



BoostCamp iOS

MOGAY

Contents

REPL

Enum

Type Safety & Inference

Extension

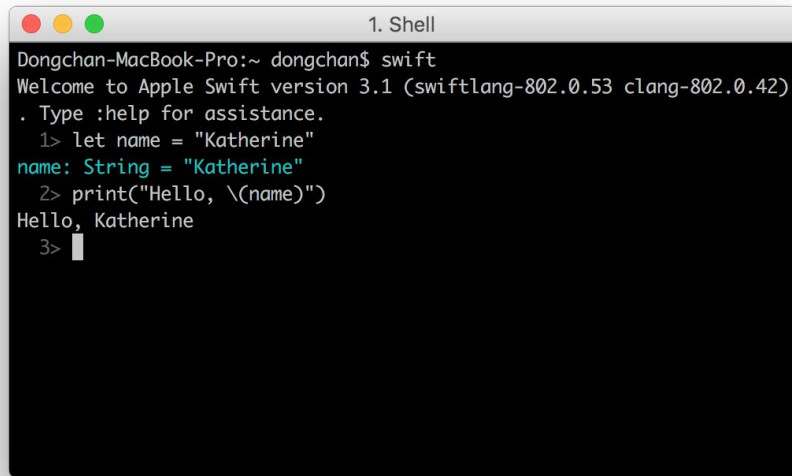
Optional

Property Observer

Closure

Read-Eval-Print-Loop (REPL)

REPL은 Read-Eval-Print Loop의 머리 글자로 “읽기-실행-출력 반복”이란 뜻. 간단하게 “스위프트 인터랙티브 명령행 실행기”이다. 직접 간단한 코드를 직접 입력하여 바로바로 결과값을 볼수있게끔 해주는 편의성을 제공해주는 툴이다. 오픈 소스 스위프트 컴파일러가 설치된 리눅스나 맥에서 터미널을 실행하고 **swift**를 실행하면 **REPL**이 실행된다.

A screenshot of a macOS terminal window titled "1. Shell". The terminal shows the execution of the Swift REPL. The prompt is "Dongchan-MacBook-Pro:~ dongchan\$ swift". The output is "Welcome to Apple Swift version 3.1 (swiftlang-802.0.53 clang-802.0.42)". The prompt changes to "1>". The user enters "let name = \"Katherine\"". The output is "name: String = \"Katherine\"". The prompt changes to "2>". The user enters "print(\"Hello, \"(name)\")". The output is "Hello, Katherine". The prompt changes to "3>".

```
Dongchan-MacBook-Pro:~ dongchan$ swift
Welcome to Apple Swift version 3.1 (swiftlang-802.0.53 clang-802.0.42)
. Type :help for assistance.
1> let name = "Katherine"
name: String = "Katherine"
2> print("Hello, \"(name)\")
Hello, Katherine
3>
```

* Java 9에서 JShell이라는 툴로 REPL을 지원할 예정

Enum

```
enum Weekday {  
    case mon  
    case tue  
    case wed  
    case thu, fri, sat, sun  
}
```

enum은 타입이므로 대문자 카멜케이스를 사용하여 이름을 정의

각 **case**는 소문자 카멜케이스로 정의

각 **case**는 그 자체가 고유의 값

각 케이스는 한 줄에 개별로도, 한 줄에 여러개로도 정의 가능

Enum

Hashable 프로토콜을 따르는 모든 타입이 원시값으로 가능

//정수 타입 뿐만 아니라 Hashable 프로토콜을 따르는 모든 타입이 원시값의 타입으로 지정될 수 있습니다.

```
enum School: String {  
    case elementary = "초등"  
    case middle = "중등"  
    case high = "고등"  
    case university  
}
```

```
print("School.middle.rawValue == \(School.middle.rawValue)")  
// School.middle.rawValue == 중등
```

