SWIFT
Command

Bill Kim(김정훈) | ibillkim@gmail.com

목차

Command

Command 패턴은 실행될 기능을 캡슐화함으로써 주어진 여러 기능을 실행할 수 있는 재사용성이 높은 클래스를 설계하는 행위(Behavior) 패턴입니다.

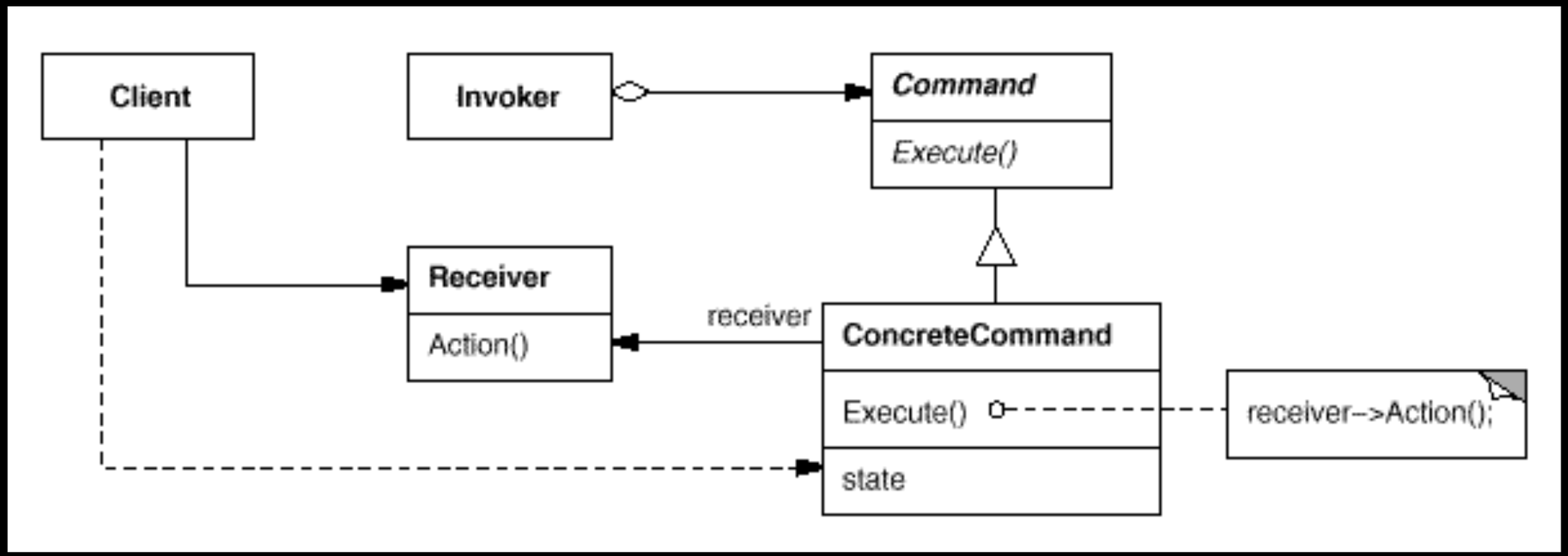
즉, 이벤트가 발생했을 때 실행될 기능이 다양하면서도 변경이 필요한 경우에 이벤트를 발생시키는 클래스를 변경하지 않고 재사용하고자 할 때 유용합니다.

실행될 기능을 캡슐화함으로써 기능의 실행을 요구하는 호출자(Invoker) 클래스와 실제 기능을 실행하는 수신자(Receiver) 클래스 사이의 의존성을 제거합니다.

따라서 실행될 기능의 변경에도 호출자 클래스를 수정 없이 그대로 사용 할 수 있도록 해줍니다.

Structure

Command 패턴을 UML로 도식화하면 아래와 같습니다.



Structure

Command : 실행될 기능에 대한 인터페이스를 선언하는 객체

ConcreteCommand : 실제로 실행되는 인터페이스를 구현하는 객체

Invoker : 기능의 실행을 요청하는 호출자 클래스 객체

Receiver : ConcreteCommand 객체에서 실행할 메서드를 구현할 때 필요한 클래스 객체, ConcreteCommand 객체의 기능을 실행하기 위해서 사용하는 수신자 클래스 객체

Implementation

구체적인 구현에 대해서 소스 코드를 통하여 살펴봅니다.

```
protocol Command {
    func execute()
}

class SimpleCommand: Command {
    private var payload: String

    init(_ payload: String) {
        self.payload = payload
    }

    func execute() {
        print("SimpleCommand: See, I can do simple things like printing (" +
payload + ")")
    }
}
```

Implementation

```
class ComplexCommand: Command {
    private var receiver: Receiver

    private var a: String
    private var b: String

    init(_ receiver: Receiver, _ a: String, _ b: String) {
        self.receiver = receiver
        self.a = a
        self.b = b
    }

    func execute() {
        print("ComplexCommand: Complex stuff should be done by a receiver
object.\n")
        receiver.doSomething(a)
        receiver.doSomethingElse(b)
    }
}
```

Implementation

```
class Receiver {
    func doSomething(_ a: String) {
        print("Receiver: Working on (" + a + ")\n")
    }

    func doSomethingElse(_ b: String) {
        print("Receiver: Also working on (" + b + ")\n")
    }
}
```

```
class Invoker {
    private var onStart: Command?
    private var onFinish: Command?

    func setOnStart(_ command: Command) {
        onStart = command
    }

    func setOnFinish(_ command: Command) {
        onFinish = command
    }

    func doSomethingImportant() {
        onStart?.execute()
        onFinish?.execute()
    }
}
```

Implementation

```
let invoker = Invoker()
invoker.setOnStart(SimpleCommand("Say Hi!"))

let receiver = Receiver()
invoker.setOnFinish(ComplexCommand(receiver, "Send email", "Save report"))

// 호출자의 실행 함수를 수정을 할 필요없이 ConcreteCommand 내의 receiver 객체를 통하여 다양한 행위를 할 수 있다.
invoker.doSomethingImportant()
```

```
// SimpleCommand: See, I can do simple things like printing (Say Hi!)
// ComplexCommand: Complex stuff should be done by a receiver object.
// Receiver: Working on (Send email)
// Receiver: Also working on (Save report)
```

References

[1] [Swift-Design Pattern] 커맨드 (Command pattern) : http://throughkim.kr/2019/09/06/swift-command/

[2] [Design Pattern] 커맨드 패턴이란 : https://gmlwjd9405.github.io/2018/07/07/command-pattern.html

[3] Design Patterns in Swift: Command Pattern : https://agostini.tech/2018/06/03/design-patterns-in-swift-command-pattern/

[4] Rethinking Design Patterns in Swift: https://khawerkhaliq.com/blog/swift-design-patterns-command-pattern/

[5] Design Patterns in Swift: Command Pattern : https://medium.com/design-patterns-in-swift/design-patterns-in-swift-command-pattern-b95a1f4bbc45

References

[6] Swift World: Design Patterns — Command : https://medium.com/swiftworld/swift-world-design-patterns-command-cc9c56544bf0

[7] [Swift] 커맨드(Command) 패턴 : https://m.blog.naver.com/PostView.nhn?blogId=horajjan&logNo=220804709260&proxyReferer=https:%2F%2Fwww.google.com%2F

[8] Swift command design pattern : https://theswiftdev.com/swift-command-design-pattern/

[9] Command in Swift : https://refactoring.guru/design-patterns/command/swift/example

[10] [Design Pattern] 커멘드(Command) 패턴 - 디자인 패턴 : https://palpit.tistory.com/204

Thank you!