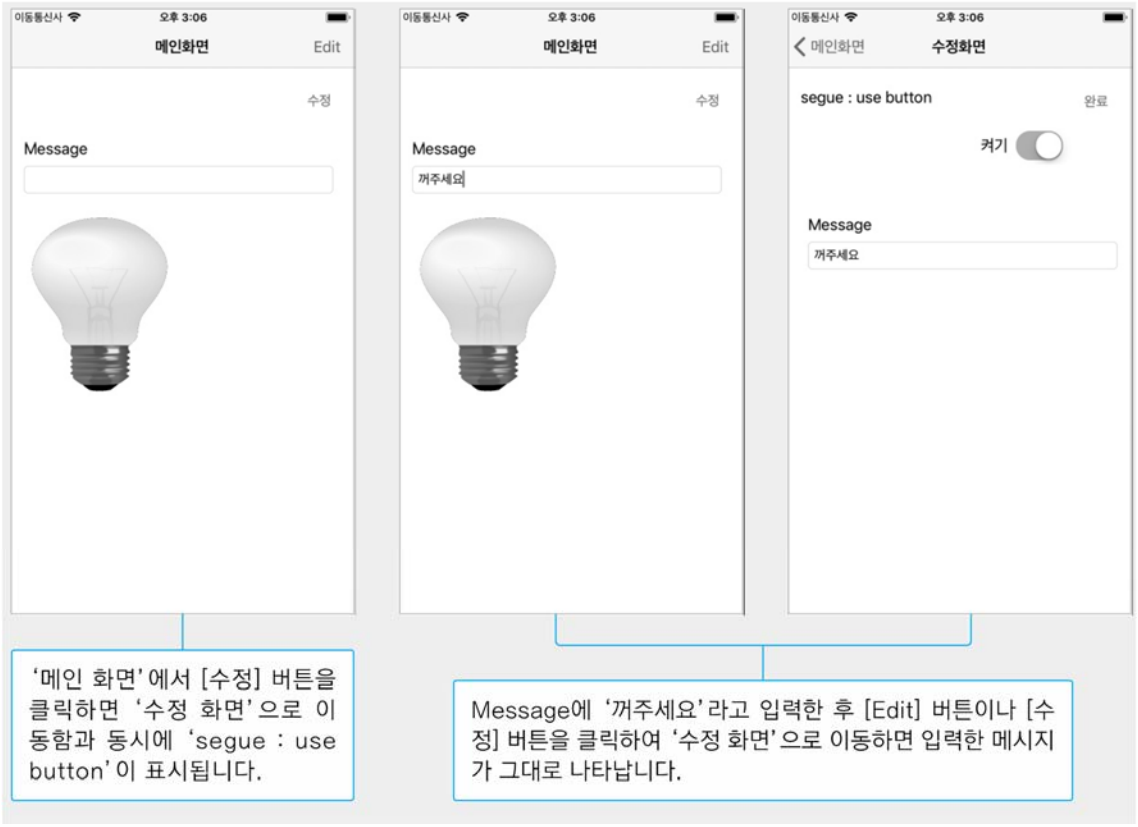
11 장. 내비게이션 컨트롤러 이용해 화면 전환하기

iOS 앱이나 안드로이드 앱에서 가장 많이 사용하는 부분이 화면 전환일 것이다. 한 화면에서 모든 동작을 구현하면 좋지만 대부분은 화면을 전환해서 앱에서 필요한 동작을 구현한다. 또한 화면이 전환되면 데이터도 함께 전달되어야 하는 경우가 대부분일 것이다.

화면 전환 방법과 데이터 전송 방법을 알아보자.



11-1 내비게이션 컨트롤러란?

화면과 화면을 전환할 때 연관성이 많거나 데이터를 서로 주고받아야 하는 경우에는 어떻게 해야할까? 그런 경우 사용하는 것이 바로 내비게이션 컨트롤러이다.

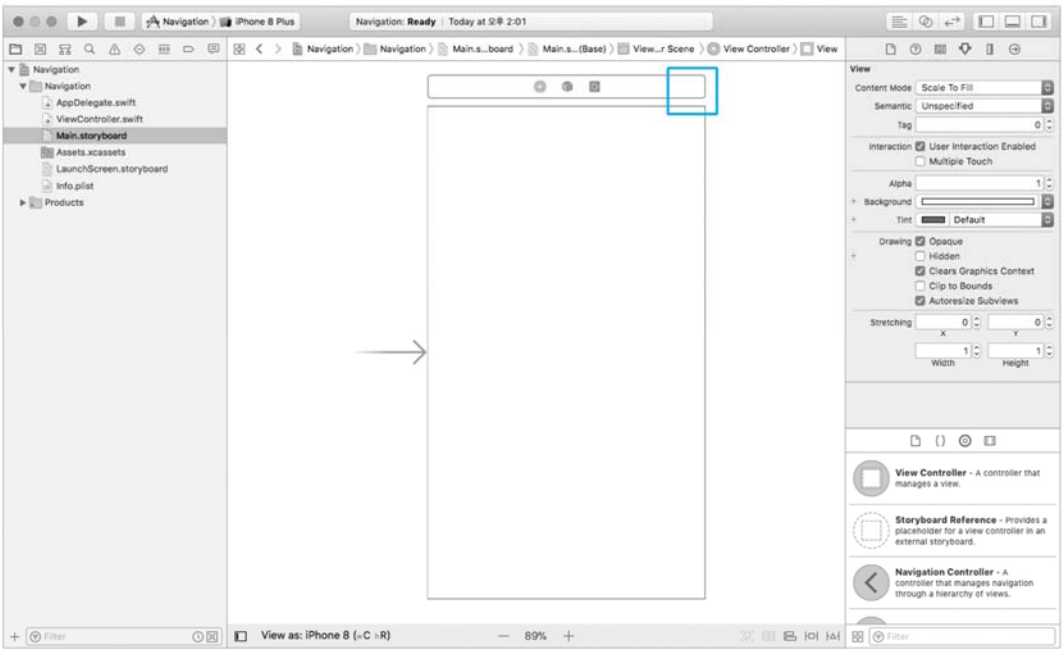
내비게이션 컨트롤러를 이용한 대표적인 앱이 미 피트이다. 메인 화면에 간단하게 총 걸음 수가 표시되고, 바 버튼을 클릭하면 사용자 정보 화면으로 전환되고, 총 걸음 수를 클릭하면 하루 걷기에 관련된 자세한 설명 화면으로 전환된다. 그리고 바 버튼을 클릭하면 메인 화면으로 돌아온다. 이처럼 내비게이션 컨트롤러를 이용하면 한 화면에서 다른 화면으로 쉽게 전환할 수 있다.

미 피트(Mi Fit) 앱

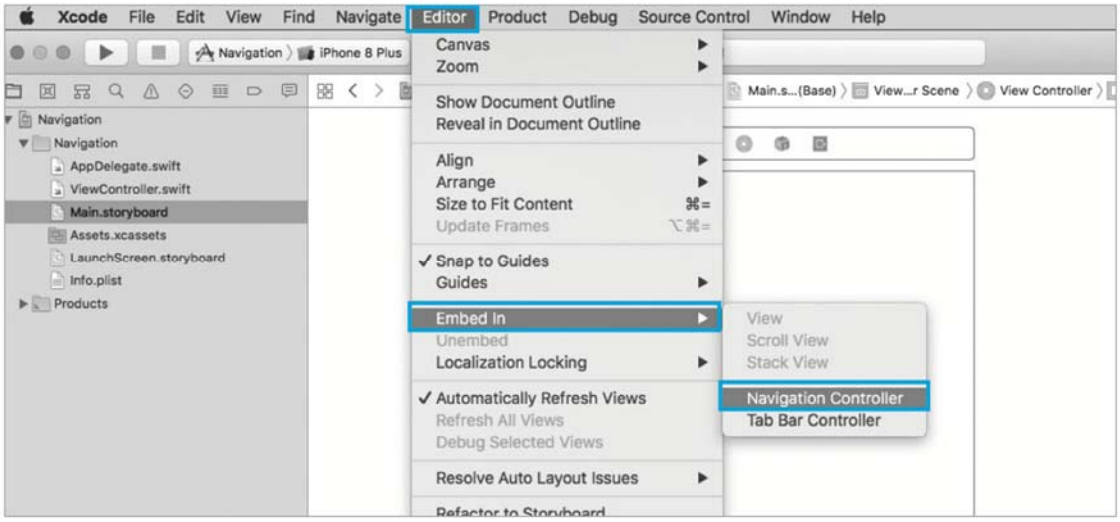
11-2. 내비게이션 컨트롤러 앱을 위한 기본 환경 구성하기

동작 화면은 '메인 화면'과 '수정 화면' 두 개로 만든다. 또한 '메인 화면'에서 '수정 화면'으로 메시지를 전달할 수 있게 하고, 전구의 상태를 '수정 화면'에서 제어할 수 있도록 구현한다.

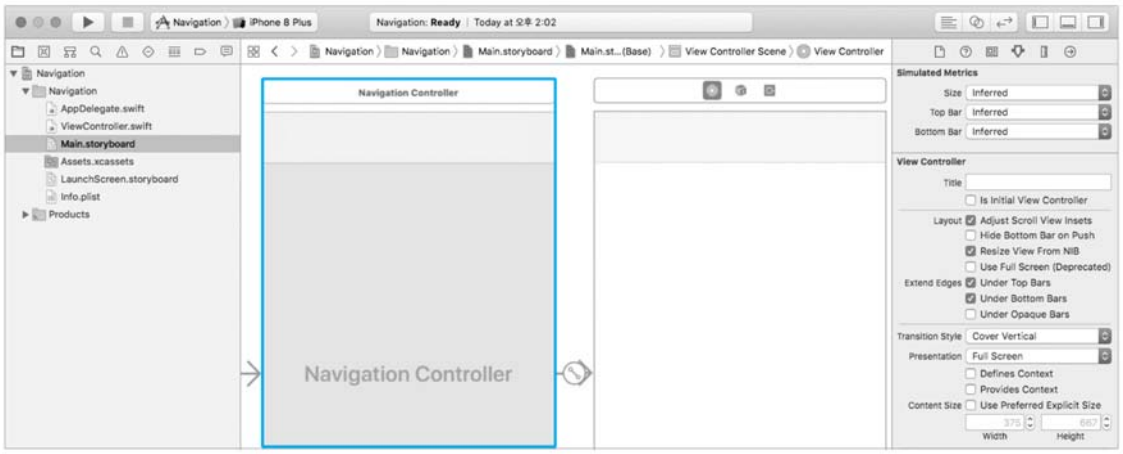
1. Xcode 를 실행한 후 'Navigation'이라는 이름으로 프로젝트를 만든다.
2. 뷰 컨트롤러 크기 조절하기
아이폰 모양의 뷰 컨트롤러 크기를 상황에 맞게 조절한다. 뷰 컨트롤러의 위쪽을 드래그하여 선택한다.



