# SWIFT
# State

Bill Kim(김정훈) | ibillkim@gmail.com

# 목차

# iterator

State(상태) 패턴은 객체 내부의 상태에 따라서 객체가 다른 행동을 할 수 있게 해주는 패턴입니다.

즉 객체의 상태만 변경해주어도 다른 행동을 하는 객체로 만들어줍니다.

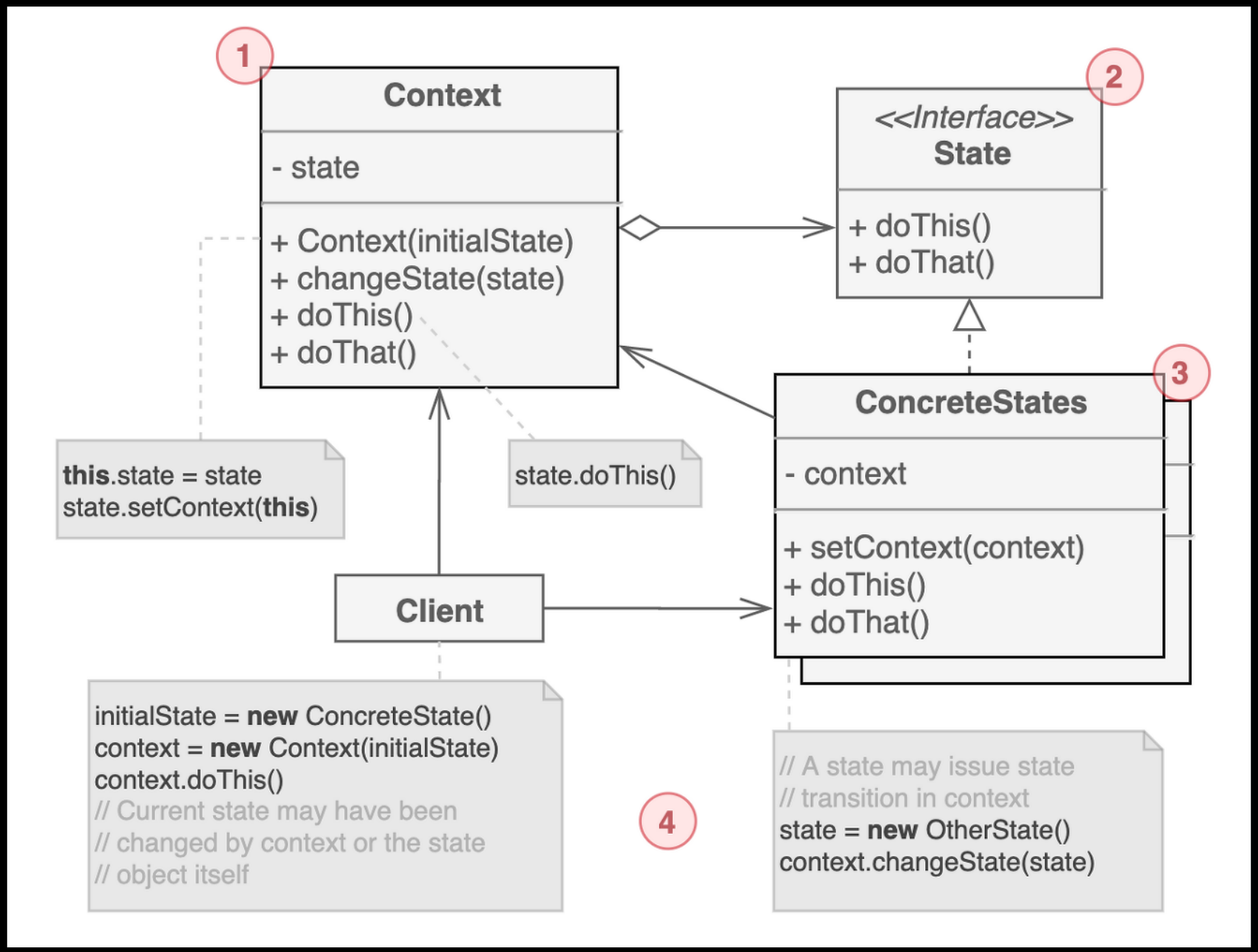
상태 패턴은 유한 상태 머신과도 밀접한 관련이 있습니다.

