# SWIFT
# Mediator

Bill Kim(김정훈) | ibillkim@gmail.com

# 목차

# Mediator

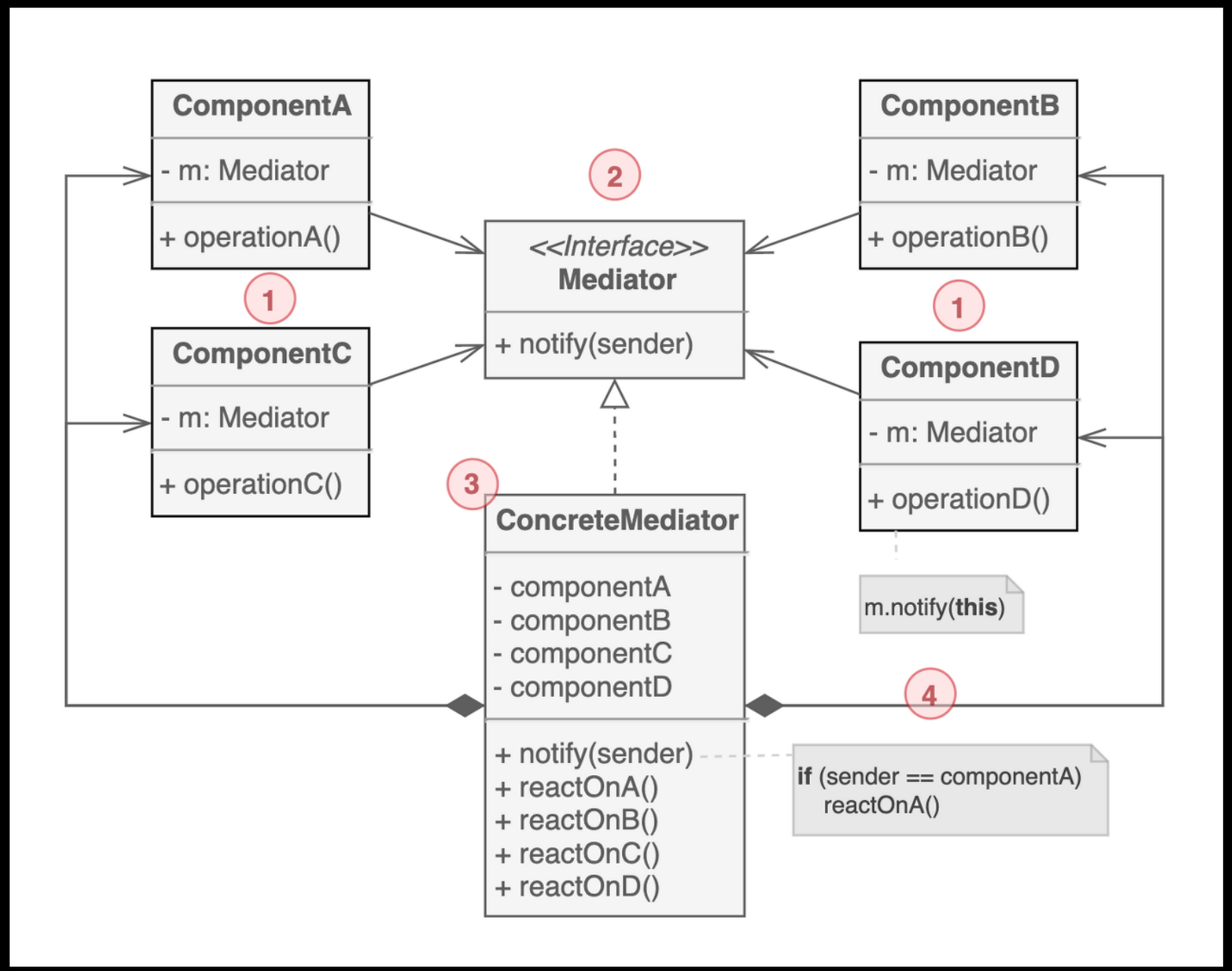
Mediator(미디에이터) 패턴은 복잡한 의존 관계를 줄이고자 할 때 유용한 행동 디자인 패턴입니다.