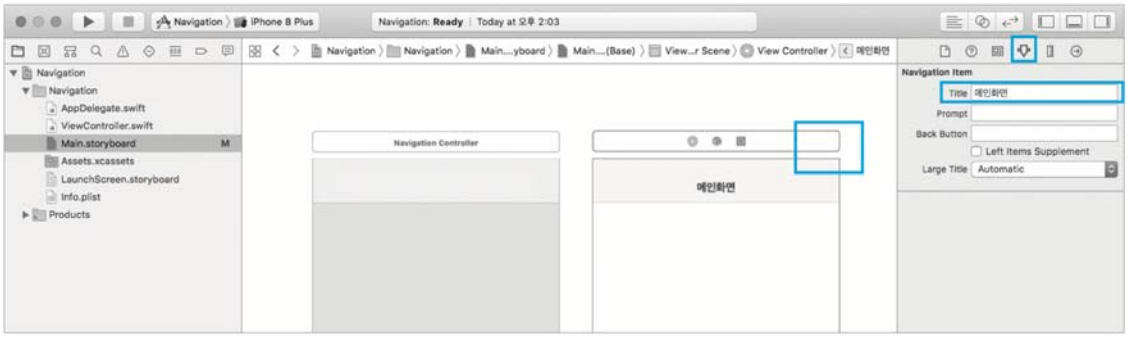
3. 내비게이션 컨트롤러를 스토리보드에 추가하기
먼저 스토리보드에 내비게이션 컨트롤러를 추가한다. 아이폰 모양의 뷰 컨트롤러를 클릭한 후 메뉴에서 [Editor->Embed in->Navigation Controller]를 클릭한다.



처음에는 없던 내비게이션 컨트롤러가 포함되었다.

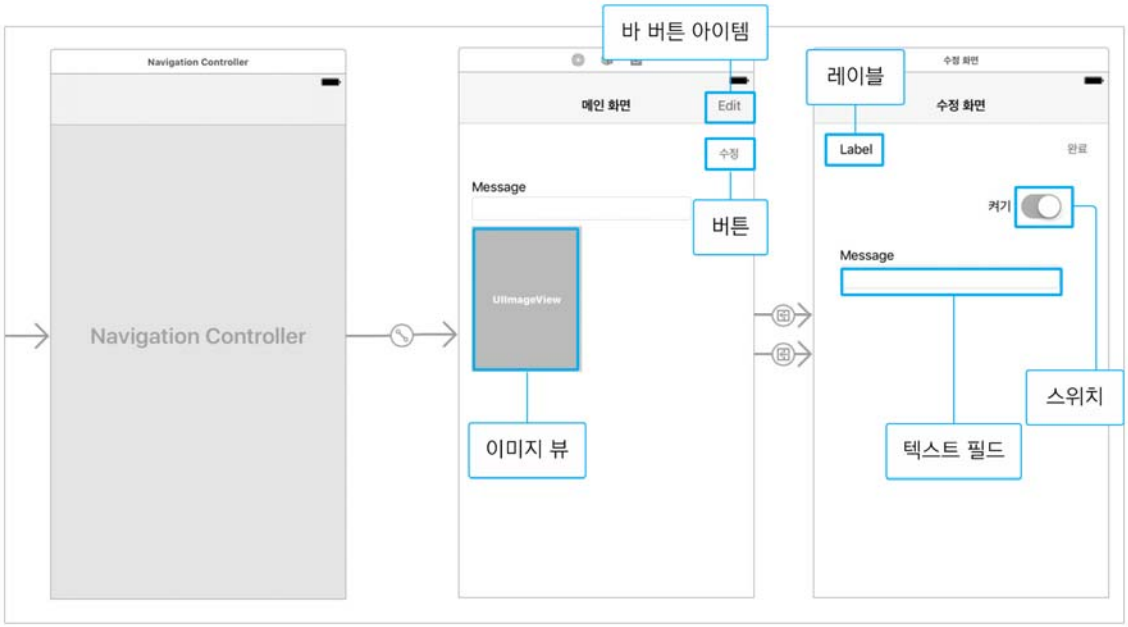


4. 내비게이션 컨트롤러의 위쪽을 드래그하여 선택한다. 오른쪽의 인스펙터 영역에서 [ATTRIBUTES INSPECTOR] 버튼을 클릭한 후 [Title]을 '메인 화면'으로 입력한다.



11-3. 뷰 추가하고 바 버튼으로 뷰 전환하기

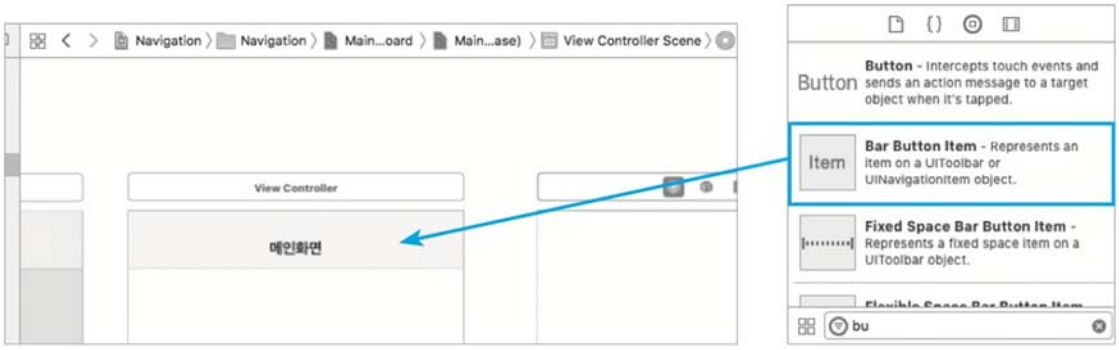
'메인 화면'에서 전환될 뷰를 추가하고 바 버튼을 누를 때 추가한 뷰로 전환되도록 구현해보자. 화면 전환을 위해 버튼(Button)과 바 버튼 아이템(Bar Button Item), 메시지 전달을 위한 텍스트 필드(Text Field), 전구 이미지를 나타낼 이미지 뷰() 그리고 전구를 켜고 끄기 위한 스위치(Switch)를 사용한다.



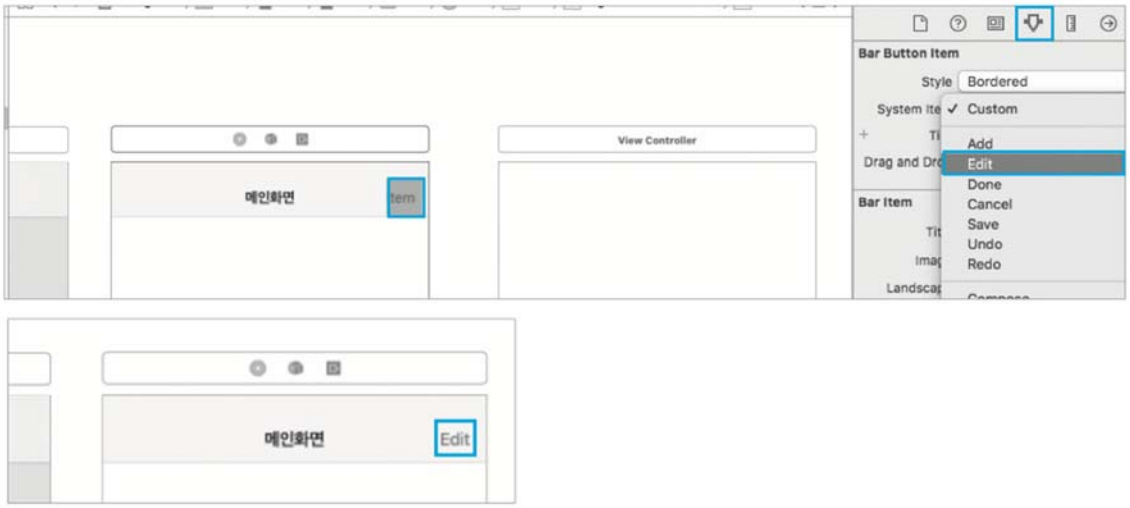
1. 뷰 추가하기

오브젝트 라이브러리에서 [뷰 컨트롤러(View Controller)]를 끌어다 '메인 화면' 컨트롤러의 오른쪽 빈 공간에 갖다 놓는다.

2. 오른쪽 아랫부분의 오브젝트 라이브러리에서 검색란에 'bar'를 입력하여 검색한 후 [바 버튼 아이템(Bar Button Item)]을 내비게이션 바의 오른쪽에 끌어다 놓는다.

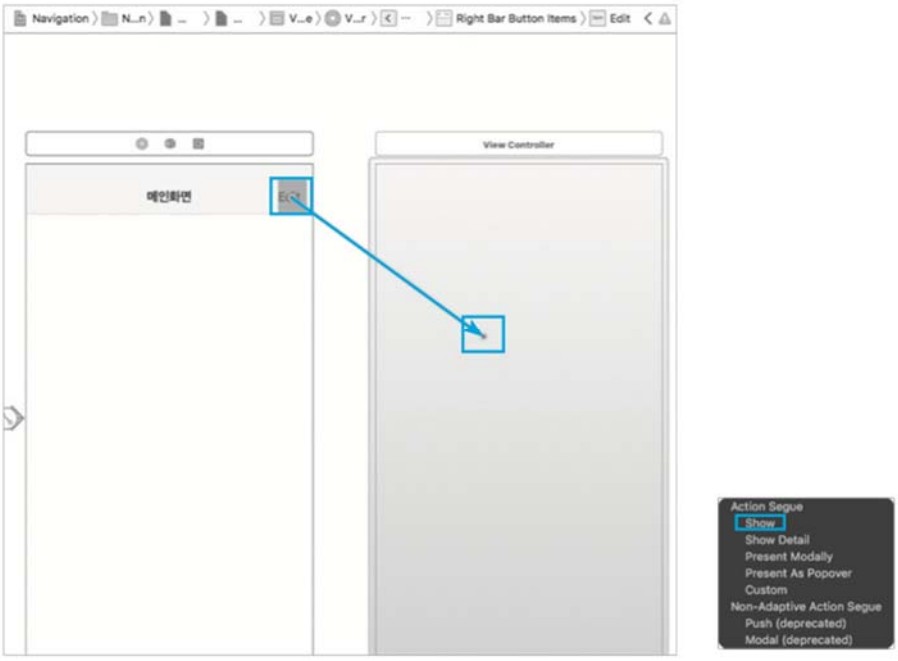


3. [바 버튼 아이템]을 선택한 상태에서 오른쪽의 인스펙터 영역에서 [Attributes inspector]버튼을 클릭한 후 '시스템 아이템(System Item)'에서 [Edit]를 선택한다.



4. 뷰 전환 작업하기

이 [Edit] 바 버튼을 마우스 오른쪽 버튼으로 클릭한 채 오른쪽의 뷰 컨트롤러에 갖다 놓으면 검은색 창이 나타나는데, 여기에서 [Action Segue]를 [Show]로 선택한다. 그러면 '메인 화면'에서 서브 화면으로 갔다가 돌아오는 형태를 취하게 된다.

