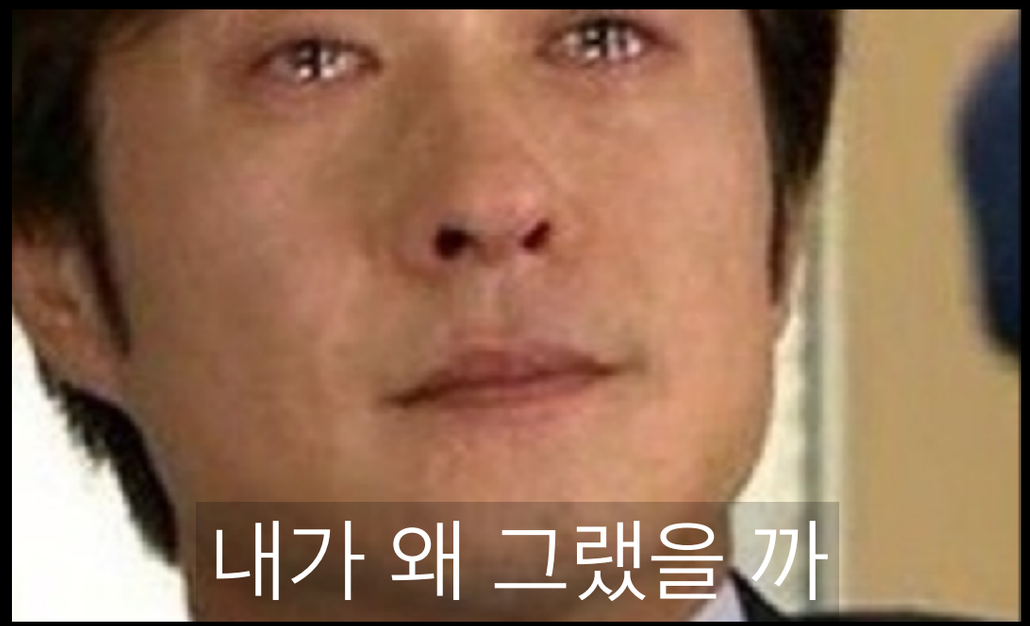
The Swift Programming Language

Protocols

osxdev (hothead)
성기평 (sungkipyung@gmail.com)



시작 전 공지

https://github.com/sungkipyung/Swift_Protocol_osxdev

Protocol Syntax

```
// 요렇게 쓰면 됩니다.
protocol SomeProtocol {
    // protocol definition goes here
}
```

Protocol Syntax

```
protocol FirstProtocol {

}

protocol AnotherProtocol {

}

// 이렇게 여러개를 한꺼번에 붙일 수 있어요.
struct SomeStructure: FirstProtocol, AnotherProtocol {
    // structure definition goes here
}
```

Protocol Syntax

```
// 클래스도 되네요!
class SomeClass: SomeSuperclass, FirstProtocol, AnotherProtocol {
    // class definition goes here
}

// enum 도 됩니다!
enum SomeEnum: FirstProtocol, AnotherProtocol {
    case born, to, study
}

// Protocl도 됩니다.!
protocol NewProtocol: FirstProtocol, AnotherProtocol {

}
```

Property Requirements

```
protocol SomeProtocol {
    var mustBeSettable: Int { get set }
    var doesNotNeedToBeSettable: Int { get }
}

// 흠 문제는 없어보입니다!
class ImplementSomeProtocol: SomeProtocol {
    internal var mustBeSettable: Int = 0
    internal var doesNotNeedToBeSettable: Int = 0
}
```

Property Requirements

```
protocol SomeProtocol {
    var mustBeSettable: Int { get set }
    var doesNotNeedToBeSettable: Int { get }
}

// 흠 문제는 없어보입니다!
class ImplementSomeProtocol: SomeProtocol {
    internal var mustBeSettable: Int = 0
    internal var doesNotNeedToBeSettable: Int = 0
}
```

Property Requirements

```
protocol SomeProtocol {
    var mustBeSettable: Int { get set }
    var doesNotNeedToBeSettable: Int { get }
}

// 호기심 발동!
// Settable하지 않게 만들면 어떻게 될까요?
class DoseNotFollowRequirements: SomeProtocol {
    private(set) internal var mustBeSettable: Int = 0


    private(set) internal var doesNotNeedToBeSettable: Int = 0
}
```

Property Requirements

```
protocol SomeProtocol {
    var mustBeSettable: Int { get set }
    var doesNotNeedToBeSettable: Int { get }
}

// 호기심 발동!
// Settable하지 않게 만들면 어떻게 될까요?
class DoseNotFollowRequirements: SomeProtocol {
    private(set) internal var mustBeSettable: Int = 0
```

