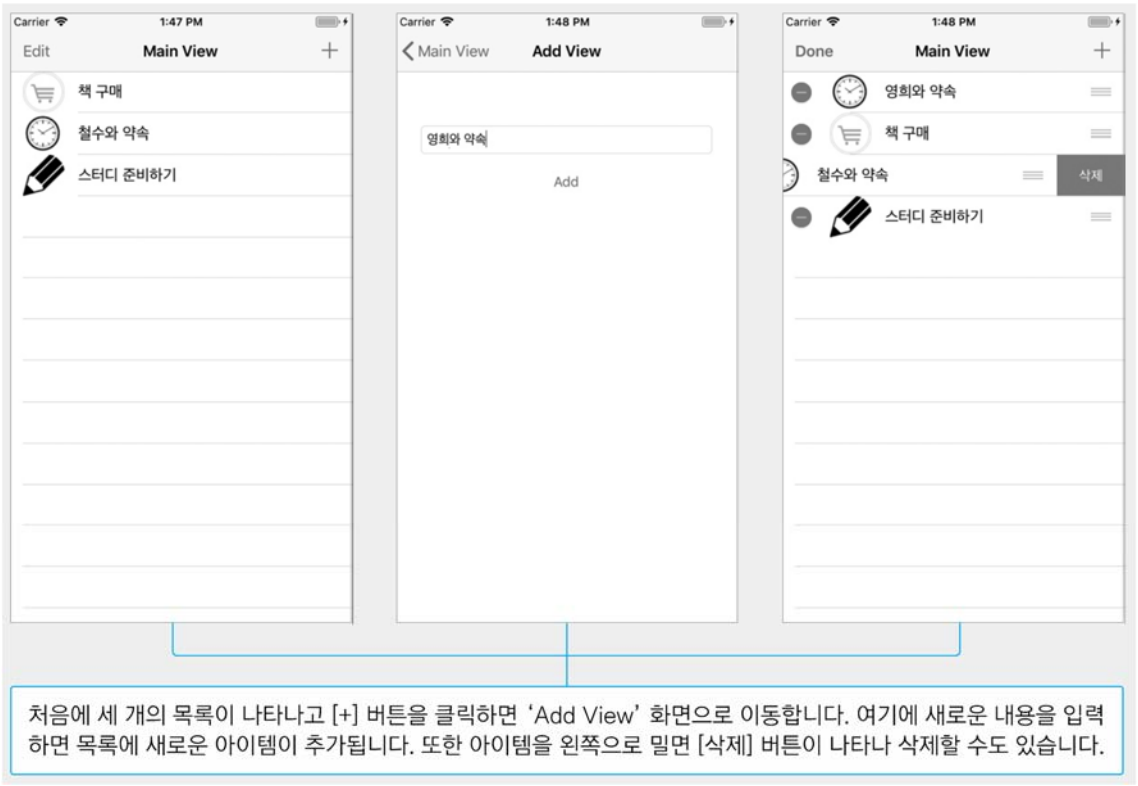
12 장. 테이블 뷰 컨트롤러 이용해 할 일 목록 만들기

알람 앱, 메모장 앱 등 아이폰 앱에서 자주 보고 익숙하게 사용하고 있는 '목록' 기능은 테이블 뷰 컨트롤러(Table View Controller)를 이용해서 구현할 수 있다. 테이블 뷰 컨트롤러에 대해 살펴 본다.

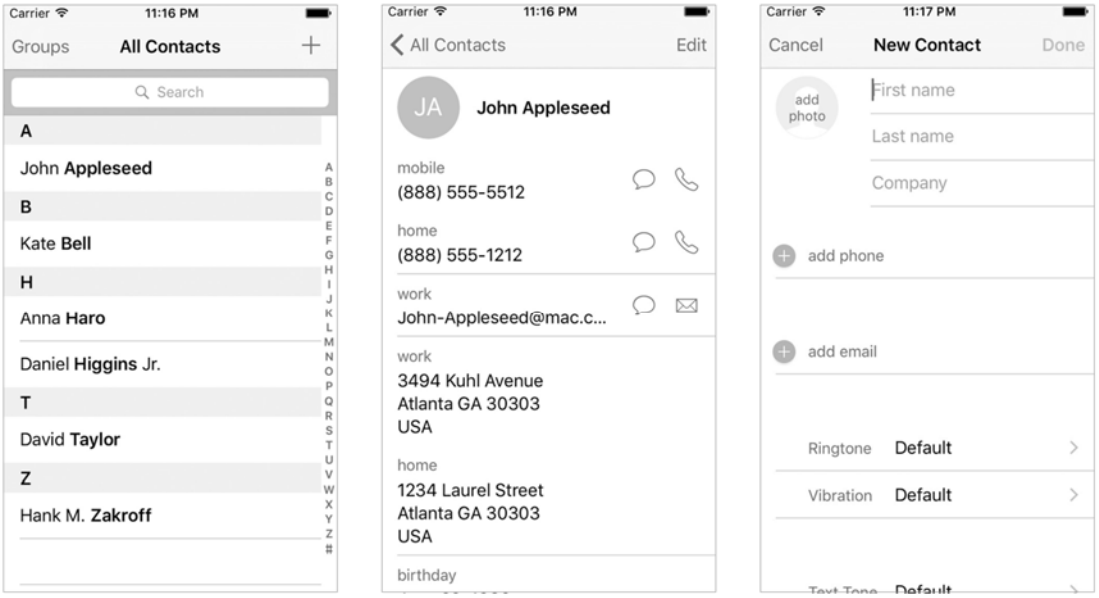
목록 기능은 항목을 추가할 수 잇는 기능뿐만 아니라 항목을 삭제하거나 이동할 수 있는 기능, 항목을 선택하면 내용을 볼 수 있는 기능을 구현한다.



12-1 테이블 뷰 컨트롤러란?

테이터를 목록 형태로 보여 주기 위한 가장 좋은 방법은 테이블 뷰 컨트롤러(Table View Controller)를 이용하는 것이다. 테이블뷰 컨트롤러는 사용자에게 목록 형태의 정보를 제공해 줄 뿐만 아니라 목록의 특정 항목을 선택하여 세부사항을 표시할 때 유용하다. 이런 테이블 뷰 컨트롤러를 잘 설명해 줄 대표적인 앱으로 알람, 메일, 연락처 등이 있다.

연락처 앱을 살펴보면 메인 화면에 이름 목록이 나타나고 이 목록 중에서 특정 사람을 클릭하면 그 사람의 세부 정보가 나타난다. 그리고 메인 화면에서 [+]버튼을 클릭하면 새 연락처를 추가할 수 있다.



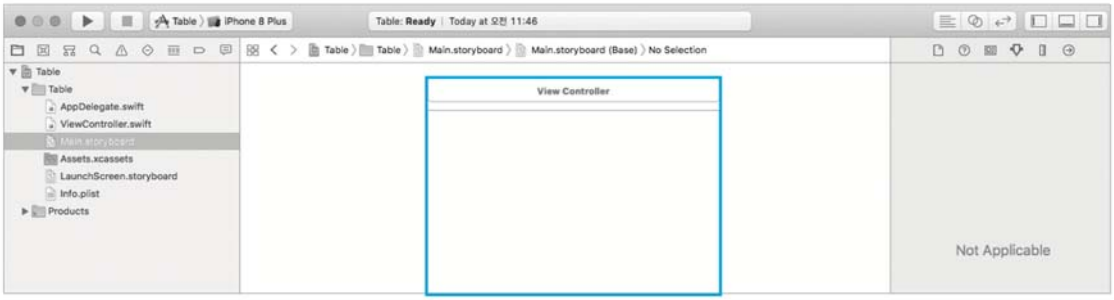
연락처 앱

12-2 할일목록앱을위한기본환경구성하기

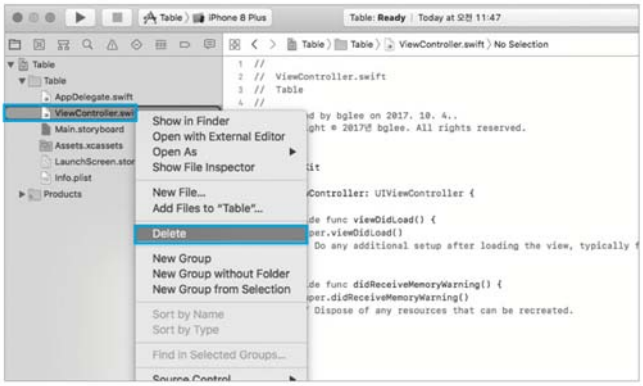
할 일 목록 앱을 만들기 위해서는 테이블 뷰 컨트롤러를 사용해야 한다. 이를 위해 스토리보드에서 기존에 있는 뷰 컨트롤러를 삭제하고 스위프트 파일도 삭제한다.

1. Xcode 를 실행한 후 'Table'이라는 이름으로 새 프로젝트를 만든다.
2. 기본 뷰 컨트롤러 삭제하기

테이블 뷰 컨트롤러를 사용해야 하므로 기존의 뷰 컨트롤러를 삭제하자. 아이폰 모양의 스토리보드를 클릭한 후 [delete]를 눌러 삭제한다.



3. 스토리보드에서 뷰 컨트롤러를 삭제하더라도 연결되어 있는 스위프트 파일은 삭제되지 않는다. 따라서 왼쪽의 내비게이터 영역의 [ViewController.swift]를 마우스 오른쪽 버튼으로 클릭한 후 [Delete]를 선택한다.



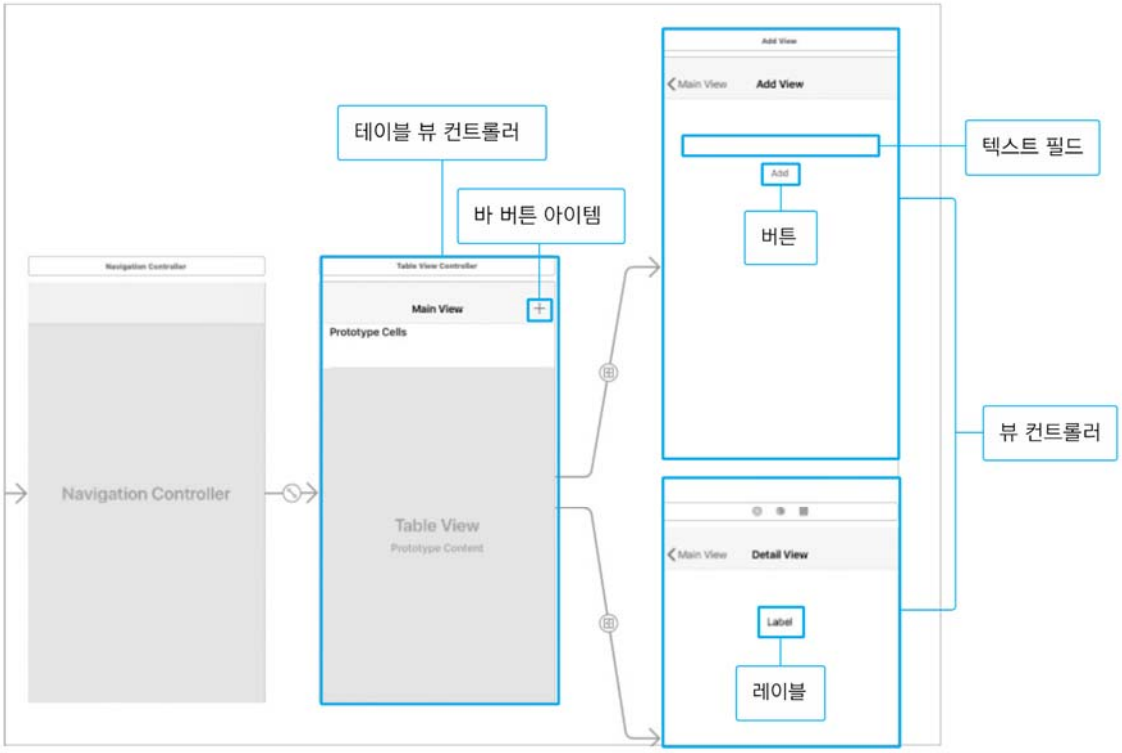
4. 스위프트 파일을 삭제할 것인지, 파일은 남겨두고 프로젝트에서 없앨 것인지를 묻는 경고 창이 나타난다. 이 예제에서는 파일이 필요 없기 때문에 [Move to Trash]를 클릭하여 삭제한다.