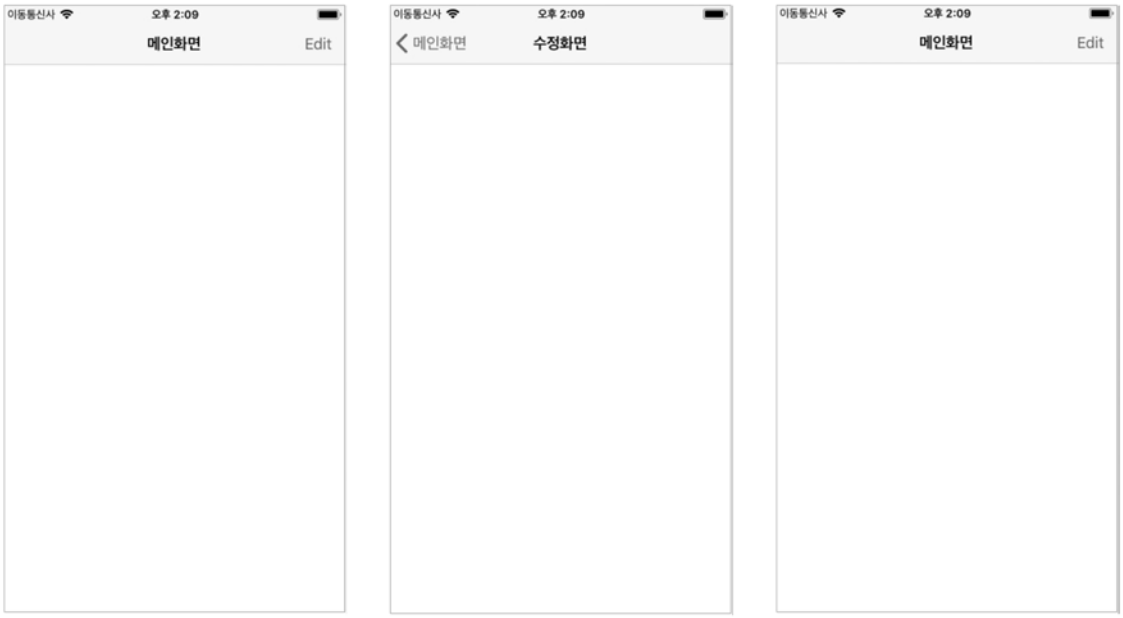


5. 세그웨이(Segue)가 생성되면 코딩이 없어도 화면 전환을 할 수 있다.

6. 오른쪽 뷰 컨트롤러의 내비게이션 바를 드래그하여 선택 후 [Attributes inspector]에서 Title 을 [수정 화면]으로 바꾼다.

7. 결과 보기

iOS 시뮬레이터를 실행하기 전 기기를 원하는 것으로 변경한다. 여기서는 [iPhone 8]으로 변경한 후 [실행] 버튼을 클릭한다. '메인 화면'에서 [Edit] 버튼을 클릭하면 '수정 화면'으로 이동하고 [메인 화면] 버튼을 클릭하면 다시 '메인 화면'으로 돌아온다. 이렇듯 세그웨이만 추가해도 화면 전환이 가능하다.

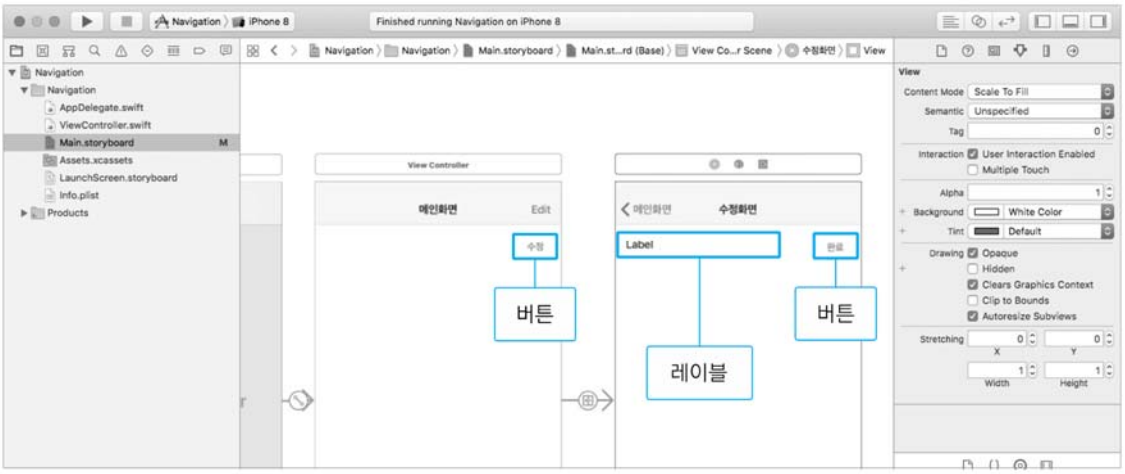


11-4. 버튼 활용해 뷰 전환하기

버튼을 활용해 뷰가 전환되도록 작업한다.

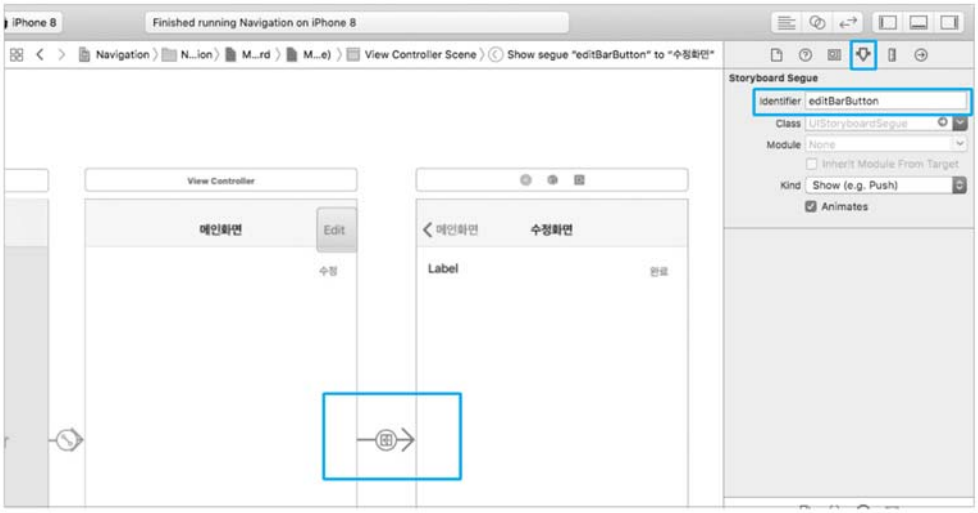
1. 스토리보드에 객체 추가하기

'수정 화면'에 레이블과 [완료]버튼을 추가한다. 그 후 '메인 화면'에서 [수정]버튼을 추가한다.

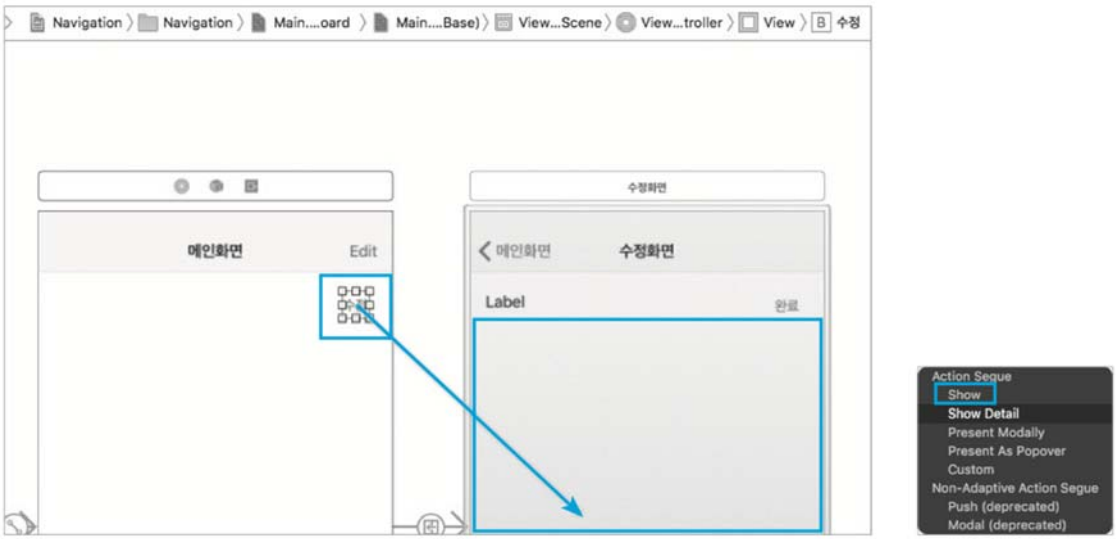


2. 뷰 전환하기

스토리보드의 중앙에 있는 세그웨이를 선택한다. 그리고 [Attributes inspector] 버튼을 클릭한 후 Identifier 을 [editBarButton]으로 수정한다. 이렇게 하면 지금 선택한 세그웨이의 아이디가 'editBarButton'으로 정의되고 나중에 나올 세그웨이와도 구분된다.

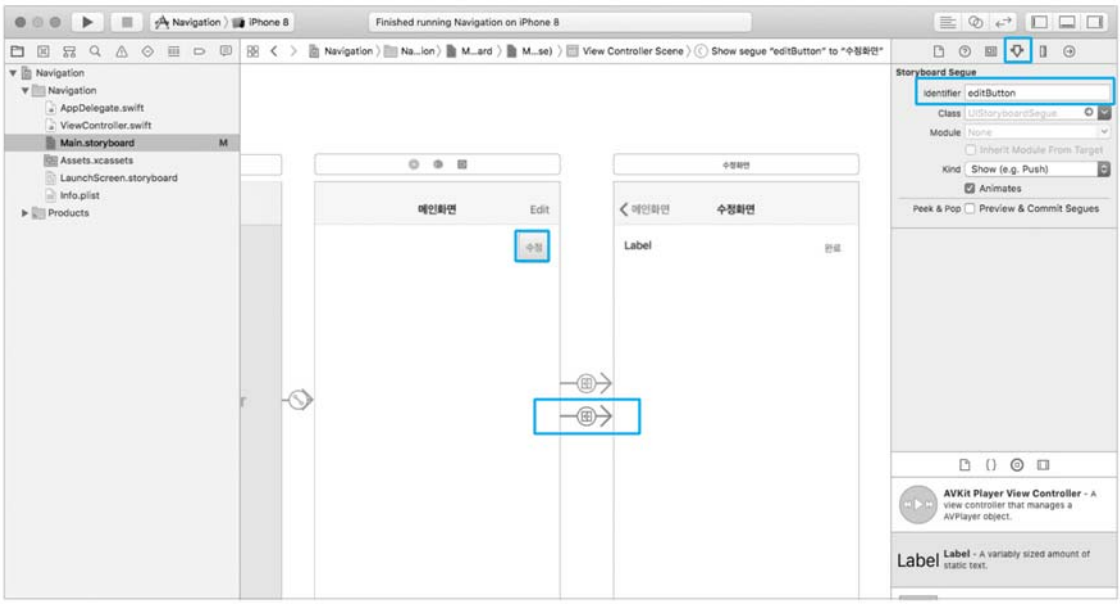


3. 마우스 오른쪽 버튼으로 '메인 화면'의 [수정] 버튼을 끌어다가 오른쪽 '수정 화면'의 뷰 컨트롤러로 갖다 놓는다. 검은색 설정 창의 [Action Segue] 에서 [Show]를 선택한다. 그러면 '메인 화면'에서 서브 화면으로 갔다가 돌아오는 형태를 취하게 된다.



