프로그래밍 패러다임

주차	19주차
스터디 일자	@2024/03/08

프로그래밍 패러다임이란?

> **패러다임의 사전적 정의**
>
> 한 시대의 사람들의 견해나 사고를 근복적으로 규정하고 있는 인식의 체계.
> 또는 사물에 대한 이론적인 틀이나 체계. 일종의 '틀'
>
> (출처 : Oxford Languages)

이를 프로그래밍 분야에 적용해본다면...

개발자는 프로그래밍과 관련된 이론적 체계 혹은 틀(패러다임)에 따라

'어떻게 프로그래밍을 할 것인가'란 의문을 스스로 던지고,

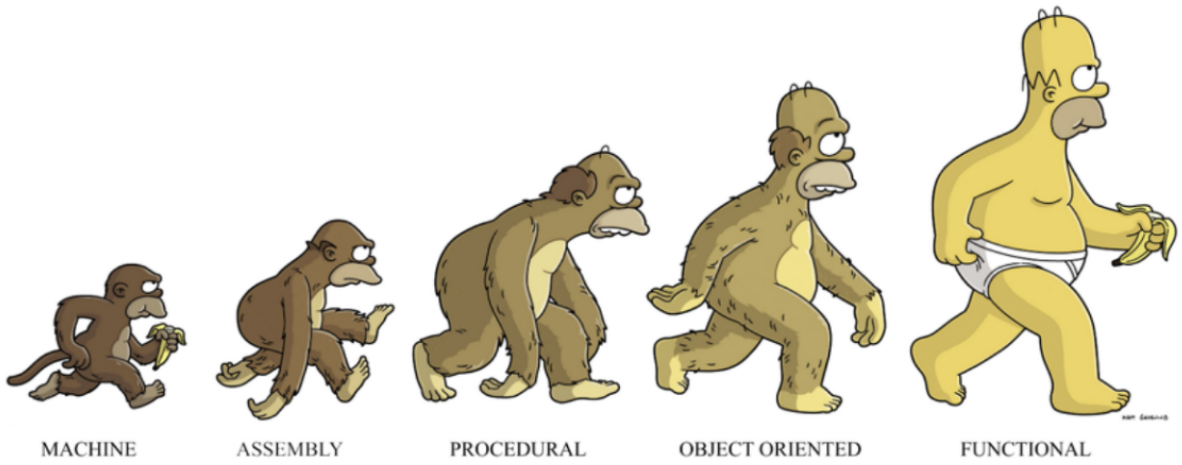
특정 관점과 방식을 바탕으로 '프로그래밍'을 실시하게 되는 셈

프로그래밍 패러다임의 변화

- 통상, *프로그래밍 패러다임* 이라 하면 아래 3가지를 의미
  - **절차적 프로그래밍** (PP, Procedure Programming)

- **객체지향 프로그래밍** (OOP, Object Oriented Programming)
- **함수형 프로그래밍** (FP, Functional Programming)

- 해당 분류는 각 패러다임이 프로그래밍 분야에서 각광받은 시점과도 동일
  - 다만, 등장 순서는 정 반대로 **함수형 → 객체지향 → 절차적** !



객체지향 프로그래밍

실세계에 존재하고 인지하고 있는 **객체(Object)**를 소프트웨어의 세계에서 표현하기 위해 객체의 핵심적인 개념 또는 기능만을 추출하는 **추상화(abstraction)**를 통해 모델링하려는 프로그래밍 패러다임

👉 우리가 주변의 **실세계에서 사물을 인지하는 방식을 프로그래밍에 접목하려는 사상**을 의미

> 💡 객체지향 프로그래밍은 보다 **유연하고 유지보수하기 쉬우며 확장성** 측면에서서도 유리한 프로그래밍을 하도록 의도되었기 때문에 대규모 소프트웨어 개발에 널리 사용되고 있다!

