2023년 2월 18일

객체지향에 대한 나의 생각

임원호

객체지향 4가지 특성

1. 추상화 (Abstraction)

불필요한 부분을 제거함으로써, 필요한 부분만을 남겨놓은 것이다. 객체지향에서 추상화의 목적은 복잡성을 낮추기 위해 사용하는 것. 추상화에 대한 사람들의 생각은 각자 다르지만, 나는 추상화의 근본적인 논리 자체가 복잡한 자료, 시스템에서 핵심적인 개념을 뽑거나 간추려내는 것을 말하는데, 핵심적이지 않다는 것은 불필요한 것이고, 핵심적인 것은 필요한 부분이라는 것이다. 그래서, 불필요한 부분을 제거함으로 써, 필요한 부분을 남기는 것이 추상화이다.

2. 다형성 (Polymorphism)

하나의 타입으로 여러 종류의 객체를 참조하는 것을 다형성이다. 프로그램 언어 각 요소들이 타입에 속하는 것이 허가되는 성질을 가르키는 것이다. 다형성을 구현하는 방법은 오버로딩, 오버라이딩이 존재한다. 또한 각 메소드는 이름으로 구분하는게 아닌, 파라미터로 다르게 구분 할 수 있는 것 또한, 다형성의 하나이다.

3. 캡슐화 (Encapsulation)

캡슐화란 객체 내부의 세부사항을 외부로 부터 감추는 것이고, 이런 세부사항을 인터페이스만 공개해서 변경하기 쉬운 코드를 만들기 위함이다. 자동차를 예로 들면,

자동차 내부의 엔진과 다른 부품과의 통신이나, 연결은 모두 보이지 않게끔, 외관으로 막아놓았다. 이처럼, 내부사항을 외부로부터 감추는 것이 캡슐화이다.

4. 상속 (Inheritance)

부모로 부터 상속받는 것을 상속이다.

객체지향의 5가지 설계 원칙 (SOLID)

1. SRP - Single Responsibility Principle (단일 책임의 원칙)

하나의 책임을 가져야 한다. 책임이란, 기능을 담당하는 것이다. 하나의 클래스는 하나의 기능 담당하여 하나의 책임을 수행해야 한다.

2. OCP : Open/Closed Principle (개방 폐쇄의 원칙)

확장에는 열려있고, 변경에는 닫혀있다.
기존 코드를 변경하지않고 기능을 추가할 수 있어야한다.

3. LSP : Liskov's Substitution Principle (리스코프 치환의 원칙)

상위 타입의 객체를 하위 타입의 객체로 치환해도 동작에 문제가 없어야한다.

4. ISP : Interface Segregation Principle (인터페이스 분리의 원칙)

많은 기능을 가진 인터페이스를 작은 단위로 분리시켜, 클라이언트에게 필요한 인터페이스들만 구현한다. 즉 클라이언트가 사용하지않는 기능에 의존하게 되면 문제가 발생하는데, 이런 문제를 해결할 수 있다.

5. DIP : Dependency Inversion Principle (의존성 역전의 원칙)

의존관계를 맺을 때 자주 변경이 일어나는 쪽이 아닌 변경이 적게 일어나는 곳에 의존하라는 의미이며 자기보다 변하기 쉬운 것에 의존하면, 변화의 영향을 많이 받기 때문에 추상화된 인터페이스나 상위 클래스를 두어서 변화에 영향을 받지않게 하기 위한 원칙이라 생각합니다.

절차지향 프로그래밍 vs 객체지향 프로그래밍

객체지향 패러다임

적절한 객체에게 적절한 책임을 할당하여 서로 메시지를 주고 받으며 협력하도록 하는 것
점점 증가하는 SW 복잡도를 낮추기 위해 객체지향 패러다임이 떠올랐다.
1. 클래스가 아닌 객체에 초점을 맞추는 것
2. 객체들에게 얼마나 적절한 역할과 책임을 할당하는 것
책임이 한곳에 집중되어 있는 방식 (getter) - 절차지향 프로그래밍
책임이 여러 객체로 적절히 분산되어 있는 방식 - 객체지향 프로그래밍
즉 하나의 메소드에서 getter를 통해 값을 가져오고 이곳에서 모든 처리가 집중되어있으면 절차지향 프로그래밍이라 생각한다. getter가 아닌 해당 책임을 가진 객체에게 메세지를 통해 협력하도록 구현하는 것이 객체지향 프로그래밍이라 생각한다.

객체 지향

High cohesion, Loose coupling - 높은 응집도의 낮은 결합도

높은 응집도의 낮은 결합도의 장점

응집도가 높은 것끼리 모아두면 변경이 생길 때 변경의 포인트가 하나로 집중이 될 수 있다. 역량 범위를 파악하는 것이 굉장히 한 곳에 집중되어 있다는 것이다. 프로그램을 하다보면, 수정에 대한 요구사항을 받을 수 있는데, 역량 범위가 어렵다는 표현을 쓴다. 그런 표현을 쓴다는 것 자체가 객체지향적으로 설계가 되지 않았다. 변경에 대한 요청이 들어왔을 때, 변경에 대한 한 부분만 수정하면, 응집도가 높다는 것이고, 변경이 생겼을 때, 다른 곳에 영향이 가지 않는다면, 낮은 결합도를 가지는 것이다. 즉 이 표현은 유지보수와 관련된 것에 크다고 생각합니다. 객체지향 설계를 한다는 것은 어떠한 변경이 생겼을 때, 빠르게 높은 응집도와 낮은 결합도를 가지고, 유연하게 대응할 수 있다.

객체지향 설계 및 구현

1. 도메인을 구성하는 객체에는 어떤 것들이 있는지 고민하기
2. 객체들 간의 관계를 고민하기
3. 동적인 객체를 정적인 타입으로 추상화해서 도메인 모델링하기 - (복잡성을 낮춤)
4. 협력을 설계한다. (적절한 객체에 적절한 책임을 할당)
5. 객체들을 포괄하는 타입에 적절한 책임을 할당 (클래스에 적절할 책임을 할당한다.)
6. 구현하기 (퍼블릭 인터페이스를 구현)
#객체지향 세계에서는 모든 객체가 능동적인 존재#