SWIFT
Observer

Bill Kim(김정훈) | ibillkim@gmail.com

목차

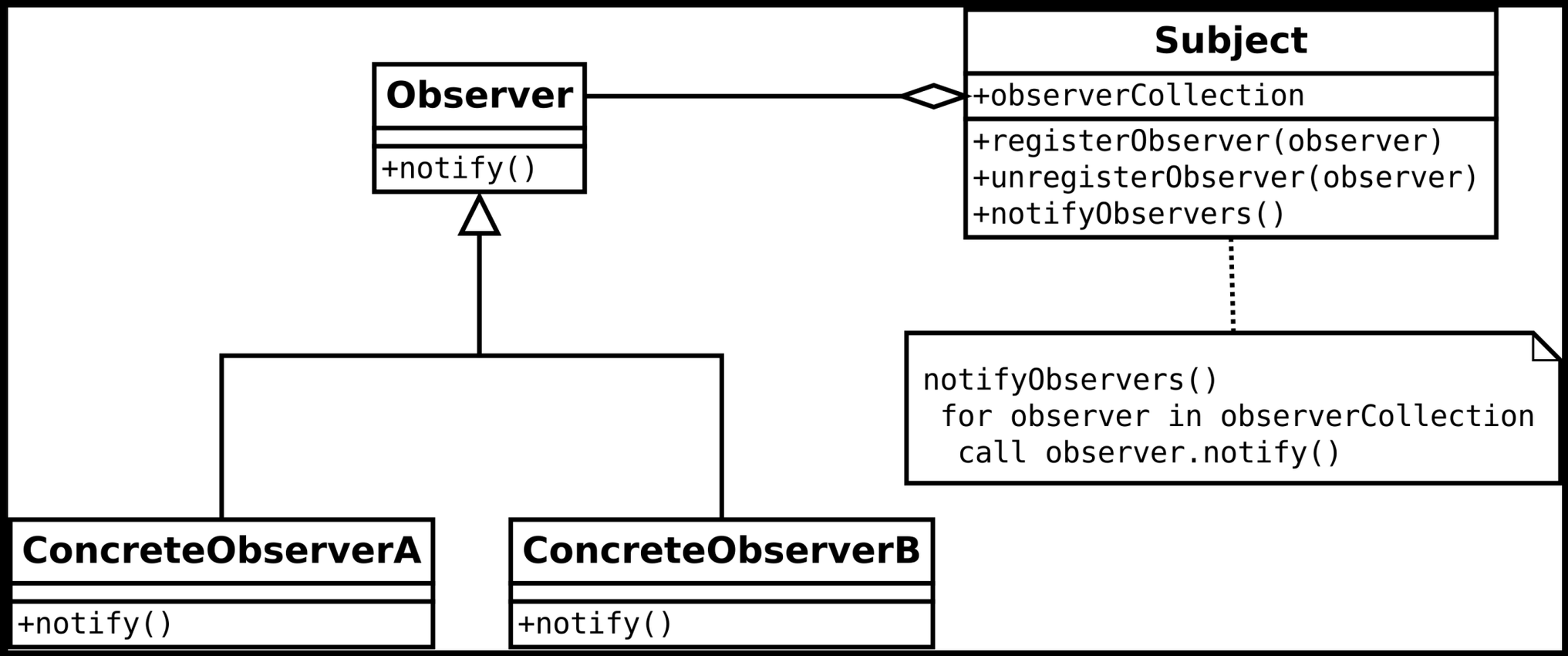
Observer

Observer(옵저버) 패턴은 특정 객체에서 발생하는 이벤트를 구독자에게 전달하는 패턴입니다.

Observer(옵저버) 패턴은 한 객체의 상태 변화에 따라 다른 객체의 상태도 연동 되도록 1 대 n 객체간 의존 관계를 구성하는 디자인 패턴입니다.

Structure

Observer 패턴을 UML로 도식화하면 아래와 같습니다.



Structure

Subject : 구독자(ConcreteObserver)가 이벤트를 받을 주요 주제 객체, 해당 Subject의 특정 이벤트 변경 시 구독하고 있는 구독자들이 해당 이벤트를 수신한다.

Observer : 구독자들의 부모가 되는 추상 클래스 객체

ConcreteObserver : Subject의 이벤트를 받게되는 구독자 객체

Implementation

구체적인 구현에 대해서 소스 코드를 통하여 살펴봅니다.

```
// 구독할 주제 객체
class Subject {
    var state: Int = { return Int(arc4random_uniform(10)) }()
    private lazy var observers = [Observer]()

    func attach(_ observer: Observer) {
        print("Subject: Attached an observer.\n")
        observers.append(observer)
    }

    func detach(_ observer: Observer) {
        if let idx = observers.index(where: { $0 === observer }) {
            observers.remove(at: idx)
            print("Subject: Detached an observer.\n")
        }
    }

    func notify() {
        print("Subject: Notifying observers...\n")
        observers.forEach({ $0.update(subject: self)})
    }

    func updateState() {
        state = Int(arc4random_uniform(10))
        notify()
    }
}
```

Implementation

```swift
// 구독자 부모 추상 객체
protocol Observer : class {
    func update(subject: ObserverSubject)
}

// 실제 구독을 하는 객체
class ConcreteObserverA : Observer {
    func update(subject: ObserverSubject) {
        print("ConcreteObserverA - updated a state : \(subject.state)\n")
    }
}

// 실제 구독을 하는 객체
class ConcreteObserverB : Observer {
    func update(subject: ObserverSubject) {
        print("ConcreteObserverB - updated a state : \(subject.state)\n")
    }
}
```

Implementation

```
let subject = ObserverSubject()

let observer1 = ConcreteObserverA()
let observer2 = ConcreteObserverB()

subject.attach(observer1) // Subject: Attached an observer.
subject.attach(observer2) // Subject: Attached an observer.

subject.updateState()
// ConcreteObserverA - updated a state : 6
// ConcreteObserverB - updated a state : 6

subject.updateState()
// ConcreteObserverA - updated a state : 1
// ConcreteObserverB - updated a state : 1

subject.detach(observer2) // Subject: Detached an observer.

subject.updateState()
// ConcreteObserverA - updated a state : 9
```

References

[1] [Swift-Design Pattern] 옵저버 패턴(Observer pattern) : http://throughkim.kr/2019/09/05/swift-observer/

[2] Observer Pattern(swift) : https://linsaeng.tistory.com/6

[3] Observer Pattern in Swift : https://magicmon.github.io/2017/07/04/Observer-Pattern/

[4] Ch 8. Oberver Pattern : https://rhammer.tistory.com/348

[5] The observer pattern in Swift : https://medium.com/swift-coding/the-observer-pattern-in-swift-97a0e6fafa58

References

[6] Swift Design Pattern: Observer Pattern : https://medium.com/99ridho/swift-design-pattern-observer-pattern-fc009b783d19

[7] 굿바이~ 옵저버 패턴 and FRP : https://hamait.tistory.com/885

[8] Observer in Swift : https://refactoring.guru/design-patterns/observer/swift/example

[9] Design pattern in Swift - Observers : https://benoitpasquier.com/observer-design-pattern-swift/

[10] 옵저버 패턴 (Observer Pattern in Swift) : https://jerome.kr/entry/observer-pattern

Thank you!