Composite Pattern

결론 및 정리

정의

한 마디로, Interface로 묶음으로써 클래스들을 Tree 로서 묶을 수 있도록 만드는 Pattern 이라고 생각하면 될 듯.
Tree 자체의 특성 중 하나가 Traversal. 그 내부에서 순회를 돌 수 있다는 것이므로, 순회도 간단하게 가능하고.

Composite Pattern은 Structural Pattern 중 하나로, **Leaf (개별 객체) 와 Composite (복합 객체) 를 동일하게 취급하도록 하여, 객체의 구조를 트리 구조로 구성하는 패턴**. 이 패턴은 클라이언트가 개별 객체와 복합 객체를 구분하지 않고 일관된 방식으로 접근할 수 있게 해줌.

- **Component**: 개별 객체와 복합 객체의 공통 인터페이스. 공통 연산들을 선언.
- **Leaf**: Component 인터페이스를 구현하는 개별 객체. 복합 객체가 아니므로, 자식을 추가하거나 제거하는 연산은 지원하지 않음.
- **Composite**: Component 인터페이스를 구현하는 복합 객체. 자식 Component들을 관리하며, 자식을 추가하거나 제거하는 연산을 포함.

구현

1. 공통 인터페이스인 Component를 정의한다다.
2. Leaf 클래스에서 Component를 구현합니다.
3. Composite 클래스에서 Component를 구현하며, 자식 Component들을 관리하는 메서드들을 추가합니다.

Context & Problem & Merit

- 개별 객체와 그 객체들의 조합(복합 객체)을 동일하게 취급하고 싶을 때 Composite Pattern을 사용합니다.
- 트리 구조로 객체를 구성하여, "계층적인 구조"를 표현하고자 할 때 유용합니다.

1. **일관성**: 클라이언트는 개별 객체와 복합 객체를 동일한 방식으로 취급할 수 있습니다.
2. **간결성**: 클라이언트는 복잡한 트리 구조를 쉽게 추가하거나 수정할 수 있습니다.
3. **확장성**: 새로운 종류의 Component나 Composite를 쉽게 추가할 수 있습니다.