4. 세그웨이가 두 개가 되었다. [Edit] 바 버튼을 이용한 세그웨이와 [수정] 버튼을 이용한 세그웨이다.

5. 세그웨이를 선택하면 그 세그웨이와 연결된 버튼이 표시되는데, [수정] 버튼이 선택되는 세그웨이를 선택한 후 [editButton]으로 수정한다.



6. 결과 보기

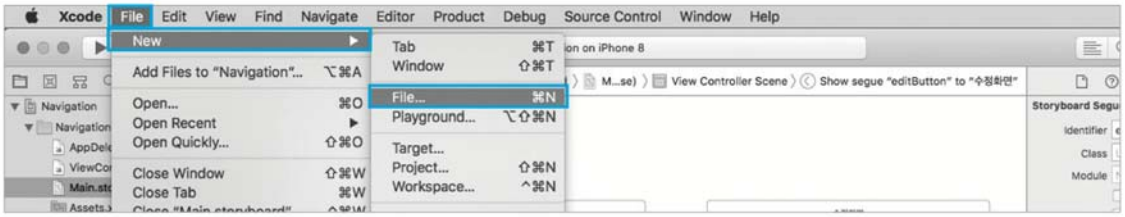
iOS 시뮬레이터에서 [실행] 버튼을 클릭한다. '메인 화면'의 [Edit] 바 버튼을 클릭해도 '수정 화면'으로 이동하고, '메인 화면'의 [수정] 버튼을 클릭해도 '수정 화면'으로 이동한다. 둘 다 세그웨이를 생성했기 때문에 어느 버튼을 클릭해도 '수정 화면'으로 이동할 수 있다.

11-5. 뷰 전환 구분하기

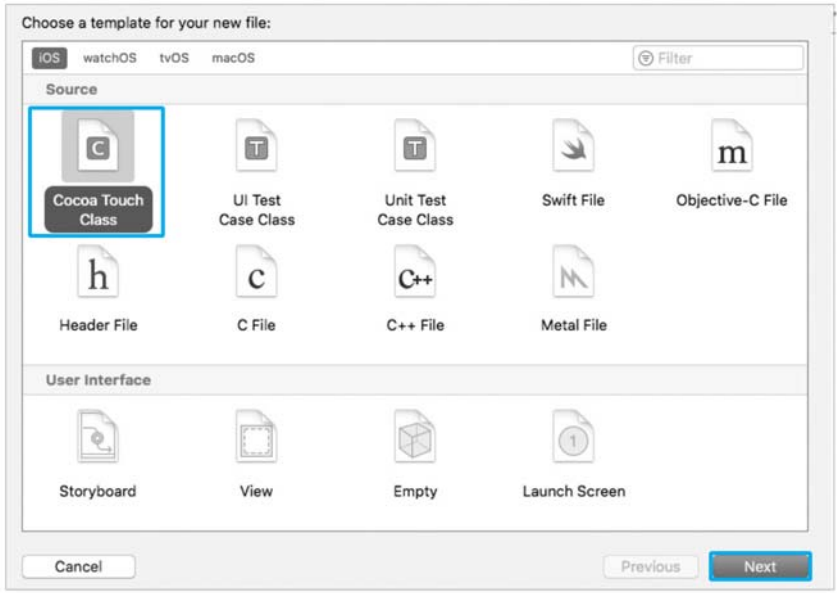
앞 절에서는 뷰의 전환만 이루어질 뿐 아무런 동작도 수행하지 않는다. 뷰가 전환되면서 특정한 동작이나 작업을 하기 위해서는 스위프트 소스 코드와 아웃렛 변수 및 액션 함수를 추가해야 한다. 앞에서는 바 버튼을 이용한 방법과 일반 버튼을 이용한 방법으로 뷰를 전환했는데, 여기서는 뷰가 전환될 때 바 버튼을 눌러 전환되었는지 일반 버튼을 눌러 전환되었는지 구분하고, 이를 화면에 나타내본다.

1. 스위프트 소스 파일 추가

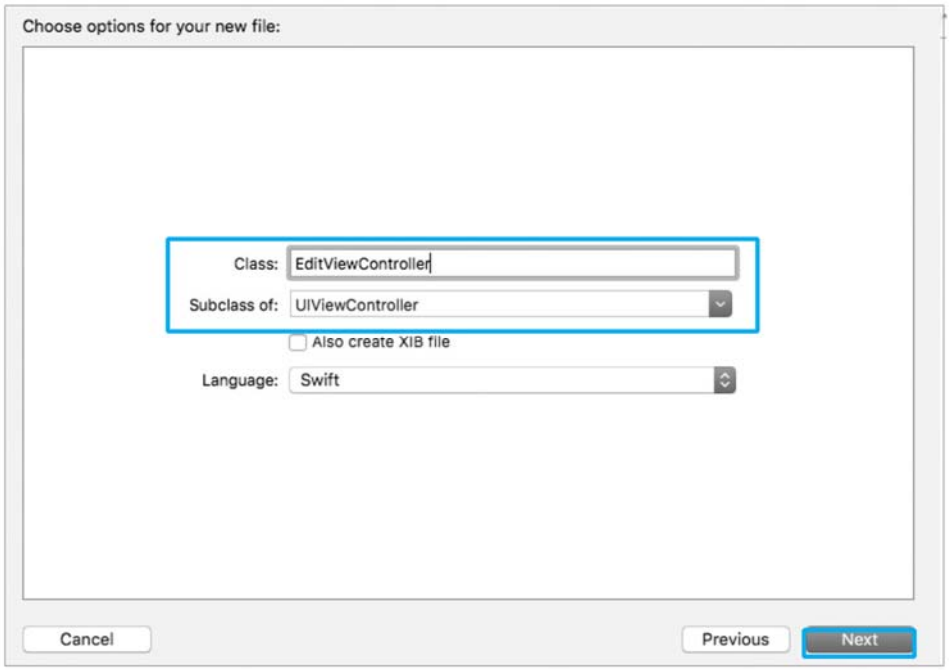
'수정 화면'의 뷰 컨트롤러를 만들었지만 이에 해당하는 뷰 컨트롤러 클래스 파일이 없다. 그래서 '수정 화면'의 뷰 컨트롤러 클래스 파일을 만들어본다. 메뉴에서 [File->New->File...] 버튼을 클릭한다.



2. 'iOS' 항목의 [Source] 탭에서 [Cocoa Touch Class]를 선택하고 [Next] 버튼을 클릭한다.



3. Class 는 [EditViewController]로, Subclass of 는 [UIViewController]로 수정한 후 [Next] 지정 폴더에 저장한다.



4. 내비게이터 영역에 [EditViewControllers.swift]가 추가된 것을 확인할 수 있다.

5. [Main.storyboard]에서 '수정 화면' 뷰 컨트롤러를 선택한 후 [Identity inspector] 버튼을 클릭해 Class 에서 [EditViewController]를 선택한다. 이로써 스토리보드의 뷰 컨트롤러와 'EditViewControllers.swift' 파일이 제대로 연결되었다. 이제부터 '메인 화면'에 관한 것은 [ViewControllers.swift]에서, '수정 화면'에 관한 것은 [EditViewControllers.swift]에서 다룬다. 이름이 비슷하니 헷갈리지 않도록 주의한다.

6. 아웃렛 변수 추가하기
아웃렛 변수와 함수를 추가하기 위해 우선 오른쪽 윗부분의 [Show the Assistant editor] 버튼을 클릭하여 보조 편집기 영역을 연다. 가운데 화면의 스토리보드 부분이 둘로 나누어지며 왼쪽에는 스토리보드, 오른쪽에는 보조 편집기 영역이 나타난다.

7. [레이블(Label)]을 마우스 오른쪽 버튼으로 선택 후 오른쪽의 보조 편집기 영역으로 드래그하면 아래 그림과 같이 연결선이 나타난다. 다음 표를 참고하여 아웃렛 변수를

연결한다.

위치	에디트 뷰 컨트롤러의 클래스(EditView Controller class) 선언문 바로 아래
연결(Connection)	Outlet
이름(Name)	lblWay
유형(Type)	UILabel


```
@IBOutlet var lblWay: UILabel!
```

8. 액션 함수 추가하기

아웃렛 변수를 추가하는 것과 같은 방법으로 버튼에 대한 액션 함수를 추가한다. 레이블의 오른쪽에 있는 [완료] 버튼을 마우스 오른쪽 버튼으로 클릭한 후 드래그해서 오른쪽의 보조 편집기 영역에 갖다 놓는다.

위치	didReceiveMemoryWarning 함수 아래
연결(Connection)	Action
이름(Name)	btnDone
유형(Type)	UIButton


```
@IBAction func btnDone(_ sender: UIButton) {
}
```

9. 뷰 전환을 구분하기 위한 코딩 작업

코딩을 위해 화면 모드를 수정한다. 오른쪽 윗부분에서 [Show the Standard editor] 버튼을 클릭한다. 그리고 왼쪽의 내비게이터 영역에서 [ViewController.swift]를 선택한다.

10. 세그웨이를 이용하여 화면을 전환하기 위해 prepare 함수를 사용한다. 뷰 컨트롤러 클래스와 아래쪽에 다음 함수를 추가한다.

11. 세그웨이의 도착 컨트롤러를 EditViewControlelr 형태를 가지는 segue.destinationViewController 로 선언한다. 그리고 세그웨이의 아이디에 따라 다르게 동작하도록 if 문 형태로 작성한다. If 문과 else 문을 이용해 각 세그웨이별로 다른 문자열을 전송할 것이다. 앞에서 작성한 함수 안에 다음 소스를 추가한다.

12. 뷰 컨트롤러에서 각 세그웨이 아이디별로 다른 문자열을 받아서 표시하기 위해 'EditViewController.swift' 파일을 수정한다. [EditViewController.swift]를 클릭한다.

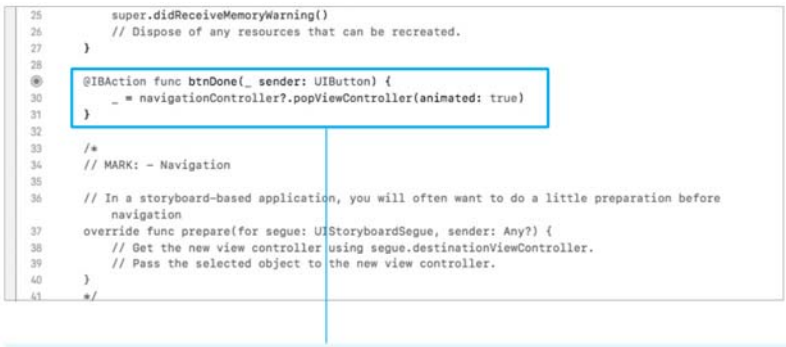
13. 레이블의 테스트를 직접 제어할 수 없기 때문에 문자열 변수 textWayValue 를 만든다. 그리고 viewDidLoad 함수에서 변수 textWayValue 값을 레이블의 텍스트로 대입하여 레이블에 출력되게 한다.


```
class EditViewController: UIViewController {
var textWayValue: String = “”
@IBOutlet var lblWay: UILabel!
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view.
    lblWay.text = textWayValue
}
```

14. 다시 [ViewController.swift] 파일을 클릭하여 선택한다. Prepare 함수를 마저 작성한다. If 문과 else 문에 다음 소스를 추가한다. 세그웨이 아이디가 'editButton'일 경우, 즉 레이블의 오른쪽 옆에 있는 버튼일 경우에는 'segue:use button' 문자열을 전송하고, 세그웨이 아이디가 'editBarButton'일 경우, 즉 바 버튼을 클릭하였을 경우에는 'segue:use Bar button' 문자열을 전송하여 '수정 화면'의 레이블이 서로 다르게 표시되게 한다.

