Delegate Protocol

아서 & 재민

목차

Delegation Pattern

- 특정 기능들을 다른 객체에 위임하여 필요한 시점에 메소드의 호출만 받는 디자인 패턴

- 객체지향 방식의 코드설계에서 효율적으로 이벤트를 관리할 수 있음
  - '프로토콜' 객체를 활용

나는 프로토콜이 뭔줄이나 알고

델리게이션 패턴에 대해 떠들어대는 것인가?

프로토콜 ?

#계약 #청사진 #설계도 #기준

프로토콜 ?

Conform : 따르다, 준수하다

프로토콜을 conform한다?
특정 객체가 갖추어야할 기능이나 속성요건을 충족한다

프로토콜 ?

1. 채 용 개 요

❒ 채용 분야 : 경호분야

❒ 채용 인원 : OO명

❒ 채용 직급 : 특정직 7급(경호주사보)

❒ 응시 자격

구 분	내 용
임 용 자 격	.대한민국 국적 소지자(대통령 등의 경호에 관한 법률 제8조) .국가공무원법 제33조(결격사유) 각 호에 해당하지 아니한 사람 .공무원임용시험령 및 기타 법령에 의해 응시자격이 정지당하지 아니한 사람
학력/성별	.제한 없음
연 령	.시험년도 기준 만30세 이하자('85. 1. 1 이후 출생자) - 제대군인 지원에 관한 법률 및 병역법 등에 의거 최대 3년까지 연장 가능 * 1년미만 근무 후 제대자는 1세, 1년이상 2년미만 근무 후 제대자는 2세, 2년이상 근무 후 제대자는 3세 연장 (2년 이상 근무 후 제대자의 경우 '82. 1. 1 이후 출생자)
병 역	.남자의 경우 병역을 필한 자(면제자 포함) - 현역의 경우 '16.12.31까지 전역가능자
신 체	.심신이 건강한 자로서 신장 및 시력 기준 적합자 - 신장 : 男 175cm 이상, 女 165cm 이상 - 시력 : 맨눈시력 0.8 이상(남녀 공통)
영 어	.공인영어성적 기준 점수 이상 보유자(2014.10. 1 이후 응시한 성적표) - 기준(택일) : 토익700, 텝스625, 토플(PBT530,CBT197,IBT71), 토익스피킹 레벨5, OPIc 레벨IM3, MATE스피킹 레벨4 이상

프로토콜을 구현할 수 있는 구현체

구조체

클래스

열거형

확장객체

구조체 vs 클래스

나는 구조체나 클래스에 대해 뭔줄이나 프로토콜 활용에 대해 떠들어대는 것인가?

구조체 vs 클래스

```
import UIKit

class CarClass {
    var col:String!
    init(color:String = "") {
        col = color
    }
}

struct CarStruct {
    var col:String!
    init(color:String = "") {
        col = color
    }
}

var car1 = CarClass(color:"red")          CarClass
var car2 = car1                           CarClass

car1.col = "black"                        CarClass
car2.col                                  "black"

var car3 = CarStruct(color:"blue")        CarStruct
var car4 = car3                           CarStruct

car3.col = "yellow"                       CarStruct
car4.col                                  "blue"
```

구조체
- Value Type

클래스
- Reference Type

구조체 vs 클래스

Inheritance

프로토콜을 준수하는 요건

프로퍼티

메소드

프로토콜 활용

```swift
import UIKit

protocol Vehicle {
    var isRunning: Bool { get set }
    mutating func start()
    mutating func turnOff()
}
```

프로토콜 활용

```
struct SportsCar: Vehicle {
    var isRunning: Bool = false

    mutating func start() {
        if isRunning {
            print("Already started!")
        } else {
            isRunning = true
            print("Vrooom")
        }
    }

    mutating func turnOff() {
        if isRunning {
            isRunning = false
            print("Crickets")
        } else {
            print("Already Dead")
        }
    }
}
```

