SWIFT
Composite

Bill Kim(김정훈) | ibillkim@gmail.com

목차

Composite

Composite(컴포지트)패턴은 클라이언트가 복합 객체(group of object) 나 단일 객체를 동일하게 취급하는 것을 목적으로 하는 패턴입니다.

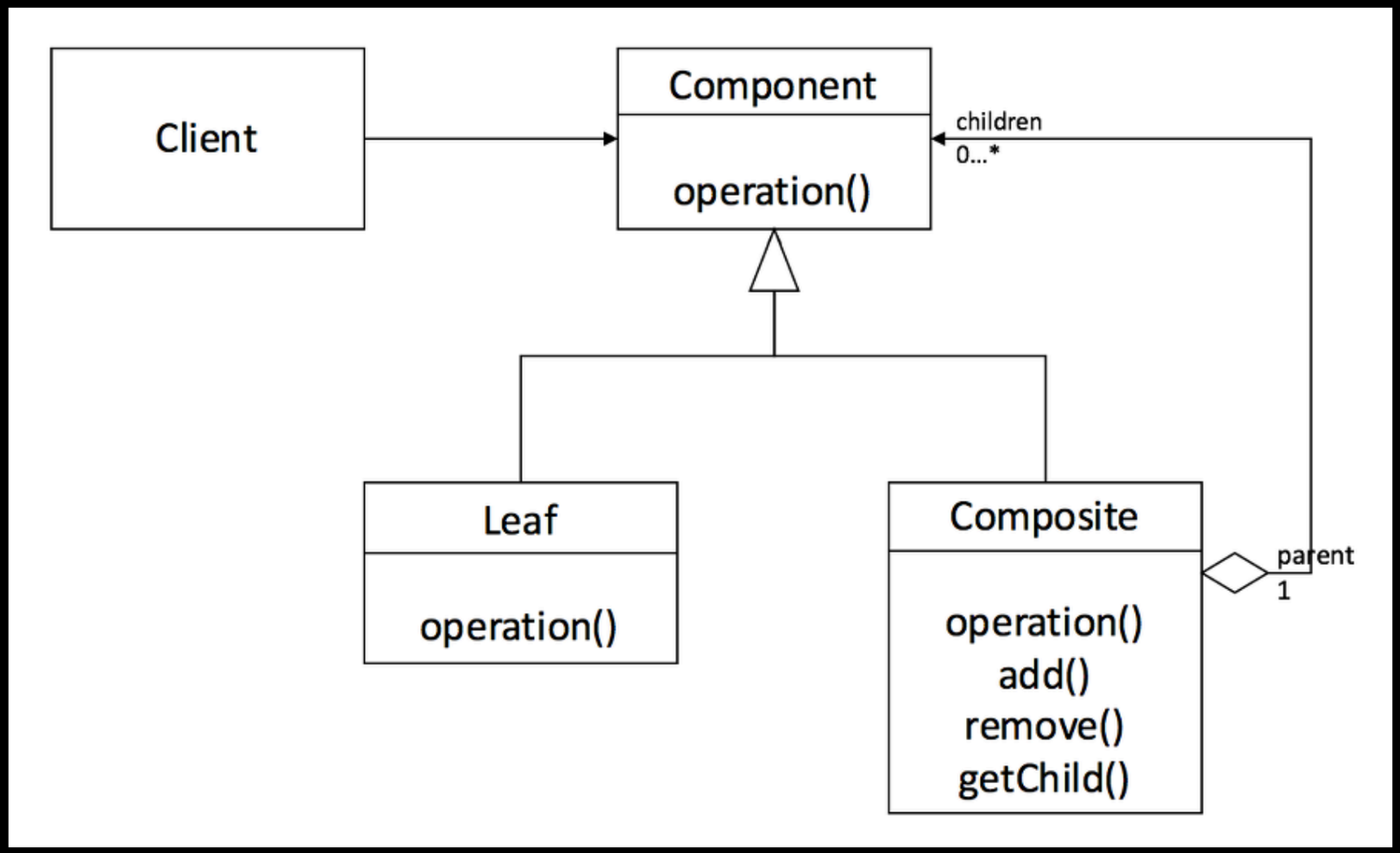
즉, 전체-부분의 관계(Ex. Directory-File)를 갖는 객체들 사이의 관계를 정의할 때 유용합니다.

