**SOLID TRONG THIẾT KẾ HỆ THỐNG**

|  |
| --- |
| **Single Responsibility Principle(SRP- nguyên tắc trách nhiệm đơn lẻ)**  **Mô tả**: Một class (hoặc module) chỉ nên có **một lý do duy nhất để thay đổi**. Điều này nghĩa là một class chỉ nên chịu trách nhiệm thực hiện **một nhiệm vụ cụ thể**. **Ví dụ**:   * Class Invoice chỉ chịu trách nhiệm quản lý thông tin hóa đơn, trong khi việc lưu trữ hóa đơn nên được tách thành class InvoiceRepository.   **Lợi ích**:   * Dễ bảo trì vì các thay đổi chỉ ảnh hưởng đến một phần nhỏ của hệ thống. |

|  |
| --- |
| ****Open/Closed Principle(OCP là nguyên tắc đóng mở)**** **Mô tả**: Một class nên được thiết kế để **mở rộng** nhưng **đóng với sửa đổi**. Nghĩa là, bạn có thể thêm tính năng mới mà không cần thay đổi mã nguồn hiện có. **Cách thực hiện**:   * Sử dụng kế thừa, interface hoặc abstraction để thêm các tính năng mới.   **Ví dụ**:   * Class Shape có các phương thức trừu tượng như calculateArea(). Khi thêm hình mới (hình tròn, hình chữ nhật), chỉ cần tạo class kế thừa từ Shape mà không sửa đổi Shape. |

|  |
| --- |
| ****Liskov Substitution Principle(LSP- nguyên tắc phân vùng Liskov)**** **Mô tả**: Các class con có thể thay thế hoàn toàn cho class cha mà không làm thay đổi hành vi của chương trình. **Điều kiện**:   * Class con không được thay đổi định nghĩa hoặc logic của class cha.   **Ví dụ**:   * Nếu class Bird có phương thức fly(), thì class con như Sparrow hay Eagle cũng phải có thể bay. Tuy nhiên, class Penguin không bay được thì không nên kế thừa từ Bird. |

|  |
| --- |
| ****Interface Segregation Principle(ISP- nguyên tắc phân tách giao diện)**** **Mô tả**: Một interface không nên bắt buộc các class triển khai những phương thức mà chúng **không sử dụng**. **Cách thực hiện**:   * Tách các interface lớn thành các interface nhỏ, phù hợp với từng mục đích sử dụng cụ thể.   **Ví dụ**:   * Thay vì một interface Animal có tất cả các phương thức như fly(), swim(), run(), hãy chia nhỏ thành các interface như Flyable, Swimmable, Runnable. |

|  |
| --- |
| ****Dependency Inversion Principle(DIP chính là nguyên tắc đảo ngược phụ thuộc)**** **Mô tả**:   * Các module bậc cao không nên phụ thuộc trực tiếp vào các module bậc thấp.   Cả hai nên phụ thuộc vào **abstraction (Interface)**.  - Modul bậc cao là các phương thức chưa logic chính , Thực hiện logic tổng thể và chức năng chính của hệ thống.  -Modul bậc thấp là các phương thức thực hiện các chi tiết kỹ thuật cụ thể như giao tiếp cơ sở dữ liệu, xử lý file, hoặc các chức năng hỗ trợ.   Cung cấp dịch vụ cho module bậc cao.   * **Abstraction** không nên phụ thuộc vào **chi tiết**, mà **chi tiết** phải phụ thuộc vào **abstraction**.   **Ví dụ**:   * Thay vì module OrderService phụ thuộc trực tiếp vào MySQLDatabase, ta nên dùng interface Database để trung gian. Khi đó, có thể dễ dàng thay thế MySQLDatabase bằng MongoDB mà không cần sửa OrderService |