5. 이제 프로젝트 내에는 'ViewController.swift'가 사라지고 텅 빈 스토리보드만 보인다,

12-3 스토리보드 꾸미기

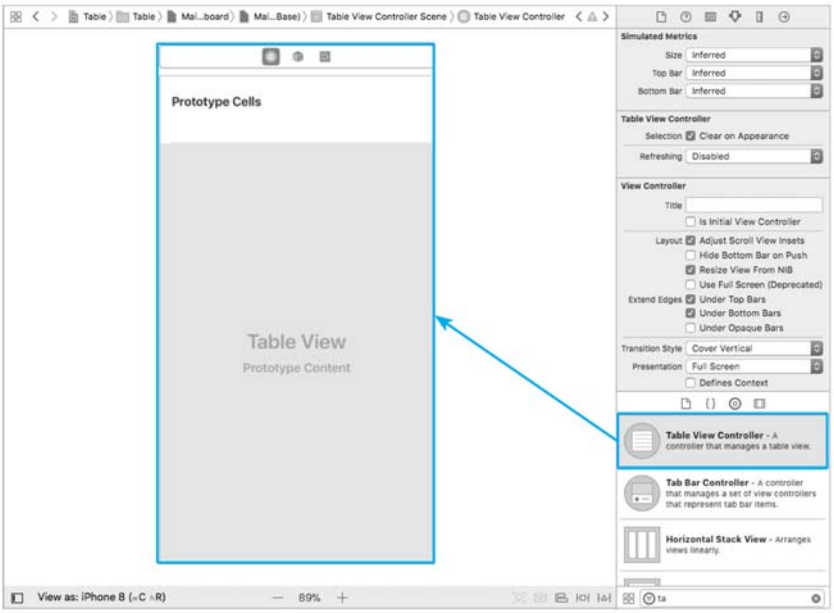
할 일 목록을 보여 주는 화면과 할 일을 추가하는 화면 그리고 목록 중 한 가지 할 일을 보여 주는 화면, 이렇게 세 개의 화면을 만들어 보자. 할 일 추가 화면에서는 텍스트 필드를 이용하여 할 일을 추가하고, 할 일을 보여주는 화면에서는 레이블을 추가한다.

이제 텅 빈 화면에 할 일 목록을 보여 주는 테이블 뷰 컨트롤러(Table View Controller)와 두개의 뷰 컨트롤러(View Controller)를 추가하고, 추가 화면으로 이동하기 위한 바 버튼 아이템(Bar Button Item), 추가할 일을 압력하기 위한 텍스트 필드(Text Field), 버튼(Button)을 추가하고 할 일을 보여 줄 레이블(Label)을 추가하여 스토리보드를 꾸며 보자.



1. 테이블 뷰 컨트롤러 추가하기

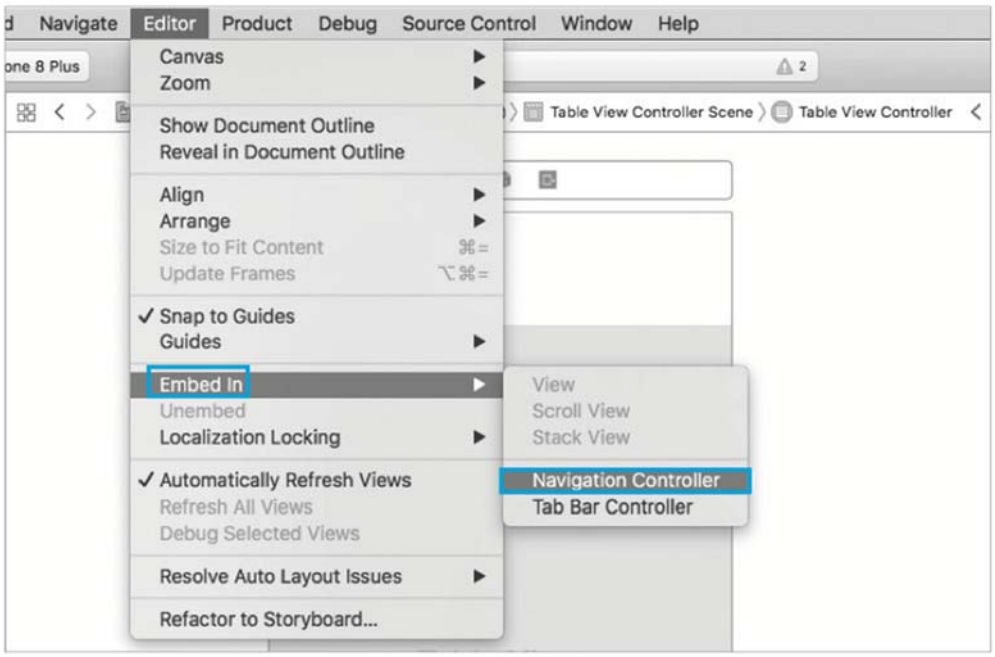
오른쪽 아랫부분의 오브젝트 라이브러리 검색란에 'ta'를 입력하여 검색한 후 [테이블 뷰 컨트롤러(Table View Controler)]를 찾아 스토리보드에 끌어 놓는다.



2. 내비게이션 컨트롤러 추가하기

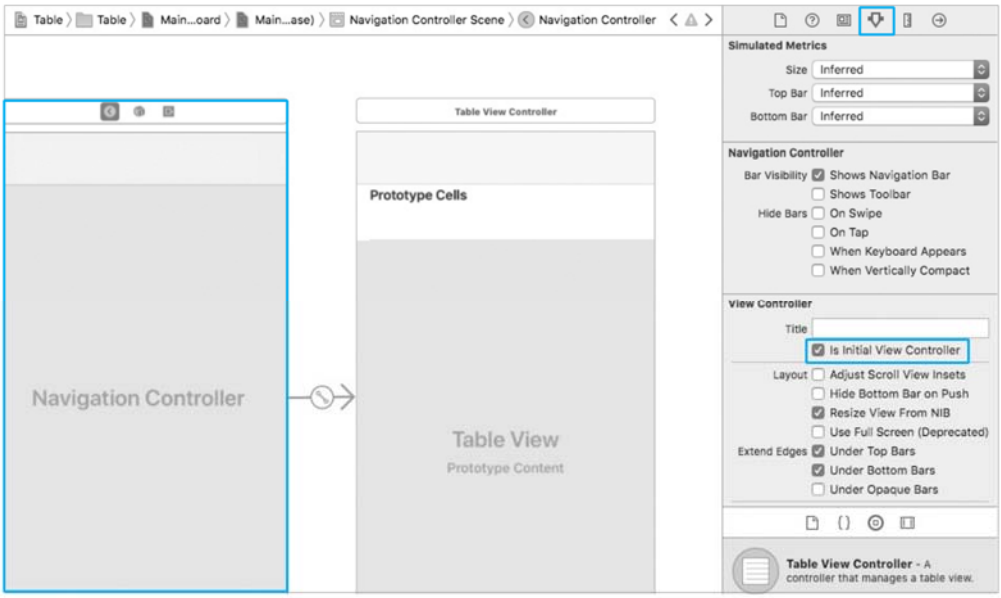
테이블에 들어갈 새로운 리스트를 추가하고 리스트를 편집하려면 두개의 뷰 컨트롤러가 필요하다. 이 두 개의 뷰로 화면을 전환하기 위해 내비게이션 컨트롤러를 추가한다.

메뉴에서 [Editor -> Embed in -> Navigation Controller]를 클릭한다.



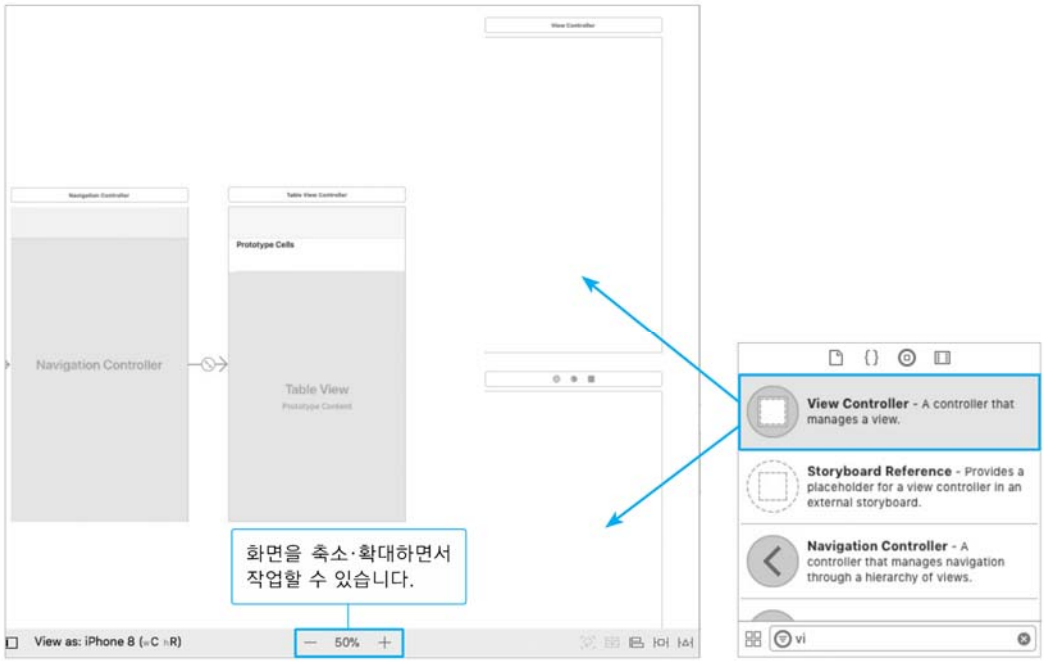
3. 테이블 뷰 컨트롤러에 내비게이션 컨트롤러가 추가된 것을 확인할 수 있다.

4. 내비게이션 컨트롤러를 드래그하여 선택하고 오른쪽 인스텍터 영역의 [Attributes inspector]에서 [is Initial View Controller]항목에 체크한다. 앱이 실행될 때 처음으로 가야 할 뷰 컨트롤러를 내비케이션 컨트롤러로 선택한다.