구조체 활용

프로토콜 활용

```
struct SportsCar: Vehicle {   Type 'SportsCar' does not conform to protocol 'Vehicle'
    var isRunning: Bool = false

    mutating func start() {
        if isRunning {
            print("Already started!")
        } else {
            isRunning = true
            print("Vrooom")
        }
    }
}
```

프로토콜 활용

```
class SemiTruck: Vehicle {
    var isRunning: Bool = false

    func start() {
        if isRunning {
            print("Already started")
        } else {
            isRunning = true
            print("Rumble")
        }
    }

    func turnOff() {
        if isRunning {
            isRunning = false
            print("put put silence")
        } else {
            print("Already Dead")
        }
    }

    func blowAirHorn() {
        print("TOOOOOOT! ")
    }
}
```

클래스 활용

프로토콜 활용

```
var car1 = SportsCar()
var truck1 = SemiTruck()

car1.start()
truck1.start()
truck1.blowAirHorn()
car1.turnOff()
truck1.turnOff()
```

```
SportsCar
SemiTruck

SportsCar
SemiTruck
SemiTruck
SportsCar
SemiTruck
```

활용 결과

```
Vrooom
Rumble
TOOOOOT!
Crickets
put put silence
TOOOOOT!
```

프로토콜 활용

```
var vehicleArray: Array<Vehicle> = [car1, truck1]
for vehicle in vehicleArray {
    if let vehicle = vehicle as? SemiTruck {
        vehicle.blowAirHorn()
    }
}
```

```
[{isRunning false}, {isRunning false}]


SemiTruck
```

동일한 프로토콜을 준수하는 객체끼리
배열로 통합도 가능

프로토콜 활용

프로토콜 타입의 특성을 이용하여
델리게이션이라는 기능을 구현함

핵심 : 프로토콜 타입으로 선언되어 할당된 객체가 무슨 기능을 가지고 있는지 알 필요가 없으며 단지 할당된 객체를 이용하여 프로토콜에 정의된 프로퍼티나 메소드를 호출

프로토콜 활용

예시

- 동일한 연료펌프를 사용하는 자동차, 비행기, 버스
- 연료펌프는 연료가 꽉 차거나 부족할 때 알려주는 기능이 있음

프로토콜 활용

```
protocol FuelPumpDelegate {
    func lackFuel()
    func fullFuel()
}
```

프로토콜 제시

연료 부족할 때 알려주기

연료가 꽉 찼을 때 알려주기

프로토콜 활용

```
class FuelPump {
    var maxGage: Double = 100
    var delegate: FuelPumpDelegate? = nil

    var fuelGage: Double {
        didSet {
            if oldValue < 10 {
                self.delegate?.lackFuel()
            } else if oldValue == self.maxGage {
                self.delegate?.fullFuel()
            }
        }
    }

    init(fuelGage: Double = 0) {
        self.fuelGage = fuelGage
    }


    func startPump() {
        while(true) {
            if self.fuelGage > 0 {
                self.jetFuel()
            } else {
                break
            }
        }
    }

    func jetFuel() {
        self.fuelGage -= 1.0
    }
}
```

연료펌프 객체

- 펌핑 시작
- 펌핑이 되는 동안 연료의 양을 줄임
- 연료 꽉차거나 부족하면 알려줌

프로토콜 활용

```
class Car: FuelPumpDelegate {
    var fuelPump = FuelPump(fuelGage: 100)

    init() {
        self.fuelPump.delegate = self
    }

    func lackFuel() {
        // 연료가 부족할 때 뭐할래?
    }

    func fullFuel() {
        // 연료가 꽉차면 뭐할래?
    }

    func start() {
        fuelPump.startPump()
    }
}
```

자동차 객체

- 연료펌프 프로토콜을 따름
- 연료가 꽉차거나 부족할 때 취할 행동 구현

Delegate Pattern

- 객체지향 프로그래밍에서
  하나의 객체가 모든 일을 처리하는 것이 아니라
  처리해야 할 일 중 일부를 다른 객체에 넘기는 것을 말한다.

