SWIFT
Visitor

Bill Kim(김정훈) | ibillkim@gmail.com

목차

Visitor

Visitor(방문자) 패턴은 객체의 구조와 기능을 분리시키는 패턴으로서 구조는 변하지 않으면서 기능을 쉽게 추가하거나 확장되어야 할 경우 사용할 수 있는 행위 관련 패턴입니다.

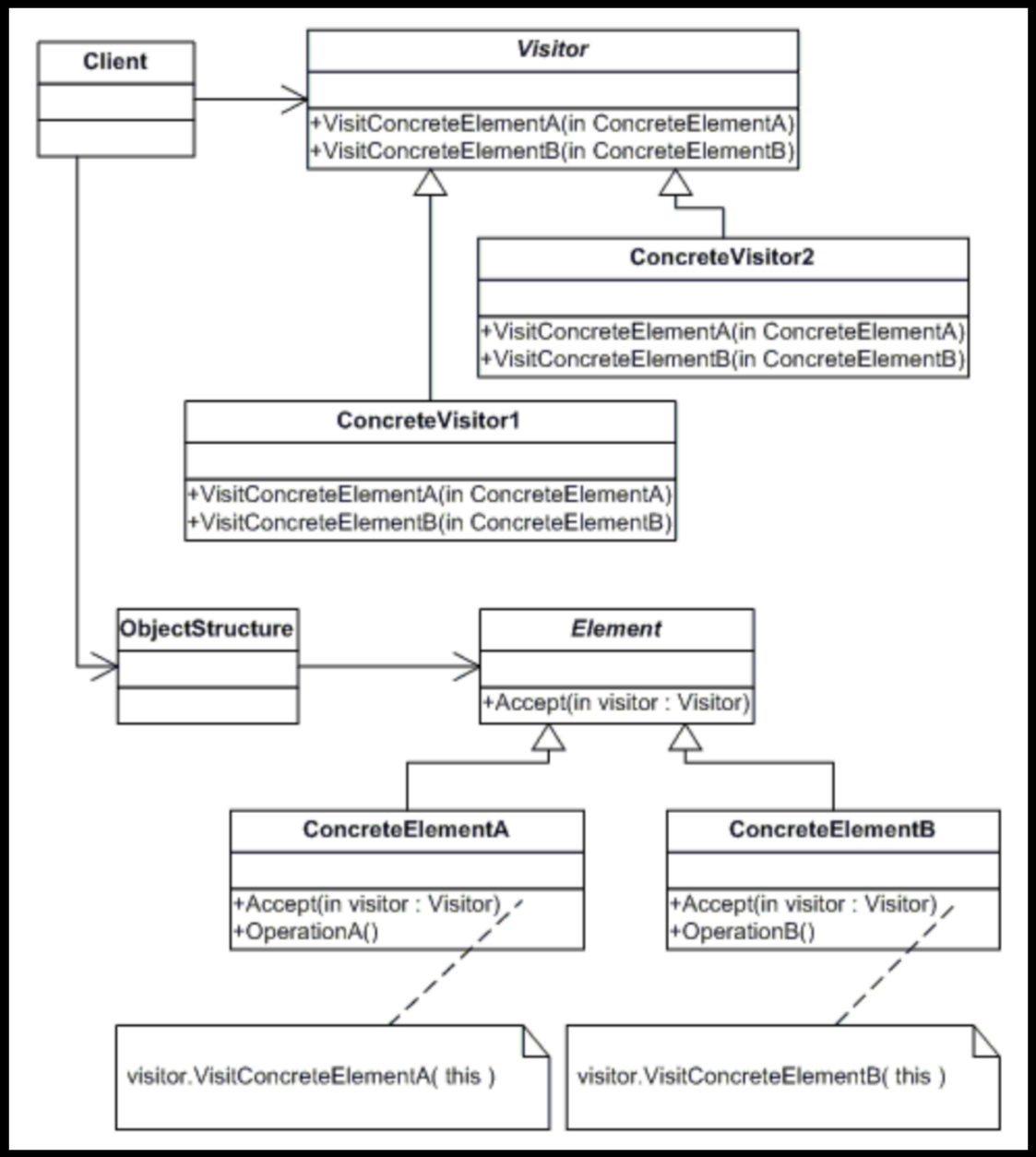
Visitor(방문자) 패턴은 컴포지트 패턴과 연동되어 사용되는 경우가 많습니다.