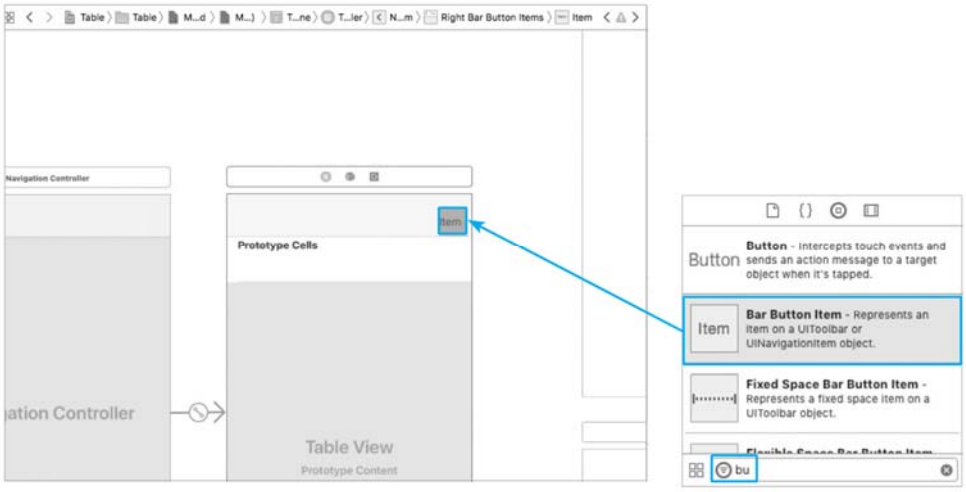
5. 뷰 컨트롤러 추가하기

오른쪽 아랫부분의 오브젝트 라이브러리에서 [뷰 컨트롤러(View Controller)]를 찾아 테이블 뷰 컨트롤러의 오른쪽에 위, 아래로 두 개를 추가한다.

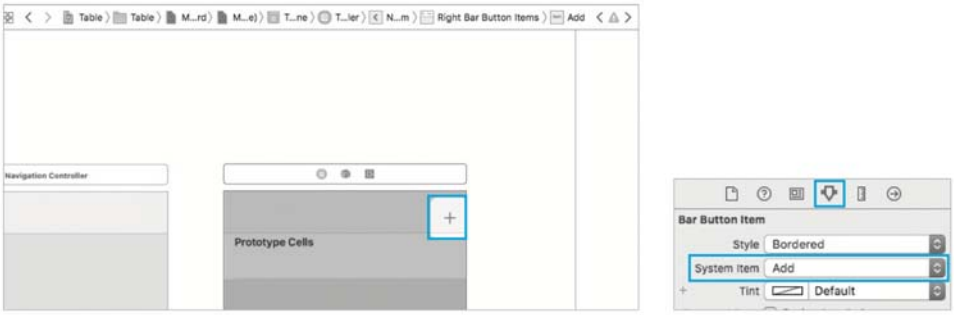


6. 뷰 전환 추가하기

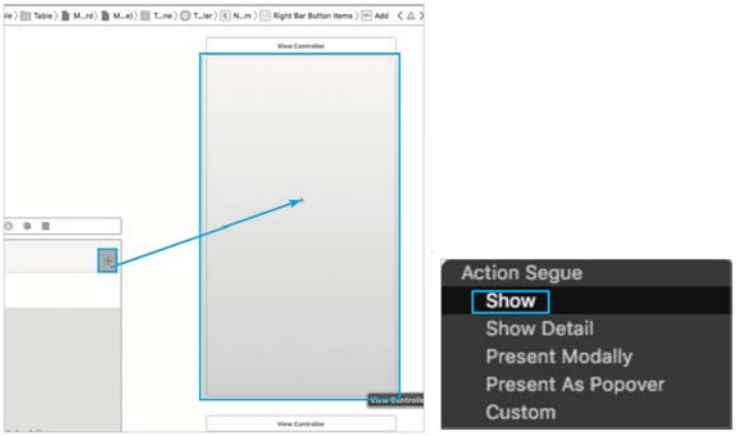
다시 오른쪽 아랫부분의 오브젝트 라이브러리에서 [바 버튼 아이템(Bar Button Item)]을 찾아 테이블 뷰 컨트롤러의 오른쪽 윗부분에 끌어다 놓는다.



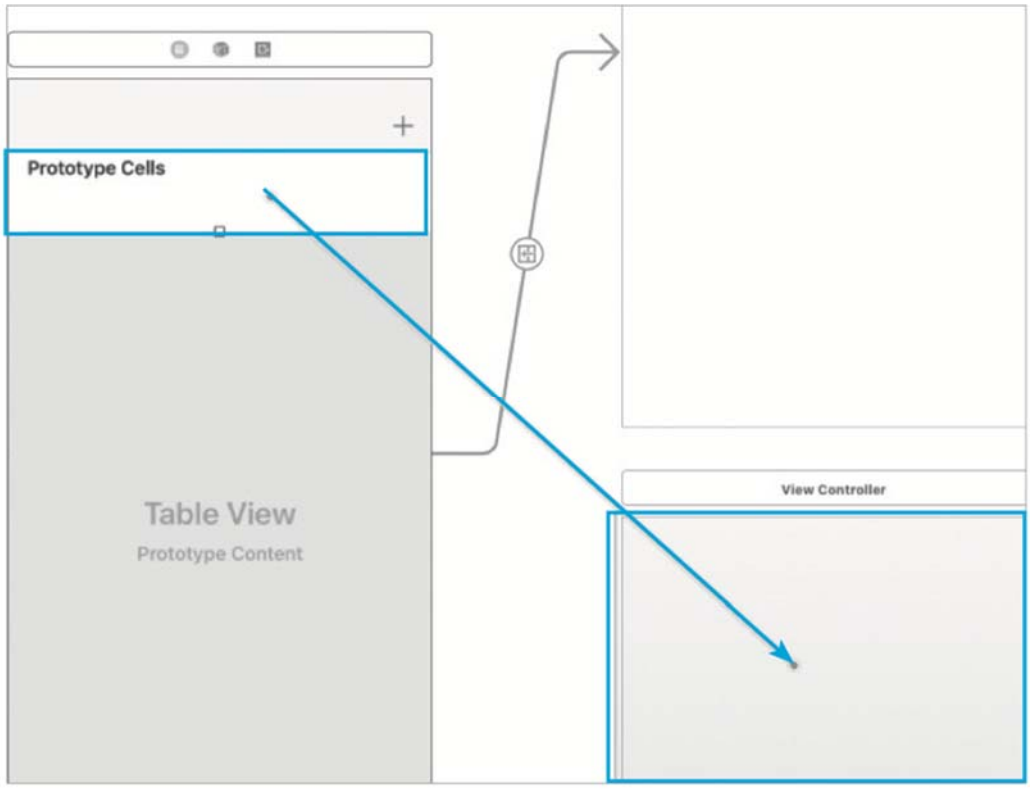
7. 이제 바 버튼 아이템의 아이콘 모양을 수정한다, 먼저 조금 전에 추가한 바 버튼 아이템을 선택한다. 그런 다음 오른쪽의 인스펙터 영역에서 [Attributes inspector]버튼을 클릭한 후 System Item 을 [Add]로 수정한다.



8. 바 버튼 아이템을 마우스 오른쪽 버튼으로 클릭하여 테이블 뷰 컨트롤러의 오른쪽 윗 부분에 있는 뷰 컨트롤러로 드래그 한다. [액션 세그웨이]에서 [Show]를 선택한다.



10. 테이블 뷰 컨트롤러 위쪽의 [Prototype Cells]를 선택한 후 마우스 오른쪽 버튼으로 클릭한 채 이번엔 아랫쪽의 뷰 컨트롤러로 드래그하여 연결한다. 뷰 컨트롤러가 전체적으로 파랗게 되면 마우스의 버튼에서 손을 뗀다.



11. 앞에서와 마찬가지로 [액션 세그웨이]에서 [Show]를 선택한다.

** 세그웨이 지정 방법 다섯 가지 **

세그웨이는 다음의 다섯 가지 방법으로 지정할 수 있다.

1) Show : Swift 2.x 에서의 [push]와 유사하다. 기본 뷰 컨트롤러를 불러올 때 새로운 뷰 컨트롤러가 스택(stack)에 푸시(push)하면서 활성화된다.

다시 이전 화면으로 돌아갈 때는 새로운 뷰 컨트롤러가 팝(pop)이 되면서 밑에 있던 기존의 뷰 컨트롤러가 활성화되는 형태이다. 이때 가장 위에 있는 뷰 컨트롤러가 활성화된다.

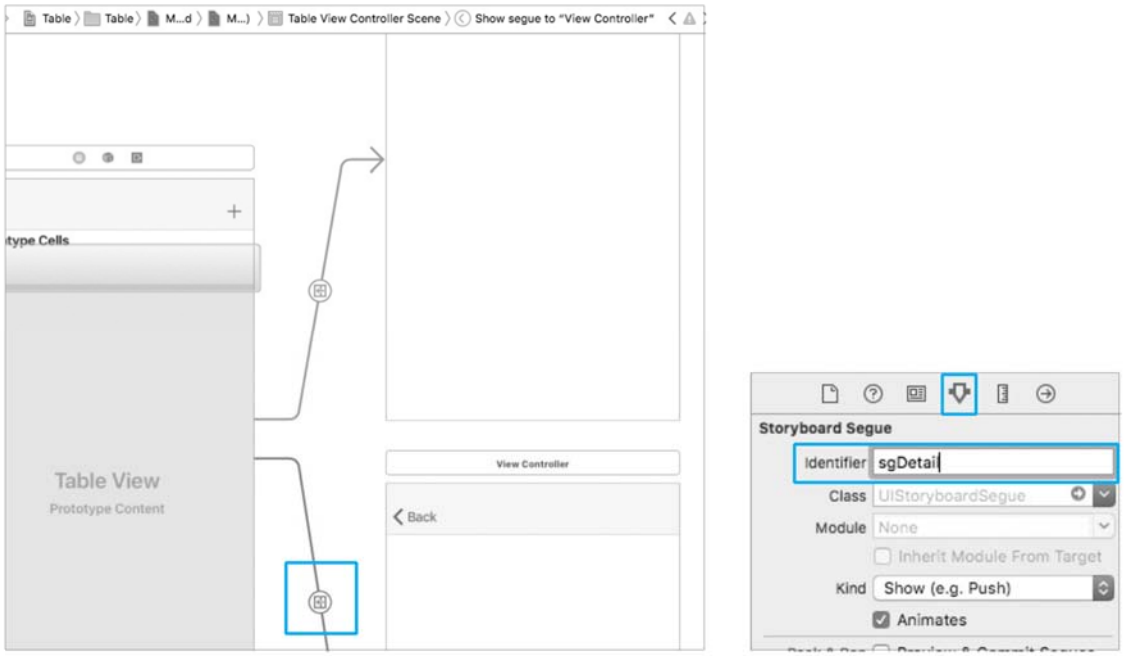
2) Show Detail : [Show]와 매우 비슷하지만 푸시[push]가 아니라 교체(replace)된다는 점이 다르다. 현재 뷰 컨트롤러 스택(stack)의 최상단 뷰를 교체한다.

3) Plasent Modally : 새로운 뷰 컨트롤러를 보여 주는 스타일과 화면 전환 스타일을 결정하여 뷰 보달(modal)형태로 준다.

4) Present As Popover : 현재 보이는 뷰 컨트롤러 위에 앵커를 가진 팦업 형태로 콘텐츠 뷰를 표시한다.

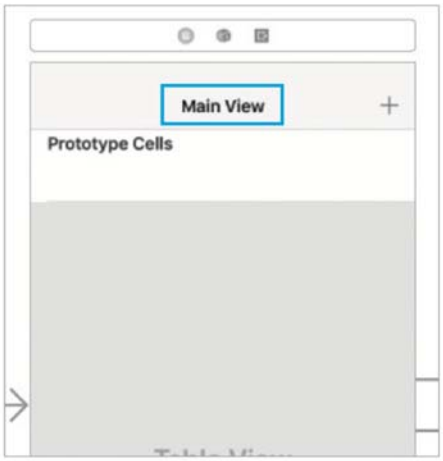
5) Custom : 개발자가 임의로 지정한 동작을 수행한다.

