **Encapsulation**:
   * **Private Fields**: Fields like vehicleId, model, baseRentalRate, and isAvailable are private.
   * **Getters/Setters**: Controlled access through getters and setters with validation (e.g., ensuring positive rental rates).
4. **Polymorphism**:
   * **Method Overriding**: Car, Motorcycle, and Truck override methods like calculateRentalCost() and isAvailableForRental().
   * **Rentable Interface**: rent() and returnVehicle() methods are polymorphic, allowing flexible vehicle rental handling.
5. **Composition**:
   * **Customer**: Has a rental history (composed of RentalTransaction objects) and may have active rentals (composed of Vehicle objects).
   * **RentalAgency**: Manages a fleet of vehicles (Vehicle objects) and processes rentals.

**Key Classes**

* **Vehicle (Abstract)**: Base class with common fields and abstract methods.
* **Car, Motorcycle, Truck**: Concrete vehicle classes implementing specific behavior for rental calculation and availability.
* **Customer**: Manages rental history and current rentals.
* **RentalTransaction**: Tracks rental details (vehicle, customer, cost, duration).
* **RentalAgency**: Manages a fleet of vehicles and processes rental transactions.

**Conclusion**

The system uses OOP principles to create a flexible, maintainable design that supports vehicle rentals with specific cost calculations, availability checks, and customer management.