Class & Struct (2)

G4ENG

11 JUL 19

Agenda

- Inheritance
- Subclassing
- Overriding
- Final
- Type Casting
- Any, AnyObject

Inheritance

- 상속이란 클래스가 구조체와 구분되는 특성 중 하나로 말 그대로 하나의 클래스가 다른 클래스에게 무엇인가를 물려줄 수 있다는 것.
- 이미 만들어진 다른 클래스의 기능이나 프로퍼티를 직접 구현하지 않고도 사용 가능, 다만 추가로 필요한 기능이나 프로퍼티만 정의하여 사용하면 됨.
- 이때 물려주는 클래스를 부모 클래스, 상속받는 클래스를 자식 클래스라고 한다.

SubClassing

```
class A {
    var name = "Class A"

    var description: String {
        return "This class name is \(self.name)"
    }

    func foo() {
        print("\(self.name)'s method foo is called")
    }
}
```

SubClassing

```
class B: A {
    var prop = "Class B"

    func boo() -> String {
        return "Class B prop = \(self.prop)"
    }
}

let b = B()
b.prop
b.boo()
// 서브클래싱을 통해 클래스 A의 자식 클래스가 된 클래스 B에는 현재 prop 프로퍼티와 boo()가 선언됨.
    기본적으로 클래스 B의 인스턴스는 자신의 클래스에서 선언된 프로퍼티와 메소드를 자유롭게 이용가능.
b.name
b.foo()

b.name = "Class C"
b.foo()
```

Overriding

- 부모 클래스로부터 상속받은 프로퍼티나 메소드를 필요에 의해 다시 구현하거나 재정의해서 사용할때 쓰는 키워드
- 부모키워드에 없는 프로퍼티나 메소드를 override한다면 에러
- 프로퍼티를 오버라이딩할 때는 상위 클래스에서 저장 프로퍼티였건, 연산 프로퍼티였건 연산 프로퍼티의 형태로 오버라이딩함

Overriding

- 프로퍼티 오버라이딩 시 허용되는 것
  - 저장 프로퍼티를 get, set 구문이 모두 있는 연산 프로퍼티로 오버라이딩
  - get, set 구문이 모두 제공되는 연산 프로퍼티를 get, set 구문이 모두 제공되는 연산 프로퍼티로 오버라이딩
  - get 구문만 제동되는 연산 프로퍼티를 get, set 구문이 모두 제공되는 연산 프로퍼티로 오버라이딩
  - get 구문만 제공되는 연산 프로퍼티를 get 구문만 제공되는 연산 프로퍼티로 오버라이딩

overriding

```
class Car: Vehicle {
    var gear = 0
    var engineLevel = 0

    override var currentSpeed: Double {
        get {
            return Double(self.engineLevel * 50)
        }
        set {
            // 아무것도 하지 않음 (newValue로 지어짐 이름)
        }
    }

    override var description: String {
        get {
            return "Car: engineLevel = \(self.engineLevel), so currentSpeed = \(self.currentSpeed)"
        }
        set {
            print("New Vehicle is \(newValue)")
        }
    }
}
```

Overriding

- 메소드 오버라이딩
  - 대상이 되는 메소드의 매개변수 개수나 타입, 그리고 반환 타입을 변경할 수 없다. 상위 클래스에서 정의된 메소드의 반환 타입을 그대로 유지해야 한다.
  - 변경할 수 있는 것은 오로지 내부 구문뿐

Overriding

```
class AutomaticCar: Car {
    override var currentSpeed: Double {
        didSet {
            self.gear = Int(currentSpeed / 10.0) + 1
        }
    }
}
```

```
class Bike: Vehicle {
    override func makeNoise() {
        print("빠라빠라빠라밤")
    }
}

let bk = Bike()
bk.makeNoise()
```

Final

- Java의 final과 비슷한 개념
- 클래스에 final을 붙이면 상속 불가
- 프로퍼티나 메소드에 붙이면 오버라이딩 불가

Type Casting

- Up Casting
  - 객체 as 변환할 타입
- Down Casting
  - 객체 as? 변환할 타입 (옵셔널 타입으로 반환)
  - 객체 as! 변환할 타입 (강제적 옵셔널 해제로 일반 타입으로 반환)

Type Casting

Vehicle1 -> Car1 -> SUV

```
let anyCar: Car1 = SUV()
let anyVehicle = anyCar as Vehicle1
```

Type Casting

```
let anySUV = anyCar as? SUV
if anySUV != nil {
    print("\(anySUV!) 캐스팅이 성공하였습니다")
}
```

```
if let anySUV = anyCar as? SUV {
    print("\(anySUV) 캐스팅 성공")
}
```

```
let anySUV1 = anyCar as! SUV
```

Any, AnyObject

- AnyObject
  - 모든 클래스를 담을 수 있는 타입
- Any
  - 모든 타입을 담을 수 있는 타입

Any, AnyObject

```
let obj: AnyObject = SUV()

if let suv = obj as? SUV {
    print("\(suv) 캐스팅 성공")
}
```

Any, AnyObject

```swift
var value: Any = "Sample String"
value = 3
value = false
value = [1,3,5,7,9]
value = {
    print("함수 실행")
}

func name(_ param: Any) {
    print("\(param)")
}

name(3)
name(false)
name([1,3,5,7,9])
name {
    print(">>>")
}
```

```
"Sample String"
3
false
[1, 3, 5, 7, 9]
() -> ()


(4 times)
```

QNA

끝