SWIFT
Protocol (1/2)

Bill Kim(김정훈) | ibillkim@gmail.com

목차

Protocols

프로토콜은 특정 기능 수행에 필수적인 요수를 청의한 청사진(blueprint)입니다.

프로토콜을 만족시키는 타입을 프로토콜을 따른다(conform)고 말합니다.

프로토콜에 필수 구현을 추가하거나 추가적인 기능을 더하기 위해 프로토콜을 확장(extend)하는 것이 가능합니다.

Protocol Syntax

프로토콜의 정의는 클래스, 구조체, 열거형 등과 유사합니다.

```
protocol SomeProtocol {
    // protocol definition goes here
}

struct SomeStructure: SomeProtocol {
    // structure definition goes here
}

class SomeClass: SomeProtocol {
    // class definition goes here
}
```

Property Requirements

프로토콜에서는 프로퍼티가 저장된 프로퍼티인지 계산된 프로퍼티인지 명시하지 않습니다.

하지만 프로퍼티의 이름과 타입 그리고 gettable, settable한지는 명시합니다. 필수 프로퍼티는 항상 var로 선언해야 합니다.

```
protocol SomeProtocol {
    // 프로퍼티의 이름과 타입 그리고 gettable, settable한지는 명시합니다.
    var mustBeSettable: Int { get set }
    var doesNotNeedToBeSettable: Int { get }
}

protocol AnotherProtocol {
    // 타입 프로퍼티는 static 키워드를 적어 선언합니다.
    static var someTypeProperty: Int { get set }
}
```

Property Requirements

```
protocol FullyNamed {
    // 하나의 프로퍼티를 갖는 프로토콜을 선언합니다.
    var fullName: String { get }
}

struct Person: FullyNamed {
    var fullName: String
}

let john = Person(fullName: "John Appleseed")
// john.fullName is "John Appleseed"

class Starship: FullyNamed {
    var prefix: String?
    var name: String
    init(name: String, prefix: String? = nil) {
        self.name = name
        self.prefix = prefix
    }
    var fullName: String {
        return (prefix != nil ? prefix! + " " : "") + name
    }
}

// 계산된 프로퍼티로 사용될 수 있습니다.
var ncc1701 = Starship(name: "Enterprise", prefix: "USS")
// ncc1701.fullName is "USS Enterprise"
```

Method Requirements

익스텐션을 이용해 존재하는 타입에 새로운 이니셜라이저를 추가 할 수 있습니다.

이 방법으로 커스텀 타입의 이니셜라이저 파라미터를 넣을 수 있도록 변경하거나 원래 구현에서 포함하지 않는 초기화 정보를 추가할 수 있습니다.

Method Requirements

```
protocol SomeProtocol {
    static func someTypeMethod()
}

protocol RandomNumberGenerator {
    func random() -> Double
}

class LinearCongruentialGenerator: RandomNumberGenerator {
    var lastRandom = 42.0
    let m = 139968.0
    let a = 3877.0
    let c = 29573.0
    func random() -> Double {
        lastRandom = ((lastRandom * a + c).truncatingRemainder(dividingBy:m))
        return lastRandom / m
    }
}

let generator = LinearCongruentialGenerator()

print("Here's a random number: \(generator.random())")
// Prints "Here's a random number: 0.3746499199817101”

print("And another one: \(generator.random())")
// Prints "And another one: 0.7290237776863283"
```

Method Requirements

mutating 키워드를 사용해 인스턴스에서 변경 가능하다는 것을 표시할 수 있습니다.