12. 이번에는 아래쪽 세그웨이를 선택한다. 그런 다음 오른쪽의 인스펙느 영역에서 [Attributes inspector]버튼을 클릭한 후 Identifier 에 'seDetail'을 입력한다. 뷰가 전환될 때 전달할 데이터가 있다면 여기서 지정한 세그웨이의 이름을 활용한다.

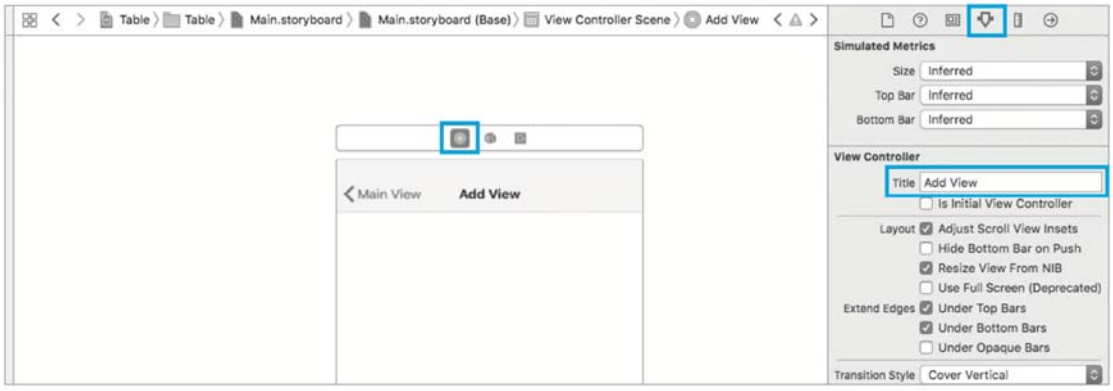


13. 각 테이블 뷰 컨트롤러에 제목을 추가한다. 테이블 뷰 컨트롤러의 위쪽을 더블 클릭한다.

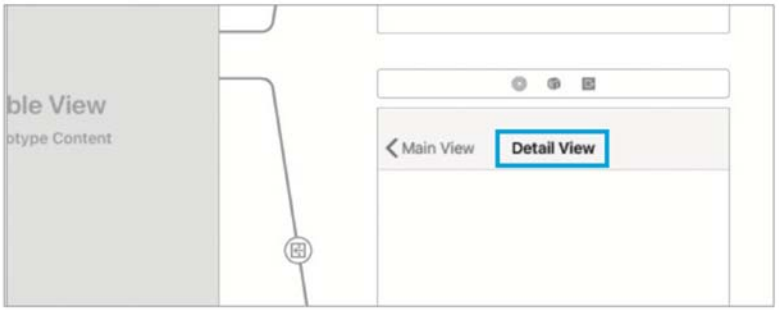
14. 테이블 뷰 컨트롤러에는 전체 리스트들을 보여 줄 것이므로 'Main View'를 입력한다.



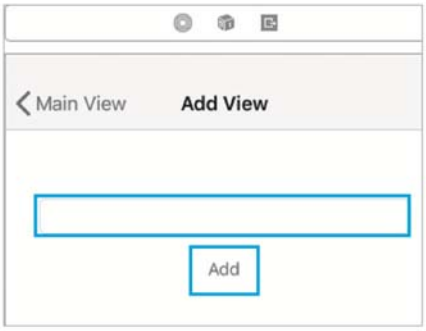
15. 이번에는 위쪽 뷰 컨트롤러의 (View Controller)를 드래그하여 선택한다.
그런 다음 오른쪽의 인스펙터 영역에서 [Atrributes inspector]버튼을 클릭한 후 Title 에 'Add View'를 입력한다.



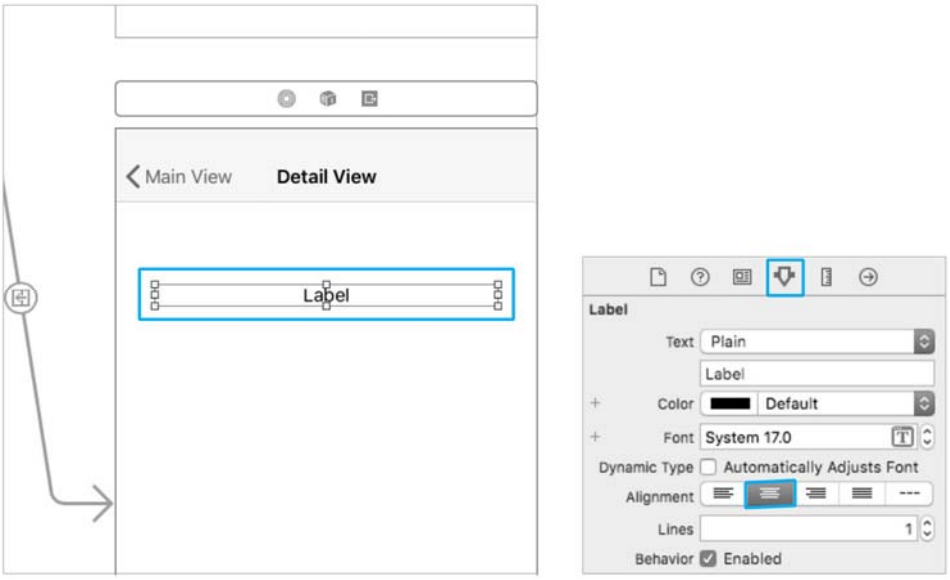
16. 같은 방법으로 아래쪽 뷰 컨트롤러의 Tilte 에 'Detail View'를 추가한다.



17. 뷰 컨트롤러에 컴포턴트 추가하기
'Add View' 뷰 컨트롤러에 텍스트 필드(Text Field)와 버튼을 오른쪽 그림과 같이 추가한다. 그리고 버튼을 더블 클릭하여 'Add'로 수정한다.



18. 'Detail View'뷰 컨트롤러에 레이블을 다음 그림과 같이 추가한다. 그리고 오른쪽 위쪽의 [Attributes inspextor]버튼을 클릭한 후 Alignment 의 두 번째 항목을 클릭한다.
이렇게 하면 레이블이 가운데 정렬로 바뀐다.

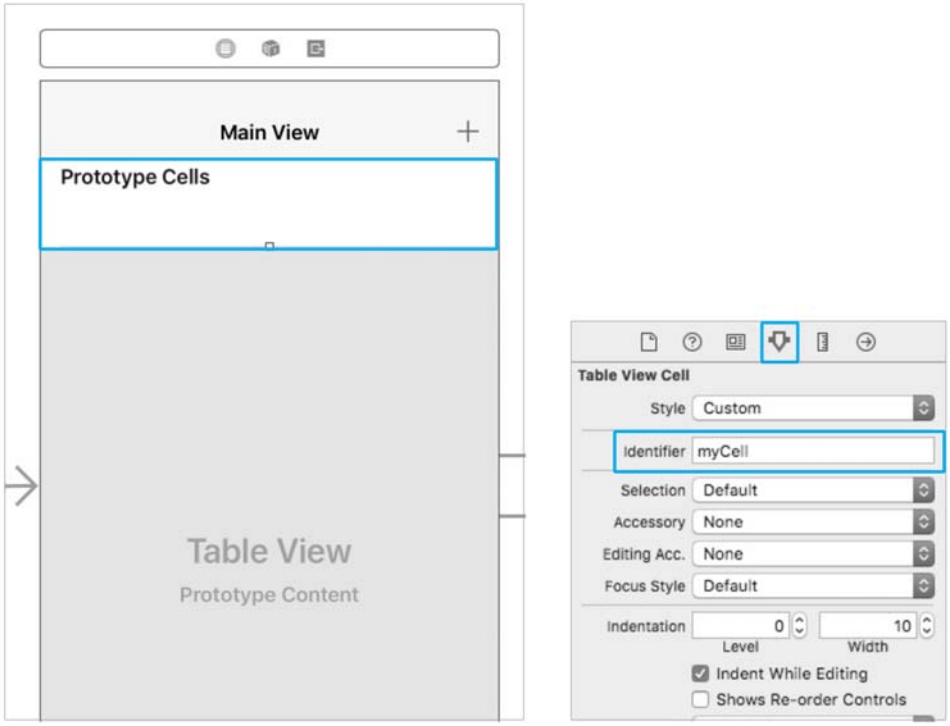


19. 스토리보드를 축소하면 다음 그림처럼 배치된다.

20. 테이블 뷰에 셀 추가하기

마지막으로 'Main View'컨트롤러에서 [Prototype Cells]를 선택하고 오른쪽의 인스팩터 영역에서 [Attributes inspector]버튼을 클릭한 후 Identifier 에 'myCell'을 입력한다.

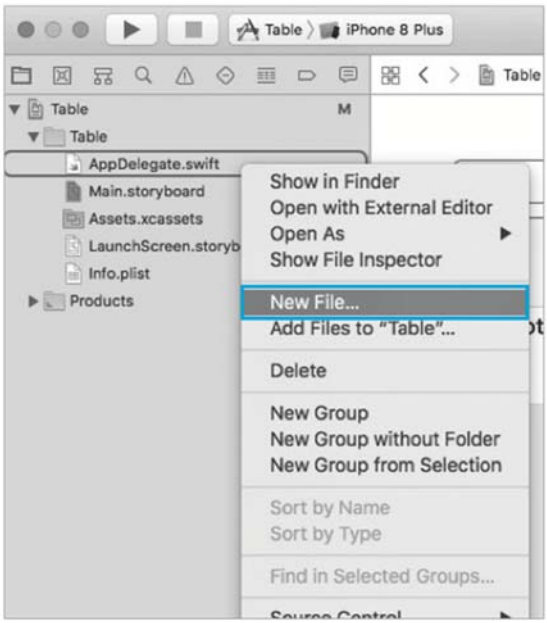
테이블 뷰 컨트롤러의 셀 이름을 'myCell'로 정한 것이다.



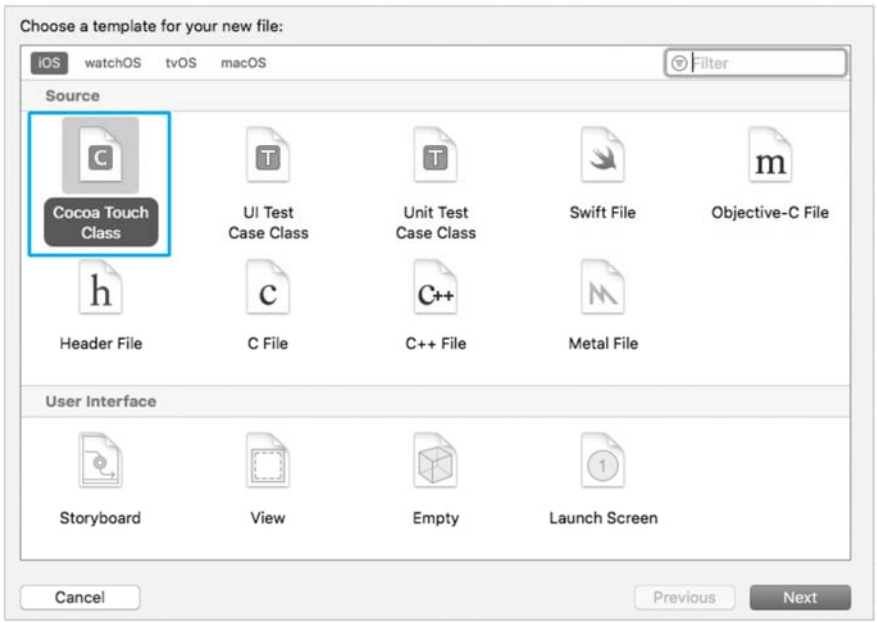
*** 스위프트 파일 추가하기 ***

1. 스위프트 파일 추가하기

스토리보드는 완성되는데 스위프트(swift)소스 파일에 없으므로 여기에서 추가한다. 왼쪽의 내비게이터 영역에서 마우스 오른쪽 버튼을 클릭한 후 [New File...]을 선택한다.