Setter for property 'mustBeSettable' must be declared internal because it matches a requirement in internal protocol 'SomeProtocol'

```
    private(set) internal var doesNotNeedToBeSettable: Int = 0
}
```

Property Requirements

```
// static property를 가지는게 조건이네요
protocol AnotherProtocol {
    static var someTypeProperty: Int { get set }
}

class FollowStaticPropertyRequirement: AnotherProtocol {
    internal var someTypeProperty: Int = 0

    //    internal static var someTypeProperty: Int = 0
}
```

Property Requirements

```
// static property를 가지는게 조건이네요
protocol AnotherProtocol {
    static var someTypeProperty: Int { get set }
}

class FollowStaticPropertyRequirement: AnotherProtocol {
    internal var someTypeProperty: Int = 0
    Type 'FollowStaticPropertyRequirement' does not conform to protocol 'AnotherProtocol'
    //    internal static var someTypeProperty: Int = 0
}
```

Method Requirements

```
protocol RandomNumberGenerator {
    func random() -> Double
}

class LinearCongruentialGenerator: RandomNumberGenerator {
    var lastRandom = 42.0
    let m = 139968.0
    let a = 3877.0
    let c = 29573.0
    func random() -> Double {
        lastRandom = ((lastRandom * a + c).truncatingRemainder(dividingBy:m))
        return lastRandom / m
    }
}
let generator = LinearCongruentialGenerator()
print("Here's a random number: \(generator.random())")
// Prints "Here's a random number: 0.37464991998171"
print("And another one: \(generator.random())")
// Prints "And another one: 0.729023776863283"
```

Mutating Method Requirements

```
protocol Togglable {
    mutating func toggle()
}

enum OnOffSwitch: Togglable {
    case off, on
    mutating func toggle() {
        switch self {
        case .off:
            self = .on
        case .on:
            self = .off
        }
    }
}
var lightSwitch = OnOffSwitch.off
lightSwitch.toggle()
// lightSwitch is now equal to .on
```

Initializer Requirements

```
protocol SomeProtocol {
    init(someParameter: Int)
}

class SomeClass: SomeProtocol {
    // initializer는 required가 필요합니다.
    required init(someParameter: Int) {
        // initializer implementation goes here
    }
}
```

Initializer Requirements

```
protocol SomeProtocol {
    func someFunction()
}

class SomeSuperClass {
    func someFunction() {
        print("someFunction called")
    }
}

class SomeSubClass: SomeSuperClass, SomeProtocol {
    // 이미 구현된 메서드라 오류가 발생하지 않습니다.
}

let obj = SomeSubClass()
obj.someFunction()
```

Initializer Requirements

```
protocol SomeProtocol {
    func someFunction()
    init()
}

class SomeSuperClass {
    func someFunction() {
        print("someFunction called")
    }

    init() {

    }
}

class SomeSubClass: SomeSuperClass, SomeProtocol {
    // someFunction은 오류가 발생하지 않습니다.
    // 하지만 initializer는 오류가 발생하는데
    // 문맥적으로 해석하면
    // initializer를 Protocol에 선언하면 override를 요구하는 것입니다.

//    override required init() {
//
//    }
}
```

Initializer requirement 'init()' can only be satisfied by a `required` initializer in non-final class 'SomeSubClass'

Initializer Requirements

```
protocol SomeProtocol {
    func someFunction()
    init()
}

class SomeSuperClass {
    func someFunction() {
        print("someFunction called")
    }

    init() {

    }
}

class SomeSubClass: SomeSuperClass, SomeProtocol {
    // someFunction은 오류가 발생하지 않습니다.
    // 하지만 initializer는 오류가 발생하는데
    // 문맥적으로 해석하면
    // initializer를 Protocol에 선언하면 override를 요구하는 것입니다.

    override required init() {

    }
}
```

Failable Initializer Requirements

```
protocol FailableInitializer {
    init?(initFail: Bool)
}

class SomeFailableInitClass: FailableInitializer {
//    required init(initFail: Bool) {
//        if initFail {
//            return nil
//        }
//    }
    required init?(initFail: Bool) {
        if initFail {
            print("init failed")
            return nil
        }
    }
    var text = "SomeText"
}

var optionalObj = SomeFailableInitClass(initFail: true)
optionalObj?.text
//optionalObj.text
```

Failable Initializer Requirements

```
class SomeFailableInitClass: FailableInitializer {
    required init(initFail: Bool) {
        if initFail {
            return nil
        }
    }
//    required init?(initFail: Bool) {
//        if initFail {
//            print("init failed")
//            return nil
//        }
//    }
    var text = "SomeText"
}

var optionalObj = SomeFailableInitClass(initFail: true)
//optionalObj?.text
optionalObj.text
```

Only a failable initializer can return 'nil'

언제? 왜? 사용해야 하나?

