Property

강사 주영민

프로퍼티

- 변수의 다른 이름
- 클래스, 구조체, 열거형등 전체적으로 사용되는 변수를 프로퍼티라고 부른다.

프로퍼티의 종류

- 저장 프로퍼티 (Stored Properties)
- 연산 프로퍼티 (Computed Properties)
- 타입 프로퍼티 (TypeProperties)

저장 프로퍼티(Stored Properties)

- 가장 일반적인 프로퍼티
- 값을 저장하는 용도로 사용된다.
- 클래스, 구조체에서만 인스턴스와 연관된 값을 저장한다.
- 초기값을 설정 할 수 있습니다.

저장 프로퍼티(Stored Properties)

```
struct FixedLengthRange {
    var firstValue: Int
    let length: Int
}
```

```
var rangeOfThreeItems:FixedLengthRange
            = FixedLengthRange(firstValue: 0, length: 3)
```

지연 저장 프로퍼티 (Lazy Stored Properties)

- 지연 저장된 속성은 처음 프로퍼티가 사용되기 전 까지 초기값이 계산되지 않은 특성을 가지고 있는 프로퍼티이다.

- 지연 저장 속성은 **lazy** keyword를 선언 앞에 작성한다.

- let은 지연 저장 프로퍼티로 설정할 수 없다.

- 초기화하는데 오래걸리거나, 복잡한 초기화 과정이 있는 변수의 경우 지연저장을 사용하면 좋다 .

연산 프로퍼티(Computed Properties)

- 실제로 값을 저장하지 않지만, get, set키워드를 통해서 값을 간접적으로 설정하거나 받을 수 있다.
- 클래스, 구조체, 열거형에서 사용할수 있는 프로퍼티이다.

연산 프로퍼티 예제

```
struct Point {
    var x = 0.0, y = 0.0
}
struct Size {
    var width = 0.0, height = 0.0
}
struct Rect {
    var origin = Point()
    var size = Size()
    var center: Point {
        get {
            let centerX = origin.x + (size.width / 2)
            let centerY = origin.y + (size.height / 2)
            return Point(x: centerX, y: centerY)
        }
        set(newCenter) {
            origin.x = newCenter.x - (size.width / 2)
            origin.y = newCenter.y - (size.height / 2)
        }
    }
}
```

연산 프로퍼티 예제

- 연산프로퍼티 자신을 바로 사용하면 어떻게 될까요?

```
struct Rect {
    //var origin = Point()
    var size = Size()
    var center: Point {
        get {
            let centerX = center.x + (size.width / 2)
            let centerY = center.y + (size.height / 2)
            return Point(x: centerX, y: centerY)
        }
        set(newCenter) {
            center = newCenter
        }
    }
}
```

Setter ValueName 미 지정

- set의 값이름 미지정시 newValue 가 기본 값으로 사용된다.

```
struct Rect {
    var origin = Point()
    var size = Size()
    var center: Point {
        get {
            let centerX = origin.x + (size.width / 2)
            let centerY = origin.y + (size.height / 2)
            return Point(x: centerX, y: centerY)
        }
        set {
            origin.x = newValue.x - (size.width / 2)
            origin.y = newValue.y - (size.height / 2)
        }
    }
}
```

Read Only 연산 프로퍼티

- 읽기 전용 연산프로퍼티 작성시 get 키워드 없이 바로 작성할수 있다.

- 쓰기 전용 연산 프로퍼티는 작성할수 없다.

```
struct Cuboid {
    var width = 0.0, height = 0.0, depth = 0.0
    var volume: Double {
        return width * height * depth
    }
}
```

Property Observers

- 프로퍼티 값의 변화를 감시하고 그에 따라 대응한다.
- 초기값이 설정된 저장 프로퍼티에서 사용 가능하다. .
- 프로퍼티의 값이 설정될때마다 호출된다.
- didSet, willSet 키워드를 통해 값 변화의 직전 직후를 감지 할수 있다.
- 값이름 미지정시 oldValue, newValue가 기본값으로 지정된다.

Property Observers 예제

```
var changeValue:Int = 0{
    didSet(oldV){
        print("oldValue \(oldV)")
    }
    willSet(willV)
    {
        print("newValue \(willV)")
    }
}

changeValue = 4
```

타입 프로퍼티(Type Properties)

- 인스턴스의 속성이 아닌, 타입에 따른 속성을 정의 할수 있다.
- static 키워드를 사용해서 값타입에서 타입 프로퍼티를 설정할수 있으며, class 키워드를 사용해서 클래스 타입에서 타입 프로퍼티를 설정할 수 있다.
- 값을 가져올때는 클래스의 이름을 통해서 가져올 수 있다.

타입 프로퍼티 예제

```
struct AudioChannel {
    static let level = 10
    static var maxLevel = 0
    var currentLevel: Int = 0 {
        didSet {
            if currentLevel > AudioChannel.level
            {
                currentLevel = AudioChannel.level
            }
            if currentLevel > AudioChannel.maxLevel
            {
                AudioChannel.maxLevel = currentLevel
            }
        }
    }
}
```

Method

- 메서드는 특정 타입에 관련된 함수를 뜻합니다.
- 함수의 문법과 같다.
- 인스턴스의 기능을 수행하는 인스턴스 메서드와 타입자체의 기능을 수행하는 타입 메서드로 나눌수 있습니다.

self Property

- 모든 인스턴스는 self 프로퍼티를 가지고 있다.
- self프로퍼티는 자기 자신을 가르키고 있는 프로퍼티 이다.
- 이를 사용해서 인스턴스 메소드 안에서 자기 인스턴스에 접근 할수 있다.

```
struct Point {
    var x = 0.0, y = 0.0
    func isToTheRightOf(x: Double) -> Bool {
        return self.x > x
    }
}
```

타입 메서드

- 타입 프로퍼티랑 마찬가지로 타입 자체에서 호출이 가능한 메서드.

- 메서드 앞에 static키워드를 사용하여 타입메서드를 작성할수 있다. 타입 프로퍼티와 마찬가지로 클래스에서는 class키워드를 사용해 타입메서드를 표현한다.

- 타입 메소드 안에서의 self는 인스턴스가 아닌 타입을 나타낸다.