30. 디자인 패턴.

- 좁은 범위의 문제 → 도구의 문제 / 일반문제X (주로).

커맨드.

- 간단한 method 호출 X
  복잡한 형태의 계산작업 호출하면서
  → 계산작업 객체 생성, 이후 호출.

값객체

- 공유로 간주해야하지만 동일성은 중요치 X
  ↳ 객체 생성과 상태 고정 → 연산시 새로운 객체 반환.
  - 객체의 복사본을 제공.
  - 시간 및 공간 측면에서 비효율적일 가능성.
  - observer 패턴 → 제어흐름이 어려워질 우려 있음.
  - 의존성 설정 / 제거 어려워질 수 있다.
- 결론적으로 수학적 값을 연산하면 value object를 사용하려는 경향. 좋다.
- 값 객체는 동등성/ 해시를 구현해야한다.

null 객체.

- 특별한 상황에선 특별한 객체를 사용.
- null 일 시 새로운 객체 반환하여 null check 제거

template method.

- 작업순서는 변하지 않는다. 각 단위에 대해.
  이런 가능성을 열어두고 싶을 경우?
  → 다른 메서드를 호출하는 메서드로만 이루어진 메서드를 만든다.

pluggable object.

- 변이(?)를 어떻게 표현할 것인가?
  - 명시적인 조건문을 사용한다.
- 그러나 명시적인 조건문은 → 중복으로 발전한다.

pluggable selector.

- 인스턴스 별로 서로 다른 메서드 호출을 원할 시
  - switch문 → bad.
  - reflection을 활용한 호출.

factory method

- object를 생성하는 method
- 객체를 생성한다는 것을 기억해야한다.

Imposter

- 새로운 구현을 도입해야한다.
- Imposter pattern의 두가지 예
  - null object
  - composite.

Composite

- 하나의 객체 → 다른 객체 collection에
  행위 전달한다. 동일하게 하려면?
- 객체컬렉션을 나타내는 객체를 단일객체의 Imposter로 구현한다.
- 동일 Interface로 단일객체 → collection으로 확장.
  - 단순한 설계가능.

수집 매개 변수.

- 여러 객체에 걸쳐 호출되는 operation의 결과를 수집하려면?
  - 결과 수집 객체를 operation의 매개변수로 추가한다.

Singletone

- 전역변수 제공X 재사용X.