Special 1

Xcode 어시스턴트 에디터를 사용하여 아웃렛과 액션 만들기

이 Special 6개 장은 iOS 5.1 업데이트 중 핵심 내용을 새로 추가한 것입니다.
《핵심만 골라 배우는 iOS 5 프로그래밍》 책과 함께 보시면 이해가 빠릅니다.
이 파일은 크리에이티브 커먼즈 저작자표시-비영리-동일조건변경허락 2.0
대한민국 라이선스에 따라 이용할 수 있습니다.

책의 앞부분 장들에서는 아웃렛과 액션에 대한 개념에 대해 알아보고 이를 이용하여 간단한 예제 애플리케이션을 만들어보았다. 11장 "**상호작용하는 iOS 아이폰 앱 만들기**" 장에서는 아웃렛과 액션을 직접 애플리케이션 코드에 추가하였다. 그렇지만 Xcode에서는 이렇게 수작업으로 아웃렛과 액션을 추가하는 방법 외에 어시스턴트 에디터(Assisipant Editor)라는 기능을 이용하여 아웃렛과 액션을 추가하는 방법을 지원한다.

이번 장에서는 Xcode의 어시스턴트 에디터를 사용하여 아웃렛과 액션을 애플리케이션 프로젝트에 적용하는 방법에 대해 알아보자.

이번 장 이후에서는 예제에서 필요한 아웃렛과 액션의 목록만 나열하도록 하겠다. 이들 아웃렛과 액션을 수작업으로 추가하거나 어시스턴트 에디터를 사용하여 추가하는 것은 독자들의 선택에 맡기겠다. 이러한 방식을 사용하는 이유는 다음과 같다.

먼저 어떤 개발자들은 아직도 직접 코딩을 통해 아웃렛과 액션을 구현하는 것을 선호한다. 또한 어시스턴트 에디터를 사용하여 아웃렛과 액션을 선언하는 방법은 애플리케이션을 개발할 때 유저 인터페이스를 가장 먼저 개발한다는 전제를 가지고 있다. 그렇지만 실제 개발에 있어서는 유저 인터페이스보다 애플리케이션 로직을 먼저 개발하는 경우도 흔하다. 또한 많은 소프트웨어 개발팀들은 유저 인터페이스는 사용자 인터페이스 전문가 혹은 그래픽 디자이너가 담당하고, 이와 동시에 개발자는 애플리케이션 코드를 작성

하는 경우가 일반적이다. 그렇지만 보통의 경우 어시스턴트 에디터를 사용하면 개발자가 아웃렛과 액션을 구현하는 데 필요한 노력을 많이 줄일 수 있다.

S1.1 어시스턴트 에디터 표시하기

기본적으로 Xcode가 실행될 때 어시스턴트 에디터가 화면에 표시되지 않는다. 따라서 View ➡ Assistant Editor ➡ Show Assistant Editor 메뉴 옵션을 사용하여 어시스턴트 에디터를 화면에 표시한다. 또한 다음 그림과 같이 Xcode의 오른편 상단의 에디터 툴바 버튼의 가운데 버튼을 선택하면 어시스턴트 에디터가 화면에 표시된다.

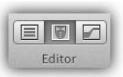

그림 S1-1

만약 여러 개의 어시스턴트 에디터 화면을 표시하는 것이 필요하다면 View ➡ Assistant Editor ➡ Add Assistant Editor 메뉴 옵션을 사용하자.

기본적으로 에디터 화면은 주 에디터 화면의 오른편에 표시될 것이다. 다음의 그림 S1-2는 어시스턴트 에디터 화면이 인터페이스 빌더 화면의 오른편에 표시된 것을 나타낸다.

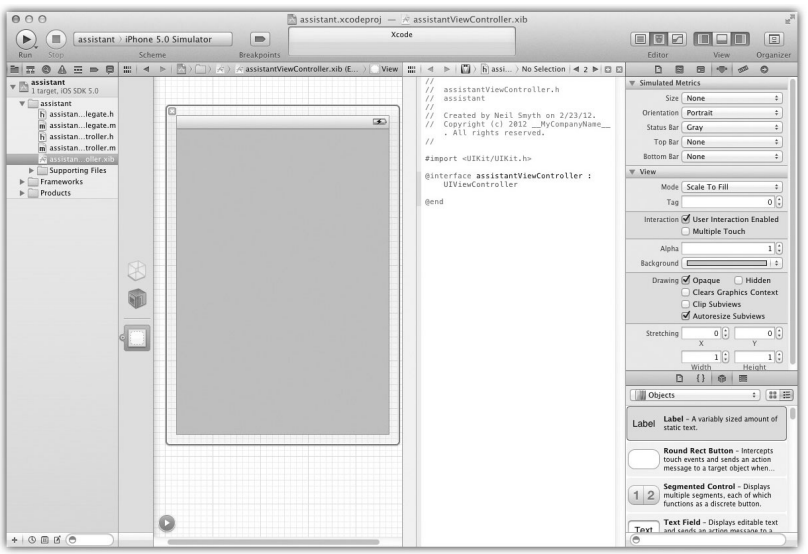

그림 S1-2

S1.2 어시스턴트 에디터 사용하기

이번 장에서는 어시스턴트 에디터를 사용하여 간단하게 아웃렛과 액션을 선언하는 예제를 만들어 볼 것이다. 이를 위해 Xcode에서 Assistant라는 이름의 Single View Application 템플릿을 이용하는 아이폰 프로젝트를 만든다. 프로젝트가 생성되면 AssistantViewController.xib 파일을 선택한다. 인터페이스 빌더 화면이 표시되면 뷰 캔버스로 버튼과 라벨을 드래그 앤 드롭한다.

