익스텐션

- 익스텐션은 구조체, 클래스, 열거형, 프로토콜 타입에 새로운 기능을 추가할 수 있다.
- 기능을 추가하려는 타입의 구현된 소스 코드를 알지 못하거나 볼 수 없다 해도, 타입만 알고 있어도 그 타입의 기능을 추가할 수 있다.
- 추가 기능 뿐만이 아니라 외부 라이브러리나 프레임워크를 가져다 쓰게되면 원본 소스를 수정을 할 수 없지만 익스텐션을 이용하면 이러한 타입들을 내가 원하는 기능을 추가할 수 있다.
- 모든 타입에 적용할 수 있다.

스위프트의 익스텐션이 타입에 추가할 수 있는 기능

- 연산 타입 프로퍼티 / 연산 인스턴스 프로퍼티
- 타입 메서드 / 인스턴스 메서드
- 이니셜라이저
- 서브스크립트
- 중첩 타입
- 특정 프로토콜을 준수할 수 있도록 기능 추가
    - double타입을 보면 많은 프로토콜을 채택하고 있지만 모든 프로토콜이 정의되어 있지 않고 익스텐션을 이용하여 채택하고 프로토콜을 준수하도록 정의하였다.

기존에 있는 기능은 추가할 수 없다.

클래스의 상속과 익스텐션 비교

클래스 상속

- 클래스타입에서만 상속 가능
- 특정 타입을 물려받아 하나의 새로운 타입을 정의하고 추가 기능을 구현하는 수직 확장

익스텐션

- 익스텐션은 구조체, 클래스, 프로토콜 등에 적용이 가능
- 익스텐션은 기존 타입에 기능을 추가하는 수평 확장

상속		익스텐션
확장	수직 확장	수평 확장
사용	클래스 타입	클래스, 구조체, 프로토콜, 제네릭 등 모든 타입
재정의	가능	불가능

정의

```
extension 확장할 타입 이름 {
    /* 타입에 추가될 새로운 기능 구현 */
}
```

```
extension 확장할 타입 이름: 프로토콜1, 프로토콜2, 프로토콜3... {
    /* 프로토콜 요구사항 구현 */
}
```

Extension 기능은 delegate와 datasource를 사용시에도 유용하게 이용한다.

구현

연산 프로퍼티 추가

```swift
extension Int {
    var isEven: Bool {
        return self % 2 == 0
    }
    var isOdd: Bool {
        return self % 2 == 1
    }
}

print(1.isEven) // false
print(2.isEven) // true
print(1.isOdd)  // true
print(2.isOdd)  // false

var number: Int = 3
print(number.isEven) // false
print(number.isOdd) // true

number = 2
print(number.isEven) // true
print(number.isOdd) // false
```

Int타입에 두 개의 연산 프로퍼티(인스턴스가 홀/짝수인지 판별)를 추가한 것
Static 키워드를 사용하여 타입 연산 프로퍼티도 추가 가능
But 저장 프로퍼티나 타입에 정의되어 있는 기존의 프로퍼티에 프로퍼티 감시자를 추가할 수 없다.

메서드 추가

```swift
extension Int {
    func multiply(by n: Int) -> Int {
        return self * n
    }
}
print(3.multiply(by: 2))  // 6
print(4.multiply(by: 5))  // 20

number = 3
print(number.multiply(by: 2))   // 6
print(number.multiply(by: 3))   // 9
```

여러 기능인 메서드들을 추가해야 될때 여러 익스텐션 블록을 나눠서 구현하는 것이 좋다.

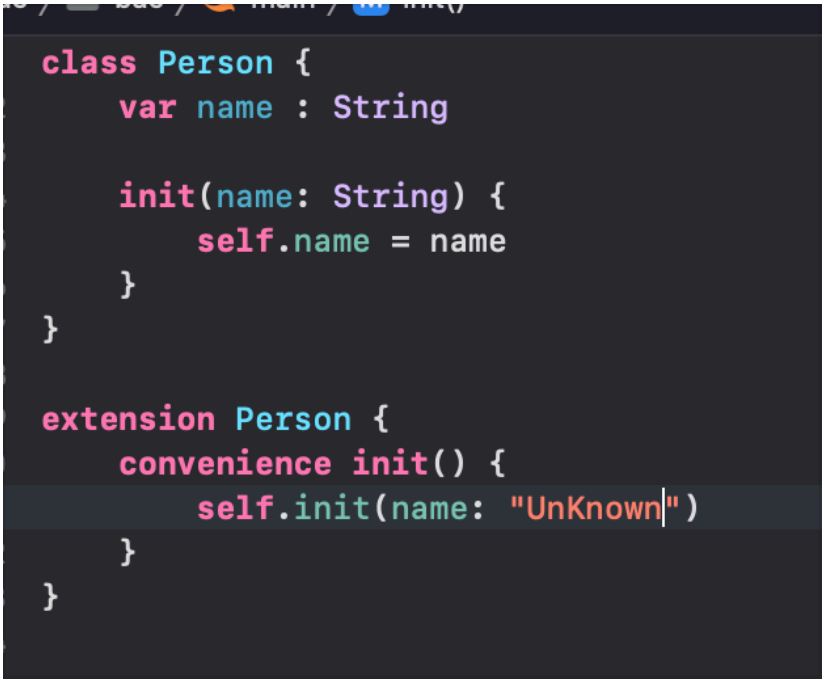
이니셜라이저 추가

```swift
extension String {
    init(int: Int) {
        self = "₩(int)"
    }

    init(double: Double) {
        self = "₩(double)"
    }
}

let stringFromInt: String = String(int: 100)
// "100"

let stringFromDouble: String = String(double: 100.0)
// "100.0"
```



```swift
class Person {
    var name : String

    init(name: String) {
        self.name = name
    }
}

extension Person {
    convenience init() {
        self.init(name: "UnKnown")
    }
}
```

- 인스턴스를 초기화(이니셜라이즈)할 때 인스턴스 초기화에 필요한 다양한 데이터를 전달받을 수 있도록 여러 종류의 이니셜라이저를 만들 수 있습니다. 타입의 정의부에 이니셜라이저를 추가하지 않더라도 익스텐션을 통해 이니셜라이저를 추가할 수 있습니다.
- 익스텐션으로 클래스 타입에 편의 이니셜라이저는 추가할 수 있지만, 지정 이니셜라이저는 추가할 수 없습니다. 지정 이니셜라이저와 디이니셜라이저는 반드시 클래스 타입의 구현부에 위치해야 합니다(값 타입은 상관없습니다).