객체지향 개발 5대 원리 : SOLID 원칙

> ***S****RP (단일 책임의 원칙 - Single Responsibility Principle)*
>
> **O***CP (개방폐쇄의 원칙 : Open Close Principle)*
>
> **L***SP (리스코브 치환의 원칙 : The Liskov Substitution principle)*
>
> **I***SP (인터페이스 분리의 원칙 : Interface Segregation principle)*
>
> **D***IP (의존성 역전의 원칙 : Dependency Inversion Principle)*

SRP - 단일 책임의 원칙

> 작성된 클래스는 **하나의 기능만 가지며** 클래스가 제공하는 모든 서비스는 그 **하나의 책임을 수행하는 데 집중**되어 있어야 한다는 원칙

이는 즉, 어떤 변화에 의해 클래스를 변경해야 하는 이유는 **오직 하나뿐이어야 함**을 의미함.

SRP 원리를 적용하면

- 무엇보다도 책임 영역이 확실해지기 때문에 한 책임의 변경에서 다른 책임의 **변경으로의 연쇄작용**에서 자유로울 수 있음.
- 책임을 적절히 분배함으로써 **코드의 가독성 향상, 유지보수 용이**의 이점을 누릴 수 있음.
- 객체지향 원리의 다른 원리들을 적용하는 **기초가 됨.**

적용 방법

리팩토링(*Refactoring: Improving the Design of Existing Code - Martin Fowler*)에서 소개하는 대부분의 위험상황에 대한 해결방법은 직/간접적으로 SRP 원리와 관련 있음.

이는 항상 코드를 최상으로 유지한다는 >>리팩토링<<의 근본정신 또한 항상 **객체들의 책임을 최상의 상태로 분배한다는 것에서 비롯**되기 때문

1️⃣ 여러 원인에 의한 변경 (Divergent change) :

Extract Class를 통해 혼재된 각 책임을 각각의 개별 클래스로 분할하여 클래스 당 하나의 책임만을 맡도록 하는 것

- 여기서 관건은 책임만 분리하는 것이 아닌 분리된 두 클래스간의 관계의 복잡도를 줄여 주어야함.
    - 만약 Extract Class된 각각의 클래스들이 유사하고 비슷한 책임을 중복해서 갖고 있다면 Extract Superclass를 사용할 수 있음.
- 이는 Extract된 각각의 클래스들의 공유되는 요소를 부모 클래스로 정의하여 부모 클래스에 위임하는 기법
  따라서 각각의 Extract Class들의 유사한 책임들은 부모에게 명백히 위임하고 다른 책임들은 각자에게 정의할 수 있음.

2️⃣ 사탄총 수술(Shotgun surgery) :

Move Field와 Move Method를 통해 책임을 기존의 어떤 클래스로 모으거나, 이럴만한 클래스가 없다면 새로운 클래스를 만들어 해결할 수 있음.
👉 산발적으로 여러 곳에 분포된 책임들을 한 곳에 모으면서 설계를 깨끗하게 하고 응집성을 높이는 작업

OCP - 개방폐쇄의 원칙

> *객체지향 소프트웨어 설계* 라는 책에서 정의한 내용으로, 소프트웨어의 구성요소(컴포넌트, 클래스, 모듈, 함수)는 **확장에는 열려있고 변경에는**

닫혀있어야 한다는 원칙

이는 즉, 변경을 위한 비용은 가능한 줄이고 확장을 위한 비용은 가능한 극대화 해야 한다는 의미

⇒ 요구사항의 변경이나 추가사항이 발생하더라도, 기존 구성요소는 수정이 일어나지 말아야 하며, 기존 구성요소를 쉽게 확장해서 재사용할 수 있어야 한다는 뜻