Visitor(방문자) 패턴의 예로서 파일 탐색기를 구현해볼수 있는데 파일 탐색기는 여기저기 폴더를 돌아다니면서 필요한 파일을 찾아서 특정 저장 공간에 그 파일들이 있는지 저장하였다가 방문이 모두 끝나면 사용자에게 결과를 보여줄 수 있습니다.

바로 탐색기가 여기서 Visitor(방문자) 입니다.

Structure

Visitor 패턴을 UML로 도식화하면 아래와 같습니다.



Structure

Element : Visitor 객체를 수용하여 특정 메소드에 방문할 수 있도록 하는 메소드를 정의하는 추상 클래스 객체

ConcreteElement : Element 객체를 상속받아 방문자가 방문할 수 있는 메소드에 대한 구현을 실제로 하는 객체

Visitor : 각 ConcreteElement 객체에 대해서 방문할 수 있는 방문자 기본 클래스로서 방문을 할 객체들에 대한 기본 인터페이스 메소드에 대한 정의를 하는 객체

ConcreteVisitor : Visitor 객체를 상속받아 방문하게 되는 객체들의 같은 인터페이스 메소드에 대한 행동을 재구현할 수 있게 해주는 서브 클래스 객체

Implementation

구체적인 구현에 대해서 소스 코드를 통하여 살펴봅니다.

```
protocol Element {
    func accept(_ visitor: Visitor)
}

class ConcreteElementA : Element {
    func accept(_ visitor: Visitor) {
        visitor.visitConcreteElementA(element: self)
    }

    func exclusiveMethodOfConcreteElementA() -> String {
        return "A"
    }
}

class ConcreteElementB: Element {
    func accept(_ visitor: Visitor) {
        visitor.visitConcreteElementB(element: self)
    }

    func specialMethodOfConcreteElementB() -> String {
        return "B"
    }
}
```

Implementation

```
protocol Visitor {
    func visitConcreteElementA(element: ConcreteElementA)
    func visitConcreteElementB(element: ConcreteElementB)
}

class ConcreteVisitor1: Visitor {
    func visitConcreteElementA(element: ConcreteElementA) {
        print(element.exclusiveMethodOfConcreteElementA() + " + ConcreteVisitor1\n")
    }

    func visitConcreteElementB(element: ConcreteElementB) {
        print(element.specialMethodOfConcreteElementB() + " + ConcreteVisitor1\n")
    }
}

class ConcreteVisitor2: Visitor {
    func visitConcreteElementA(element: ConcreteElementA) {
        print(element.exclusiveMethodOfConcreteElementA() + " + ConcreteVisitor2\n")
    }

    func visitConcreteElementB(element: ConcreteElementB) {
        print(element.specialMethodOfConcreteElementB() + " + ConcreteVisitor2\n")
    }
}
```

Implementation

```
func visit(elements: [Element], visitor: Visitor) {
    /* for element in elements
    {
        element.accept(visitor)
    }*/

    elements.forEach({ $0.accept(visitor) })
}
```

```
let elements: [Element] = [ConcreteElementA(), ConcreteElementB()]

let visitor1 = ConcreteVisitor1()
visit(elements: elements, visitor: visitor1)
// A + ConcreteVisitor1
// B + ConcreteVisitor1

let visitor2 = ConcreteVisitor2()
visit(elements: elements, visitor: visitor2)
// A + ConcreteVisitor2
// B + ConcreteVisitor2
```

References

[1] [디자인 패턴] 13. 방문자 패턴 (Visitor Pattern) : https://itchipmunk.tistory.com/373

[2] 방문자 패턴(Visitor Pattern) : https://copynull.tistory.com/146

[3] Visitor in Swift : https://refactoring.guru/design-patterns/visitor/swift/example

[4] ⑬ 디자인 패턴(Design Pattern) - Visitor : https://gorakgarak.tistory.com/466

[5] c++ - 방문자 디자인 패턴 및 다중 계층 클래스 계층 : https://www.python2.net/questions-439189.htm

References

[6] [소프트웨어/디자인패턴] 방문자 패턴(Visitor Pattern) : https://sticky32.tistory.com/entry/소프트웨어디자인패턴-방문자-패턴Visitor-Pattern

Thank you!