SWIFT
Extensions

Bill Kim(김정훈) | ibillkim@gmail.com

목차

Extensions

Swift에서는 익스텐션(Extensions)를 이용하여 클래스, 구조체, 열거형 혹은 프로토콜 타입에 대한 기능을 확장할 수 있습니다.

익스텐션을 활용하여 아래와 같은 기능을 할 수 있습니다.

익스텐션은 타입에 새 기능을 추가할 수 있지만 오버라이드는 할 수 없습니다.

1) 계산된 인스턴스 프로퍼티와 계산된 타입 프로퍼티의 추가
2) 인스턴스 메소드와 타입 메소드의 추가
3) 새로운 이니셜라이저 제공
4) 서브스크립트 정의
5) 중첩 타입의 선언과 사용
6) 특정 프로토콜을 따르는 타입 만들기
7) 코드를 분리

Extension Syntax

```swift
class SomeType { }

protocol SomeProtocol { }
protocol AnotherProtocol { }

// 익스텐션은 extension 키워드를 사용해 선언합니다.
extension SomeType {
    // new functionality to add to SomeType goes here
}

// 하나의 익스텐션에서 현재 존재하는 타입에 한개 이상의 프로토콜을 따르도록 확장할 수 있습니다.

// 익스텐션을 정의하여 존재하는 타입에 새 기능을 추가하면,
// 그 기능은 익스텐션을 정의하기 이전에 생성한 인스턴스를 포함한 존재하는
// 모든 해당 타입의 인스턴스에서 사용 가능합니다.
extension SomeType : SomeProtocol, AnotherProtocol {
    // implementation of protocol requirements goes here
}
```

Computed Properties

익스텐션을 이용해 존재하는 타입에 계산된 인스턴스 프로퍼티와 타입 프로퍼티를 추가할 수 있습니다.

```
extension Double {
    var km: Double { return self * 1_000.0 }
    var m: Double { return self }
    var cm: Double { return self / 100.0 }
    var mm: Double { return self / 1_000.0 }
    var ft: Double { return self / 3.28084 }
}

let oneInch = 25.4.mm

print("One inch is \(oneInch) meters")
// Prints "One inch is 0.0254 meters"

let threeFeet = 3.ft

print("Three feet is \(threeFeet) meters")
// Prints "Three feet is 0.914399970739201 meters”

let aMarathon = 42.km + 195.m

print("A marathon is \(aMarathon) meters long")
// Prints "A marathon is 42195.0 meters long"
```

Initializers

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.

Initializers

```
struct Size {
    var width = 0.0, height = 0.0
}
struct Point {
    var x = 0.0, y = 0.0
}
struct Rect {
    var origin = Point()
    var size = Size()
}
```

```
extension Rect {
    init(center: Point, size: Size) {
        let originX = center.x - (size.width / 2)
        let originY = center.y - (size.height / 2)
        self.init(origin: Point(x: originX, y: originY), size: size)
    }
}

// Rect 구조체에서 모든 프로퍼티의 기본 값을 제공하기 때문에 Rect구조체는
// 기본 이니셜라이저와 멤버쪽 이니셜라이저를 자동으로 제공 받아 사용할 수 있습니다.

// 기본적으로 제공되는 이니셜라이저를 사용해 초기화를 한 예제입니다.
let defaultRect = Rect()
let memberwiseRect = Rect(origin: Point(x: 2.0, y: 2.0), size: Size(width: 5.0, height: 5.0))

// Rect에서 확장한 이니셜라이저를 사용한 코드는 다음과 같이 사용할 수 있습니다.

let centerRect = Rect(center: Point(x: 4.0, y: 4.0), size: Size(width: 3.0, height: 3.0))
// centerRect's origin is (2.5, 2.5) and its size is (3.0, 3.0)
```

Methods

익스텐션을 이용해 존재하는 타입에 인스턴스 메소드나 타입 메소드를 추가할 수 있습니다.

```
extension Int {
    func repetitions(task: () -> Void) {
        for _ in 0..<self {
            task()
        }
    }
}

// repetitions(task:) 메소드는 () -> Void 타입의 하나의 인자를 받고 파라미터와 반환 값이 없는 함수입니다.
// 함수를 실행하면 함수 안의 task를 숫자 만큼 반복 실행합니다.

3.repetitions {
  print("Hello!")
}
// Hello!
// Hello!
// Hello!
```

Methods

익스텐션에서 추가된 인스턴스 메소드는 인스턴스 자신(self)을 변경할 수 있습니다.

