객체지향 프로그래밍 개요

(Object-Oriented-Programming Overview)

이미지출처 : https://geetikakaushik2020.medium.com/what-is-object-oriented-programming-7f14c5147ee5

객체지향 프로그래밍(Object-Oriented Programming, OOP)은 프로그래밍에서 데이터를 처리하는 방법들을 객체라는 단위로 구분하여 프로그램을 구성하는 방법론이다. 객체지향 프로그래밍은 여러 기법들을 사용하여 프로그램을 마치 실제 세계의 객체들이 상호작용하는 것처럼 모델링함으로써 복잡한 문제를 보다 쉽게 해결할 수 있도록 돕는다. 이러한 객체지향 프로그래밍의 주요 개념 및 기법에는 클래스, 객체, 캡슐화, 추상화 , 상속, 다형성 등이 있다.

1) 클래스와 객체

클래스(Class): 객체를 생성하기 위한 템플릿 또는 설계도이다. 클래스는 객체의 기본 형태를 정의하며, 이에는 객체가 가지는 속성(변수)과 행동(메서드)이 포함된다.

객체(Object): 클래스에 정의된 속성과 행동을 가지는 실체이다. 예를 들어, '동물'이라는 클래스가 있다면, '강아지', '고양이' 등은 이 클래스의 인스턴스(객체)가 될 수 있다. 각 객체는 클래스에서 정의한 속성과 행동을 가지되 각기 다른 값을 가질 수 있다.

2) 캡슐화(Encapsulation)

캡슐화는 객체의 속성과 메서드를 하나로 묶고 실제 구현 내용 일부를 외부에서 접근하지 못하도록 숨기는 기법이다. 이를 통해 객체의 데이터를 보호하고, 외부에서 임의로 변경할 수 없도록 하여 데이터의 무결성을 유지할 수 있다.

3) 상속(Inheritance)

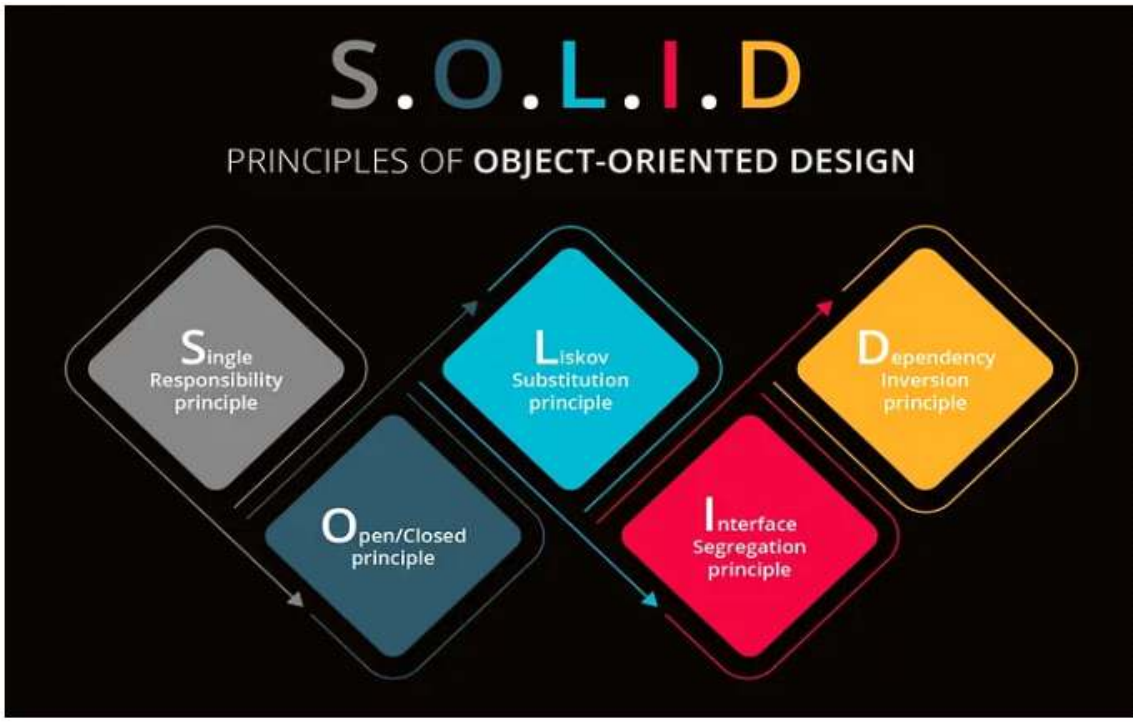
상속은 한 클래스가 다른 클래스의 속성과 메서드를 물려받는 기능이다. 이를 통해 코드의 재사용성을 높이고 중복을 줄일 수 있다. 예를 들어, '동물' 클래스에서 공통적인 속성과 행동을 정의하고, '강아지' 클래스에서는 '동물' 클래스의 특성을 상속받아 강아지만의 특성을 추가로 정의할 수 있다.

