**설계원칙 : SOLID**

* 단일 책임 원칙 (Single responsibility principle) : SRP
* 개방 폐쇄 원칙 (Open-closed principle) : OCP
* 리스코프 치환 원칙 (Liskov substitution principle) : LSP
* 인터페이스 분리 원칙 (Interface segregation principle) : ISP
* 의존 역전 원칙 (Dependency inversion principle) : DIP

1. **단일 책임 원칙(SRP)** : 클래스는 단 한 개의 책임을 가져야 한다.
   1. 클래스가 변경되는 이유는 단 한가지여야 한다.
   2. 원칙을 어기면 재사용을 어렵게 하는 문제가 발생함.
   3. 단일 책임 원칙을 준수하면 재사용성을 높일 수 있다.
   4. 서로 다른 이유로 바뀌는 책임들이 한 클래스에 함께 포함되어 있다면 이 클래스는 단일 책임 원칙을 어기고 있다고 볼 수 있다.
   5. 단일 책임 원칙을 지키는지 확인할 때에는 사용자를 확인해본다.
   6. 클래스의 사용자들이 서로 다른 메서드들을 사용한다면 그들 메서드는 각각 다른 책임에 속할 가능성이 높고, 따라서 책임 분리 후보가 될 수 있다.
2. **개방 폐쇄 원칙(OCP)** : 확장에는 열러 있어야 하고, 변경에는 닫혀 있어야 한다.
   1. 기능을 변경하거나 확장할 수 있으면서, 그 기능을 사용하는 코드는 수정하지 않는다.
   2. 추상화를 통해서 가능하게 할 수 있다.
   3. 상속을 이용해서 가능하게 할 수 있다.
   4. 추상화와 다형성(상속)을 이용해서 개방 폐쇄 원칙을 구현. 그렇기 때문에 제대로 지켜지지 않으면 원칙을 어기게 된다.
      1. 다운 캐스팅을 한다. instanceof() 와 같은 타입 확인 연산자가 사용된다면 지켜지지 않았을 가능성이 높다.
      2. 비슷한 if - else 블록이 존재한다.
   5. 변경의 유연함과 관련된 원칙
   6. 코드에 대한 변화 요구가 발생하면, 변화와 관련된 구현을 추상화해서 개방 폐쇄 원칙에 맞게 수정할 수 있는지 확인하는 습관을 갖자.
3. **리스코프 치환 원칙(LSP)** : 상위 타입의 객체를 하위 타입의 객체로 치환해도 상위 타입을 사용하는 프로그램은 정상적으로 동작해야 한다.
   1. 개방 폐쇄 원칙을 받쳐 주는 다형성에 관한 원칙을 제공
   2. 리스코프 치환 원칙이 지켜지지 않으면 개방 폐쇄 원칙 또한 지켜지지 않은 것
   3. 대표 예 : 직사각형 - 정사각형 문제
   4. 리스코프 치환 원칙은 기능의 명세(계약)과 확장에 대한 것이다.
   5. 기능 실행의 계약과 관련해서 흔히 발생하는 위반 사례
      1. 명시된 명세에서 벗어난 값을 리턴한다.
      2. 명시된 명세에서 벗어난 익셉션을 발생한다.
      3. 명시된 명세에서 벗어난 기능을 수행한다.
   6. 하위 타입은 상위 타입에서 정의한 명세를 벗어나지 않는 범위에서 구현해야 한다.
   7. 리스코프 치환 원칙을 어기게 되는 이유는 추상화가 덜 되었기 때문
   8. 리스코프 치환 원칙을 지키지 않으면 기능을 확장하기가 어렵게 된다.
4. 인터페이스 분리 원칙(ISP) : 인터페이스는 그 인터페이스를 사용하는 클라이언트를 기준으로 분리해야 한다.
   1. 클라이언트는 자신이 사용하는 메서드에만 의존해야 한다.
   2. 인터페이스 분리 원칙은 결국 인터페이스와 콘크리트 클래스의 재사용성을 높여 주는 효과도 갖는다.
   3. 인터페이스 분리 원칙은 클라이언트 입장에서 인터페이스를 분리하라는 원칙
   4. 인터페이스를 분리하는 기준이 클라이언트가 된다는 것. 클라이언트가 사용하는 기능을 중심으로 인터페이스를 분리. 클라이언트로부터 발생하는 인터페이스 변경의 여파가 다른 클라이언트에 미치는 영향을 최소화
5. **의존 역전 원칙(DIP)** : 고수준 모듈은 저수준 모듈의 구현에 의존해서는 안 된다. 저수준 모듈이 고수준 모듈에서 정의한 추상 타입에 의존해야 한다.
   1. 고수준 모듈 : 어떤 의미 있는 단일 기능을 제공하는 모듈
   2. 저수준 모듈 : 고수준 모듈의 기능을 구현하기 위해 필요한 하위 기능의 실제 구현
   3. 저수준 모듈이 변경되더라도 고수준 모듈은 변경되지 않는 것
   4. 저수준 모듈이 고수준 모듈을 의존하게 만들면 된다. 어떻게? 추상화
   5. 의존 역전 원칙은 앞서 리스코프 치환 원칙과 함께 개방 폐쇄 원칙을 따르는 설계를 만들어 주는 기반이 되는 것
   6. 의존 역전 원칙은 소스 코드에서의 의존을 역전시키는 원칙
   7. 인터페이스는 저수준 모듈 보다는 고수준 모듈 입장에서 만들어짐. 그래서 의존 역전 원칙이 됨.
   8. 런타임의 의존이 아닌 소스 코드의 의존을 역전시킴으로써 변경의 유연함을 확보할 수 있도록 만들어 주는 원칙이지, 런타임에서의 의존을 역전시키는 것은 아니다.
   9. 의존 역전 원칙은 타입의 소유도 역전시킨다.
   10. 타입 소유 역전을 하지 않을 경우, 상세 구현의 독립적 배포가 안 된다.
   11. 개방 폐쇄 원칙을 클래스 수준뿐만 아니라 패키지 수준까지 확장시켜 주는 디딤돌이 된다.