2019.06.09 Clean Code.

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

절차/객체 차이
- procedural ⇒ 프로시저들이 데이터를 공유하는 형태 ⇒ Data의 변경에 많은 함수들이 영향을 받음
- OOP ⇒ Data와 제기하는 함수는 외부에 영향을 안끼침 (Interface의 변경만 없다면 외부에 영향 X)

⇒ 유지보수가 되야한다면 OOP를 하는게 좋음

⇒ 단순 script 정도는 procedural을 써도 괜찮은 듯
(shell 같은..)

※ 객체는 본래 Data가 아닌 기능의 관점에서 생각해야 한다. (ex. Article Service vs Write Article Service)
↳ 기능이기에 WriteArticle이 더 좋은 naming + How 보다 what으로 정의해야한다.
(JsonRequestParser / RequestParser)

"역할은 책임의 집합" ⇒ 객체는 역할을 가진다.

객체지향 설계 과정

1. 기능 제공할 객체 후보 선정
2. 객체간 메시지 흐름 연결

) 반복 → 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 꺼내면 X)

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

"Law of Demeter" (얘 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 Query로 이루어진다.

- Command → 상태는 변경되나 Return type X (간혹 있을 수 있음)
  Query → 상태 변경 X Return type O.

⇒ 상태는 변경하고 값을 반환하는 함수는 좋지 않음

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

Interface → ① IC 쉬움
② 나중에 추가되지 않는 다른 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 조기 된다 (test 도 포함하고..)

"Concrete class"는 변화가 생기면 다 수정해야함
"Interface"는 내부 함수가 바뀌지 않는 이상 구현체가 바뀌어도 종합하는 부분만 수정하면 ok.

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