Swift 프로그래밍

12. 구조체와 Enum

Tacademy

CONTENTS

학습 목표

- 구조체와 프로퍼티, 메소드를 작성하고 구조체와 클래스의 차이를 이해한다.
- Enum 타입과 원소 타입을 이해하고 사용할 수 있다.
- 타입 내부에 타입을 정의하는 중첩 타입을 이해할 수 있다.

1. 구조체

객체지향 프로그래밍

- 애플리케이션에서 다루는 데이터와 행위
- 클래스로 작성하기
  - 클래스와 메소드, 프로퍼티
- 클래스 외 다른 선택
  - 구조체
  - Enum

구조체 정의

- **struct** [구조체 이름] **{}**
- 좌표를 다루는 구조체 **Point** 작성
- 프로퍼티와 메소드

```
struct Point {
    var x = 0
    var y = 0

    func description() -> String {
        return "Point : \(x), \(y)"
    }
}
```

구조체 사용

❖ 구조체 객체

- 객체 생성과 사용

```
var p1 = Point()
p1.x = 10
p1.y = 20
```

- 프로퍼티를 자동으로 초기화하는 객체 생성

```
var p2 = Point(x: 3, y: 5)
```

—> initializer가 자동 생성됨

구조체 초기화

❖ 구조체 객체 초기화

◉ 초기화가 필요한 프로퍼티

```
struct Point {
    var x : Int
    var y : Int
}
```

◉ 프로퍼티 자동 초기화 **Initializer**

```
~~var obj = Point()~~
var obj2 = Point(x : 10, y : 10)
```

구조체 초기화

❖ 구조체 객체 초기화

◉ **Initializer** 작성

```
struct Point {
    var x : Int
    var y : Int
    init() {
        self.x = 0
        self.y = 0
    }
}
```

◉ 프로퍼티 자동 초기화 **Initializer** 사용 불가

```
var obj = Point()
var obj2 = Point(x : 10, y : 10)
```

구조체 초기화

❖ 구조체의 Initializer

- Initializer 없이 프로퍼티 초기화 가능

❖ 구조체의 Designated Initializer

- 상속 없으므로 부모 클래스로 초기화 위임 없음

❖ 구조체의 Convenience Initializer

- convenience 키워드 생략
- Initializer delegation 작성

구조체 초기화

❖ 구조체의 **Initializer**

```
struct MyStruct {
    var value : Int

    // Designated Initializer
    init(v : Int) {
        self.value = v
    }

    // convenience Initializer
    init() {
        self.init(v:0)      // initializer delegation
    }
}
```

Mutating

❖ 구조체의 프로퍼티 수정

- 초기화 메소드에서 설정 가능 // initializer
- 구조체 외부에서 설정 가능 // 구조체 객체 생성 후
- ~~구조체 내부에서 설정 가능~~

❖ 구조체 프로퍼티 수정 메소드

- 메소드 선언에 **mutating** 추가

Mutating

- 구조체 내부에서 데이터 수정
  - 컴파일 에러 발생

```
struct Point {
    var x = 0
    var y = 0
    func moveTo(x : Int, y : Int) {
        self.x = x // 에러
        self.y = y // 에러
    }
}
```

Mutating

- 구조체 내부에서 데이터 수정
  - 메소드에 **mutating** 선언

```
struct Point {
    var x = 0
    var y = 0
    mutating func moveTo(x : Int, y : Int) {
        self.x = x
        self.y = y
    }
}
```

다른 구조체 타입 : Array

❖ 배열 : struct Array

◉ 배열 내용 유지 - 새로운 배열 생성 API

```
func map<U>(transform: (T) -> U) -> [U]
func reverse() -> [T]
func filter(includeElement: (T) -> Bool) -> [T]
```

배열의 각 원소를 파라미터로 전달하는 클로저를 사용하여 값을 계산한 새로운 배열을 생성

◉ 배열 내용 변경 API

```
mutating func insert(newElement: T, atIndex i: Int)
mutating func remove(at index: Int) -> T
```

정적 메소드와 정적 프로퍼티

- **static** 키워드

```
struct MyStruct {
    static func staticFunc() {
        print("정적 메소드")
    }

    static var staticProperty : Int!
}

MyStruct.staticFunc()
MyStruct.staticProperty = 10
```

2. 클래스와 구조체

클래스와 구조체

- 구조체와 클래스 모두 있는 것
  - 프로퍼티와 메소드
  - 초기화(Initializer)
- 구조체에는 없고 클래스에 있는 것
  - 상속과 재정의
  - ARC

클래스와 구조체

- 레퍼런스(Reference) 타입과 밸류(Value) 타입

클래스	구조체
레퍼런스 타입	밸류 타입

클래스와 구조체

❖ 객체 대입

클래스 : 참조 복사

구조체 : 복사

- 클래스 : 객체 참조 추가

```
let obj1 = MyClass()      r.c. +1 (1)
let obj2 = obj1           r.c. +1 (2)
```

