Swift 프로그래밍

9. 객체 초기화

Tacademy

CONTENTS

학습 목표

- 객체를 사용 전에 초기화가 필요하고 수동으로 초기화가 필요한 상황을 이해할 수 있다.
- Designated Initializer, Convenience Initializer를 작성할 수 있다.
- Failable Initializer와 deinit을 이해할 수 있다.

1. Initializer

초기화

- 모든 객체는 사용하기 전에 초기화
- 프로퍼티 초기화
  - 초기값과 함께 선언된 프로퍼티 — 자동
  - 옵셔널 타입의 프로퍼티 — 자동 (nil)
  - 초기값이 없고, 옵셔널 타입이 아닌 프로퍼티!! — initializer

객체 준비

❖ 객체 사용 준비

◉ 클래스 선언

```
class Rectangle {
   var width = 0
   var height = 0
   var name : String?    nil
}
```

◉ 객체 생성

```
var obj = Rectangle()
obj.width
obj.height
obj.name
```

객체 준비

❖ 객체 사용 준비

- 클래스 선언
- 초기값이 없고 옵셔널 타입이 아닌 프로퍼티

```
class MyClass {
    var value : Int
}
```

에러

- 객체가 생성되면?

```
var obj = MyClass()
obj.value = ???
```

사용 불가

Initializer

❖ 객체 사용에 앞서

- 객체 사용하려면 초기화 필요
- 옵셔널/초기값이 있는 프로퍼티 : 자동 초기화
- 옵셔널 타입이 아니고 초기값이 없는 프로퍼티 : 수동 초기화 **Initializer 사용**

❖ Initializer

- 객체 초기화 코드 작성
- 하나의 클래스에 다수의 **Initializer** 작성

```
init() {
   // 객체 초기화
}
```

**failable initializer가 아닌 경우,
return value 없음**

Initializer 작성과 객제 생성

❖ Initializer 작성

- 초기화가 필요한 프로퍼티 초기화

```
class Rectangle {
    var width : Int
    var height : Int
    init() {
        width = 0
        height = 0
    }
}
```

- 객체 생성 : Initializer 형태에 맞게

```
let obj = Rectangle()
```

Initializer 작성과 객체 생성

❖ 파라미터가 있는 Initializer

```
class Rectangle {
    var width : Int
    var height : Int

    init(width : Int, height : Int) {
        self.width =  width
        self.height = height
    }
}
```

◉ 객체 생성 : Initializer 형태에 맞게

```
let obj1 = Rectangle(width:10, height:20)
let obj2 = Rectangle()  에러
```

Initializer의 종류

❖ Designated Initializer

- 객체 초기화를 단독으로 완료 가능
- 모든 초기화가 필요한 프로퍼티 초기화
- 클래스에 반드시 1개 이상 필요 (Convenience Initializer를 사용하는 경우)

❖ Convenience Initializer

- 단독으로 초기화 불가능
- 일부 프로퍼티만 초기화
- 다른 Initializer를 이용해서 초기화
- 중복 코드 방지

Designated Initializer

◉ 모든 값 초기화

```
class MyClass {
   var a : Int
   var b : Int

   init(a : Int, b : Int) {
      self.a = a
      self.b = b
   }
}
```

Designated Initializer

- 초기화 코드 먼저 작성

```
init(a : Int, b : Int) {
   self.a = a
   // 초기화가 끝나기 전에 다른 메소드 호출은 에러
   otherPrepare()
   self.b = b
}
```

초기화 안 된 프로퍼티를 다른 메소드에서 사용할 가능성이 있기 때문

Convenience Initializer

- 단독으로 객체 초기화 불가
- 초기화가 필요한 모든 프로퍼티를 초기화하지 않음
- 다른 초기화 메소드에 의존(**Initializer Delegation**)

❖ Initializer Delegation

- 다른 **init** 메소드 호출하기 `self.init( … )`
- 다양한 객체 생성 방법 제공 **-> init** 메소드 다수
- 초기화 코드의 중복 방지, 재사용 높이기

Convenience Initializer

◉ 초기화 위임 이후에 다른 초기화 동작 작성

```
convenience init(파라미터) {
   // 초기화 위임   self.init( … )
   // 초기화 코드   다른 프로퍼티 초기화
}
```

❖ **Initializer Delegate** 방향

Convenience Initializer

```
class MyClass {
   var a, b : Int
   init() {                    designated initializer
      a = 0
      b = 0
   }
   init(a:Int, b:Int) {        designated initializer
      self.a = a
      self.b = b
   }
   convenience init(b:Int) {
      self.init()    // Initializer delegation
      self.b = b
   }
}
```

Convenience Initializer

❖ 객체 생성

```
// Designated Initializer로 객체 생성
var obj1 = MyClass()
var obj2 = MyClass(a: 1, b: 2)

// Convenience Initializer로 객체 생성
var obj3 = MyClass(b: 2)
```

2. Failable Initializer

Failable Initializer

❖ 객체 생성, 초기화 과정 실패

- 출생년도가 미래인 **Person** 객체
- 학번 규칙이 맞지 않는 **Student** 객체
- 크기가 음수인 **Rectangle** 객체

❖ 초기화 실패의 결과는?

- **nil** 반환 - **Failable Initializer**

```
let obj = Rectangle(width:-10, height:-20)
obj // nil
```

Failable Initializer

❖ 작성 방법

- **Initializer** 와 동일
- 조건 체크 - 오류상황에 **nil** 반환

❖ **Initializer** 이름

init?, init**!**

Failable Initializer 작성

- 1900년 이전 출생한 사람은 없는 시스템
- Initializer의 파라미터 조건 검사

```
init?(birthYear : Int) {
    if birthYear <= 1900 {
        return nil
    }
    else {
        self.birthYear = birthYear
    }
}
```

Failable Initizlier로 객체 생성

- 객체 생성 메소드 - 반환 타입이 옵셔널

```
var obj1 = Person(birthYear: -1990) // nil, Optional
var obj2 = Person(birthYear: 1990)
```

Person? or Optional<Person>

- if-let 바인딩

```
if let obj = Person(birthYear: 1990) {
}
```

- 강제 언래핑

```
var obj2 = Person(birthYear: -1800)!
```

런타임 에러

Failable Initizlier로 객체 생성

- 암시적 언래핑 옵셔널로 작성 가능

```
init!(birthYear : Int) {
    // 초기화 코드
}
```

- 생성된 객체의 타입

```
let obj : Person! = Person(birthYear : 1999)
```

obj.birthYear // obj가 nil이면 런타임 에러

3. 객체 해제

해제 메소드

❖ 객체의 메모리 해제

- 객체가 메모리에서 해제되면서 호출
- 이름 **deinit**
- 파라미터, 리턴 타입 없음

```
class MyClass {
    deinit {
        // 객체 해제 시 동작
    }
}
```

<주의>
struct에는 deinit이 없음
deinit 뒤에 ( )가 없어야 함

deinit

- 객체 생성, 해제

```
var obj : MyClass! = MyClass()
// 객체 해제
obj = nil     // deinit 실행됨
```

학습정리

지금까지 [객체 초기화]에 대해서 살펴보았습니다.

Initializer

수동으로 초기화 코드 작성이 필요한 상황과
Initializer 작성하기, Designated Initializer와 Convenience Initializer

Failable Initializer

객체 생성을 실패하면 nil을 반환하는 Failable Initializer 작성과 사용 방법

객체 해제

객체가 해제되면서 동작하는 deinit