SWIFT
Iterator

Bill Kim(김정훈) | ibillkim@gmail.com

목차

iterator

Iterator(반복자) 패턴은 데이터의 타입(리스트, 스택, 트리 등)을 드러내지 않고 컬렉션 요소를 순회할 수 있는 행동 디자인 패턴입니다.

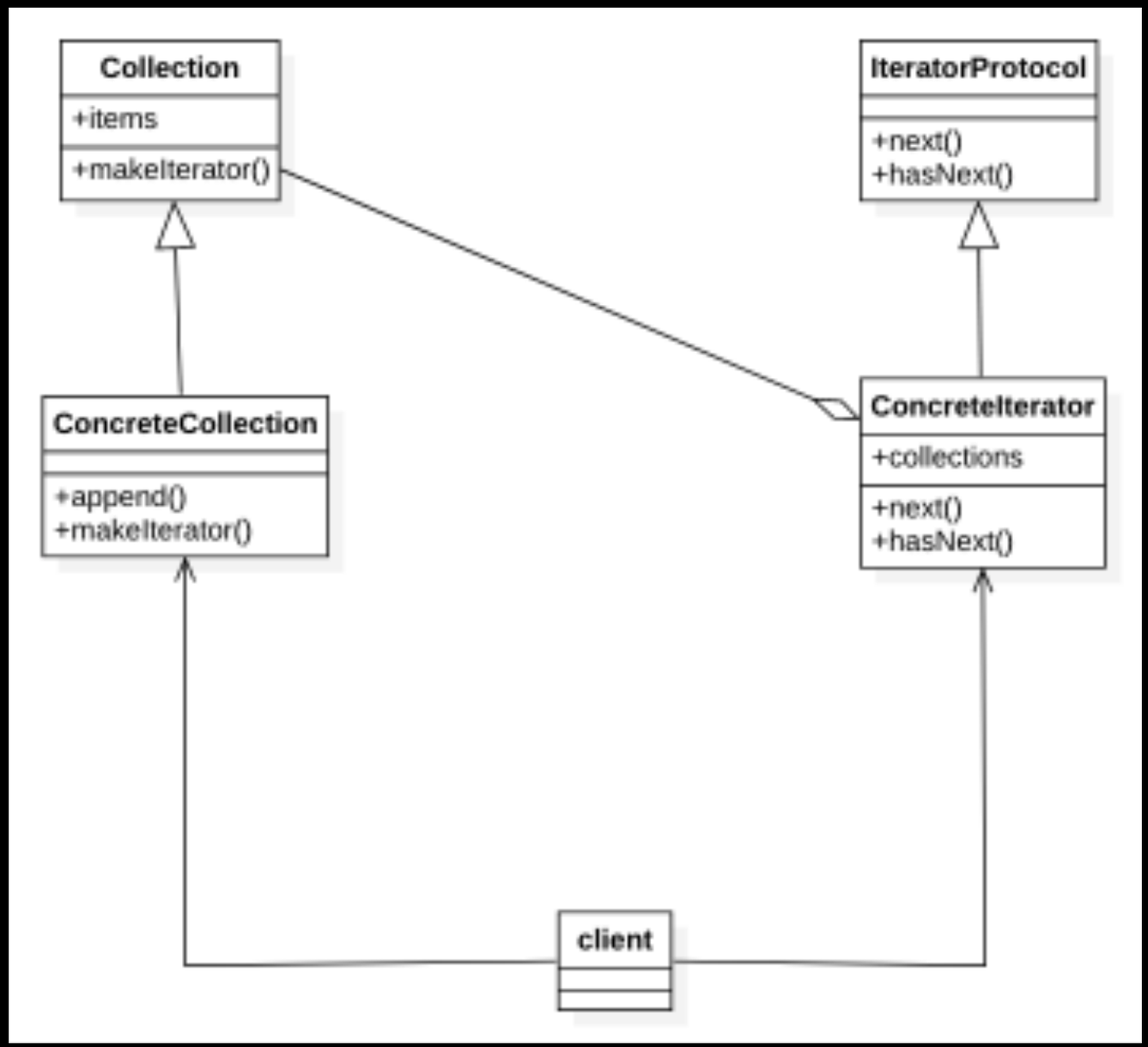
Iterator(반복자) 패턴의 주요 아이디어는 컬렉션의 순회 동작을 별도의 반복자 객체로 분리하는 것입니다.