Delegate Pattern

- 델리게이트 프로토콜을 구현한다는 것은 프로토콜명을 덧붙이는 것만을 의미하지 않는다.

- 프로토콜에 정의된 메소드를 실질적으로 정의하는 것까지를 포함한다.

- 이어지는 작업은 해당 객체의 **델리게이트 속성**을 **뷰 컨트롤러와 연결**해야 한다

- 델리게이트 속성이란, 델리게이트 메소드가 구현되어 있는 객체를 의미한다.

- 특정 이벤트 발생 -> 델리게이트 메소드가 구현되어 있는 객체를 찾는다 -> 이를 위한 참조포인터가 저장되는 곳이 delegate 속성이다. -> delegate 속성에 저장된 인스턴스가 델리게이트 메소드를 구현한 것으로 인식하고, 필요한 메소드를 호출하는 것이다.

UITextFieldDelegate 사용 예제

```
class ViewController: UIViewController, UITextFieldDelegate {

    @IBOutlet var tf: UITextField!

    override func viewDidLoad() {
        /**
         * 입력값 속성 설정
         */
        self.tf.placeholder = "값을 입력하세요"
        self.tf.keyboardType = UIKeyboardType.alphabet
        self.tf.keyboardAppearance = UIKeyboardAppearance.dark
        self.tf.returnKeyType = UIReturnKeyType.join // 리턴키 타입은 "Join"
        self.tf.enablesReturnKeyAutomatically = true // 리턴키 자동 활성화 "On"

        /**
         * 스타일 설정
         */
        self.tf.borderStyle = UITextBorderStyle.line // 테두리 스타일 - 직선
        self.tf.backgroundColor = UIColor(white:0.87, alpha:1.0) // 배경 색상
        self.tf.contentVerticalAlignment = .center // 수직 방향 텍스트 배열 위차 - 가운데
        self.tf.contentHorizontalAlignment = .center // 수평 방향 텍스트 배열 위치 - 가운데
        self.tf.layer.borderColor = UIColor.darkGray.cgColor // 테두리 색상 - 회색
        self.tf.layer.borderWidth = 2.0 // 테두리 두께 - 1.0 픽셀

        //텍스트 필드를 최초 응답자로 지정
        self.tf.becomeFirstResponder()

        // Delegate 지정
        self.tf.delegate = self

}
```

- 텍스트필드의 **delegate**는 텍스트필드에 특정 이벤트가 발생했을 때 알려줄 대상 객체를 가리키는 속성이다.

  이 속성에 대입된 **self**는 **현재의 뷰 컨트롤러 인스턴스**를 의미한다.

- 텍스트필드가 자신의 델리게이트 객체에게 특정 이벤트가 발생했음을 알려줄 때에는 **델리게이트 메소드**를 이용한다. 이벤트마다 호출하기로 약속된 메소드가 정해져 있는데, 이를 현재 지정된 델리게이트 객체에서 찾아 호출한다.
  이를 위해 먼저 **델리게이트 객체에 지정된 메소드가 구현되어 있는지 확인한 후**, **구현되어 있다면 필요한 인자값을 담아 메소드를 호출**하고, **구현되어 있지 않다면 그대로 종료**하는 방식이 델리게이트 패턴이다.

텍스트필드에서 미리 정해진 특정 이벤트가 발생하면 현재의 뷰 컨트롤러에게 알려달라는 요청을 뜻한다.