4) 다형성(Polymorphism)

다형성은 같은 이름의 메서드가 다른 클래스에서 다른 작업을 수행할 수 있도록 하는 개념이다. 예를 들어, '동물' 클래스에 '소리내기'라는 메서드가 있을 때, '강아지' 클래스에서는 '멍멍', '고양이' 클래스에서는 '야옹'이라는 소리를 내도록 같은 메서드를 다르게 구현할 수 있다.

5) 추상화(Abstraction)

추상화는 복잡한 실세계 문제를 간단한 모델로 변환하는 과정이다. 이 과정에서 프로그램을 보다 이해하기 쉽게 만든다. 이는 프로그램의 설계와 유지보수를 효율적으로 만들어 준다.
예를 들어, '동물'이라는 개념을 추상화하여 '동물' 클래스를 만들 수 있다. 이 클래스에는 '먹기', '움직이기' 같은 공통적인 행동이 정의될 수 있으며, 실제 동물(예: 강아지, 고양이)은 이 '동물' 클래스를 상속받아 각각의 특성에 맞는 구체적인 행동 방식을 구현한다.



이미지 출처: https://sakuntha.medium.com/solid-principles-of-object-oriented-design-9ba978676916

SOLID는 객체 지향 설계의 다섯 가지 기본 원칙을 나타내는 약어로, 소프트웨어의 유지보수성과 확장성을 높이는 데 목적이 있다. SOLID 원칙을 따르면, 소프트웨어는 더욱 견고하고, 유연하며, 유지보수가 쉬워진다. 이를 통해 소프트웨어 개발 과정에서 발생할 수 있는 많은 문제들을 예방할 수 있다.

1) S: Single Responsibility Principle (단일 책임 원칙)

각 클래스는 하나의 기능만을 가지며, 클래스가 변경되어야 하는 이유는 단 하나여야 한다. 이 원칙은 클래스의 복잡성을 줄이고, 시스템의 유지보수성을 향상시킨다.

2) O: Open/Closed Principle (개방/폐쇄 원칙)

소프트웨어의 구성요소(클래스, 모듈, 함수 등)는 확장에 대해서는 열려 있어야 하며, 수정에 대해서는 닫혀 있어야 한다. 즉, 기존의 코드를 변경하지 않고도 객체의 행동을 변경하거나 확장할 수 있어야 한다.

3) L: Liskov Substitution Principle (리스코프 치환 원칙)

프로그램에서 부모 클래스의 인스턴스 대신 자식 클래스의 인스턴스를 사용해도 프로그램의 정확성이 변하지 않아야 한다. 이는 상속 관계에 있는 클래스 간의 호환성을 의미한다.

4) I: Interface Segregation Principle (인터페이스 분리 원칙)

클라이언트는 자신이 사용하지 않는 인터페이스에 의존하면 안 된다. 큰 단일 인터페이스보다는, 필요한 인터페이스만을 구현하도록 여러 개의 작은 인터페이스로 분리하는 것이 좋다.

5) D: Dependency Inversion Principle (의존 역전 원칙)

고수준 모듈은 저수준 모듈에 의존해서는 안 되며, 둘 다 추상화에 의존해야 한다. 또한, 추상화는 세부 사항에 의존해서는 안 되며 세부 사항은 추상화에 의존해야 한다.