코딩 교육 메뉴얼 (5)

코딩 교육 메뉴얼은 현재 객체지향 프로그래밍을 수강하고 있는 학생들에게 프로그래밍 공부를 헤매지 않고 지속적으로 공부할 수 있을 지 고민하여, 주기적으로 코딩 공부에 대한 가이드라인이나 간략한 노하우를 알려주기 위해 작성되고 있습니다.

이번 시간에는 유명한 객체지향 설계 5대 원칙을 정리해보는 시간을 갖도록 하겠습니다. 개발자 면접 단골 질문이기도 하고, 앞으로 공부를 이어가면서 자주 언급될 수 있는 중요한 개념인 만큼 객체지향 프로그래밍을 설명하면서 빼놓을 수 없는 개념이죠.

SOLID 원칙

1. 단일 책임 원칙 (Single Responsiblity Principle)

모든 클래스는 각각 하나의 책임만 가져야 하며, 클래스는 그 책임을 완전히 캡슐화해야 하는 원칙을 의미합니다. 결국 캡슐화를 통해 외부로부터 안전하게 보호되는 객체는 단일한 책임을 가지고 맡은 역할을 수행해야 합니다. 예를 들어 사칙연산 함수를 가지고 있는 계산 클래스가 있다면, 이 상태의 계산 클래스는 오직 사칙연산 기능만을 책임져야 합니다. 또한, 이 클래스를 수정하게 된다면 그 이유는 사직연산 함수와 관련된 문제 뿐입니다.

2. 개방-폐쇄 원칙 (Open Closed Principle)

소프트웨어 요소는 확장에는 열려있으나 수정에는 닫혀있어야 하는 원칙입니다.

기존의 코드를 변경하지 않으면서(Closed), 기능을 추가할 수 있도록(Open) 설계가 되어야 한다는 원칙을 말합니다.

3. 리스코프 치환 원칙 (Liskov Substitution Principle)

프로그램의 객체는 프로그램의 정확성을 깨뜨리지 않으면서 하위 타입의 인스턴스로 바꿀 수 있어야 한다는 원칙입니다.

다형성에서 하위 클래스는 인터페이스 규약을 다 지켜야 한다는 것, 다형성을 지원하기 위한 원칙, 인터페이스를 구현한 구현체는 믿고 사용하려면, 이 원칙이 필요하죠.

예를 들어, 자동차 인터페이스의 엑셀은 앞으로 가라는 기능이라면 이걸 뒤로 가게 구현하면 LSP 위반이 되죠. 아무리 느리더라도 앞으로 움직이게 구현해야 합니다.

4. 인터페이스 분리 원칙 (Interface Segregation Principle)

특정 클라이언트를 위한 인터페이스 여러 개가 범용 인터페이스 하나보다 낫다는 원칙입니다.

다시 말해 하나의 일반적인 인터페이스보다 여러개의 구체적인 인터페이스가 낫다는 것이죠.

5. 의존 역전 원칙 (Dependency Inversion Principle)

의존 관계를 맺을 때 변화하기 쉬운 것 또는 자주 변화하는 것보다는 변화하기 어려운 것, 거의 변화가 없는 것에 의존해야 합니다.

한마디로 구체적인 클래스보다 인터페이스나 추상 클래스와 관계를 맺으라는 것이죠.

Code tip

마지막으로 간단한 코딩 노하우를 소개해드리려고 합니다.

다양한 팁이 있고, 공부를 해나가며 자신만의 노하우가 쌓이겠지만 오늘 소개해드릴 것은

'코드 레벨 맞추기' 입니다.

쉽게 얘기하자면, 같은 수준의 코드끼리 구현을 해야한다는 것인데, 이는 객체지향 설계 원칙을 준수하며 코드 설계하는데 전반으로 사용됩니다.

예를 들어, 외부로부터 메시지를 받아 필요한 데이터를 문자열로 응답해주는 메서드가 있다고 합시다.

그렇다면 그 메서드 내에서 메시지를 수신하고 이를 통해 필요한 데이터를 찾아 응답해주는 로직이 필요하겠죠. 또한 여기서 찾은 데이터가 보내고자 하는 형식과 맞지 않다면 이를 변환해줄 필요가 있을 수 있습니다. 그러나 필요한 데이터를 찾는 로직과 변환 로직은 그 메서드에 모두 담아서는 안됩니다.

메서드가 가진 수신 및 응답의 역할과 코드 레벨이 맞지 않기 때문입니다.

보다 나은 설계는 데이터를 찾는 로직과 변환 로직은 별도의 메서드로 만들어 하나의 메서드는 각자 하나의 역할만을 충실히 이행하고 확장에 열려있고 수정에 닫혀있을 수 있도록 설계하는 것입니다.