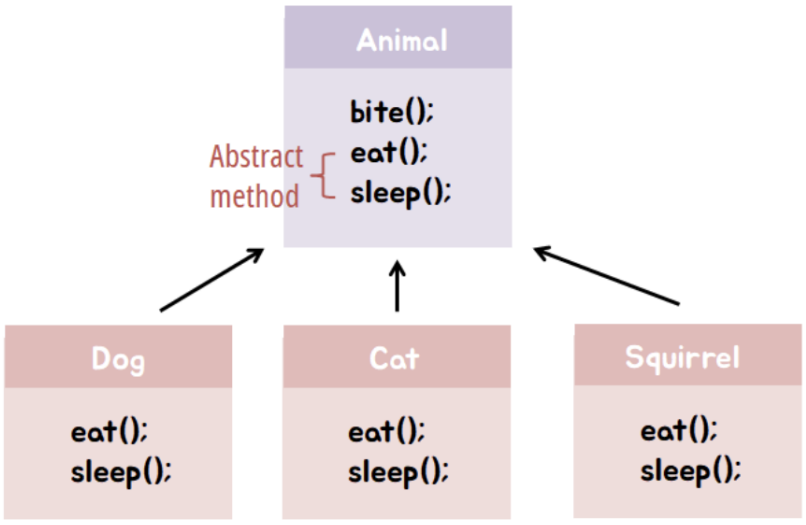
OCP 원리를 적용하면

- 관리 가능하고 재사용 가능한 코드를 만드는 기반이 됨.
- 새로운 변경 사항이 발생했을 때 유연하게 코드를 추가함으로써 애플리케이션의 기능을 큰 힘을 들이지 않고 확장할 수 있음.

적용 방법

1. 변경(확장)될 것과 변하지 않을 것을 엄격히 구분
2. 이 두 모듈이 만나는 지점에 인터페이스를 정의
3. 구현에 의존하기보다 정의한 인터페이스에 의존하도록 코드를 작성

이 과정은 즉, **추상화**다!



LSP - 리스코브 치환의 원칙

> 서브 타입은 언제나 기반 타입과 호환될 수 있어야 한다.
> 즉, 서브 타입은 기반 타입이
> **약속한 규약을 지켜야 한다는 원칙**

이는 즉, 부모 객체와 이를 상속한 자식 객체가 있을 때 부모 객체를 호출하는 동작에서 **자식 객체가 부모 객체를 완전히 대체**할 수 있어야 한다는 의미

객체지향 언어에선 객체의 상속이 일어나고, 이 과정에서 부모/자식 관계가 정의됨.
자식 객체는
부모 객체의 특성을 가지며, 이를 토대로 확장할 수 있음.
하지만, 이 과정에서 무리하거나
객체의 의의와 어긋나는 확장으로 인해 잘못된 방향으로 상속되는 경우가 생김.

리스코프 치환 원칙(LSP)은 올바른 상속을 위해 자식 객체의 확장이 **부모 객체의 방향을 온전히 따르도록 권고하는 원칙**

적용 방법

1. 만약 두 개체가 똑같은 일을 한다면 둘을 하나의 클래스로 표현하고 이들을 구분할 수 있는 필드를 둠.
2. 똑같은 연산을 제공하지만, 이들을 약간씩 다르게 한다면 공통의 인터페이스를 만들고 둘이 이를 구현함. * 인터페이스 상속
3. 공통된 연산이 없다면 완전 별개인 2개의 클래스를 만듦.

ISP - 인터페이스 분리의 원칙

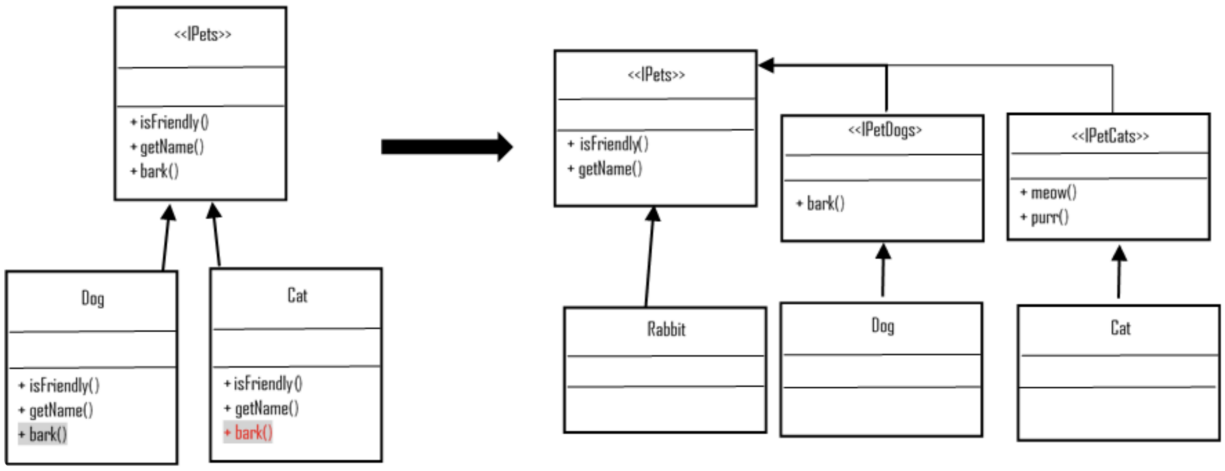
> 한 클래스는 자신이 사용하지 않는 인터페이스는 구현하지 말아야 한다는 원칙

