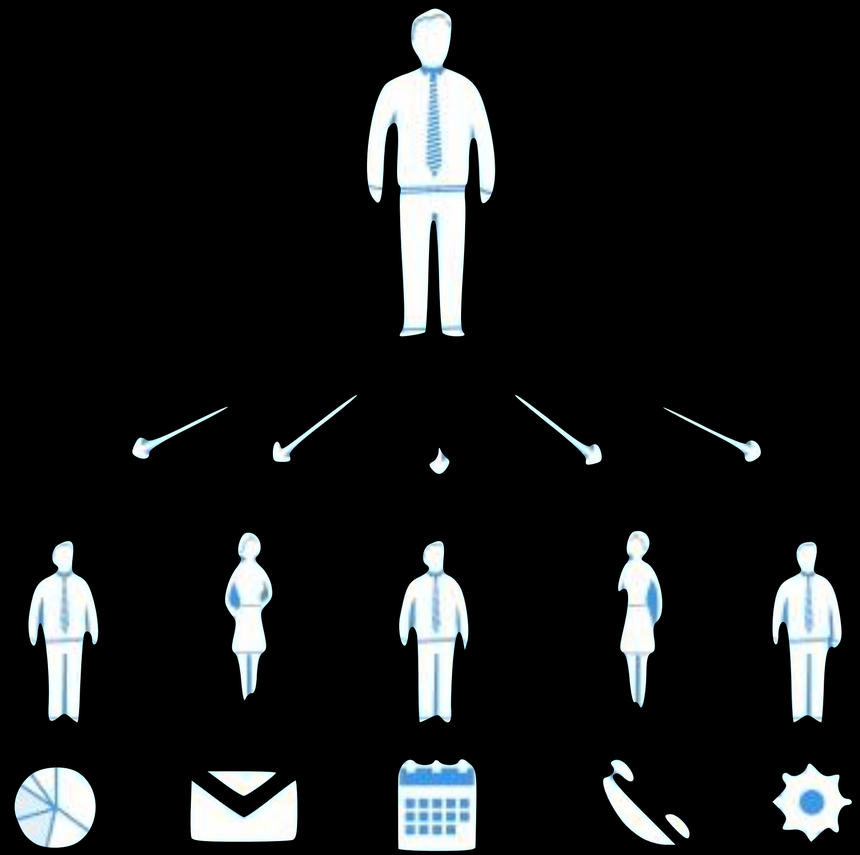
Delegate Parttern

G4ENG

05 AUG 19

Agenda

- Pattern
- Delegate Pattern
- First Responder
- TextField Delegate
- Image Picker Delegate



Pattern

- 일반적인 의미로는 반복해서 나타나는 사건이나 형태
- 객체지향 프로그래밍에서는 디자인 패턴이라는 합성 용어로 사용되는 경우가 대부분이다
- 여기서 디자인이란 프로그램 구성에 대한 디자인 즉, 구조적 설계를 의미

Pattern

- 컴퓨터 프로그래밍 공학에서는 다양한 설계 문제를 합리적으로 해결하기 위한 목적으로 디자인 패턴들이 등장
- 특정 객체의 값이 바뀌었을 때 여러 객체에 이를 알려주기 위한 방법, 구조를 바꾸지 않고 기능을 변경하거나 개선하기 위한 방법 등이 이에 해당하는 문제들
- 프로그래밍 과정에서 직면하게 되는 설계 문제를 해결해야 할 때, 디자인 패턴 중에서 적절한 것을 선택하여 그에 따라 프로그램을 구조화하면 쉽게 해결 가능

\

- 디자인 패턴의 종류
  - Factory Pattern
  - Observer Pattern
  - Singleton Pattern
  - Iterator Pattern
  - Delegate Pattern

