Extension

GANG

08 APR 19

(오프화이트 아님ㅎ)

Extension

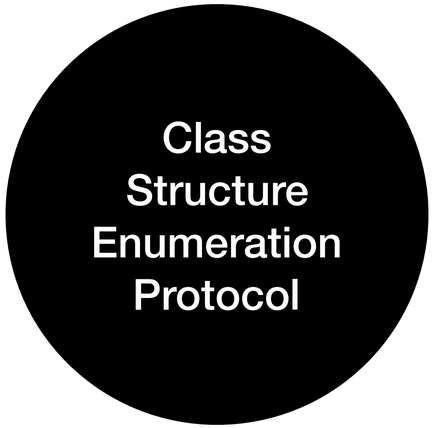
GAENG

08 APR 19

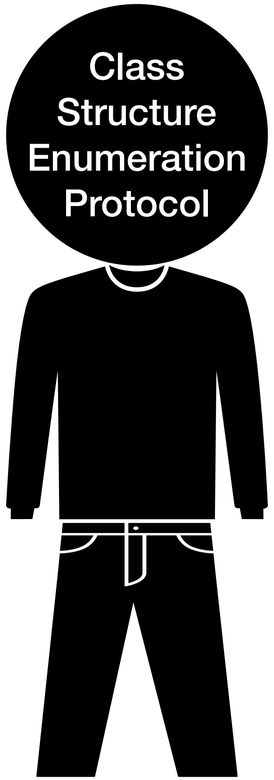
Agenda

- What is Extension?
- Syntex
- Computed Properities
- Initializers
- Methods
- Subscripts
- Nested Types

What is Extension?



What is Extension?



Extension은 타입에 새 기능을 추가할 수 있지만 오버라이드는(override)는 할 수 없습니다.

Extension Syntax

`extension` 키워드 사용하여 선언

```
extension SomeType {
    // new functionality to add to SomeType goes here
}
```

하나의 익스텐션에서 현재 존재하는 타입에 한개 이상의 프로토콜을 따르도록 확장할 수 있음

```
extension SomeType: SomeProtocol, AnotherProtocol {
    // implementation of protocol requirements goes here
}
```

Computed Properties

Extension을 이용해 존재하는 타입에 계산된 인스턴스 프로퍼티를 추가할 수 있음

```
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"
```

Initializers

Extension을 이용해 기존 타입에 새로운 이니셜라이저를 추가 가능. 이 방법으로 커스텀 타입의 이니셜라이저 파라미터를 넣을 수 있도록 변경하거나 원래 구현에서 포함하지 않는 초기화 정보를 추가 가능

Initializers

Size와 Point 구조체 정의 후 이 둘을 사용하는 Rect 구조체 정의. Rect는 모든 프로퍼티의 기본 값을 제공 받기에 Rect는 기본 이니셜라이저와 멤버쪽 이니셜라이저를 자동으로 제공 받아 사용 가능.

```
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()
}
```

기본적으로 제공되는 이니셜라이저를 사용한 초기화 예

```
let defaultRect = Rect()
let memberwiseRect = Rect(origin: Point(x: 2.0, y: 2.0), size: Size(width: 5.0, height: 5.0))
```

Initializers

Rect를 추가적인 이니셜라이저를 제공하기 위해 extension을 이용한 예

```
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)
    }
}
```

Extension을 이용한 이후 사용 예

```
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

extension을 이용해 존재하는 타입에 인스턴스 매소드나 타입 매소드를 추가할 수 있다

Int 타입에 repetitions라는 인스턴스 매소드를 추가한 예제

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

repetitions(task:) 매소드는 () -〉 Void 타입의 하나의 인자를 받고 파라미터와 반환 값이 없는 함수

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

Subscripts

Extension을 이용해 존재하는 타입에 새로운 서브스크립트를 추가 가능

다음 예제는 Swift의 built-in 타입에 integer 서브스크립트를 추가한 예제. 서브스크립트 [n]은 숫자의 오른쪽에서부터 n번째 위치하는 정수를 반환

```
extension Int {
    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
    }
}
746381295[0]
// returns 5
746381295[1]
// returns 9
746381295[2]
// returns 2
746381295[8]
// returns 7
```

Nested Types

extension을 이용해 존재하는 클래스, 구조체, 열거형에 중첩 타입을 추가할 수 있음
Int에 중첩형 enum을 추가한 예제. 열거형 Kind는 Int를 음수, 0, 양수로 표현

```
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
        }
    }
}
```