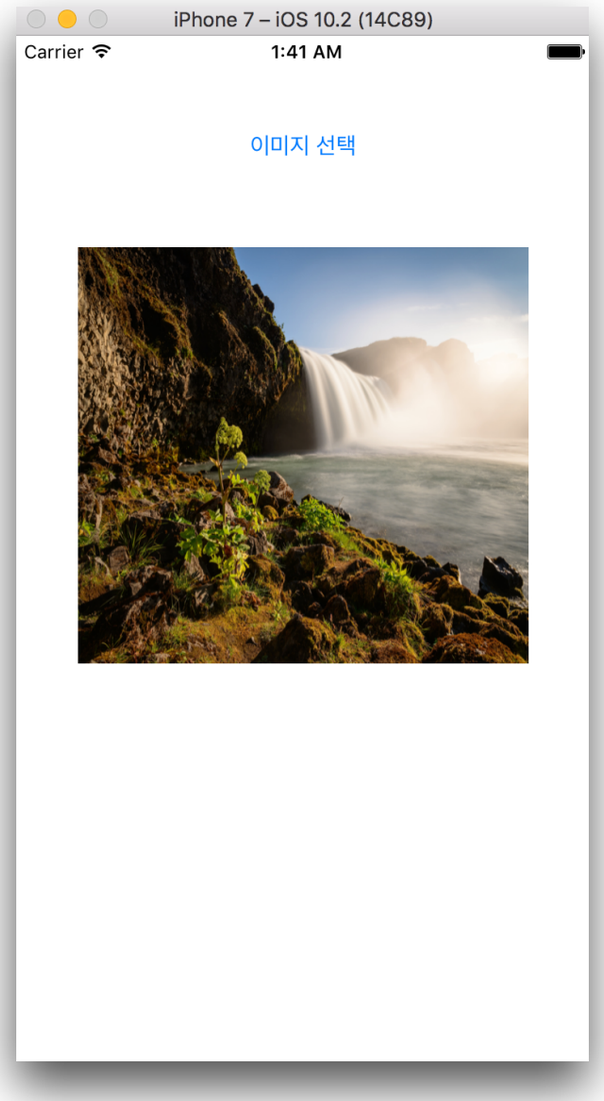
Image Picker Controller

- Image Picker Controller는 카메라나 앨범 등을 통해 이미지를 선택할 때 사용하는 컨트롤러

- 델리게이트 패턴을 활용하는 대표적인 객체

- 사용자가 카메라로 사진을 촬영하거나 앨범에서 사진을 선택하면 그 이미지 정보를 델리게이트로 지정된 객체에 메소드 호출을 통해 인자값으로 전달해주어 우리가 선택한 이미지를 사용할 수 있도록 해 준다.

UIImagePickerControllerDelegate, UINavigationControllerDelegate 사용




```
class ViewController: UIViewController, UIImagePickerControllerDelegate,
    UINavigationControllerDelegate {

    @IBOutlet var imageView: UIImageView!

    @IBAction func pick(_ sender: Any) {

        // 이미지 피커 컨트롤러 인스턴스 생성
        let picker = UIImagePickerController()

        picker.sourceType = .photoLibrary // 이미지 소스로 사진 라이브러리 선택
        picker.allowsEditing = true // 이미지 편집 기능 On

        // Delegate 지정
        picker.delegate = self

        self.present(picker, animated: false) // 이미지 피커 컨트롤러 실행
    }


    // 이미지 피커에서 이미지를 선택하지 않고 취소했을 때 호출되는 메소드
    func imagePickerControllerDidCancel(_ picker: UIImagePickerController) {

        picker.dismiss(animated: false) // 이미지 피커 컨트롤러 창 닫기

        let alert = UIAlertController(title: "",
                                      message: "이미지 선택이 취소되었습니다",
                                      preferredStyle: .alert)

        alert.addAction(UIAlertAction(title: "확인", style: .cancel))
        self.present(alert, animated: false)

    }

    // 이미지 피커에서 이미지를 선택했을 때 호출되는 메소드
    func imagePickerController(_ picker: UIImagePickerController,
        didFinishPickingMediaWithInfo info: [String : Any]) {

        picker.dismiss(animated: false) { () in

            let img = info[UIImagePickerControllerEditedImage] as? UIImage
            self.imageView.image = img

        }// 이미지 피커 컨트롤러 창 닫기
    }
```