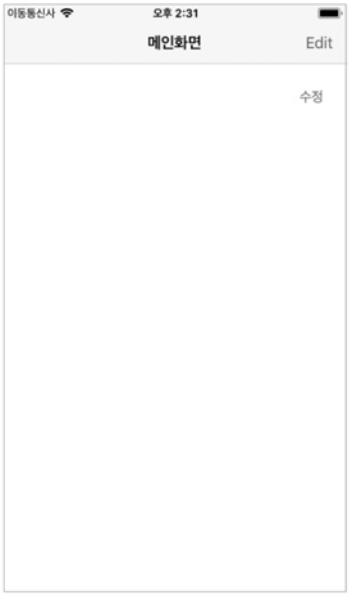
```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    let editViewController = segue.destination as!
        EditViewController
    if segue.identifier == “editButton” {
        editViewController.textWayValue = “segue : use button”
    } else if segue.identifier == “editBarButton” {
        editViewController.textWayValue = “segue : use Bar button”
    }

}
```

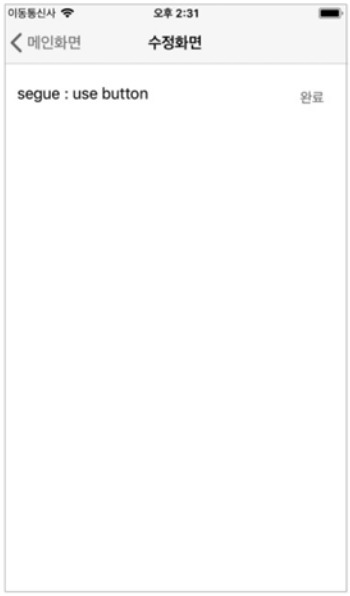
15. 다시 'EditViewController.swift'파일을 선택하여 '수정 화면'의 [완료] 버튼을 클릭하면 '메인 화면'으로 이동하게 코드를 삽입한다. 앞에서는 실습에서 뷰를 전환하기 위해 세그웨이를 추가할 때 [Action Segue]를 'Show' 형태로 했다. 따라서 되돌아갈 때는 'pop'의 형태로 해야 한다.




```
@IBAction func btnDone(_ sender: UIButton) {
    _ = navigationController?.popViewController(animated: true)
}
```

16. iOS 시뮬레이터에서 [실행] 버튼을 클릭한다. '메인 화면'에서 [수정] 버튼을 클릭하면 '수정 화면'으로 이동함과 동시에 'segue:use button'이 표시되고, '메인 화면'에서 [Edit] 버튼을 클릭하면 '수정 화면'으로 이동함과 동시에 'segue:use Bar Button'이 표시된다. 이처럼 '메인 화면'에서 '수정 화면'으로 데이터 전송이 가능하면 세그웨이별로 다른 데이터 전송도 가능하다. 또한 '수정화면'에서 [완료] 버튼을 클릭해도 '메인 화면'으로 이동한다.





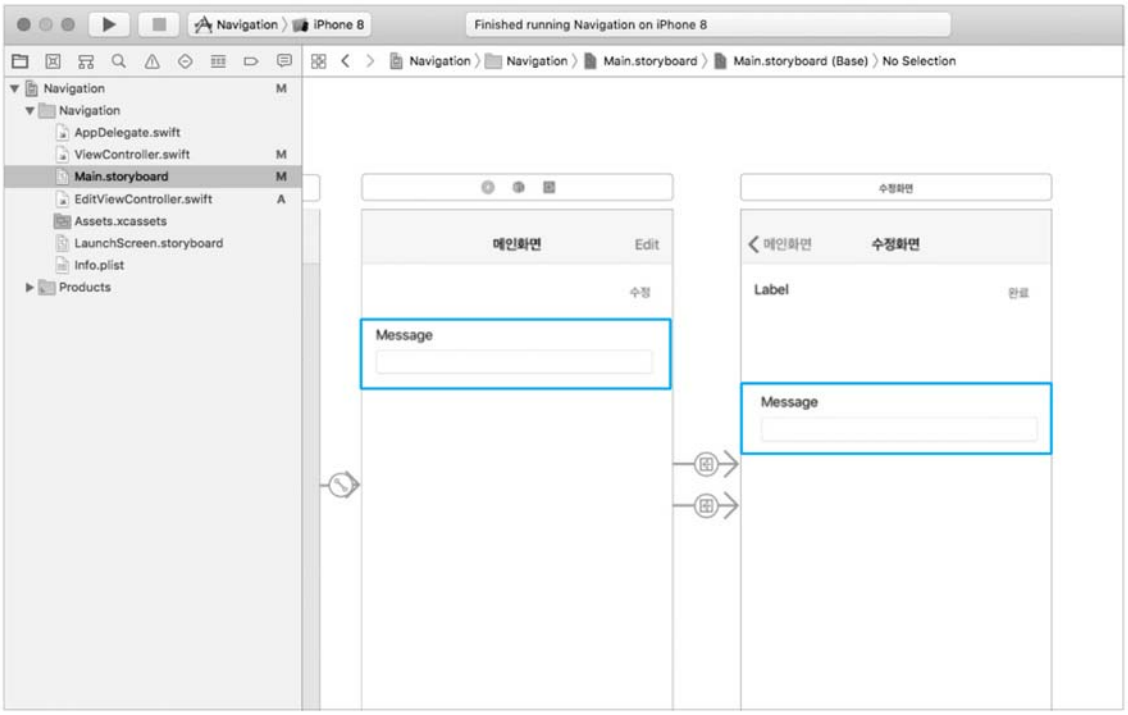
11-6. 뷰 전환과 함께 메시지 전달하기

'수정 화면'에서 메시지를 입력하고 [완료] 버튼을 클릭하면 '메인 화면'에 그 메시지가 전달되도록 구현해본다. 반대로 '메인 화면'에서 '수정 화면'으로 메시지를 전달하는 작업도 해본다.

데이터 전송은 메시지를 전달하는 형태로 한다.

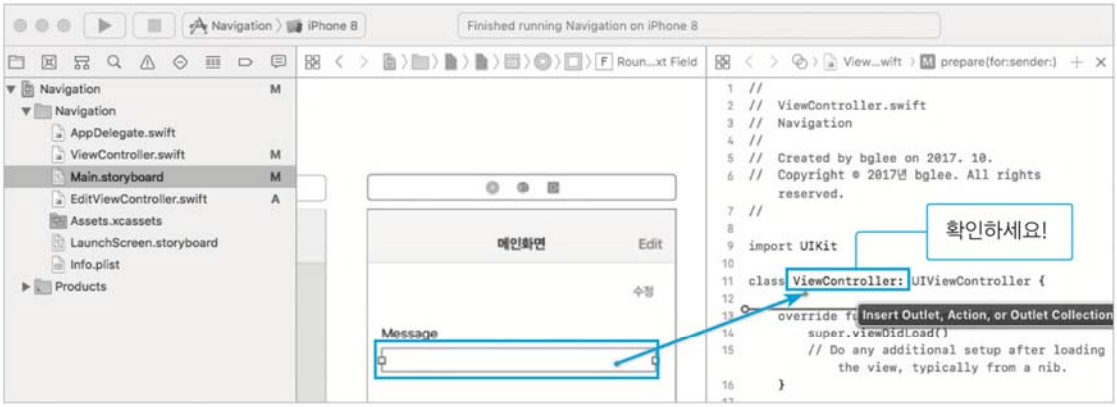
1. 메시지 전달을 위해 스토리보드에 추가 작업 하기

'메인 화면'에 레이블(Label'을 추가한 후 'Message'라고 수정하고, 텍스트 필드(Text Field)를 추가한다. command+C 를 눌러 이 둘을 복사한 후 command+V 를 눌러 '수정 화면'에 붙여넣고 아래 그림과 같이 중앙에 오도록 수정한다.



2. 아웃렛 변수와 함수 추가

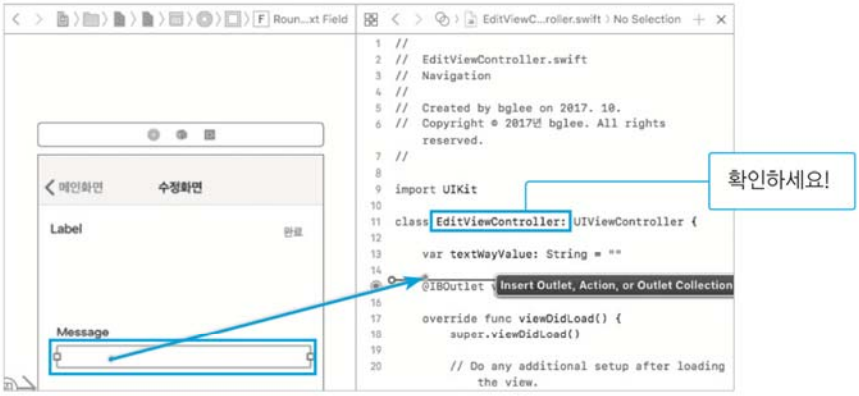
'메인 화면'의 텍스트 필드를 마우스 오른쪽 버튼으로 클릭한 후 오른쪽의 보조 편집기 영역으로 드래그하면 아래 그림과 같이 연결선이 나타난다. 드래그한 연결선을 다음 표를 참고하여 연결한다.



위치	뷰 컨트롤러 클래스 선언문 바로 아래
연결(Connection)	Outlet
이름(Name)	txMessage
유형(Type)	UITextField

3. '메인 화면'의 텍스트 필드 아웃렛 변수가 추가되었다.

4. 이번엔 '수정 화면'의 텍스트 필드를 마우스 오른쪽으로 클릭한 후 오른쪽의 보조 편집기 영역으로 드래그하여 연결한다. 드래그 하기 전에 오른쪽 보조 편집기 영역이 '에디트 뷰 컨트롤러 클래스(EditViewController class)'인지 확인한다.