모든 클래스간의 복잡한 로직(상호작용)을 캡슐화하여 하나의 클래스에 위임하여 처리하는 패턴으로서 비슷한 패턴으로는 Facade 패턴과 Observer 패턴 등이 있습니다.

결론적으로 커뮤니케이션을 하고자 하는 객체가 있을 때 서로가 커뮤니케이션 하기 복잡한 경우 이를 해결해주고 서로 간 쉽게 해주며 커플링을 약화시켜주는 패턴입니다.

# Structure

Mediator 패턴을 UML로 도식화하면 아래와 같습니다.



# Structure

Mediator : 여러 Component 중재해주는 인터페이스를 가지고 있는 추상 클래스 객체

ConcreteMediator : Component 객체들을 가지고 있으면서 중재해주는 역할을 하는 객체

Component : Mediator 객체에 의해서 관리 및 중재를 받을 기본 클래스 객체들

# Implementation

구체적인 구현에 대해서 소스 코드를 통하여 살펴봅니다.

```
protocol Mediator : AnyObject {
    func notify(sender: BaseComponent, event: String)
}

class ConcreteMediator : Mediator {
    private var component1: Component1
    private var component2: Component2

    init(_ component1: Component1, _ component2: Component2) {
        self.component1 = component1
        self.component2 = component2

        component1.update(mediator: self)
        component2.update(mediator: self)
    }

    func notify(sender: BaseComponent, event: String) {
        if event == "A" {
            self.component2.operationC()
        }
        else if (event == "D") {
            self.component1.operationB()
            self.component2.operationC()
        }
    }
}
```

# Implementation

```
class BaseComponent {
    fileprivate weak var mediator: Mediator?

    init(mediator: Mediator? = nil) {
        self.mediator = mediator
    }

    func update(mediator: Mediator) {
        self.mediator = mediator
    }
}

class Component1 : BaseComponent {
    func operationA() {
        print("operationA")
        mediator?.notify(sender: self, event: "A")
    }

    func operationB() {
        print("operationB")
        mediator?.notify(sender: self, event: "B")
    }
}

class Component2 : BaseComponent {
    func operationC() {
        print("operationC")
        mediator?.notify(sender: self, event: "C")
    }

    func operationD() {
        print("operationD")
        mediator?.notify(sender: self, event: "D")
    }
}
```

# Implementation

```
let component1 = Component1()
let component2 = Component2()

let mediator = ConcreteMediator(component1, component2)

component1.operationA()
// operationA
// operationC
```

```
component2.operationD()
// operationD
// operationB
// operationC
```

# References


[1] Mediator in Swift : https://refactoring.guru/design-patterns/mediator/swift/example

[2] Swift World: Design Patterns — Mediator : https://medium.com/swiftworld/swift-world-design-patterns-mediator-e6b3c35d68b0

[3] [Design Pattern] 중재자(Mediator) 패턴 - 디자인 패턴 : https://palpit.tistory.com/201

[4] Mediator Pattern Case Study : https://www.vadimbulavin.com/mediator-pattern-case-study/

[5] 중재자 패턴 (Mediator Pattern in Swift) : https://jerome.kr/entry/mediator-pattern

# References


[6] Mediator Pattern in Swift : https://coding.tabasoft.it/ios/mediator-pattern-in-swift/

[7] 중재자 패턴(Mediator Pattern) : https://www.crocus.co.kr/1542

[8] Design Patterns in Swift: Mediator : https://codereview.stackexchange.com/questions/125725/design-patterns-in-swift-mediator

[9] [디자인 패턴] 12. 중재자 패턴 ( Mediator Pattern ) : https://itchipmunk.tistory.com/372

[10] [자바 디자인 패턴 이해] 16강 중재자 패턴 (Mediator) : https://www.youtube.com/watch?v=7imEWnkVFFg

Thank you!