// 열거형의 원시값 타입이 String일 때, 원시값이 지정되지 않았다면
// case의 이름을 원시값으로 사용합니다
print("School.university.rawValue == \(School.university.rawValue)")
// School.middle.rawValue == university

switch의 비교값에 열거형 타입이 위치할 때
모든 열거형 케이스를 포함한다면
default를 작성할 필요 없음

```
// switch의 비교값에 열거형 타입이 위치할 때  
// 모든 열거형 케이스를 포함한다면  
// default를 작성할 필요가 없습니다  
switch day {  
case .mon, .tue, .wed, .thu:  
    print("평일입니다")  
case Weekday.fri:  
    print("볼금 파티!!")  
case .sat, .sun:  
    print("신나는 주말!!")  
}
```

Enum

원시값을 통한 초기화

`rawValue`를 통해 초기화 할 수 있음.

`rawValue`를 통해 초기화 한 인스턴스는 옵셔널 타입.

// `rawValue`를 통해 초기화 한 열거형 값은 옵셔널 타입이므로 `Fruit` 타입이 아닙니다

//let apple: Fruit = Fruit(rawValue: 0)

let apple: Fruit? = Fruit(rawValue: 0)

// if let 구문을 사용하면 `rawValue`에 해당하는 케이스를 곧바로 사용할 수 있습니다

if let orange: Fruit = Fruit(rawValue: 5) {

print("rawValue 5에 해당하는 케이스는 \orange입니다")

} else {

print("rawValue 5에 해당하는 케이스가 없습니다")

} // rawValue 5에 해당하는 케이스가 없습니다

메서드 추가

```
enum Month {  
    case dec, jan, feb  
    case mar, apr, may  
    case jun, jul, aug  
    case sep, oct, nov  
  
    func printMessage() {  
        switch self {  
            case .mar, .apr, .may:  
                print("따스한 봄~")  
            case .jun, .jul, .aug:  
                print("여름 더워요~")  
            case .sep, .oct, .nov:  
                print("가을은 독서의 계절!")  
            case .dec, .jan, .feb:  
                print("추운 겨울입니다")  
        }  
    }  
}
```

Month.mar.printMessage()

Type Inference (타입 추론)

swift : OK!!

```
var doubleType = 1.23
```

java : ????

```
doubleType = 22.3;
```

Type Safety

swift : No!!!!

```
5 var doubleType : Double = 1.23
6 var floatType : Float = 32.3
7
8 doubleType = floatType
```

Cannot assign value of type 'Float' to type 'Double'

Java : Ok..

```
double doubleType = 1.23;
float floatType = 3.2f;

doubleType = floatType;
```


Extension

기존에 있는 클래스, 구조체, 열거형, 프로토콜 타입에 새로운 기능을 추가!!

swift 확장이 할 수 있는 것들

- 계산 인스턴스 프로퍼티와 계산 타입 프로퍼티 추가
- 인스턴스 메소드와 타입 메소드를 정의
- 새로운 초기화를 제공
- 서브스크립트 정의
- 새로 중첩된 타입을 정의하고 사용
- 기존 타입에 프로토콜을 준수 (conform)

Optional

안정성을 문법적으로 보장

값이 있을 수도 없을 수도 있음

변수나 상수 등에 꼭 값이 있다는 것을 보장하지 않음

It can be null이라는 부연 설명할 필요가 없이 컴파일 할 때 오류를 걸러낼 수 있다.

자바에는 **Optional**을 언어 차원에서 지원해주는 것이 아니라 **java.util**에서 지원한다.