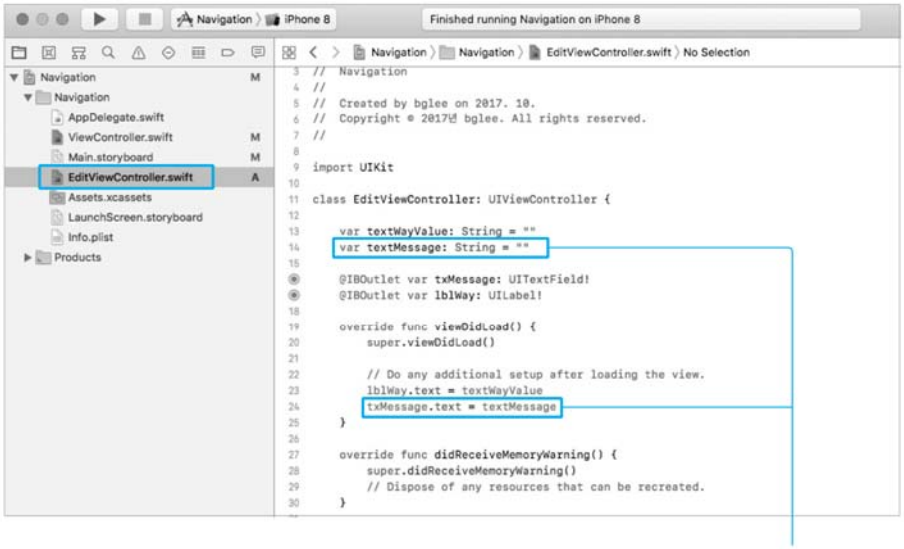


위치	에디트 뷰 컨트롤러 클래스 선언문 바로 아래
연결(Connection)	Outlet
이름(Name)	txMessage
유형(Type)	UITextField

5. '수정 화면'의 텍스트 필드 아웃렛 변수가 추가되었다.

6. 메시지를 전달하기 위한 코딩 작업

코딩을 위해 화면 모드를 수정한다. 오른쪽 윗부분에서 [Show the Standard editor] 버튼을 클릭한 후 왼쪽의 내비게이터 영역에서 [EditViewController.swift]를 선택한다. 그리고 텍스트 필드에 적은 내용을 나타내기 위해 다음 코드를 추가한다.




```
var textMessage: String = "" ❶
⋮
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view.

    lblWay.text = textWayValue
    txMessage.text = textMessage ❷
}
```

1) '수정 화면'에서 직접 텍스트 필드의 텍스트를 제어 할 수 없기 때문에 변수 textMessage 를 문자열로 만든다.

2) viewDidLoad 함수에서 변수 textMessage 값을 텍스트 필드의 텍스트로 대입하여 텍스트 필드에 나타나게 한다.

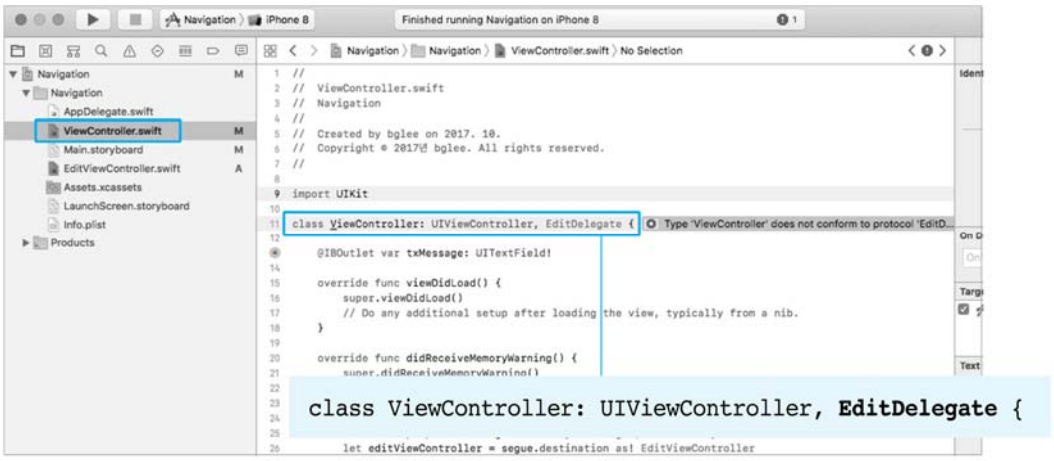
7. 프로토콜 형태의 델리게이트를 추가하기 위해 import 문과 클래스문 사이에 공백을 만든다.

8. 공백 안에 아래와 같이 EditDelegate 프로토콜을 작성한다.

```
//
//  EditViewController.swift
//  Navigation
//
//  Created by bglee on 2017. 10.
//  Copyright © 2017년 bglee. All rights reserved.
//

import UIKit

protocol EditDelegate {
    func didMessageEditDone(_ controller: EditViewController, message: String)
}

class EditViewController: UIViewController {

    var textWayValue: String = ""
    var textMessage: String = ""

    @IBOutlet var txMessage: UITextField!
    @IBOutlet var lblWay: UILabel!
```

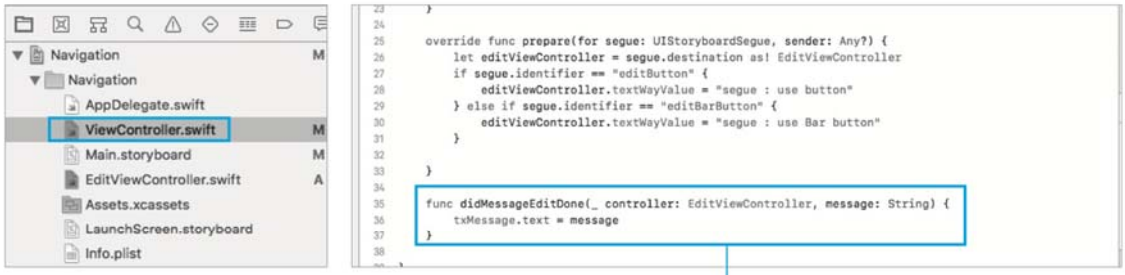
```
protocol EditDelegate {
    func didMessageEditDone(_ controller: EditViewController, message: String)
}
```

9. 왼쪽의 내비게이터 영역에서 [ViewController.swift[를 선택한다. 그런 다음 앞에서 프로토콜로 작성한 EditDelegate 를 뷰 컨트롤러 클래스 선언문 안에 넣어 상속을 받는다.



```
class ViewController: UIViewController, EditDelegate {
```

10. 이렇게 프로토콜을 상속받으면 프로토콜에서 정의한 함수를 무조건 만들어야 한다. 만들지 않으면 에러가 발생한다.



```
func didMessageEditDone(_ controller: EditViewController, message: String) {
    txMessage.text = message
}
```

11. 이 didMessageEditorDone 함수는 에디트 뷰 컨트롤러에서 함수를 호출하며 메시지(message 를 전달해주는데, 이 메시지의 스트링 값을 메인 화면의 텍스트 필드에 텍스트로 보여준다. 즉, '수정 화면'의 데이터를 '메인 화면'에 전달하여 보여주는 것이다. 따라서 앞에서 만든 함수 안에 다음 소스를 입력한다.

12. 다시 왼쪽의 내비게이터 영역에서 [EditViewController.swift]를 선택한다. 그리고 delegate 변수를 생성한다.

13. 같은 에디트 뷰 컨트롤러레서 이번에는 btnDone 함수를 수정해 보자. '수정 화면'의 btnDone 함수에서 didMessageEditDone 을 호출하면서 '수정 화면'의 텍스트 필드 내용을 메시지 문자열(String)로 전달합니다. 다시 말해 '수정 화면'의 텍스트 필드의 내용, 즉 데이터를 '메인 화면'으로 전달한다.


```
@IBAction func btnDone(_ sender: UIButton) {
    if delegate != nil {
        delegate?.didMessageEditDone(self, message: txMessage.text!)
    }
    _ = navigationController?.popViewController(animated: true)
}
```

14. 마지막으로 왼쪽의 내비게이터 영역에서 [ViewController.swift]를 선택한다. 그 다음 prepare 함수의 마지막 부분에 다음 코드를 추가한다.

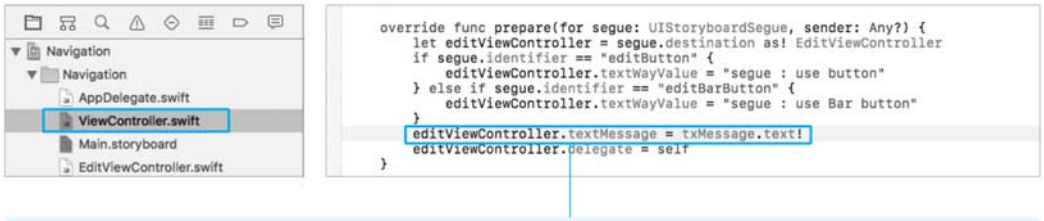

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        let editViewController = segue.destination as! EditViewController
      ⋮
       editViewController.delegate = self
}
```

15. 결과 보기
[실행] 버튼을 클릭한 후 '메인 화면'에서 메시지가 비어있는 것을 확인하고 '수정 화면'으로 이동한 후 메시지 창에 '수정 완료'를 입력한다. [완료] 버튼을 클릭하면 '메인 화면'에 동일한 내용이 나타난다. '수정 화면'의 데이터를 역으로 '메인 화면'으로 이동하였다.

*** '메인 화면'에서 '수정 화면'으로 메시지 전달하기 ***
메시지를 입력한 후 버튼을 클릭하면 '메인 화면'의 메시지가 '수정 화면'의 메시지로 이동되도록 코딩한다.

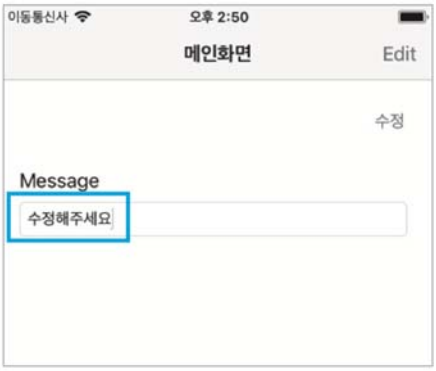
1. 버튼 이용해 '수정 화면'으로 메시지를 전달하기 위한 코딩 작업
코딩을 이용해 화면 모드를 수정한다. 오른쪽 상단에서 [Show the Standard editor] 버튼을 클릭 후 왼쪽의 내비게이터 영역에서 [ViewController.swift]를 선택한다. 이제 prepare 함수를 수정한다.

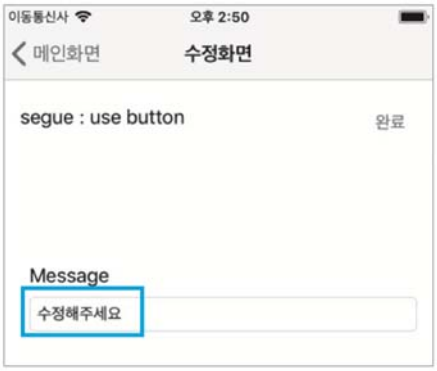
2. prepareForSegue 함수에 다음 소스를 추가한다. editViewController, 즉 '수정 화면'의 textMessage 에 '메인 화면'의 텍스트 필드 내용이 전달된다.



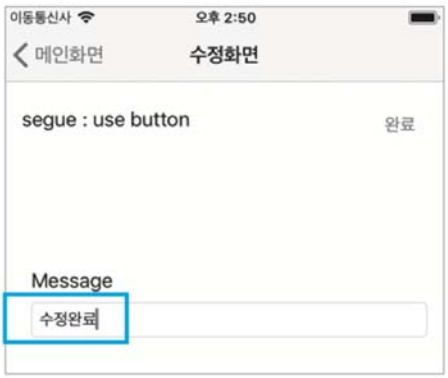

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    let editViewController = segue.destination as! EditViewController
        ⋮
    editViewController.textMessage = txMessage.text!
    editViewController.delegate = self
}
```

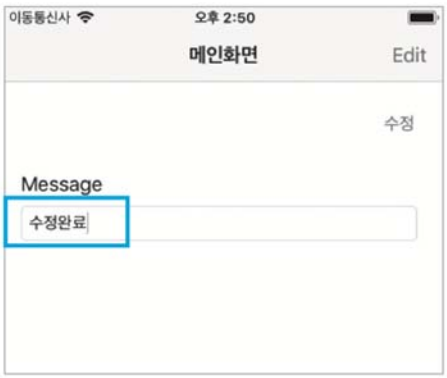
3. 결과 보기
iOS 시뮬레이터에서 [실행] 버튼을 클릭한다. '메인 화면'에서 메시지에 '수정해주세요'라고 입력한 후 [Edit] 혹은 [수정] 버튼을 클릭하면 '수정 화면'의 메시지 창에 '수정해주세요'가 나타난다. '메인 화면'의 데이터가 '수정 화면'으로 이동된 것이다.