이는 즉, 어떤 클래스가 다른 클래스에 종속될 때에는 가능한 최소한의 인터페이스만을 사용해야 한다는 의미

SRP가 **클래스의 단일책임**을 강조한다면, **ISP**는 **인터페이스의 단일책임**을 강조

☝ **특정 클라이언트를 위한 인터페이스 여러 개가 범용 인터페이스 하나보다 나음.**

구현할 객체에게 무의미한 메소드의 구현을 방지하기 위해 반드시 필요한 메소드만을 상속/구현하도록 권고하는데,
만약 상속할 객체의 규모가 너무 크다면 해당 객체의 메소드를
작은 인터페이스로 나누는 것이 좋다는 것이 ISP 원칙!



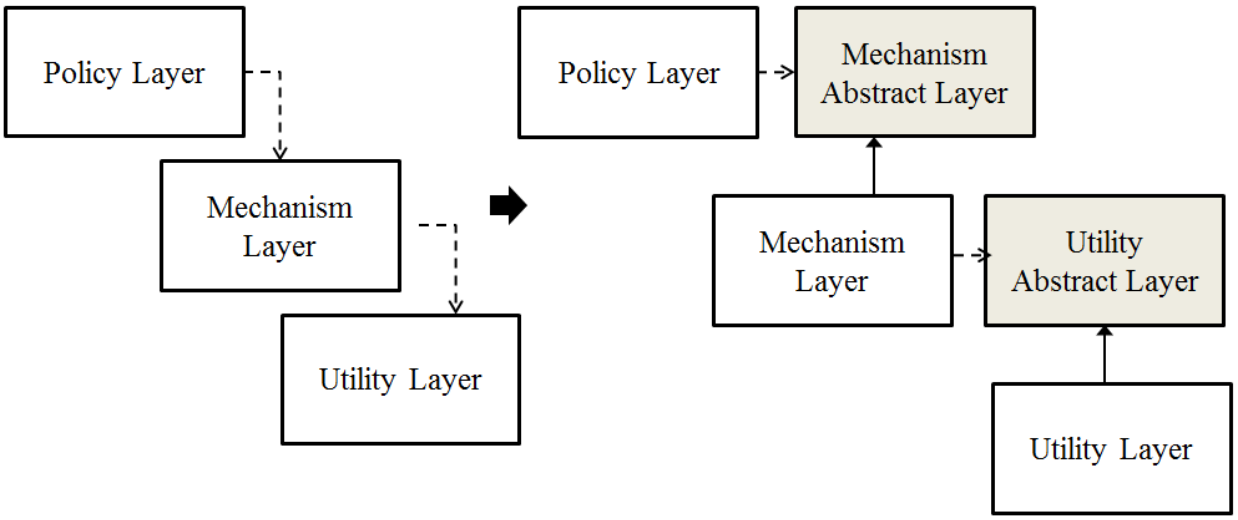
DIP - 의존성 역전의 원칙

> 의존성 역전 원칙이란 객체는 저수준 모듈보다 **고수준 모듈에 의존해야한다는 원칙**
>
> - 고수준 모듈: 인터페이스와 같은 객체의 형태나 추상적 개념
> - 저수준 모듈: 구현된 객체

이는 즉, **추상화에 의존해야지, 구체화에 의존하면 안된다는 의미**

적용 방법

- 상위 레벨의 레이어가 하위 레벨의 레이어를 바로 의존하게 하는 것이 아니라 이 둘 사이에 존재하는 **추상 레벨을 통해 의존해야 함.**
- 이를 통해서 상위레벨의 모듈은 하위레벨의 모듈로 의존성에서 벗어나 그자체로 재사용 되고 **확장성도 보장 받을 수 있음.**



참고

https://www.nextree.co.kr/p6960/

https://iosdevlime.tistory.com/entry/CSBasic-좀-더-나은-프로그램을-위해-프로그래밍-패러다임

https://inpa.tistory.com/entry/OOP-💠-아주-쉽게-이해하는-OCP-개방-폐쇄-원칙