SWIFT
Chain of Responsibility

Bill Kim(김정훈) | ibillkim@gmail.com

목차

Chain of Responsibility

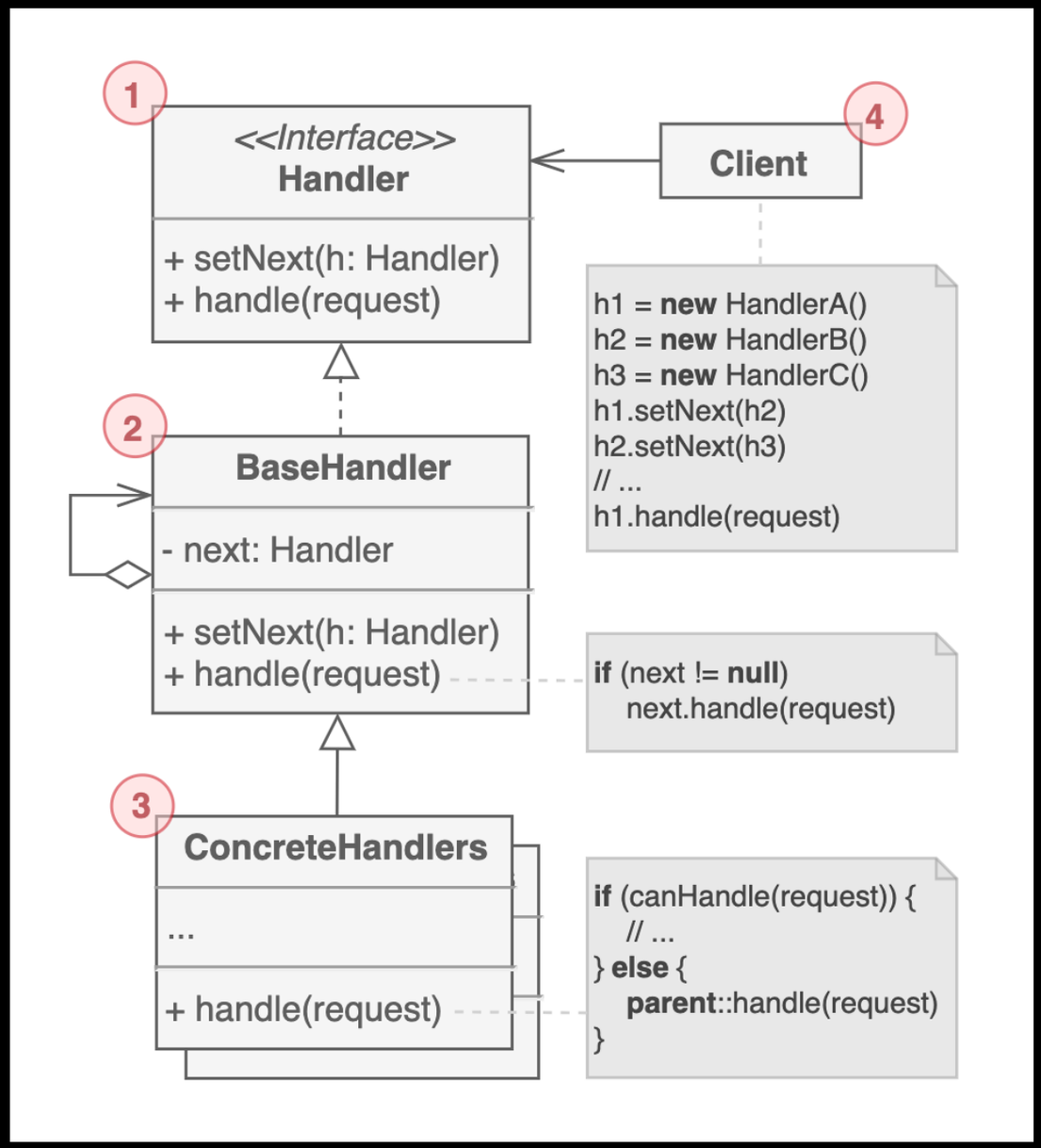
Chain of Responsibility(책임 연쇄 패턴) 패턴은 연속적인 핸들러를 따라 요청을 전달하도록 하는 행동 디자인 패턴입니다.

요청을 받으면 핸들러는 요청을 처리하거나 다음 핸들러로 요청을 전달합니다.