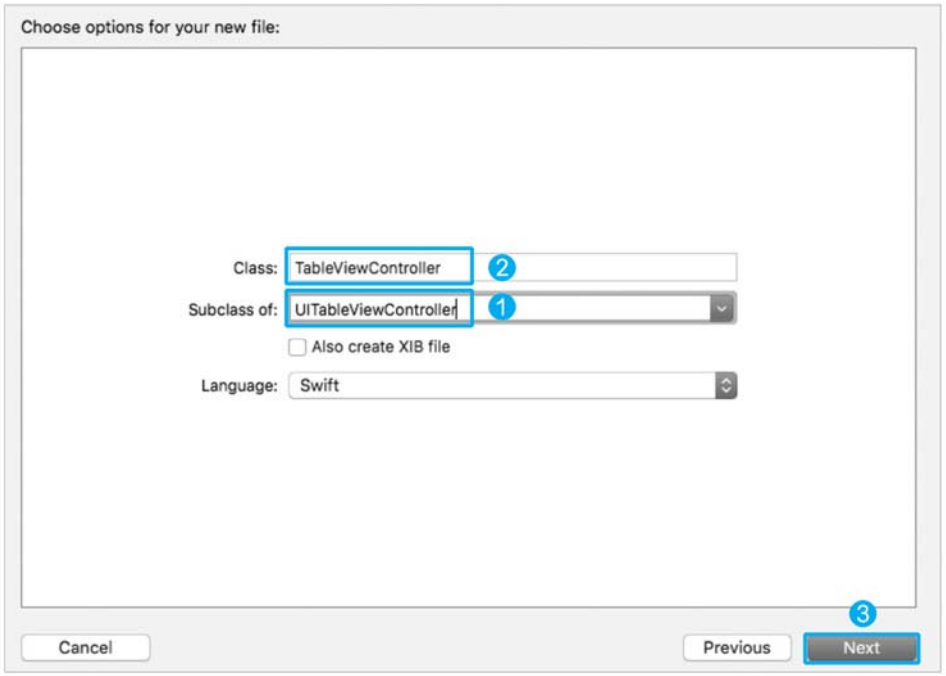


2. 다음 화면에서 [Swift File]이 아니라 [Cocoa Touch Class]를 선택한다.



3. 서브 클래스를 [UITableViewController]로 선택하면 클래스명이 자동으로 'TableViewController'로 바뀐다. [Next] 버튼을 클릭하고 저장할 폴더를 클릭하면 파일이

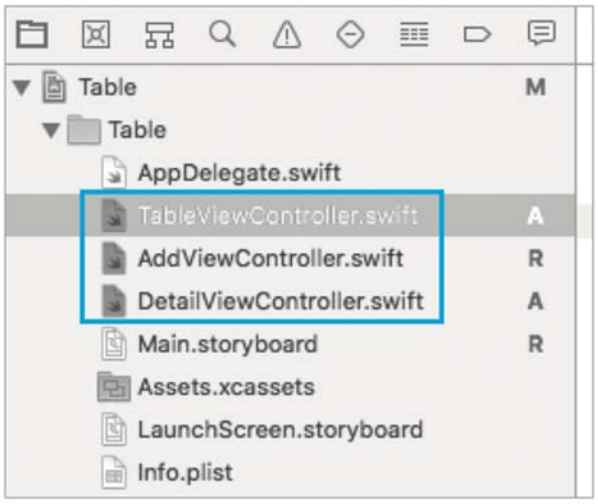
생성된다.



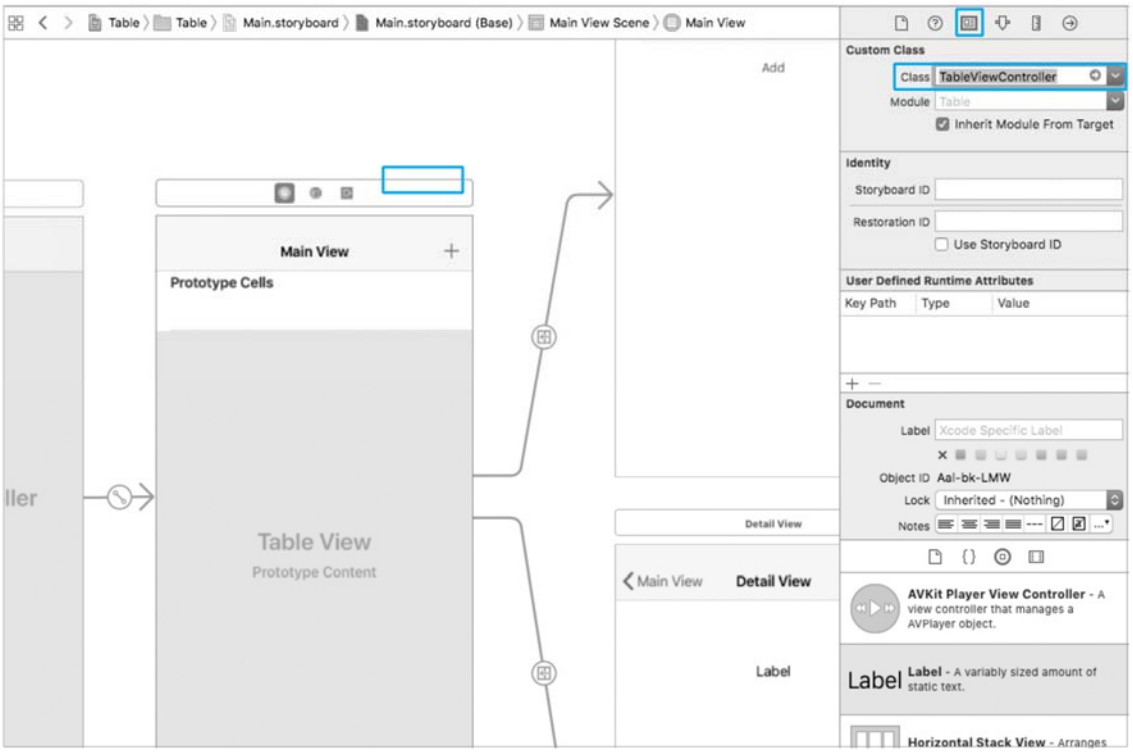
4. 클래스명은 'AddViewController'이고 서브 클래스는 [UIViewController]인 파일을 생성한다.

5. 마지막으로 동일한 방법으로 클래스명은 'DetailViewController'이고, 서브 클래스는 [UIViewController]인 파일을 생성한다.

6. 세 개의 스위프트 파일이 생성된다.



7. 다시 스토리보드로 돌아가서 'Main View' 컨트롤러를 선택하고 [Identity inspector]버튼을 클릭한 후 클래스(Class)를 [TableViewController]로 선택한다. 그러면 테이블 뷰 컨트롤러와 'TableViewController.swift'파일이 연결된다.



8. 'Add View'컨트롤러를 선택하고 [Indentity inspector]버튼을 클릭한 후 클래스(Class)를 [AddViewController]로 선택한다. 그러면 뷰 컨트롤러와 'AddViewController.swift'파일이 연결된다.

9. 'Detail View'컨트롤러를 선택하고 [Identity inspector]버튼을 클릭한 후 클래스(Class)를 [DetailViewController]로 선택한다. 그러면 뷰 컨트롤러와 'DetailViewController.swift'파일이 연결된다.

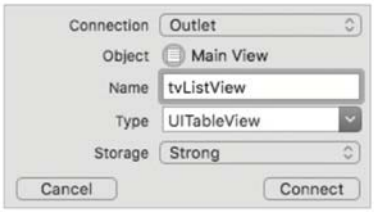

Table
Table
Main.storyboard
Main.storyboard (Base)
Detail View Scene
Detail View
Table View
Prototype Content
Main View
Detail View
Label
Custom Class
Class
DetailViewController
Module
Table
Inherit Module From Target
Identity
Storyboard ID
Restoration ID
Use Storyboard ID
User Defined Runtime Attributes
Key Path
Type
Value
Document
Label
Xcode Specific Label
Object ID
EIG-9x-RIV
Lock
Inherited - (Nothing)

12-4 아웃렛 변수와 액션 함수 추가하기

1. 왼쪽의 내비게이터 영역에서 [Main.storyboard]를 선택하고 [Show the Assistant editor] 버튼을 클릭해 화면을 분할한다. 왼쪽 창에서 'Main View'뷰 컨트롤러를 선택한 후 오른쪽 보조 편집기 영역의 클래스명을 보고 파일이 'TableViewController.swift'인지를 확인한다. 그리고 [테이블 뷰(Table View)]를 마우스 오른쪽 버튼으로 선택한 후 드래그해서 오른쪽 보조 편집기 영역의 클래스 선언문 바로 아래에 갖다 놓는다.

아래 표를 참고하여 아웃렛 변수의 이름과 유형을 설정한다.

위치	테이블 뷰 컨트롤러의 클래스 선언문 바로 아래
연결(Connection)	Outlet
이름(Name)	tvListView
유형(Type)	UITableView




```
@IBOutlet var tvListView: UITableView!
```

3. 왼쪽 창에서 'Add View'뷰 컨트롤러를 선택한 후 오른쪽 보조 편집기 영역의 클래스명을 보고 파일이 'AddViewController.swift'인지를 확인한다. 그리고 [텍스트 필드]를 마우스 오른쪽 버튼으로 선택한 루 드래그해서 오른쪽 보조 편집기 영역의 클래스 선언문 바로 아래에 갖다 놓는다.

아래 표를 참고하여 아웃렛 변수의 이름과 유형을 설정한다.

위치	애드 뷰 컨트롤러의 클래스 선언문 바로 아래
연결(Connection)	Outlet
이름(Name)	tfAddItem
유형(Type)	UITextField


```
@IBOutlet var tfAddItem: UITextField!
```

5. [Add]버튼을 마우스 오른쪽 버튼으로 선택한 후 드래그해서 오른쪽 보조 편집기 영역의 didReceiveMemoryWarning 함수 아래에 갖다 놓는다.

아래 표를 참고하여 이름과 유형을 설정한다, 이번엔 버튼에 액션을 추가할 것이므로 [Action]을 선택한다.

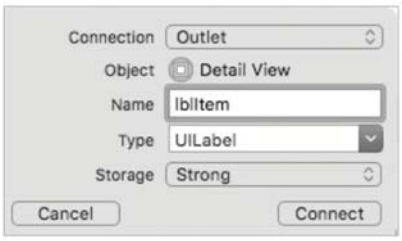
위치	didReceiveMemoryWarning 함수 아래
연결(Connection)	Action
이름(Name)	btnAddItem
유형(Type)	UIButton


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

7. 마지막으로 왼쪽 창에서 'Detail View'뷰 컨트롤러를 선택한 후 오른쪽 보조편집기 영역의 클래스명을 보고 파일이 'DetailViewController.swift'인지를 확인 한다. 그리고 [레이블(Label)]을 마우스 오른쪽 버튼으로 선택한 후 드래그해서 오른쪽 보조편집기 영역의 디테일 뷰 컨트롤러 클래스 선언문 바로 아래에 갖다 놓는다.

아래 표를 참고하여 아웃렛 변수의 이름과 유형을 설정한다.

