Design Pattern

- □□ □□ □□

패턴 접근 방식

생성 패턴

- Factory Method: 객체 생성을 서브클래스에 위임하고자 할 때.
- Abstract Factory: 관련 객체들의 제품군을 생성하고자 할 때.
- Builder: 복잡한 객체의 생성 과정을 단계별로 나누고자 할 때.
- Prototype: 객체를 복사하여 새로운 객체를 생성하고자 할 때.
- Singleton: 특정 클래스의 인스턴스를 하나만 생성하고자 할 때.

구조적 패턴

- Adapter: 기존 클래스를 수정하지 않고 호환성을 제공하고자 할 때.
- Bridge: 구현과 추상화를 분리하여 독립적으로 변형할 수 있도록 할 때.
- Composite: 객체들을 트리 구조로 구성하여 부분-전체 계층을 표현하고자 할 때.
- Decorator: 객체에 동적으로 새로운 기능을 추가하고자 할 때.
- Facade: 복잡한 시스템에 단순한 인터페이스를 제공하고자 할 때.
- Flyweight: 많은 수의 작은 객체들을 효율적으로 관리하고자 할 때.
- Proxy: 객체에 대한 접근을 제어하고자 할 때.

행동 패턴

- Chain of Responsibility: 요청을 처리할 수 있는 객체를 연결하여 요청을 순차적으로 처리하고자 할 때.
- Command: 실행될 작업을 캡슐화하여 작업의 실행을 객체화하고자 할 때.
- Interpreter: 언어의 문법을 클래스로 표현하고자 할 때.
- Iterator: 집합체 요소들을 순차적으로 접근하고자 할 때.
- Mediator: 객체 간의 상호작용을 캡슐화하여 객체들이 서로 직접 통신하지 않도록 할 때.
- Memento: 객체 상태를 저장하고 복원하고자 할 때.
- Observer: 객체의 상태 변화에 따라 다른 객체들이 통지받고자 할 때.
- State: 객체의 상태에 따라 행위를 변경하고자 할 때.
- Strategy: 알고리즘을 캡슐화하여 동적으로 변경하고자 할 때.
- Template Method: 알고리즘의 골격을 정의하고 세부 사항을 서브클래스에서 구현하고자 할 때.
- Visitor: 객체 구조를 변경하지 않고 새로운 기능을 추가하고자 할 때.