때로는 CoR 혹은 Chain of Command라고도 불리기도 합니다.

Structure

Chain of Responsibility 패턴을 UML로 도식화하면 아래와 같습니다.



Structure

Handler : 기본이 되는 부모 추상 클래스로서 다음 핸들러 및 핸들러에 대한 정의만 구현이 되는 객체

BaseHandler : Handler 객체를 상속받아 핸들러를 가지고 있으면서 다음 핸들러와 다음 핸들러를 호출하는 구현을 가지고 있는 클래스

ConcreteHandler : BaseHandler 객체를 상속 받아서 최종 핸들에 대한 처리를 구현하는 서브 클래스 객체

Implementation

구체적인 구현에 대해서 소스 코드를 통하여 살펴봅니다.

```
protocol Handler : class {
    @discardableResult // 결과를 사용하든 안하든 경고가 안 뜨도록 해주는 지시어
    func setNext(handler: Handler) -> Handler
    func handle(request: String) -> String?
}

class BaseHandler : Handler {
    var nextHandler: Handler?

    func setNext(handler: Handler) -> Handler {
        self.nextHandler = handler
        return handler
    }

    func handle(request: String) -> String? {
        return nextHandler?.handle(request: request)
    }
}
```

Implementation

```
class MonkeyHandler : BaseHandler {
    override func handle(request: String) -> String? {
        if (request == "Banana") {
            return "Monkey : I'll eat the " + request
        } else {
            return nextHandler?.handle(request: request)
        }
    }
}

class DogHandler: BaseHandler {
    override func handle(request: String) -> String? {
        if (request == "MeatBall") {
            return "Dog : I'll eat the " + request
        } else {
            return nextHandler?.handle(request: request)
        }
    }
}

class CatHandler: BaseHandler {
    override func handle(request: String) -> String? {
        if (request == "Fish") {
            return "Cat : I'll eat the " + request
        } else {
            return nextHandler?.handle(request: request)
        }
    }
}
```

Implementation

```
func handler(handler: Handler) {

    let food = ["Nut", "Banana", "Fish"]

    food.forEach({ let result = handler.handle(request: $0)
        if(result == nil) {
            print($0 + " was left untouched.")
        }
        else {
            print(result!)
        }})
}
```

Implementation

```
let monkey = MonkeyHandler()
let dog = DogHandler()
let cat = CatHandler()

monkey.setNext(handler: dog).setNext(handler: cat)

// Chain: Monkey > Dog > Cat
handler(handler: monkey)

// Nut was left untouched.
// Monkey : I'll eat the Banana
// Cat : I'll eat the Fish

// Subchain: Dog > Cat
handler(handler: dog)

// Nut was left untouched.
// Banana was left untouched.
// Cat : I'll eat the Fish
```

References

[1] Chain of Responsibility in Swift : https://refactoring.guru/design-patterns/chain-of-responsibility/swift/example

[2] Design Patterns in Swift: Chain of Responsibility Pattern : https://medium.com/design-patterns-in-swift/design-patterns-in-swift-chain-of-responsibility-pattern-f575c85a43c

[3] Design Patterns in Swift: Chain of Responsibility : https://agostini.tech/2018/05/27/design-patterns-in-swift-chain-of-responsibility/

[4] Chain of responsibility : https://riptutorial.com/swift/example/26459/chain-of-responsibility

[5] 책임 연쇄 패턴 (Chain of Responsibility Pattern in Swift) : https://jerome.kr/entry/chain-of-responsibility-pattern

References

[6] Design Patterns in Swift: Chain of Responsibility : https://viblo.asia/p/design-patterns-in-swift-chain-of-responsibility-WAyK8V9nlxX

[7] A Design Pattern Story in Swift - Chapter 14: Chain Of Responsibility : http://audreyli.me/2015/07/13/a-design-pattern-story-in-swift-chapter-14-chain-of-responsibility/

[8] 책임연쇄패턴 : Chain of Responsibility : https://eastroot1590.tistory.com/entry/책임연쇄패턴-Chain-of-Responsibility

[9] Design Patterns in Swift: Chain of Responsibility : https://codereview.stackexchange.com/questions/125723/design-patterns-in-swift-chain-of-responsibility

[10] 책임 연쇄 패턴 : https://ko.wikipedia.org/wiki/책임_연쇄_패턴

Thank you!