위치	디테일 뷰 컨트롤러의 클래스 선언문 바로 아래
연결(Connection)	Outlet
이름(Name)	lblItem
유형(Type)	UILabel



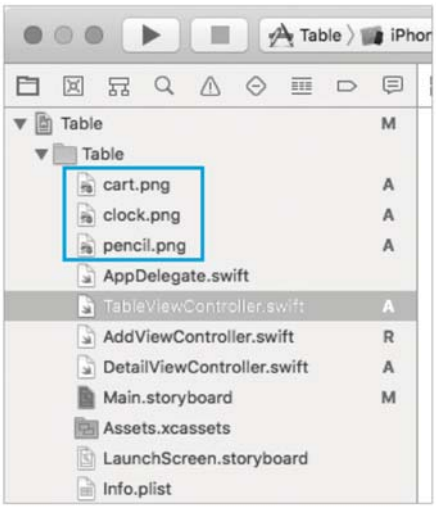

```
@IBOutlet var lblItem: UILabel!
```

12-5 테이블목록보여주기

앱을 실행했을 때 기본적인 아이콘이 포함된 세 개의 목록이 나타나게 하려고 한다. 목록과 함께 아이콘으로 사용될 이미지를 추가하고 목록을 보여주는 코드를 작성해 보자.

1. 코딩을 위해 화면 모드를 수정한다. 오른쪽 윗 부분에서 [Show the Standard editor] 버튼을 클릭한다.

2. 목록 앱을 만들 때 사용할 이미지 파일인 'cart.png', 'clock.png' 그리고 'pencil.png'파일을 추가한다. 그리고 이미지가 [Target Membership]에 추가되었는지 확인한다.

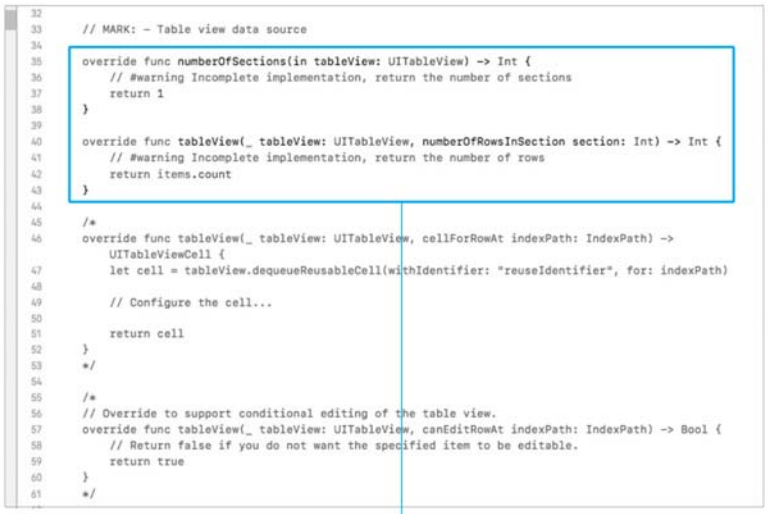


3. 왼쪽의 내비게이터 영역에서 [TableViewController.swift]를 선택한다. 그리고 아래 소스를 입력하여 앞에서 추가한 이미지 파일을 외부 변수인 'items'와 'ItemsImageFile'로 선언한다. 이렇게 하면 모든 클래스에서 이미지를 사용할 수 있다.

```
import UIKit

var items = ["책 구매", "철수와 약속", "스터디 준비하기"]
var itemsImageFile = ["cart.png", "clock.png", "pencil.png"]

class TableViewController: UITableViewController {
```

1) 외부 변수인 items 의 내용을 각각 "책 구매', '철수의 약속', '스터디 준비하기'로 지정한다.
2) 외부 변수인 이미지 파일은 각각 "cart.png", "clock.png","pencil.png"이다.

4. 소스의 아래쪽에 있는 두 개의 함수를 조금씩 바꿔 보자.



```
override func numberOfSections(in tableView: UITableView) -> Int {
    // #warning Incomplete implementation, return the number of sections
    return 1 ❶
}

override func tableView(_ tableView: UITableView, numberOfRowsInSection
    section: Int) -> Int {
    // #warning Incomplete implementation, return the number of rows
    return items.count ❷
}
```

1) 보통은 테이블 안에 색션이 한 개이므로 numberOfSections 의 리턴 값을 1 로 한다.
2) 섹션당 열의 개수는 Items 의 개수이므로 tableView

5. 이번에는 tableView(_tableView: UITableView, cellForRowAt indexPath indexPath: indexPath) -> UITableViewCell 함수를 수정하자. 이 함수 앞에서 선언한 변수의 내용을 셀에 적용하는 함수로, 이후에도 이렇게 주석 처리된 함수를 계속 사용할 것이다,

6. 이 함수의 앞 뒤에 있는 '/*'과 '*/'을 지워서 주석을 제거한다. 이렇게 하면 주석이었던 글이 함수로 활성화 된다.

7. 셀의 레이블과 이미지에 앞에서 선언한 변수가 적용되도록 수정한다.

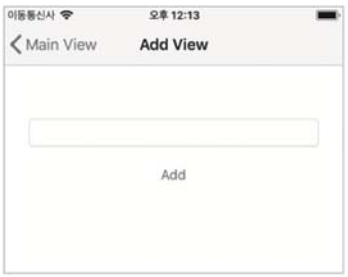
```
43
44      override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
45          // #warning Incomplete implementation, return the number of rows
46          return items.count
47      }
48
49
50      override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
            UITableViewCell {
51          let cell = tableView.dequeueReusableCell(withIdentifier: "myCell", for: indexPath)
52
53          cell.textLabel?.text = items[(indexPath as NSIndexPath).row]
54          cell.imageView?.image = UIImage(named: itemsImageFile[(indexPath as NSIndexPath).row])
55
56          return cell
57      }
58
59
60      /*
61      // Override to support conditional editing of the table view.
62      override func tableView(_ tableView: UITableView, canEditRowAt indexPath: IndexPath) -> Bool {
63          // Return false if you do not want the specified item to be editable.
64          return true
```

```
override func tableView(_ tableView: UITableView, cellForRowAt
    indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "myCell",
        for: indexPath)

    cell.textLabel?.text = items[(indexPath as NSIndexPath).row] (1)
    cell.imageView?.image = UIImage(named: itemsImageFile[(indexPath as
    NSIndexPath).row]) (2)

    return cell
}
```

1) 셀의 텍스트 레이블에 앞에서 선언한 items 을 대입한다. 그 내용은 "책 구매","철수와 약속","스터디 준비하기"이다.

2) 셀의 이미지 뷰에 앞에서 선언한 itemsImageFile("cart.png","clock.png","pencil.png")을 대입한다.

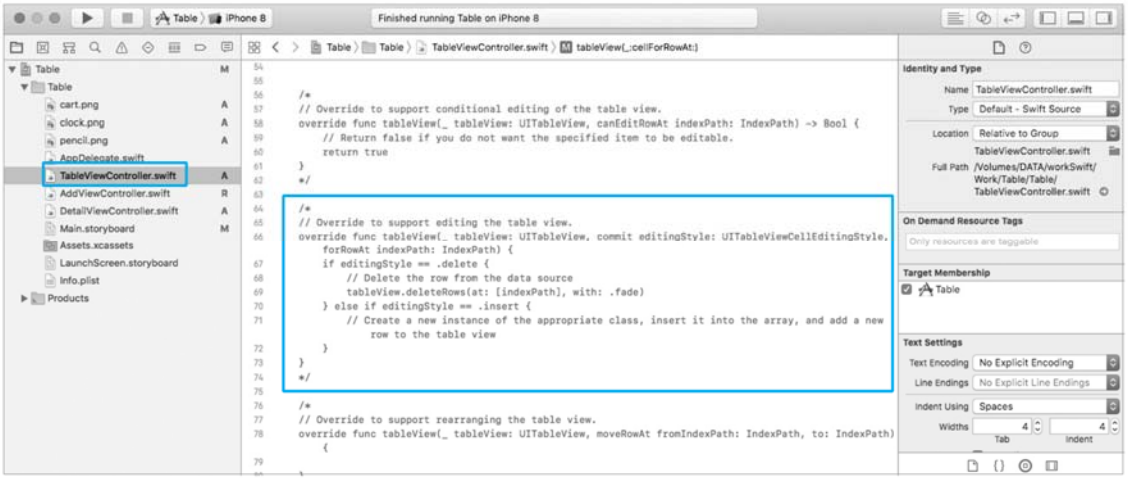
8. 결과 보기



12-6 목록 삭제하기

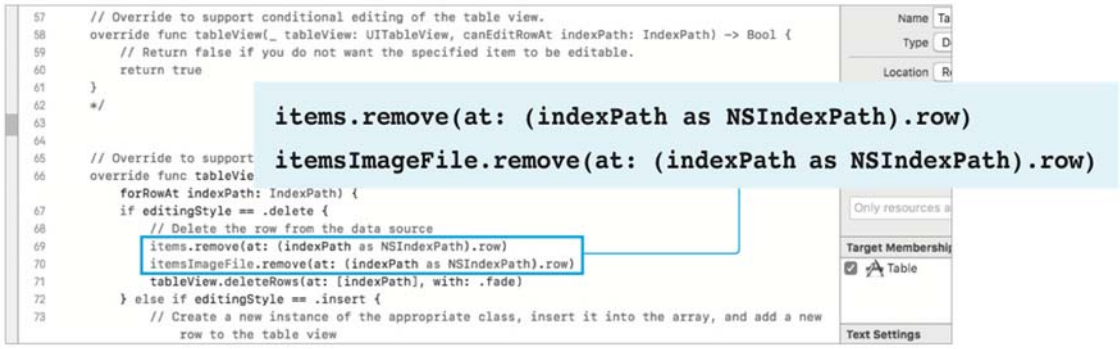
1. [TableViewController.swift]를 선택한 후 함수를 수정한다.

주석으로 되어 있는 tableView(_ tableView: UITableView, commit editingStyle: UITableView CellEditingStyle, forRowAt indexPath: IndexPath) 함수 주석 제거 후 코드 추가한다.



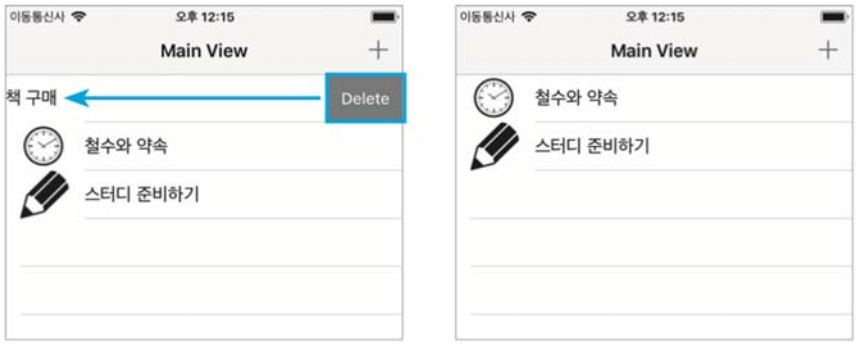
2. 함수의 앞뒤에 있는 '/*'과 '*/'을 지워서 주석문을 활성화한다.

3. 이 함수 안에 다음과 같이 선택한 셀을 삭제하는 코드를 삽입한다.



4. '목록 삭제'결과 보기

다시 [실행]버튼을 클릭한다. 한 셀을 왼쪽으로 밀면[Delete]버튼이 나타나고 이를 클릭하면 항목이 사라진다.



