


2019.06.09 Clem Code.

why clem code? → Code는 한번 작성 10번 정도 읽힌다. (Running Code)

절차/구조
러미

procedural ⇒ 프로그램은 데이터의 위치를 통해 ⇒ 데이터의 변경이 많은 함수의 영향을 받는다
oop ⇒ 데이터와 지시하는 함수 외부의 영향은 분리된 (Interface의 변경만 있다면 외부의 영향 X)

⇒ 유지보수가 쉬워야 하면 oop은 하위 종족

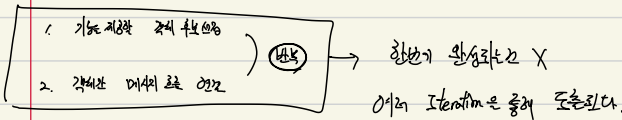
⇒ 다른 script 같은 procedural은 서로 관련이 없다
(서로 같은...)

ex. 객체는 본래 데이터가 아닌 기능을 위해서 생겨나서 생긴다 (ex. Article Service Vs Write Article Service)

↳ 기능 자체에 WriteArticle 이라 같은 naming + How to do what 으로 정의해준다.
(Json Request Param / Request Param)

"객체는 책임의 집합" ⇒ 객체는 약한느 가된다.

객체지향 설계 과정



Encapsulation → 내부의 변화가 외부에 영향을 주면 X

Tell Don't Ask ⇒ 객체에게 요청하지 말고 만능하다

(은행원이 손님에게 "카드를 좀" 라고만 말하고, 스스로 리미트에 카드를 가져온다 X)



Customer.hasCard() 0
Customer.getCard().getCard() != null X

"Law of Demeter" (안 알려줘...)

→ Command (tell) ⇒ 객체 내부 상의 모든 변경

Command vs Query

→ Query (Ask) ⇒ 객체 내부 상의 모든 변경 X (Free of side effect)

⇒ 객체 내부 상의 Command or Query 로 이루어진다.

↗ 객체별 Return Type이 다르

✓ Command → 상수는 변경되지 않 Return type X (간혹 f 같은)

Query → 상수 변경 X Return type O.

⇒ 상수는 변경되지 않는 불변하는 것은 좋지 않다

Poly morphism → extends
2개의 implements ⇒ (재사용)
↳ 객체와 객체의 재사용

Interface → TC 사용
① 나중에 추가될 수 있는 다른 내용 X
② 객체의 재사용 → "유연"

"상수" = 재사용이 불가능!

↓

"재사용"이 아닌 (의존) 인 ⇒ extends는 가장 강한 의존

Interface는 동적 상수 사용에 선제가 필요하다 (test로 생각해...)!

"concrete class"는 변수가 없음 다 수정해야함

"Interface"는 내부 변수가 바뀌지 않는 이상 객체가 바뀌어도 결과는 똑같음 수정하면 ok.

Interface type = new ImplementType()!

(의존) 상