이 mutating 키워드는 값타입 형에만 사용합니다.

```
protocol Togglable {
    mutating func toggle()
}

enum OnOffSwitch: Togglable {
    case off, on
    mutating func toggle() {
        switch self {
        case .off:
            self = .on
        case .on:
            self = .off
        }
    }
}

var lightSwitch = OnOffSwitch.off

lightSwitch.toggle()
print(lightSwitch) // lightSwitch is now equal to .on
```

Initializer Requirements

프로토콜에서 필수로 구현해야하는 이니셜라이저를 지정할 수 있습니다.

```
protocol SomeProtocol {
    init(someParameter: Int)
}

class SomeClass: SomeProtocol {
    // 클래스에서 프로토콜 필수 이니셜라이저의 구현
    // 프로토콜에서 특정 이니셜라이저가 필요하다고 명시했기 때문에 구현에서
    // 해당 이니셜라이저에 required 키워드를 붙여줘야 합니다.
    // 클래스 타입에서 final로 선언된 것에는 required를 표시하지 않도 됩니다.
    // final로 선언되면 서브클래싱 되지 않기 때문입니다.

    required init(someParameter: Int) {
        // initializer implementation goes here
    }
}
```

Initializer Requirements

```swift
protocol SomeProtocol {
    init()
}

class SomeSuperClass {
    init() {
        // initializer implementation goes here
    }
}
```

```swift
class SomeSubClass: SomeSuperClass, SomeProtocol {
    // 특정 프로토콜의 필수 이니셜라이저를 구현하고, 수퍼클래스의 이니셜라이저를 서브클래싱하는 경우
    // 이니셜라이저 앞에 required 키워드와 override 키워드를 적어줍니다.
    required override init() {
        // initializer implementation goes here
    }
}
```

Protocols as Types

프로토콜도 하나의 타입으로 사용 가능합니다. 그렇기 때문에 다음과 같이 타입 사용이 허용되는 모든 곳에 프로토콜을 사용할 수 있습니다.

- 함수, 메소드, 이니셜라이저의 파라미터 타입 혹은 리턴 타입
- 상수, 변수, 프로퍼티의 타입
- 컨테이너인 배열, 사전 등의 아이템 타입

Protocols as Types

```swift
protocol RandomNumberGenerator {
    func random() -> Double
}

class LinearCongruentialGenerator: RandomNumberGenerator {
    var lastRandom = 42.0
    let m = 139968.0
    let a = 3877.0
    let c = 29573.0

    func random() -> Double {
        lastRandom = ((lastRandom * a + c).truncatingRemainder(dividingBy:m))
        return lastRandom / m
    }
}

class Dice {
    let sides: Int
    let generator: RandomNumberGenerator

    init(sides: Int, generator: RandomNumberGenerator) {
        self.sides = sides
        self.generator = generator
    }

    func roll() -> Int {
        return Int(generator.random() * Double(sides)) + 1
    }
}


var d6 = Dice(sides: 6, generator: LinearCongruentialGenerator())
    for _ in 1...5 {
        rint("Random dice roll is \(d6.roll())")
    }
    // Random dice roll is 3, Random dice roll is 5, Random dice roll is 4
    // Random dice roll is 5, Random dice roll is 4
```

References

[1] [Swift]Protocols 정리 : http://minsone.github.io/mac/ios/swift-protocols-summary

[2] Protocols : https://docs.swift.org/swift-book/LanguageGuide/Protocols.html

[3] Swift) Protocols (4) : https://zeddios.tistory.com/347

[4] Swift Protocol 적재적소에 사용하기 : https://academy.realm.io/kr/posts/understanding-swift-protocol/

[5] Swift 4.2 Protocol 공식 문서 정리 : https://medium.com/@kimtaesoo188/swift-4-2-protocol-공식-문서-정리-f3a97c6f8cc2

References

[6] 프로토콜 (Protocols) : https://jusung.gitbook.io/the-swift-language-guide/language-guide/21-protocols

[7] 오늘의 Swift 상식 (Protocol) : https://medium.com/@jgj455/오늘의-swift-상식-protocol-f18c82571dad

[8] [Swift] Protocol [01] : https://baked-corn.tistory.com/24

[9] [Swift] Protocol [02] : https://baked-corn.tistory.com/26

[10] Swift - 프로토콜 지향 프로그래밍 : https://blog.yagom.net/531/

Thank you!