### **1. Payment Processing System**

**Principles:**

* Single Responsibility (SRP)
* Open/Closed (OCP)
* Dependency Inversion (DIP)

**Scenario:**You are designing a payment processing system for an online platform. The system must support multiple payment methods, such as Credit Card and PayPal, while remaining flexible for future payment methods.

**Requirements:**

1. **Single Responsibility Principle:**
   * A class PaymentDetails should encapsulate payment-specific information such as Amount, Currency, and TransactionId.
   * Payment processing logic should be separated into specific classes.
2. **Open/Closed Principle:**
   * Define an interface IPaymentProcessor with a method such as ProcessPayment(PaymentDetails details).
   * Implement this interface in classes like CreditCardProcessor and PayPalProcessor.
   * Adding new payment methods should only require creating a new implementation of IPaymentProcessor.
3. **Dependency Inversion Principle:**
   * Create a high-level class PaymentService that depends on the IPaymentProcessor interface.
   * Use dependency injection to provide the specific IPaymentProcessor implementation to PaymentService.

### **2. Vehicle Rental Service**

**Principles:**

* Interface Segregation (ISP)
* Liskov Substitution (LSP)

**Scenario:**You are designing a vehicle rental service for Cars, Trucks, and Bikes. Each vehicle has unique capabilities:

* Cars can carry passengers.
* Trucks can carry cargo.
* Bikes are drivable but do not carry passengers or cargo.

**Requirements:**

1. **Interface Segregation Principle:**
   * Define small, specific interfaces such as:
     + IDrivable with a method like Drive().
     + IPassengerCarrier with a method like GetPassengerCapacity().
     + ICargoCarrier with a method like GetCargoCapacity().
   * Ensure that each vehicle type (e.g., Car, Truck, Bike) implements only the interfaces relevant to its functionality.
2. **Liskov Substitution Principle:**
   * Create a base interface IVehicle with methods like StartEngine() or GetDescription().
   * Ensure that all vehicle types (Car, Truck, Bike) can substitute the IVehicle interface without breaking the system.

### **3. E-Commerce Discount and Tax Calculation System**

**Principles:**

* **Single Responsibility Principle (SRP)**
* **Open/Closed Principle (OCP)**
* **Liskov Substitution Principle (LSP)**

**Scenario:**Design a system for an e-commerce platform that calculates both discounts and taxes on orders. The platform offers multiple product categories (e.g., electronics, clothing, groceries), and different rules apply for each category. Additionally, there are various types of customers (e.g., regular, premium) and regional tax rules (e.g., country, state-specific tax rates).

Your task is to create a flexible and extensible design that can easily accommodate new discount rules and tax rates without modifying the core logic of the system.

### **Requirements:**

1. Single Responsibility Principle (SRP)

* **Discounts** and **taxes** should be calculated in separate services.
  + **IDiscountCalculator** should focus only on calculating discounts based on product categories or customer types.
  + **ITaxCalculator** should handle calculating taxes based on the region and product category.
  + **No class should have more than one responsibility.** For example, the same class should not calculate both discounts and taxes.

2. Open/Closed Principle (OCP)

* The system should be designed in a way that allows you to **add new types of discounts and tax calculations** without modifying the existing code.
  + You should be able to **add new discount calculators** (e.g., seasonal discounts, bulk discounts, holiday promotions) or **new tax calculation methods** (e.g., luxury tax, state tax, VAT) by simply adding new classes that implement the IDiscountCalculator or ITaxCalculator interfaces.
  + The existing system should remain **unchanged** even if new types of discounts or taxes are introduced. New classes should extend or implement existing abstractions but not modify existing logic.

3. Liskov Substitution Principle (LSP)

* All subclasses of IDiscountCalculator and ITaxCalculator should be **substitutable** without affecting the correctness of the system.
  + For example, the system should work correctly whether you are using a BulkDiscountCalculator or a PremiumCustomerDiscountCalculator in place of the general IDiscountCalculator interface.
  + Similarly, any tax calculator (e.g., StateTaxCalculator or LuxuryTaxCalculator) that implements ITaxCalculator should be interchangeable with the base interface without breaking the system.