구조체와 열거형 메소드 중 자기 자신(self)를 변경하는 인스턴스 메소드는 원본 구현의 mutating 메소드와 같이 반드시 mutating으로 선언돼야 합니다.

```
extension Int {
    mutating func square() {
        self = self * self
    }
}

var someInt = 3
someInt.square()

print(someInt)
// someInt is now 9
```

Subscripts

익스텐션을 이용해 존재하는 타입에 새로운 서브스크립트를 추가할 수 있습니다.

```swift
extension Int {

    // Swift의 built-in 타입에 integer 서브스크립트를 추가한 예제입니다.
    // 서브스크립트 [n]은 숫자의 오른쪽에서부터 n번째 위치하는 정수를 반환합니다.
    subscript(digitIndex: Int) -> Int {
        var decimalBase = 1
        for _ in 0..<digitIndex {
            decimalBase *= 10
        }

        return (self / decimalBase) % 10
        // 10 * n번째 수로 현재 수를 나눈 것의 나머지
        // 1인 경우 746381295 % 10 -> 5가 나머지
        // 2인 경우 746381295 % 10 -> 9가 나머지
    }
}

print(746381295[0]) // returns 5
print(746381295[1]) // returns 9
print(746381295[2]) // returns 2
print(746381295[8]) // returns 7
print(746381295[9]) // 9로 처리할 수 있는 자릿 수를 넘어가면 0을 반환
```

Nested Types

익스텐션을 이용해 존재하는 클래스, 구조체, 열거형에 중첩 타입을 추가할 수 있습니다.

```
extension Int {
    enum Kind {
        case negative, zero, positive
    }
    var kind: Kind {
        switch self {
        case 0:
            return .zero
        case let x where x > 0:
            return .positive
        default:
            return .negative
        }
    }
}
```

Nested Types

```
func printIntegerKinds(_ numbers: [Int]) {
    for number in numbers {

        // number.kind가 이미 switch에서 Int.Kind 타입이라는 것을 알고 있기 때문에
        // 안의 case에서 kind의 축약형인 .negative, .zeo, .positive로 사용할 수 있습니다.

        switch number.kind {
        case .negative:
            print("- ", terminator: "")
        case .zero:
            print("0 ", terminator: "")
        case .positive:
            print("+ ", terminator: "")
        }
    }
    print("")
}

printIntegerKinds([3, 19, -27, 0, -6, 0, 7]) // + + - 0 - 0 +
```

References

[1] 익스텐션 (Extensions) : https://jusung.gitbook.io/the-swift-language-guide/language-guide/20-extensions

[2] [Swift]Extensions 정리 : http://minsone.github.io/mac/ios/swift-extensions-summary

[3] 애플 스위프트(Apple Swift) - 확장(Extensions) : https://m.blog.naver.com/PostView.nhn?blogId=seotaji&logNo=220300304821&proxyReferer=https:%2F%2Fwww.google.com%2F

[4] 확장(Extensions) : https://kka7.tistory.com/126

[5] Swift - Extensions : https://www.tutorialspoint.com/swift/swift_extensions.htm

References

[6] Extensions In Swift Explained : https://learnappmaking.com/swift-extensions-how-to/

[7] Four Clever Uses of Swift Extensions : https://cocoacasts.com/four-clever-uses-of-swift-extensions

[8] Swift - 확장(Extensions) : http://seorenn.blogspot.com/2014/06/swift-extensions.html

[9] Extensions : https://wlaxhrl.tistory.com/27

[10] 8 Useful Swift Extensions : https://www.hackingwithswift.com/articles/141/8-useful-swift-extensions

Thank you!