다형성 구글 검색

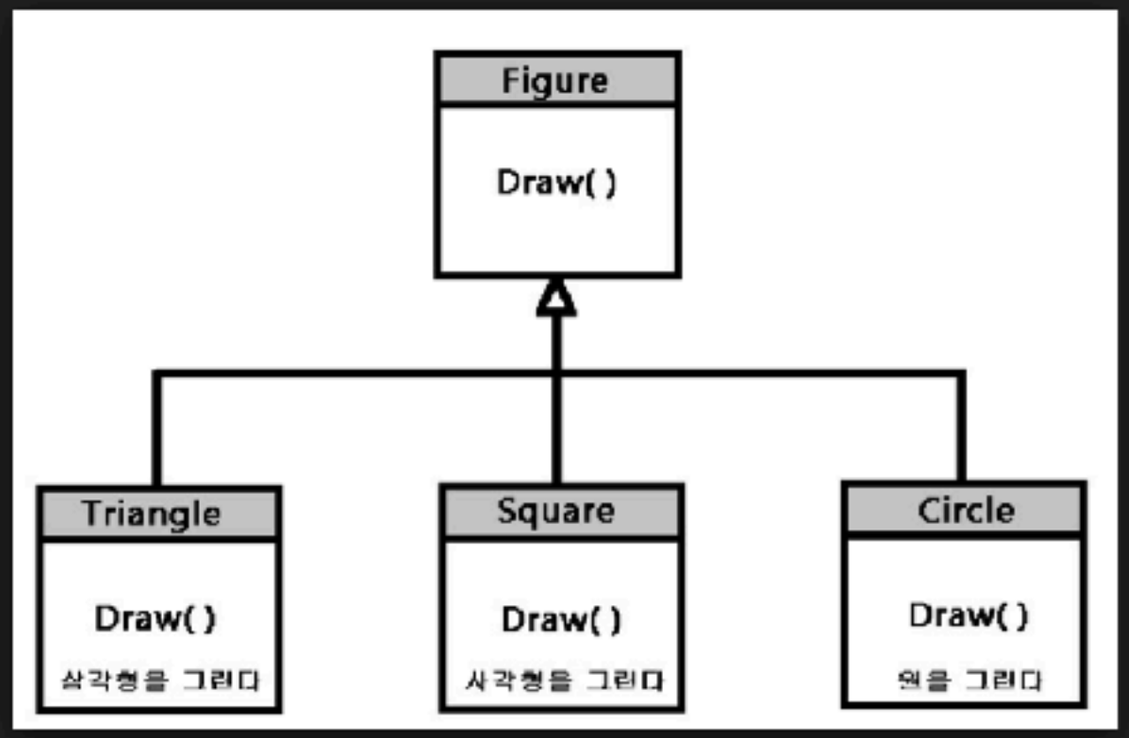


Figure
Draw( )
Triangle
Draw( )
삼각형을 그리다
Square
Draw( )
사각형을 그리다
Circle
Draw( )
원을 그리다


C++] 상속과 다형성 (Inheritance & Polymorphism)
Tistory · 719 × 469 · 이미지로 검색
Figure 클래스는 하위 클래스에서 모형을 그리는데 사용될 수 있는 draw 함수를 가지고 있다. 그러나 Figure 클래스에는 실제 도형을 그리는 함수 구현 부분을 정의할 ...
페이지 방문
이미지 보기
공유
관련 이미지
더보기

좋은 예제가 있는데 집에서 보세요

https://developer.apple.com/library/prerelease/content/documentation/Swift/Conceptual/Swift_Programming_Language/Protocols.html#//apple_ref/doc/uid/TP40014097-CH25-ID275

Class-Only Protocols

```
// gallery view controller를 띄워서
// 사진을 하나 선택 해온다고 가정해볼께요.
protocol GalleryViewControllerDelegate {
    func galleryViewController(_ vc: GalleryViewController, didSelectImage image: UIImage)
}

class GalleryViewController: UIViewController {
    var delegate: GalleryViewControllerDelegate?

    func didSelectImage(_ image: UIImage) {
        delegate?.galleryViewController(self, didSelectImage: image)

        /**
         circular reference
         */
        dismiss(animated: true, completion: nil)

    }
}

class MyViewController: UIViewController {
//    @IBAction
    func touchUpPresentGalleryButton(_ sender: UIButton) {
        guard let destination = self.storyboard?.instantiateViewController(withIdentifier: "GalleryViewController") as?
GalleryViewController else { return }

        destination.delegate = self

        self.present(destination, animated: true, completion: nil)
    }
}

extension MyViewController: GalleryViewControllerDelegate {
    func galleryViewController(_ vc: GalleryViewController, didSelectImage image: UIImage) {
        // TODO: Handle selected image
    }
}
```