그런 다음 '수정 화면'에서 메시지 창에 '수정 완료'를 입력한 후 [완료] 버튼을 클릭하면 '메인 화면'에 동일한 내용이 나타난다. 즉 '수정 화면'의 데이터가 '메인 화면'으로 이동된 것이다.





11-7. '수정 화면'에서 '메인 화면'의 전구 제어하기

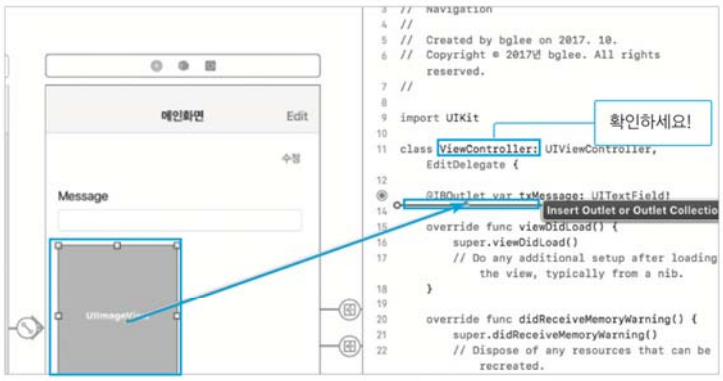
텍스트 문자열뿐만 아니라 다양한 변수를 전달하여 여러 정보를 전달하고 정보 값을 변경하여 제어할 수 있도록 해본다. '메인 화면'에 전구 이미지를 넣고 '수정 화면'에서 이 전구를 켜거나 꺼본다.

1. 전구를 제어하기 위해 스토리 보드에 추가 작업 하기

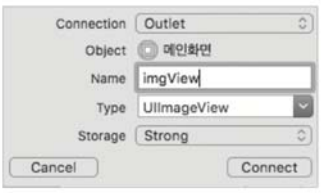
'메인 화면'에 이미지 뷰(Image View)를 추가하고, '수정 화면'에는 [켜기] 레이블(Label)과 스위치(Switch)를 추가한다.

2. 아웃렛 변수 추가하기

'메인 화면'의 [이미지 뷰(Image View)]를 마우스 오른쪽 버튼으로 클릭한 후 오른쪽의 보조 편집기 영역으로 드래그하면 연결선이 나타난다. 이때 오른쪽 보조 편집기 영역이 '뷰 컨트롤러 클래스'인지를 확인한다.



위치	앞에서 만든 텍스트 필드의 아웃렛 변수 바로 아래
연결(Connection)	Outlet
이름(Name)	imgView
유형(Type)	UIImageView



3. 이미지 뷰 아웃렛 변수가 생성되었다.

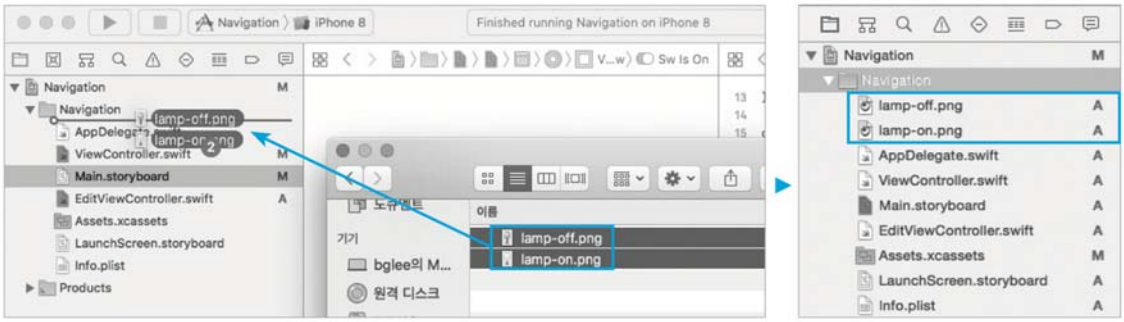
4. '수정 화면'의 빈 공간을 클릭한 후 같은 방법으로 [스위치]를 오른쪽 보조 편집기 영역으로 드래그한다. 드래그 하기 전 오른쪽 보조 편집기 영역이 '에디트 뷰 컨트롤러(EditViewController) 클래스'인지를 먼저 확인한다.

위치	앞에서 만든 레이블의 아웃렛 변수 바로 아래
연결(Connection)	Outlet
이름(Name)	swIsOn
유형(Type)	UISwitch

5. 스위치의 아웃렛 변수가 생성되었다.

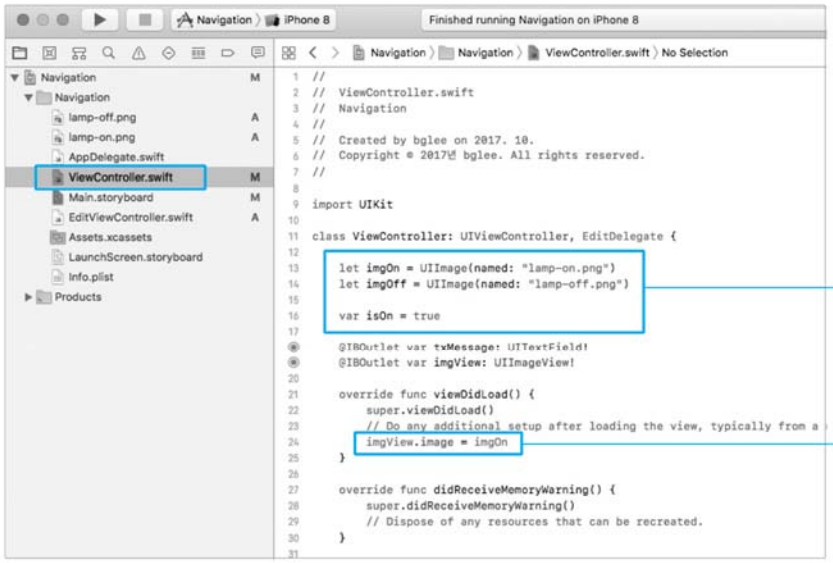
6. 이미지 추가하기

전구 이미지인 'lamp-on.png' 파일과 'lamp-off.png' 파일을 왼쪽의 내비게이터 영역에 추가한다. 두 파일을 끌어다 놓은 후 [Finish] 버튼을 클릭하면 파일이 추가된다. 이미지가 [Target Membership]에 추가되었는지 확인한다.



7. 전구 초기 화면을 세팅하기 위한 변수 초기화

오른쪽 윗부분에서 [Show the Standard editor] 버튼을 클릭한다. 왼쪽의 내비게이터 영역에서 [ViewController.swift]를 선택한 후 다음 소스를 입력한다.




```
let imgOn = UIImage(named: "lamp-on.png")
let imgOff = UIImage(named: "lamp-off.png")

var isOn = true
:
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a nib.
    imgView.image = imgOn
}
```

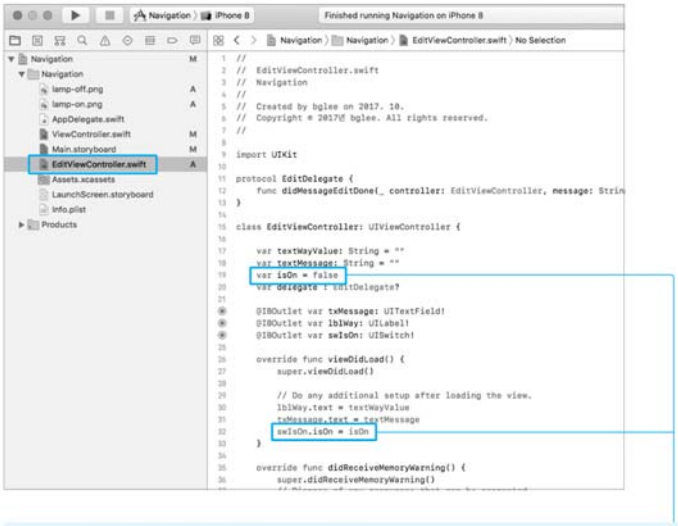
1) UIImage 타입의 변수 imgOn 과 imgOff 를 선언하고, 각 변수에 이미지 파일명을 입력한다.

이미지 파일명은 앞에서 내비게이터 영역에 추가한 이미지의 파일명을 입력한다.

2) 전구가 켜져 있는지를 나타내는 변수이다. True 값을 주어 켜져 있음을 나타낸다.

3) 스토리보드에 추가한 오브젝트인 이미지 뷰에 방금선언한 imgOn 을 대입한다, 이렇게 하면 앱을 실행했을 때 보이는 첫 화면에서 켜져 있는 전구의 모습을 볼 수 있다.

8. 다시 왼쪽의 내비게이터 영역에서 [EditViewControllers.swift]를 선택한 후 소스를 적절한 위치에 입력한다.



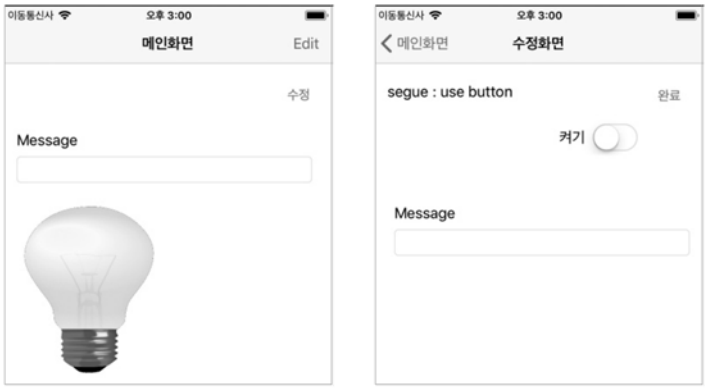
```
var textWayValue: String = ""
var textMessage: String = ""
var isOn = false ❶
var delegate : EditDelegate?
⋮
override func viewDidLoad() {
    super.viewDidLoad()

    // Do any additional setup after loading the view.
    lblWay.text = textWayValue
    txMessage.text = textMessage
    swIsOn.isOn = isOn ❷
}
```

1) '수정화면'에서 직접 스위치를 제어할 수 없기 때문에 변수 isOn 을 만든다.