옵셔널 강제 추출 / 옵셔널 바인딩

```
var team: String? = "mogay"
```

```
var teamUnwrapping: String = team!
```

```
if var teamName = team {
```

```
    print("\(teamName)") // mogay
```

```
} else {
```

```
    print("nil")
```

```
}
```

```
guard let teamName = team else {
```

```
    return
```

```
}
```

옵셔널 체이닝

옵셔널 체이닝(Optional chaining)은 현재 옵셔널이 nil이 될 수 있는 프로퍼티, 메소드, 서브스크립트를 조회하고 호출하는 과정

옵셔널 체이닝에 값이 있으면, 프로퍼티, 메소드, 스크립트 호출은 성공

옵셔널이 nil이면, 프로퍼티, 메소드, 스크립트 호출은 nil을 반환

여러개를 함께 연결 할 수 있고, 연결된 어떤 링크가 nil이면, 전체 체인(chain)은 실패하게 된다

`array?.isEmpty`의 결과로 나올 수 있는 값은 `nil`, `true`, `false`가 됩니다.

`isEmpty`의 반환값은 `Bool`인데, 옵셔널 체이닝으로 인해 `Bool?`을 반환하도록 바뀐 것이죠.

옵셔널 체이닝을 사용하면 이 코드를 더 간결하게 쓸 수 있습니다.

```
let isEmptyArray = array?.isEmpty == true
```

혹시 감이 오시나요? 옵셔널 체이닝은 옵셔널의 속성에 접근할 때, 옵셔널 바인딩 과정을 ? 키워드로 줄여주는 역할을 합니다. 다음과 같이 3가지 경우의 수를 생각해봅시다.

- `array` 가 `nil` 인 경우

```
array?.isEmpty
~~~~~
여기까지 실행되고 `nil`을 반환합니다.
```

- `array` 가 빈 배열인 경우

```
array?.isEmpty
~~~~~
여기까지 실행되고 `true`를 반환합니다.
```

- `array` 에 요소가 있는 경우

```
array?.isEmpty
~~~~~
여기까지 실행되고 `false`를 반환합니다.
```

Property Observer

프로퍼티 감시자를 이용하면 프로퍼티 값이 변경될 때 원하는 동작을 수행할 수 있다

- didSet

새로운 값이 저장 된 직후 호출

- willSet

값이 저장되기 직전에 호출

상위 클래스 초기화가 호출된 후에,
하위클래스 초기화에서 프로퍼티가 설정할때 호출

상위클래스 초기화가 호출되기 전에,
클래스가 자체 프로퍼티를 설정하는 동안에는
호출되지 않는다

```
// Property Observer
class StepCounter {
    var totalSteps : Int = 0 {
        willSet(newTotalSteps) {
            print("About to set totalSteps to \(newTotalSteps)")
        }
        didSet {
            if totalSteps > oldValue {
                print("Added \(totalSteps - oldValue) steps")
            }
        }
    }
}
```

Closure

Closure : 독립적인 기능의 블록

```
sortedName = names.sorted(by: { (s1: String, s2: String) -> Bool in  
    return s1 > s2  
})
```

```
sortedName = names.sorted(by: { (s1: String, s2: String) -> Bool in return s1 > s2 } )
```

```
sortedName = names.sorted(by: { s1, s2 in return s1 > s2 } )
```

Type Inference

```
sortedName = names.sorted(by: { s1, s2 in s1 > s2 } )
```

Implicit Returns

```
sortedName = names.sorted(by: { $0 > $1 } )
```

Shorthand Argument

Closure

Closure : 독립적인 기능의 블록

```
sortedName = names.sorted(by: > )
```

Operator Methods

```
sortedName = names.sorted() { $0 > $1 }
```

Trailing Closure

다른 언어의 `lambda`와 비슷한 기능

Trailing Closure 의 경우 코드의 가독성을 상승

Swift 의 특징을 살릴 수 있는 프로그래밍 방법

고차함수 (filter, reduce, map)

Designed for Safety : Optional, Optional Binding, Optional Chaining

클로저를 이용해 선언과 동시에 속성 설정

```
fileprivate var topRecodeLabel : UILabel! = {  
    var topRecodeLabel = UILabel()  
    topRecodeLabel.text = "최고기록"  
    topRecodeLabel.textColor = UIColor.green  
    topRecodeLabel.translatesAutoresizingMaskIntoConstraints = false  
    return topRecodeLabel  
}()
```


감사합니다