또한 클라이언트는 전체와 부분을 구분하지 않고 동일한 인터페이스를 사용할 수 있습니다.

부분과 전체의 계층을 표현하기 위해 객체들을 모아 트리 구조로 구성합니다.

Composite

Composite 패턴을 UML로 도식화하면 아래와 같습니다.



Structure

Component : 집합 관계에 정의될 모든 객체에 대한 공통 인터페이스를 정의합니다.

Leaf : 가장 말단의 객체, 즉 자식이 없는 객체를 나타냅니다. 객체 합성에 가장 기본이 되는 객체의 행동을 정의합니다.

Composite : 자식이 있는 구성 요소에 대한 행동을 정의합니다. 자신이 복합하는 요소들을 저장하면서, Component 인터페이스에 정의된 자식 관련 연산을 구현합니다.

Client : Component 인터페이스를 통해 복합 구조 내의 객체들을 조작합니다.

Implementation

구체적인 구현에 대해서 소스 코드를 통하여 살펴봅니다.

```
protocol Component {
    var parent: Component? { get set }

    func add(component: Component)
    func remove(component: Component)

    func isComposite() -> Bool
    func operation() -> String
}

extension Component {
    func add(component: Component) {}
    func remove(component: Component) {}
    func isComposite() -> Bool {
        return false
    }
}

class Leaf : Component {
    var parent: Component?

    func operation() -> String {
        return "Leaf"
    }
}
```

Implementation

```
class Composite : Component {
    var parent: Component?

    private var children = [Component]()

    func add(component: Component) {
        var item = component
        item.parent = self
        children.append(item)
    }

    func remove(component: Component) {
        if children.count > 0 {
            children.removeLast()
        }
    }

    func isComposite() -> Bool {
        return true
    }

    func operation() -> String {
        /*
            let result = children.map( { (component: Component) -> String in
                return component.operation()
            })
        */
        let result = children.map({ $0.operation() })

        return "Branch(" + result.joined(separator: " ") + ")"
    }
}
```

Implementation

```
let tree = Composite()
print("Tree : " + tree.operation())
// Tree : Branch()

let branch1 = Composite()
branch1.add(component: Leaf())
branch1.add(component: Leaf())

let branch2 = Composite()
branch2.add(component: Leaf())

tree.add(component: branch1)
print("Tree : " + tree.operation())
// Tree : Branch(Branch(Leaf Leaf))

tree.add(component: branch2)
print("Tree : " + tree.operation())
// Tree : Branch(Branch(Leaf Leaf) Branch(Leaf))

tree.add(component: Leaf())
print("Tree : " + tree.operation())
// Tree : Branch(Branch(Leaf Leaf) Branch(Leaf) Leaf)
```

Implementation

```
tree.remove()
print("Tree(remove) : " + tree.operation())
// Tree(remove) : Branch(Branch(Leaf Leaf) Branch(Leaf))

tree.remove()
print("Tree(remove) : " + tree.operation())
// Tree(remove) : Branch(Branch(Leaf Leaf))

tree.remove()
print("Tree(remove) : " + tree.operation())
// Tree(remove) : Branch()
```

References

[1] 컴포지트 패턴(Composite Pattern) : https://jdm.kr/blog/228

[2] Composite in Swift : https://refactoring.guru/design-patterns/composite/swift/example

[3] Swift Solutions: Composite : https://medium.com/swiftcraft/swift-solutions-composite-pattern-fb3f08a30d08

[4] [Design Pattern] 컴포지트(Composite) 패턴 - 디자인 패턴 : https://palpit.tistory.com/194

[5] Design Patterns in Swift: Composite Pattern : https://agostini.tech/2018/06/17/design-patterns-in-swift-composite-pattern/

References

[6] [디자인 패턴] 10. 컴포지트 패턴 (Composite Pattern) : https://itchipmunk.tistory.com/370

[7] 컴포지트 패턴(Composite Pattern) :: 마이구미 : https://mygumi.tistory.com/343

[8] Composite Design Pattern : https://sourcemaking.com/design_patterns/composite

[9] Design Pattern Ch3 in Swift Composite : https://www.slideshare.net/ChihyangLi/design-pattern-ch3-in-swi-composite

[10] Design Pattern 09 - Composite Pattern(복합체 패턴) : https://dev-gloomyfox.tistory.com/m/entry/Design-Pattern-09-Composite-Pattern복합체-패턴?category=799990

Thank you!