Delegate Pattern

- 객체지향 프로그래밍에서 하나의 객체가 모든 일을 처리하는 것이 아니라 처리해야 할 일 중 일부를 다른 객체에 넘기는 것을 말함
- 한마디로 정리하면 “ 객체 너에게 모든 것을 일임할게. 뷰 컨트롤러인 나에게는 특정 이벤트가 발생했을 때 알려주기만 하면 돼”

Delegate Pattern

- 텍스트 필드는 델리게이트 패턴을 사용하는 대표적인 객체 중의 하나
- 기본적인 기능은 델리게이트 패턴 없이도 사용할 수 있지만
- 입력값을 제어하는 등의 고급 기능을 구현하고 싶을 때에는 델리게이트 패턴을 적용해야 한다

Delegate Pattern

- 델리게이트 패턴을 적용하려면 다음과 같은 두 가지 작업이 필요
  1. 텍스트 필드에 대한 델리게이트 프로토콜 구현
  2. 텍스트 필드의 델리게이트 속성을 뷰 컨트롤러에 연결

First Responder

- UIWindow는 이벤트가 발생했을 때 우선적으로 응답할 객체를 가리키는 최초 응답자라는 포인터를 가짐
- 최초 응답자에 관한 메소드
  - becomeFirstResponder()
  - resignFirstResponder()

First Responder

```
self.tf.becomeFirstResponder()
```

```
textField.resignFirstResponder()
```

TextField Delegate

Topics

Managing Editing

`func textFieldShouldBeginEditing(UITextField) -> Bool`

Asks the delegate if editing should begin in the specified text field.

Declaration

```
optional func textFieldShouldBeginEditing(_ textField: UITextField) -> Bool
```

Parameters

`textField`
The text field in which editing is about to begin.

Return Value

`true` if editing should begin or `false` if it should not.

Discussion

The text field calls this method when the user performs an action that would normally initiate the editing of the text field's text. Implement this method if you want to prevent editing from happening in some situations. For example, you could use this method to prevent the user from editing the text field's contents more than once. Most of the time, you should return `true` to allow editing to proceed.

If you do not implement this method, the text field acts as if this method had returned `true`.

Availability

iOS 2.0+
tvOS 9.0+

TextField Delegate

```
// delegate 지정
self.tf.delegate = self
// 텍스트 필드의 delegate는 텍스트 필드에 특정 이벤트가 발생했을 때 알려줄 대상 객체를 가르키는 속성. 이 속성에 self는 현재의 뷰
    컨트롤러 인스턴스를 의미.
```

TextField Delegate

```swift
// MARK: - UITextFieldDelegate
extension ViewController: UITextFieldDelegate {

    // 텍스트 필드의 편집이 시작된 후 호출
    func textFieldShouldBeginEditing(_ textField: UITextField) -> Bool {
        print("텍스트 필드의 편집이 시작됨")
        return true // false를 리턴하면 편집되지 않음
    }

    // 텍스트 필드의 내용이 삭제될 때
    func textFieldShouldClear(_ textField: UITextField) -> Bool {
        print("텍스트 필드의 내용이 삭제됨")
        return true
    }

    // 텍스트 필드의 내요잉 변경될 때 호출
    func textField(_ textField: UITextField, shouldChangeCharactersIn range: NSRange, replacementString string: String)
        -> Bool {
        print("텍스트 필드의 내용이 \(string)으로 변경됨")
        if Int(string) == nil { // 입력된 값이 숫자가 아니면
            // 현재 텍스트 필드에 입력된 길이와 더해질 문자열 길이의 합이 10을 넘는다면 반영하지 않음
            if (textField.text?.count)! + string.count > 10 {
                return false
            }
            else {
                return true
            }
        }
        else {
            return false
        }
    }

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        textField.resignFirstResponder()
        print("텍스트 필드의 편집 종료")
        return true
    }

    // 텍스트 필드의 편집이 종료될 때 호출
    func textFieldShouldEndEditing(_ textField: UITextField) -> Bool {
        print("텍스트 필드의 편집이 종료됨")
        return true
    }

    // 텍스트 필드의 편집이 종료되었을 때 호출
    func textFieldDidEndEditing(_ textField: UITextField) {
        print("편집 종료")
    }

}
```

Image Picker Delegate

Topics

Closing the Picker

```
func imagePickerController(UIImagePickerController, didFinishPickingMediaWithInfo: [UIImagePickerController.InfoKey : Any])
```

Tells the delegate that the user picked a still image or movie.