반복자 객체는 알고리즘을 구현할뿐만 아니라 현재 위치나 남은 요소 등 모든 순회 세부 사항을 캡슐화합니다.

이 덕분에 여러 반복자가 동일한 컬렉션을 서로 독립적으로 순회할 수 있게 되는 것입니다.

Structure

Iterator 패턴을 UML로 도식화하면 아래와 같습니다.



Structure

Collection : 실제 데이터를 가지고 있는 추상 클래스 객체

ConcreteCollection : Collection 객체를 상속받아 데이터를 관리하고 반복자를 생성할 수 있는 객체

Iterator : 기본 반복자 기능 인터페이스를 가지는 추상 클래스 객체

ConcreteIterator : Iterator 객체를 상속받아 Collection 객체들을 관리하며 반복자 관련 기능을 수행하는 객체

Implementation

구체적인 구현에 대해서 소스 코드를 통하여 살펴봅니다.

```
protocol Collection
{
    var items:[Any] { get set }
    func makeIterator() -> ConcreteIterator
}

class ConcreteCollectionA : Collection {
    var items = [Any]()

    func append(_ item: String) {
        self.items.append(item)
    }

    func makeIterator() -> ConcreteIterator {
        return ConcreteIterator(self)
    }
}

class ConcreteCollectionB : Collection {
    var items = [Any]()

    func append(_ item: Int) {
        self.items.append(item)
    }

    func makeIterator() -> ConcreteIterator {
        return ConcreteIterator(self)
    }
}
```

Implementation

```
protocol Iterator {
    func next() -> Any?
    func hasNext() -> Bool
}

class ConcreteIterator : Iterator {
    private let collections : Collection
    private var index = 0

    init(_ collections: Collection) {
        self.collections = collections
    }

    func next() -> Any? {
        defer { index += 1 }
        return index < collections.items.count ? collections.items[index] : nil
    }

    func hasNext() -> Bool {
        if index < collections.items.count {
            return true
        }
        else {
            return false
        }
    }
}
```

Implementation

```
let words = ConcreteCollectionA()
words.append("First")
words.append("Second")
words.append("Third")

let numbers = ConcreteCollectionB()
numbers.append(1)
numbers.append(2)
numbers.append(3)
```

```
let iterator1 = words.makeIterator()

while (iterator1.hasNext()) {
    print(iterator1.next()!)
    // First
    // Second
    // Third
}

let iterator2 = numbers.makeIterator()

while (iterator2.hasNext()) {
    print(iterator2.next()!)
    // 1
    // 2
    // 3
}
```

Implementation

```
let iterator3 = ConcreteIterator(words)

while (true) {
    guard let collection = iterator3.next() else { break }
    print(collection)
    // First
    // Second
    // Third
}
```

```
let iterator4 = ConcreteIterator(numbers)

while (true) {
    guard let collection = iterator4.next() else { break }
    print(collection)
    // 1
    // 2
    // 3
}
```

References

[1] [Swift]Iterator 패턴 구현하기 : http://minsone.github.io/mac/ios/swift-sequence

[2] 이터레이터 패턴 : http://www.incodom.kr/이터레이터_패턴

[3] Iterator Design Pattern in Swift Universe : https://medium.com/@lazarevzubov/iterator-design-pattern-in-swift-universe-34accb6fafd6

[4] Iterator in Swift : https://refactoring.guru/design-patterns/iterator/swift/example

[5] Design Patterns in Swift: Iterator Pattern : https://agostini.tech/2018/06/10/design-patterns-in-swift-iterator-pattern/

References

[6] Iterator design pattern in Swift : https://theswiftdev.com/iterator-design-pattern-in-swift/

[7] Design Patterns in Swift: Iterator Pattern : https://viblo.asia/p/design-patterns-in-swift-iterator-pattern-6J3ZgmBA5mB

[8] 반복자 패턴 (Iterator Pattern in Swift) : https://jerome.kr/entry/iterator-pattern

[9] [Design Pattern] 반복자(Iterator) 패턴 - 디자인 패턴 : https://palpit.tistory.com/200

[10] Intermediate Design Patterns in Swift : https://www.raywenderlich.com/2102-intermediate-design-patterns-in-swift

Thank you!