Class-Only Protocols

```
// delegate가 weak가 되어야만 상호 참조를 막을 수 있어요.
protocol GalleryViewControllerDelegate {
    func galleryViewController(_ vc: GalleryViewController, didSelectImage image:
UIImage)
}

class GalleryViewController: UIViewController {
    weak var delegate: GalleryViewControllerDelegate?
    ! 'weak' may only be applied to class and class-bound protocol types, not 'GalleryViewControllerDelegate'
    func didSelectImage(_ image: UIImage) {
        delegate?.galleryViewController(self, didSelectImage: image)

        /**
         circular reference
         */
        dismiss(animated: true, completion: nil)

    }
}
```

Class-Only Protocols

```
// class only protocol만이 weak로 선언 될 수 있네요.
// 일종의 컴파일러에게 주는 힌트인가 봅니다.
protocol GalleryViewControllerDelegate: class {
    func galleryViewController(_ vc: GalleryViewController, didSelectImage image:
UIImage)
}

class GalleryViewController: UIViewController {
    weak var delegate: GalleryViewControllerDelegate?

    func didSelectImage(_ image: UIImage) {
        delegate?.galleryViewController(self, didSelectImage: image)

        /**
         circular reference
         */
        dismiss(animated: true, completion: nil)

    }
}
```

Protocol Composition

```
protocol Named {
    var name: String { get }
}

protocol Aged {
    var age: Int { get }
}

struct Person: Named, Aged {
    var name: String
    var age: Int
}
struct Object: Named {
    var name: String
}

func wishHappyBirthday(to celebrator: Named & Aged) {
    print("Happy birthday, \(celebrator.name), you're \(celebrator.age)!")
}

let birthdayPerson = Person(name: "Malcolm", age: 21)
wishHappyBirthday(to: birthdayPerson)
// Prints "Happy birthday, Malcolm, you're 21!"

let someObj = Object(name: "SomeObj")
//wishHappyBirthday(to: someObj)
```

Checking for Protocol Conformance

```
protocol HasArea {
    var area: Double { get }
}
class Circle: HasArea {
    let pi = 3.1415927
    var radius: Double
    var area: Double { return pi * radius * radius }
    init(radius: Double) { self.radius = radius }
}
class Country: HasArea {
    var area: Double
    init(area: Double) { self.area = area }
}
class Animal {
    var legs: Int
    init(legs: Int) { self.legs = legs }
}
let objects: [AnyObject] = [
    Circle(radius: 2.0),
    Country(area: 243_610),
    Animal(legs: 4)
]
for object in objects {
    if let objectWithArea = object as? HasArea {
        print("Area is \(objectWithArea.area)")
    } else {
        print("Something that doesn't have an area")
    }
}
// Area is 12.5663708
// Area is 243610.0
// Something that doesn't have an area
```

Optional Protocol Requirements

```
@objc protocol CounterDataSource {
    @objc optional func increment(forCount count: Int) -> Int
    @objc optional var fixedIncrement: Int { get }
}

class Counter {
    var count = 0
    var dataSource: CounterDataSource?

    func increment() {
        // increment? 는 Optional chaining이 적용된 것을 볼 수 있습니다.
        // increment protocol은 optional method 거든요

        if let amount = dataSource?.increment?(forCount: count) {
            count += amount
        } else if let amount = dataSource?.fixedIncrement {
            count += amount
        }
    }
}

class ThreeSource: NSObject, CounterDataSource {
    let fixedIncrement = 3
    // increment() was implemented
}

var counter = Counter()
counter.dataSource = ThreeSource()
for _ in 1...4 {
    counter.increment()
    print(counter.count)
}
// 3
// 6
// 9
// 12
```

Providing Default Implementations

```
protocol TextRepresentable {
    var textualDescription: String { get set }
}

protocol PrettyTextRepresentable: TextRepresentable {
    var prettyTextualDescription: String { get }
}

extension PrettyTextRepresentable {
    var prettyTextualDescription: String {
        return textualDescription
    }
}
```

Adding Constraints to Protocol Extensions

```
protocol TextRepresentable {
    var textualDescription: String { get }
}

extension Collection where Iterator.Element: TextRepresentable {
    var textualDescription: String {
        let itemsAsText = self.map { $0.textualDescription }
        return "[" + itemsAsText.joined(separator: ", ") + "]"
    }
}

class Person: TextRepresentable {
    internal var textualDescription: String {
        return "My name is \(name)"
    }

    var name: String

    init(name: String) {
        self.name = name
    }
}

var peoples: [Person] = [Person(name: "Tom"), Person(name: "Bob")]
//var peoples: [TextRepresentable] = [Person(name: "Tom"), Person(name: "Bob")]
let result = peoples.textualDescription
print(result)
// "[My name is Tom, My name is Bob]"
```

Reference

- Protocols (apple document)
- https://github.com/sungkipyung/Swift_Protocol_osxdev