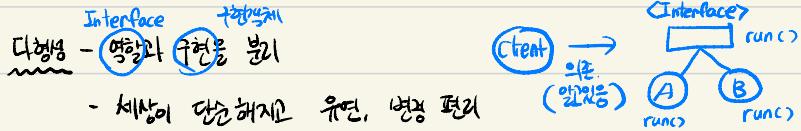


개체지향 설계와 스프링

- 스프링이란? 스프링 부트, 스프링 프레임워크 등을 모두 포함한 스프링 생태계
- 핵심 개념 : 개체지향언어가 가진 강력한 특징을 살려내는 프레임워크 (DI, IOC 컨테이너)
 - 좋은 개체지향 프로그래밍이란? 다형성 - 역할과 구현을 분리
 - 세상이 단순해지고 유연, 변경 편리
 - 클라이언트를 변경하지 않고 서버의 구현기능을 유연하게 변경 가능.



client — ①
client — ②

⇒ 결국 인터페이스를 잘 설계하는게 중요

- 스프링과 객체지향

IOC, DI 는 다형성을 활용하여 역할과 구현을 편리하게 해줌.

다형성이 주가로..

- 좋은 객체지향 설계의 5가지 원칙 (SOLID)

1. SRP (single responsibility principle) : 한 클래스는 하나의 책임만 가져야 한다. 변화 있을 때 고민해야
2. OCP (open-closed principle) : 개방-폐쇄 원칙 / 소프트웨어 요소는 확장에는 열려있으나 변경에는 닫혀있어야 한다.
 - : 위에 예제로 보면. 클라이언트-A
 - 클라이언트-B
 - 클라이언트-C

클라이언트가 구현클래스를 선택하기 때문에
클라이언트 코드를 바꿔야 한다. ⇒ OCP 원칙에 어긋남.

: 이 문제점을 해결하기 위해선 객체를 생성하고 연관관계를 맺어주는
별도의 코드, 설정자가 필요하다. = 스프링 컨테이너
3. LSP (Liskov Substitution Principle) : 하위 클래스는 인터페이스 규약을 지켜야 한다. ex. 엑셀은 앞으로
가는 이동.
4. ISP (Interface Segregation Principle) : 인터페이스 분리 규칙.

: 특정 클라이언트를 위한 인터페이스 설계자가 별용 인터페이스
하나보다 낫다.

- * DIP (Dependency Inversion Principle) : 의존관계 역전 원칙.
- : 추상화에 의존. 구체화에 의존 X
 - : 즉, 인터페이스에 의존. 구현클래스에 의존 X
 - : 앞에 예시에서 Member Repository m = new MemoryMemberRepository()
 - 클라이언트가 구체객체를 알고있음. (DIP 위반)

⇒ 대형 프로젝트는 OCP, DIP 를 지킬 수 있다.

문서가 더 필요.

그래서 고민한게...

* 개발자 향과 스피킹

DI 와 DI 컨테이너로 대형성 + OCP + DIP 가능하게 지원.

+ 스피킹 언어 자체로 OCP, DIP 를 지원해서 코드를 짜면 어느새 DI 컨테이너를 만들고 있는 것.

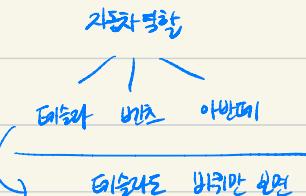
1장 정리.

모든 설계에 명함과 구현을 분리하자.

이상적으로 모든 설계에 인터페이스를 부여하자.

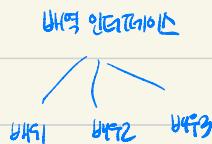
ex. 자동차로 치면

공연으로 치면



비록 Interface.

|
 비록



실무 고민