다만 조건문에 기반한 상태 머신의 단점은, 상태의 수와 상태에 따른 동작이 추가될 때에 크게 드러납니다.

대부분의 메서드에서 굉장히 지저분한 조건문이 들어가게 됩니다.
하지만 상태 패턴은 가능한 모든 상태에 대해서 클래스로 추출하는 방법을 사용합니다.

# Structure

State 패턴을 UML로 도식화하면 아래와 같습니다.



# Structure

Context : 하나의 상태를 갖고 모든 동작은 State 객체에 위임하는 객체입니다.

State : State 객체는 상태에 따른 필요한 기능(메서드)를 선언하는 추상 클래스 객체입니다.

ConcreteState : State 객체를 상속받은 실제 클래스로서 상태에 따른 기능을 실제로 구현하는 객체입니다. 해당 객체의 기능을 통하여 전환하여야 하는 상태를 정할 수 있습니다.

# Implementation

구체적인 구현에 대해서 소스 코드를 통하여 살펴봅니다.

```
class Context {
    private var state: State

    init(_ state: State) {
        self.state = state
        changeState(state: state)
    }

    func changeState(state: State) {
        //print("change state to " + String(describing: state))
        self.state = state
        self.state.update(context: self)
    }

    func request1() {
        state.handle1()
    }

    func request2() {
        state.handle2()
    }
}

protocol State : class {
    func update(context: Context)

    func handle1()
    func handle2()
}
```

# Implementation

```
class ConcreteStateA : State {
    private(set) weak var context: Context?

    func update(context: Context) {
        self.context = context
    }

    func handle1() {
        print("ConcreteStateA handles request1.")
        context?.changeState(state: ConcreteStateB())
    }

    func handle2() {
        print("ConcreteStateA handles request2.\n")
    }
}

class ConcreteStateB : State {
    private(set) weak var context: Context?

    func update(context: Context) {
        self.context = context
    }

    func handle1() {
        print("ConcreteStateB handles request1.\n")
    }

    func handle2() {
        print("ConcreteStateB handles request2.")
        context?.changeState(state: ConcreteStateA())
    }
}
```

# Implementation

```
let context = Context(ConcreteStateA())

context.request1() // ConcreteStateA handles request1.
context.request1() // ConcreteStateB handles request1.

context.request2() // ConcreteStateB handles request2.
context.request2() // ConcreteStateA handles request2.
```

# References


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

[2] Design Patterns in Swift: State Pattern : https://agostini.tech/2018/05/13/design-patterns-in-swift-state/

[3] Rethinking Design Patterns in Swift: State Pattern : https://khawerkhaliq.com/blog/swift-design-patterns-state-pattern/

[4] Applying the State Pattern in Swift : https://medium.com/@vel.is.lava/applying-the-state-pattern-in-swift-5fed17751239

[5] 상태 패턴 (State Pattern in Swift) : https://jerome.kr/entry/state-pattern

# References


[6] Modelling state in Swift : https://www.swiftbysundell.com/articles/modelling-state-in-swift/

[7] Design Patterns in Swift: State Pattern : https://viblo.asia/p/design-patterns-in-swift-state-pattern-gDVK2jkwKLj

[8] [디자인 패턴] 11. 스테이트 패턴 ( State Pattern ) : https://itchipmunk.tistory.com/371

[9] The State Design Pattern vs. State Machine : https://www.codeproject.com/articles/509234/the-state-design-pattern-vs-state-machine

[10] [Design Pattern] 상태(State) 패턴 - 디자인 패턴 : https://palpit.tistory.com/203

Thank you!