2025-09-05

SOLID 원칙

Single Responsibility Principle (단일 책임 원칙, SRP)

- 클래스는 단 하나의 책임만 가짐
- `유지보수성`: 객체 하나당 한가지 일만 신경 쓰면 됨
- `테스트 용이성`: 독단적 테스트 가능
- `확장성`: 타 클래스를 건드리지 않고 확장 가능

Open/Closed Principle (개방-폐쇄 원칙, OCP)

- 기능을 추가할 때 코드를 수정하지 않고 확장
- **확장에는 열림(Open), 수정할 필요는 닫힘(Closed)**

Liskov Substitution Principle (리스코프 치환 원칙, LSP)

- **자식 클래스는 부모 클래스를 대체할 수 있어야 한다**
- 자식은 부모의 메서드를 오버라이딩
- 다형성의 장점을 살리고, 코드 변경 시 오류를 최소화

Interface Segregation Principle (인터페이스 분리 원칙, ISP)

- **사용하지 않는 메서드에 의존하지 않아야 한다**
- 필요한 인터페이스를 작게 분리
- **클라이언트의 독립성**
- **변경에 강함**: 인터페이스가 작고 특정 기능에 집중되어 있어, 변경 사항이 영향X
- **단일 책임 원칙(SRP)과의 연결성**: 인터페이스가 작고 명확할수록 책임이 분리되고, 유지보수가 쉬워짐

Dependency Inversion Principle (의존성 역전 원칙, DIP)

- 추상화는 세부사항에 의존하지 않아야 한다. 세부사항이 추상화에 의존
- **상위 레벨 하위 레벨 구현**에 의존 X
- **구현체 교체 가능**
- **유연하고 확장 가능**: 변경할 때 상위 모듈 변경 X
- 디자인 패턴, 인터페이스 등으로 구현 가능

OOP 핵심 개념

1. 추상화 (Abstraction)

객체를 단순화하여 핵심 속성이나 기능만 나타내는 것

2. 캡슐화 (Encapsulation)

객체의 속성과 메서드를 하나의 논리적 단위로 묶고, 외부에서 객체가 제공하는 메서드를 통해서만 접근하도록 제한

3. 상속 (Inheritance)

상위(부모) 클래스의 속성과 메서드를 하위(자식) 클래스가 물려받아 사용하는 것입니다.

4. 다형성 (Polymorphism)

같은 메서드 호출이더라도, 객체의 실제 타입에 따라 다른 동작을 수행하는 특성입니다.

> 설계 원칙이 중요한 이유
>
> - **코드 유지보수성**: 각 객체의 책임 분명해짐
> - **확장성과 유연성**: 결합도는 낮추고 유연도는 높인다
>   ex) 인터페이스 사용
> - **팀 협업과 커뮤니케이션**
> - **의존성과 결합도를 줄여 재사용성 높임**