5. 'Delete'를 한글로 수정하기

한발 더 나아가 영문으로 사용된 'Delete'를 한글인 '삭제'로 바꿔 보자. 앞에서 계속 작업해온 [TableViewController.swift]를 선택한 후 다음 함수를 추가한다.

```
override func tableView(_ tableView: UITableView, titleForDelete
        ConfirmationButtonForRowAt indexPath:
        IndexPath) -> String? {
    return "삭제"
}
```

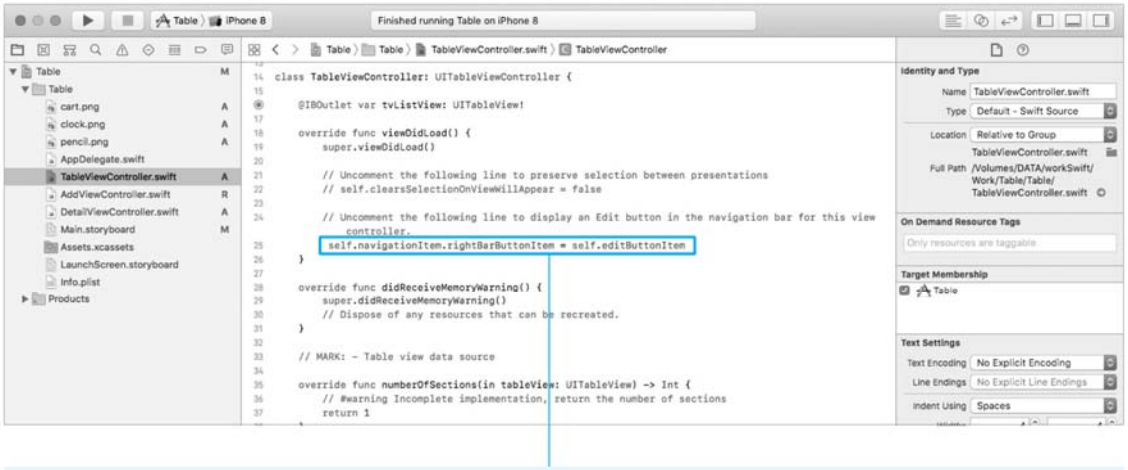
6. 버튼 이름 수정 후 다시 '목록 삭제 동작'결과 보기

다시 [실행] 버튼을 클릭한다. 그런 다음 한 셀을 왼쪽으로 밀면 [Delete]대신 [삭제] 버튼이 나타나고 이를 클릭하면 항목이 사라진다.

12-7 바 버튼으로 목록 삭제하기

목록을 밀어서 삭제하는 방법 외에 바 버튼을 이용하는 방법도 있다. 이번에는 [Edit]바 버튼을 만들고 다시 목록을 삭제해 보자.

1. 계속해서 [TableViewController.swift]를 선택한 후 수정한다. viewDidLoad 함수에서 self.navigationItem.rightBarButtonItem = self.editButtonItem 앞의 "//"와 뒤의 '()'를 지운다.



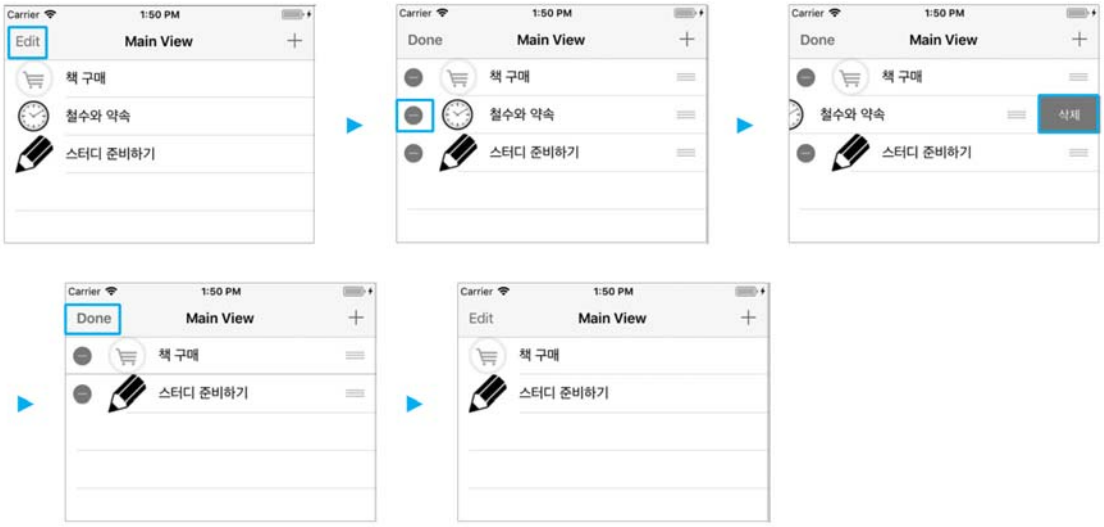
```
override func viewDidLoad() {
    super.viewDidLoad()

    // Uncomment the following line to preserve selection between presentations
    // self.clearsSelectionOnViewWillAppear = false

    // Uncomment the following line to display an Edit button in the ...
     self.navigationItem.rightBarButtonItem = self.editButtonItem
}
```

2. 오른쪽 화면에는 이미 [Add]버튼이 있으니 [edit]버튼은 왼쪽에 추가하겠다.

```
self.navigationItem.leftBarButtonItem = self.editButtonItem
```

3. 결과 보기

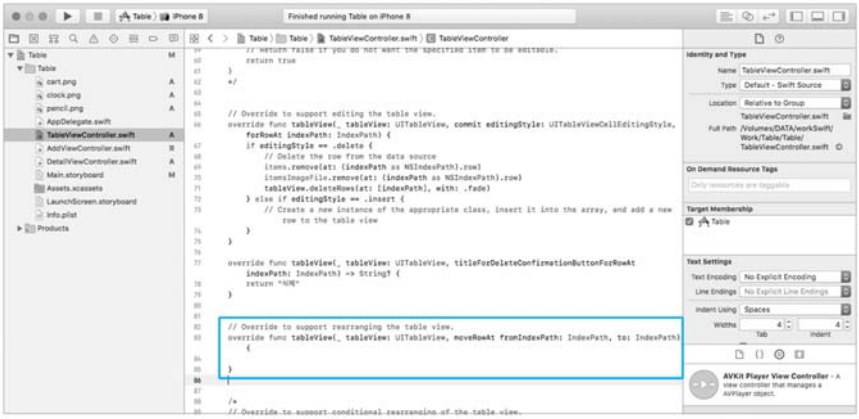
다시 [실행]버튼을 클릭한다. 그런 다음 왼쪽에 생성된 [Edit]버튼을 클릭하면 왼쪽에 붉은 원이 나타난다. 그 붉은 원모양의 버튼을 클릭하면 [삭제]이 나타나고 이를 클릭하면 삭제된다.



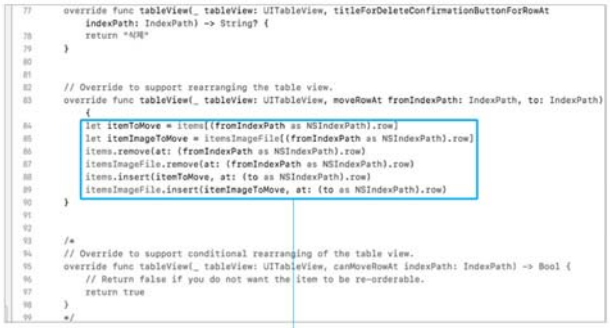
12-8 목록 순서 바꾸기

목록을 적다 보면 우선 순위로 중요한 목록을 위로 올리고 싶어질 때가 있다. 이를 위해서는 목록의 순서를 바꾸는 기능이 필요하다.
앞에서 만든 [Edit]버튼을 활용해서 이번엔 목록을 바꿔 보자.

1. [TableViewController.swift]를 선택한 후 수정하자.
tableView(_ tableView: UITableView, moveRowAt fromIndexPath: IndexPath
, to: IndexPath) 함수 주석 제거한다.
이 함수를 사용하면 목록을 옮길 수 있다.



2. 함수 안에 하나의 목록을 선택하여 다른 곳으로 이동하는 소스를 추가한다. 먼저 변수를 만들어서 이동할 변수를 기억한 후 이동할 목록을 삭제하고 변수에 저장된 내용을 이동한 곳으로 삽입한다.



```
override func tableView(_ tableView: UITableView, moveRowAt
    fromIndexPath: IndexPath, to: IndexPath) {

    let itemToMove = items[(fromIndexPath as NSIndexPath).row]
    let itemImageToMove = itemsImageFile[(fromIndexPath as NSIdexPath).row]
    items.remove(at: (fromIndexPath as NSIndexPath).row)
    itemsImageFile.remove(at: (fromIndexPath as NSIndexPath).row)
    items.insert(itemToMove, at: (to as NSIndexPath).row)
    itemsImageFile.insert(itemImageToMove, at: (to as NSIndexPath).row)
}
```

1) 이동할 아이템의 위치를 itemToMove 에 저장한다.
2) 이동할 아이템의 이미지를 itemImageToMove 에 지정한다.
3) 이동할 아이템을 삭제한다, 이때 삭제한 아이템 뒤의 아이템들의 인덱스가 재정렬된다.
4) 이동할 아이템의 이미지를 삭제한다. 이때 삭제한 아이템 이미지 뒤의 아이템 이미지들의 인덱스가 재 정렬된다.
5) 삭제된 아이템을 이동할 위치로 삽입한다. 또한 삽입한 아이템 뒤의 아이템들의 인덱스가 재정렬된다.
6) 삭제된 아이템의 이미지를 이동할 위치로 삽입한다. 또한 삽입한 아이템 이미지 뒤의 아이템 이미지들의 인덱스가 재 정렬된다.

3. 결과 보기

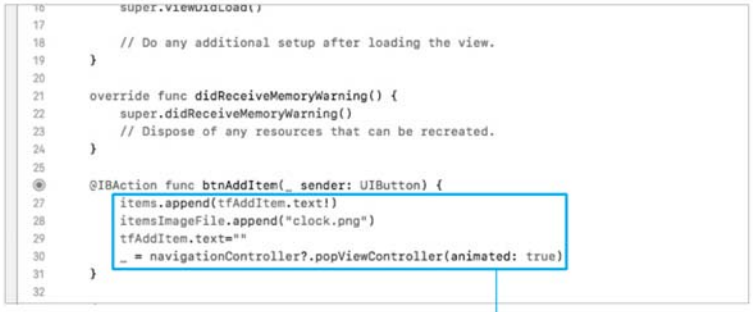
앱을 실행해서 [Edit]버튼을 클릭해 보자. 목록 오른쪽에 있던 버튼이 '순서 바꾸기'버튼으로 바꾸어 있는 것을 확인 할 수 있다. 옮기고 싶은 목록을 끌어다 놓고 [Done]버튼을 클릭해 저장한다.

12-9 새 목록 추가하기

이제 처음에 만든 'Add View'와 [+] 버튼을 활용하여 새 목록을 추가하는 기능을 구현해 보자

1. 새 목록을 추가하는 코드를 작성해 보자.
[AddView Controller.swift]를 선택한 후 btnAddItem(_sender:UIButton)함수를 수정한다.

2. 다음 소스를 btnAddItem 함수 안에 입력한다. 각 소스의 의미는 다음과 같다.