Declaration

```
optional func imagePickerController(_ picker: UIImagePickerController, didFinishPickingMediaWithInfo info:
[UIImagePickerController.InfoKey : Any])
```

Parameters

`picker`
The controller object managing the image picker interface.

`info`
A dictionary containing the original image and the edited image, if an image was picked; or a filesystem URL for the movie, if a movie was picked. The dictionary also contains any relevant editing information. The keys for this dictionary are listed in `Editing Information Keys`.

Discussion

Your delegate object's implementation of this method should pass the specified media on to any custom code that needs it, and should then dismiss the picker view.

When editing is enabled, the image picker view presents the user with a preview of the currently selected image or movie along with controls for modifying it. (This behavior is managed by the picker view prior to calling this method.) If the user modifies the image or movie, the editing information is available in the `info` parameter. The original image is also returned in the `info` parameter.

If you set the image picker's `showsCameraControls` property to `false` and provide your own custom controls, you can take multiple pictures before dismissing the image picker interface. However, if you set that property to `true`, your delegate must dismiss the image picker interface after the user takes one picture or cancels the operation.

Implementation of this method is optional, but expected.

Availability

iOS 3.0+

Image Picker Delegate

```swift
@IBAction func pick(_ sender: Any) {
    // 이미지 피커 컨트롤러 생성
    let picker = UIImagePickerController()
    picker.sourceType = .photoLibrary
    picker.allowsEditing = true

    // 델리게이트 지정
    picker.delegate = self

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

Image Picker Delegate

```swift
// MARK:- 이미지 피커 컨트롤러 델리게이트 메소드
// 익스텐션은 클래스를 대신해서 프로토콜을 구현할 수 있기 때문에, 델리게이트 패턴에 사용되는 프로토콜을 익스텐션에서 구현하면 하나의 뷰 컨트롤러
    클래스에 여러 프로토콜 메소드가 난립하는 것을 방지할 수 있다.
extension ViewController: UIImagePickerControllerDelegate {

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

        // 왜 picker.presentingViewController?.dismiss를 안씀?
        // 어짜피 가리키는 대상이 자기 자신이므로 self써도 무방.
        // 하지만 picker를 쓴건 self.presentingViewController쪽으로 알아서 연결시켜줌
        picker.dismiss(animated: false) { () in
            // 알림창 호출
            let alert = UIAlertController(title: "", message: "이미지 선택이 취소됨", preferredStyle: .alert)

            alert.addAction(UIAlertAction(title: "확인", style: .cancel, handler: nil))
            self.present(alert, animated: false)
        }
        // 충돌을 피하고자 dismiss 끝나고 할 행동을 completion 매개변수에 클로저로 넣어준다
    }

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

        // 이미지 피커 컨트롤러 창 닫기
        picker.dismiss(animated: false) { () in
            // 이미지를 이미지뷰에 복사
            let img = info[UIImagePickerController.InfoKey.editedImage] as? UIImage
            // 딕셔터리 타입으로 정의된 매개변수 info에는 사용자가 선택한 이미지 정보가 담겨서 전달되기 때문에, 앞에서 살펴본 이미지 관련
                키를 사용해서 원하는 이미지 정보를 추출. 위에서 사용하고 있는 것은 편집된 이미지 데이터다. 이를 위해
                UIImagePickerControllerEditedImage 상수값이 사용됨
            // 읽어온 값은 이미지 데이터를 담고 있지만 아직 범용 객체인 Any 타입이다. 따라서 UIImage로 다운캐스팅해야 사용 가능
            self.imgView.image = img
        }
    }
}
```

QNA