2) '수정화면'에서 직접 스위치를 제어할 수 없기 때문에 변수 isOn 값을 스위치의 On 에 대입하여 스위치 값에 출력되게 한다. 현재 isOn 이 false 값을 가지므로 스위치는 꺼져 있는 모습을 보인다.

9. 결과 보기

[실행] 버튼을 클릭한 후 '메인 화면'에서 [수정] 버튼을 클릭하면 '수정 화면'으로 이동한다. '수정 화면'을 보면 스위치가 꺼져 있는 것을 확인할 수 있다.



10. 코드 작성

'메인 화면' 전구의 온/오프(On/Off) 상태를 '수정 화면'에서 그대로 인식해 표시되도록 수정한다. 왼쪽의 내비게이터 영역에서 [ViewControllers.swift]를 선택한 후 소스를 추가한다.


```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        :
    editViewController.textMessage = txMessage.text!
    editViewController.isOn = isOn
    editViewController.delegate = self
}
```

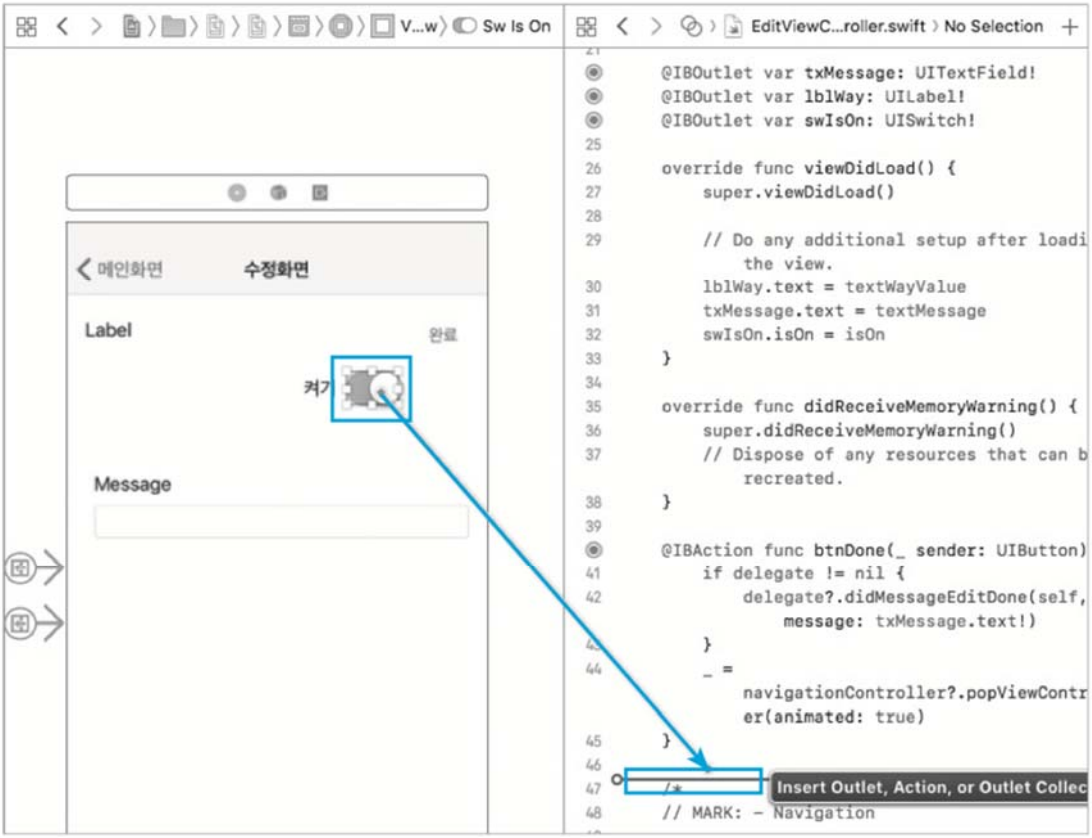
1) prepare 함수에서 editViewController, 즉 '수정화면'의 isOn 에 '메인 화면'의 상태를 전달한다.

11. 결과 보기

[실행] 버튼을 클릭한 후 '메인 화면'에서 [수정] 버튼을 클릭하면 '수정 화면'으로 이동한다. '수정 화면'을 보면 스위치가 켜져있는 것을 확인할 수 있다. '메인 화면'의 전구와 '수정 화면'의 스위치 상태가 동일해진 것이다.

12. 전구를 제어하기 위한 함수 추가

스위치에 대한 액션 함수를 추가한다. [스위치]를 마우스 오른쪽 버튼으로 클릭한 후 드래그해서 오른쪽의 보조 편집기 영역에 갖다놓는다. 드래그 하기 전에 오른쪽 보조 편집기 영역이 '에디트 뷰 컨트롤러 클래스(EditViewController)'인지 먼저 확인해야 한다.

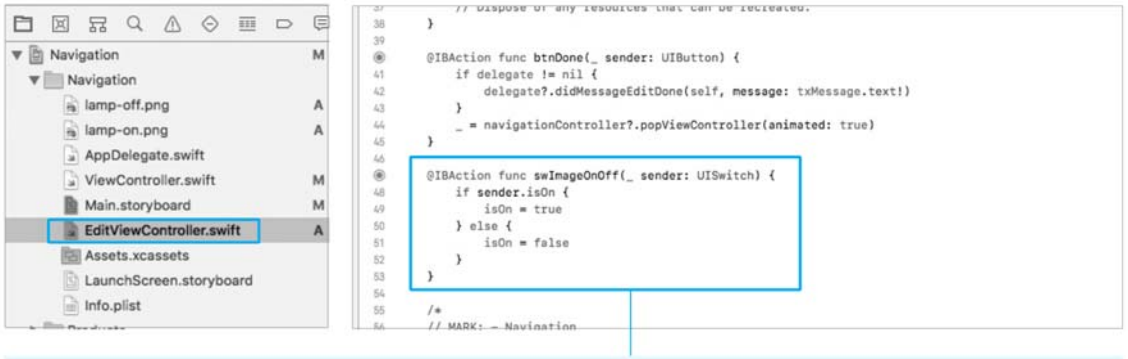


위치	앞에서 자동 생성된 didReceiveMemory Warning 함수 바로 아래
연결(Connection)	Action
이름(Name)	swImageOnOff
유형(Type)	UISwitch

13. swImageOnOff 함수가 생성되었다.

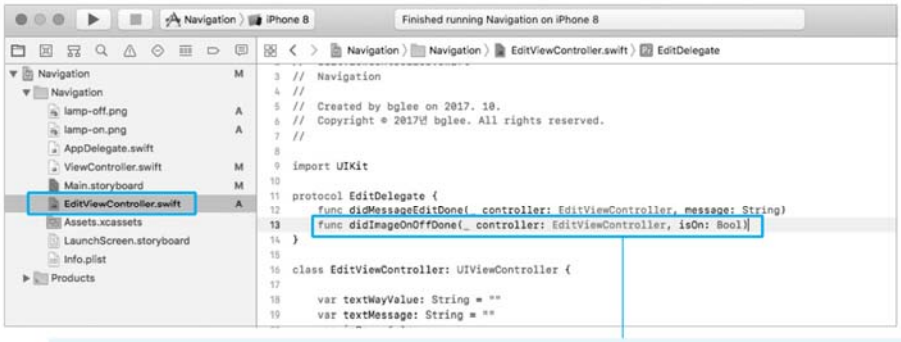
14. 전구를 제어하기 위한 코딩 작업

코딩을 위해 화면 모드를 수정한다. 오른쪽 윗부분에서 [Show the Standard editor] 버튼을 클릭한다. 왼쪽의 내비게이터 영역에서 [EditViewController.swift]를 선택한 후 swImageOnOff 함수를 수정한다. 스위치 상태를 확인하여 isOn 변수에 true 또는 false를 설정한다. 켜져있으면 true 이다.




```
@IBAction func swImageOnOff(_ sender: UISwitch) {
    if sender.isOn {
        isOn = true
    } else {
        isOn = false
    }
}
```

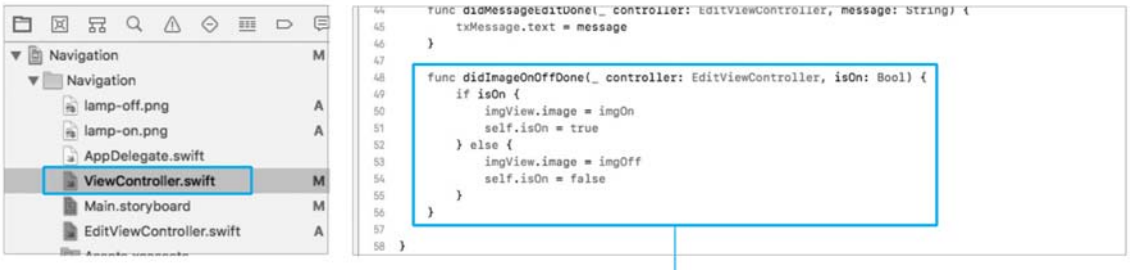
15. '수정 화면'의 스위치 상태를 '메인 화면'으로 보내기 위해 델리게이트에 didImageOnOffDone 함수를 추가한다.




```
func didImageOnOffDone(_ controller: EditViewController, isOn: Bool)
```

16. 왼쪽의 내비게이터 영역에서 [ViewController.swift]를 선택한 후 소스의 가장 아래 있는 '}' 바로 위에 didImageOnOffDone 함수를 추가한다. 이 didImageOnOffDone 함수는 에디트 뷰 컨트롤러에서 함수를 호출하며 그때 isOn 을 전달해준다.

17. 이 isOn 을 '메인 화면'의 isOn 으로 보여주면 이 값이 이미지 뷰의 이미지에 적용되어 켜진 전구와 꺼진 전구로 표시된다. 즉, '수정 화면'의 스위치 값을 '메인 화면'에 전달하여 켜진 전구 또는 꺼진 전구를 표현한다. 스위치 값에 따라 전구를 켜고 끄기 위해서 if else 문을 사용한다. didImageOnOffDone 함수 안에 소스를 입력한다.




```
func didImageOnOffDone(_ controller: EditViewController, isOn: Bool) {
    if isOn {
        imgView.image = imgOn
        self.isOn = true
    } else {
        imgView.image = imgOff
        self.isOn = false
    }
}
```

18. 왼쪽의 내비게이터 영역에서 [EditViewController.swift]를 선택한다. btnDone 함수를 수정할 것이다. '수정 화면'의 btnDone 함수에서 didImageOnOffDone 을 호출화면서 '수정 화면'의 스위치 상태를 isOn 으로 전달한다. 즉 '수정 화면'의 스위치 상태를 '메인 화면'으로 전달한다.

19. 전체 결과 화면

이제 [실행] 버튼을 클릭한 후 '메인 화면'에서 [수정] 버튼을 클릭하면 '수정 화면'으로 이동함과 동시에 'segue:use button'이 표시되고, '메인 화면'에서 [Edit] 버튼을 클릭하면 '수정 화면'으로 이동함과 동시에 'segue:use Bar button'이 표시된다.