2019.06.09 Clean Code.

why clean code? → Code는 한번 작성시 10번 정도 읽힌다 (Running Code)

편차/객체 차이 (
procedural ⇒ 프로시저들이 데이터를 공유하는 형태 ⇒ Data의 변경에 맞는 함수들이 영향을 받음 → 요구사항 변경시 데이터 구조에 영향을 미침 (모든 코드의 변경 유발)
OOP ⇒ Data 와 제기하는 함수는 외부에 영향을 안미침 (Interface의 변경만 없다면 외부에 영향 X) → 중요

⇒ 유지보수가 되어야한다면 oop를 하는게 좋음
⇒ 단순 script 정도는 procedural 은 써도 괜찮은 듯
(shell 같은..)

※ 객체는 분리 Data가 아닌 기능 관점에서 생각해야 한다 (ex. Article Service vs Write Article Service)
↳ 기능이기에 WriteArticle 이 더 좋은 naming + How 보다 what 으로 정의해야한다.
(JsonRequestParser / RequestParser)

"역할은 책임의 집합" ⇒ 객체는 역할을 가진다.

객체지향 설계 과정
1. 기능 제공할 객체 후보 선정	반복
2. 객체간 메시지 흐름 연결	

→ 한번에 완성되는건 X
여러 Iteration을 통해 도출된다.

Encapsulation → 내부의 변화가 외부에 영향을 주면 X
Tell! Don't Ask ⇒ 객체에게 요청하지 말고 말하라.
(은행원이 손님에게 "카드를 줘" 라고 말하지, 스스로 지갑에서 카드를 꺼내오진 X)

Customer.hasCard() O
Customer.getWallet().getCard() != null X

"Law of Demeter" (안 나중에..)

Command vs Query
→ Command (tell) ⇒ 객체 내부 상태를 변경 → 대부분 Return Type이 없음
→ Query (Ask) ⇒ 객체의 상태를 변경 X (Free of side Effect)
⇒ 하나의 함수는 Command or Query 로 이루어진다.

∨ Command → 상태는 변경되나 Return type X (간혹 있을 수 있음)

Query → 상태 변경 X Return type O.

⇒ 상태는 변경하고 값도 반환하는 함수는 좋지 않음

Polymorphism → extends
2가지 implements ⇒ (재사용) ↳ 객체 간의 관계의 재사용

Interface → ① TC 쉬움
② 나중에 추가되지 않는 다는 보장 X
③ 관계의 재사용 → "유연"

"상속"은 재사용이라 볼 수 있나?
⇓
"재사용"이 아닌 (의존)임 ⇒ extends는 가장 강한 의존

Interface를 통해 상위 수준의 설계가 도출이 된다 (test로 고려하고..)

"Concrete class"는 변화가 생기면 다 수정해야 함

"Interface"는 내부 함수가 바뀌지 않는 이상 구현체가 바뀌므로 주입하는 부분만 수정하면 ok.

InterfaceType type = new ImplementType();
(교체 가능)

extends → ① super class 변경이 다수의 sub class에 영향을 미친다
② 다중상속 불가
③ 상속 자체를 잘못 사용하는 경우가 있다
↳ 불변 규칙이 깨질 수 있음

→ Composition이 더 좋은 듯 (큰 이유가 없다면 ...)