**2. SOLID 원칙 (객체지향 5대 원칙)**

**1) SRP (Single Responsiblity Principle, 단일 책임 원칙)**

* **클래스나, 함수는 단 하나의 책임(기능)만을 가져야 한다.**
* 클래스, 함수가 비대해지면 이를 분리시킬 필요가 있다.
* 산탄총 수술
  + 하나의 책임이 여러 개의 클래스로 분산되어 있는 경우
  + 요구 사항이 변경될 시, 분산된 책임을 가지고 있는 모든 부분을 살펴야 한다.

**2) OCP (Open-Closed Principle, 개방-폐쇄 원칙)**

* 기존 코드 변경에는 닫혀있고, 추가나 확장에는 열려있어야 한다.
* **자주 변경될 수 있는 내용은 수정하기 쉽게 설계**해야 하고,  
  자주 변경되지 않을 내용은 수정에 영향받지 않게 설계해야 한다.

**3) LSP (Liskov Substitution Principle, 리스코프 치환 원칙)**

* 자식 클래스는 부모 클래스에서 가능한 행위를 수행할 수 있어야 한다.
* 파생 클래스를 만들 때, 이게 **정말 올바른 상속의 관계를 갖는지 생각**해봐야 한다.
* 대표적인 예. [Rectangle, Square 클래스](https://nesoy.github.io/articles/2018-03/LSP)

**4) DIP (Dependency Inversion Principle, 의존 역전 원칙)**

* 의존 관계를 맺을 때, 변화하기 쉬운 것 보단 **변화하기 어려운 것에 의존**해야 한다.
* 변화하기 쉬운 것 = 구체적인 것 (클래스, 서브 클래스 인스턴스)  
  변화하기 어려운 것 = 추상적인 것 (**추상 클래스, 인터페이스**)
* 즉, {인터페이스 or 추상클래스} {변수 명} = {서브 클래스 인스턴스} 꼴이 되어야 한다.
* 의존성 주입 (Dependency Injection) 기술

**5) ISP (Interface Segregation Principle, 인터페이스 분리 원칙)**

* 클라이언트가 자신이 이용하지 않는 메서드에 의존하지 않아야 한다.
* 큰 덩어리의 인터페이스들을 구체적이고 작은 단위들로 분리시킴으로써 **클라이언트들이 꼭 필요한 메서드들만 이용할 수 있게 한다.**
* 다시 말해, 인터페이스를 클라이언트에 특화되도록 분리시키라는 설계 원칙이다.

**잠깐 환기하는 객체지향 3대 특징**

**캡슐화**

* 객체의 필드(속성), 메소드를 하나로 묶고, 실제 구현 내용을 외부에 감추는 것
* 정보의 손상과 오용 사전 방지.
* **다른 객체와의 독립성 유지**로 전체적인 코드 유지보수 및 변경에 유연

**상속**

* 상속이란 **기존 클래스를 재사용**하는 것
* 개발시간 절약, 중복 코드 감소 및 유지보수에 용이해짐

**다형성**

* **하나의 타입에 여러 객체를 대입**함으로써 다양한 기능을 이용할 수 있도록 하는 것
* 구체적으로는, 부모 클래스 또는 인터페이스의 타입 변환, 오버로딩과 오버라이딩