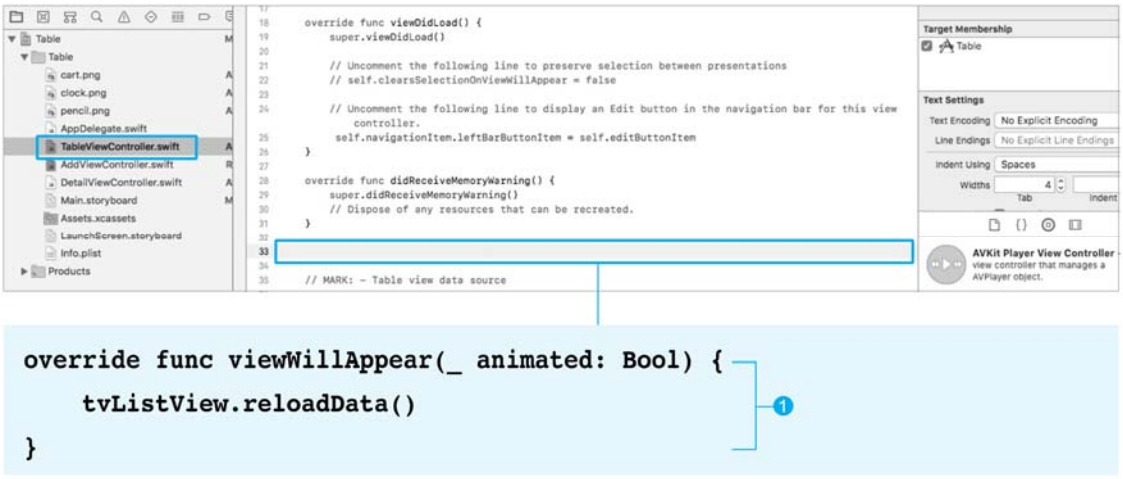

```
@IBAction func btnAddItem(_ sender: UIButton) {
    items.append(tfAddItem.text!) —1
    itemsImageFile.append("clock.png") —2
    tfAddItem.text="" —3
    _ = navigationController?.popViewController(animated: true) —4
}
```

1) items 에 텍스트 필드의 텍스트 값을 추가된다.
2) itemsImageFile 에는 무조건 'clock.png'파일을 추가한다.
3) 텍스트 필드의 내용을 지운다.
4) 루프 뷰 칸트롤러, 즉 테이블 뷰로 돌아간다.

3. "새 목록 추가"결과 보기
앱을 실행한다. 화면 오른쪽에 있는 [+]버튼을 클릭하면 'Add View'로 이동하고, 텍스트 필드에 내용을 입력한 후 [Add]버튼을 클릭하면 다시 'Main View'로 돌아오는 것까지는 잘 구현되어 있으나 내용이 추가되지 않은 것을 확인 할 수 있다.

4. 새 목록 추가 동작 코딩하기
3 번 과정의 결과 보기에서와 같이 내용이 추가되지 않았는데 이것이 제대로 작동되도록 수정해 보겠다. [TableViewController.swift]를 선택한 후 아래쪽 박스로 표시된 부분에 viewWillAppear(_animated: Bad)함수를 추가한다.

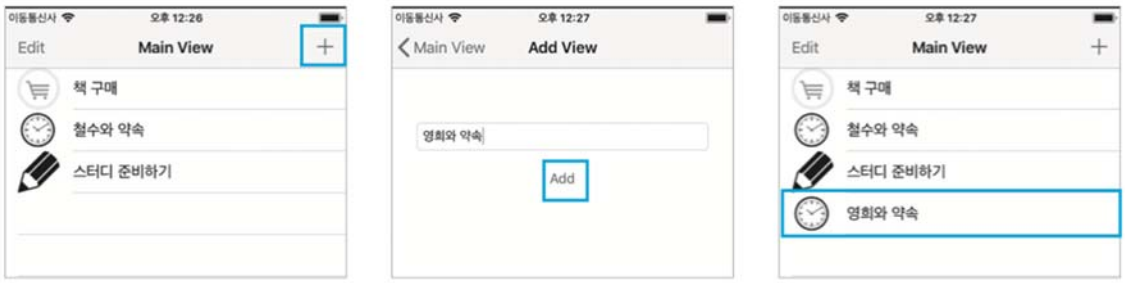
이 함수는 뷰가 전환될 때 호출되는 함수로, 리스트가 추가되어 'Main View'로 돌아올 때 호출되며 추가된 내용을 리스트에 보여준다.



1) tvListView.reloadData 함수를 추가하여 테이블 뷰를 다시 불러온다. 다시 말해 추가된 내용을 목록으로 불러온다.

5. '새 목록 추가 동작'결과 보기

앱을 실행한다. 화면 오른쪽에 있는 [+]버튼을 클릭하면 'Add View'로 이동하고, 텍스트 필드에 내용을 입력한 후 [Add] 버튼을 클릭하면 'Main View'로 돌아온다. 이제는 내용을 제대로 추가된 것을 확인 할 수 있다.



12-10 목록의세부내용보기

마지막으로 목록의 아이템을 선택하면 'Detail View'로 이동하고 그 내용을 보여 주도록 코드를 작성해 본다.

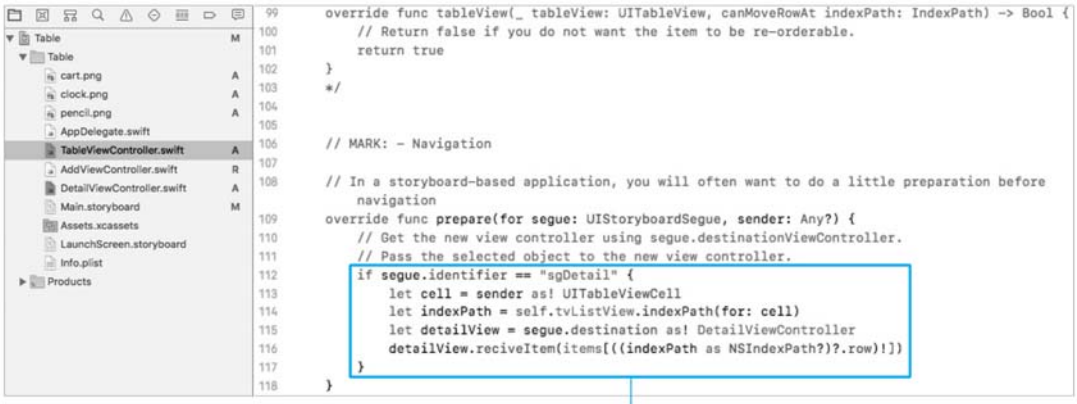
1. [DetailViewController.swift]를 선택한 후 다음 소스를 추가한다.

```
class DetailViewController: UIViewController {

    var receiveItem = "" ❶

    @IBOutlet var lblItem: UILabel!
    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
        lblItem.text = receiveItem ❷
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    func receiveItem(_ item: String) ❸
    {
        receiveItem = item
    }
```

1) Mian View 에서 받을 텍스트를 위해 receiveItem 를 선언한다.
2) 뷰가 노출될 때마다 이 내용을 레이블의 텍스트로 표시한다.
3) Main View 에서 변수를 받기 위한 함수를 추가한다.

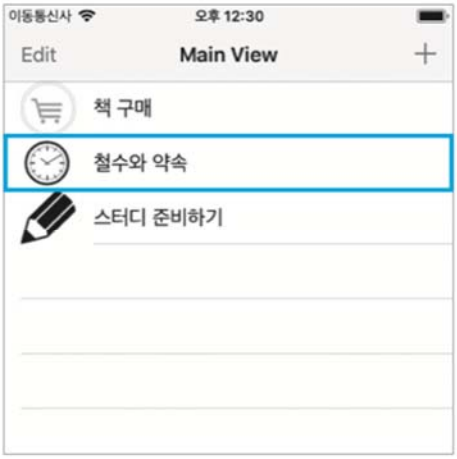
2. [TableViewController.swift]를 선택한 후 수정한다,
prepare(for segue: UIStoryboardSegue, sender: Any?)함수의 주삭을 제거 한다.
이 함수는 세그웨이를 이용하여 뷰를 이동하는 함수이다.

3. 세그웨이를 이용하여 뷰를 전환하는 것과 같은 방법을 사용한다.
다만 TableViewCell 의 indexPath 를 구하는 부분이 추가된다.




```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    // Get the new view controller using segue.destinationViewController.
    // Pass the selected object to the new view controller.
    if segue.identifier == "sgDetail" {
        let cell = sender as! UITableViewCell
        let indexPath = self.tvListView.indexPath(for: cell)
        let detailView = segue.destination as! DetailViewController
        detailView.receiveItem(items[((indexPath as NSIndexPath?)?.row)!])
    }
}
```

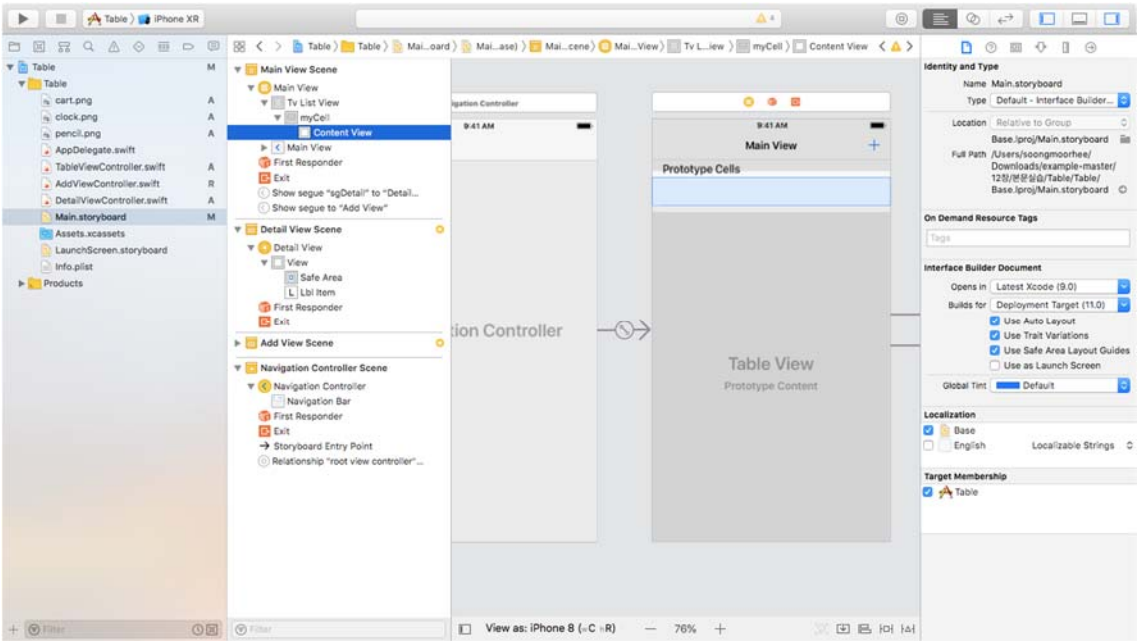
4. 결과 확인



알아두면 좋은 정보

1. Document Outline 이용해 객체 선택하기

'Document Outline'을 열어보면 [Table View Cell]이 선택된 것을 확인 할 수 있다. 이렇게 스토리보드에서 어떤 것을 선택했는지 알고 싶을 때는 [Document Outline]을 열어보면 확인할 수 있다. 또한 스토리보드에서 객체 선택이 쉽지 않을 경우에는 이 [Document Outline]에서 [myCell]을 선택하면 스토리보드에서 [Prototype Cells]를 쉽게 선택할 수 있다.



2. XIB 파일이란?

XIB 파일이란 인터페이스의 저장 포맷으로, Xcode3.0 버전부터 도입되었다. 지금은 XIB 파일을 스토리보드가 대체하고 있다. 이전 버전의 경우에는 뷰 컨트롤러 클래스 파일과 XIB 파일이 일대일로 존재했고, 스토리보드에서 버튼이나 텍스트 필드등을 추가하는 과정을 XIB 파일을 이용해 작업하고 저장했다. 하지만 Xcode4.2 버전부터 스토리보드 기능이 생겼고, 지금은 XIB 파일은 거의 사용하지 않고 있다. 따라서 'TableViewController'를 생성할 때는 XIB 파일을 생성할 필요가 없다,