- 구조체 : 복사

```
let struct1 = MyStructure()
let struct2 = struct1
```

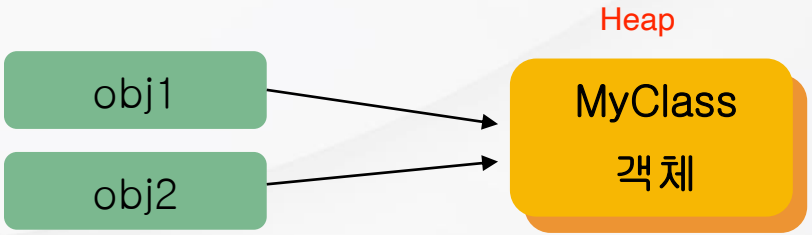
레퍼런스 타입(class)

❖ 객체 공유

```
class MyClass {
  var value = 0
}

var obj1 = MyClass()
var obj2 = obj1 참조복사

obj1.value = 10
obj2.value
```



밸류 타입(struct)

❖ 객체 복사

```
struct MyStruct {
  var value = 0
}

var obj3 = MyStruct()
var obj4 = obj3   객체복사

obj3.value = 10
obj4.value
```

참조 타입 안에 값 타입이 있는 경우
- class 안에 struct 프로퍼티가 존재
- 참조 타입이 할당 해제 되기 전에 값 타입도 할당 해제되지 않게 하기 위해 내부의 값 타입도 힙에 저장됨
- 클로저 내부에 값 타입이 있는 경우도 해당

값 타입 안에 참조 타입이 있는 경우
- struct 안에 class 프로퍼티가 존재
- 값 타입은 힙에 할당되지 않지만, 내부에 참조 타입이 있으므로 참조 카운트를 처리해 줘야 함

Swift의 밸류 타입

❖ 밸류 타입

- Int, Double, Float, Bool
- String, Character
- Array
- Dictionary
- Set
- Struct, Enum

클래스와 구조체

❖ 클래스와 구조체 중 무엇을 사용할까?

- **iOS, macOS**에서 사용하는 프레임워크 - 클래스 종속적인 **API**

❖ 프로토콜 기반의 프로그래밍**(POP)**

WWDC에서의 발표
개발자 컨퍼런스

구조체와 프로토콜을 이용한 프로그래밍 기법

3. Enum

Enum Type

❖ 열거형(Enumeration Type)

- 원소 중 하나의 값
- **Enum** 정의

```
enum Day {
  case am
  case pm
}

enum Pet {
  case cat, dog, other
}
```

- **Enum** 사용

```
var now : Day
now = Day.am
now = Day.pm
now = Day.morning // Error
```

Enum과 switch

- switch와 사용
- 타입 정보가 있는 곳 - Enum 이름 생략 가능

```
var now = Day.am
switch now {
case .am:   // Day.am
    print("오전")
case .pm:
    print("오후")
}
```

Enum 타입

❖ Enum 원소 타입

- 원소 타입 사용하기
- Int 타입 원소 값 설정 생략

```
enum Pet : Int {
    case cat = 0, dog, other
}
```

1 2

없으면 0부터 1씩 증가됨

- Int타입 외 값 설정 생략 불가

```
enum Device : String {
  case phone = "휴대폰", pad = "패드"
}
```

Enum 타입

❖ 원소 타입이 있는 Enum

- 원소의 값 : **rawValue**
- **rawValue**에서 **Enum** 생성 (옵셔널)

```
var ael = Pet(rawValue: 0)
```
Pet?

// Optional(cat)
(rawValue 값이 없으면 nil)

- **Enum**에서 **rawValue** 얻기

```
ael?.rawValue
```

// 0

Enum : 프로퍼티와 메소드

❖ 프로퍼티(계산)와 메소드

```
enum Pet : Int {
    case cat, dog

    var name : String {
        switch self {
        case .cat:
            return "고양이"
        case .dog:
            return "강아지"
        }
    }

    func description() -> String {
        return self.name
    }
}
```

```
var raz = Pet.cat

raz.name

raz.description()
```

4. 중첩 타입

중첩 타입

❖ 타입 내부에 타입 정의

- 클래스, 구조체, **Enum** 내부에 타입 정의

```
struct Rectangle {
    struct Point {
        var x, y : Int
    }
    struct Size {
        var width, height : Int
    }
    var origin : Point
    var size : Size

    init() { // 초기화 코드 }
}
```

중첩 타입

❖ 중첩 타입 접근

```
let point = Rectangle.Point(x: 10, y: 10)
let size = Rectangle.Size(width: 100, height: 100)
```

학습정리

지금까지 [구조체와 Enum]에 대해서 살펴보았습니다.

구조체

구조체의 정의와 프로퍼티, 메소드 작성 방법. 구조체 객체의 초기화

클래스와 구조체

클래스의 대안으로 구조체 사용. 구조체와 클래스의 차이점와 비슷한 점

Enum

Enum 정의하는 방법과 사용. Enum내 프로퍼티와 메소드 작성 방법

중첩 타입

타입 내부에 타입을 정의하는 방법과 사용하는 방법