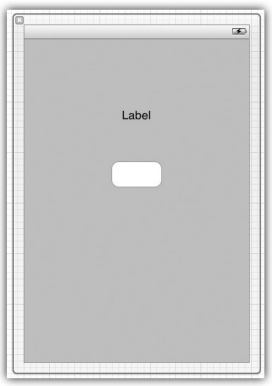

그림 S1-3

애플리케이션은 버튼을 눌렀을 때 메시지를 라벨에 표시한다. 이를 위해 버튼이 호출할 액션 메서드와 라벨의 문자열 속성의 접근을 위한 아웃렛의 구현이 필요하다. 11장 "**상호작용하는 iOS 아이폰 앱 만들기**"에서 배운 내용을 기반으로 직접 뷰 컨트롤러에 아웃렛과 액션의 선언과 코드 구현을 할 수 있을 것이다. 그렇지만 이번에는 어시스턴트 에디터를 사용하여 이러한 작업을 수행해보자.

S1.3 어시스턴트 에디터를 사용하여 아웃렛 추가하기

인터페이스 빌더 화면이 표시되고 있는 상태라면 앞쪽의 그림에서 본 것처럼 어시스턴트 에디터 화면을 표시해보자. 라벨 오브젝트를 위한 아웃렛을 만들기 위해 그림 S1-4와 같이 라벨을 Ctrl-클릭하여 어시스턴트 에디터 화면의 @interface 명령어 바로 아래쪽으로 드래그한다.

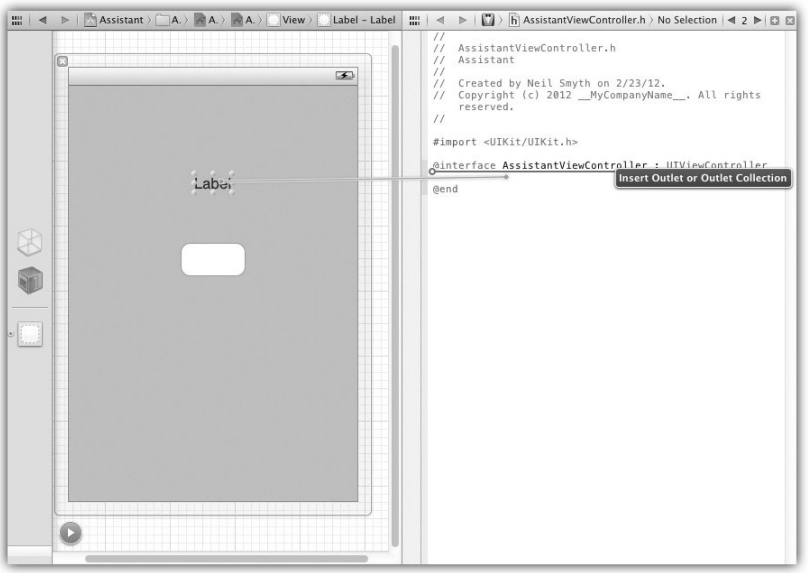

그림 S1-4

마우스 버튼에서 손을 떼면 그림 S1-5와 같은 구성화면이 표시되어 선언할 아웃렛에 대한 세부항목을 설정할 수 있다.

그림 S1-5

아웃렛을 설정하는 것이므로 Connection 메뉴는 당연히 Outlet이 선택되어 있다. Type과 Storage 값 역시 아웃렛 설정에 맞게 선택되어 있다. 따라서 아웃렛의 이름만 설정해 주면 된다. Name 필드에 statusLabel을 입력하고 Connect 버튼을 클릭한다.

연결이 설정되면 AssistantViewController.h 파일을 선택하고 어시스턴트 에디터에 의

해 선언된 아웃렛 속성이 추가된 것을 확인한다.

```
#import <UIKit/UIKit.h>

@interface AssistantViewController : UIViewController
@property (strong, nonatomic) IBOutlet UILabel *statusLabel;

@end
```

AssistantViewController.m 파일에는 어시스턴트 에디터에 의해 해당하는 아웃렛에 대한 synthesize 명령어가 추가되었음을 확인할 수 있다.

```
#import "AssistantViewController.h"

@implementation AssistantViewController
@synthesize statusLabel;
.
.
@end
```

아주 간단하게 아웃렛을 선언이 끝났다. 이제 액션에 대해 알아보자.

S1.4 어시스턴트 에디터를 사용하여 액션 추가하기

어시스턴트 에디터를 이용하여 액션을 선언하는 방법은 아웃렛의 경우와 동일하다. 다시 한 번 AssistantViewController.xib 파일을 선택하고 버튼을 Ctrl-클릭한다. 어시스턴트 에디터 화면의 @interface 명령어 아래쪽으로 드래그한 후 릴리즈한다. 세부항목을 설정하기 위한 화면이 표시될 것이다. 이번에는 액션을 만들 것이므로 Connection 메뉴가 Action으로 되어 있는지 확인한다. 액션의 이름을 buttonTouched라고 입력하고 Event가 Touch Up Inside로 설정되었는지 확인한다. 액션 메서드에서 버튼 오브젝트에 접근할 필요가 없으므로 Arguments 메뉴는 None으로 설정한다.

그림 S1-6

Connect 버튼을 클릭하여 액션을 만든다.

AssistantViewController.h 파일을 선택하고 어시스턴트 에디터에 의해 액션이 선언되었음을 확인한다.

```
#import <UIKit/UIKit.h>

@interface AssistantViewController : UIViewController
- (IBAction)buttonTouched;
@property (strong, nonatomic) IBOutlet UILabel *statusLabel;

@end
```

또한 AssistantViewController.m 파일을 선택하고 액션 메서드의 템플릿이 추가되어 있음을 확인하자.

```
- (IBAction)buttonTouched {
}
```

액션 메서드에 statusLabel 아웃렛을 사용하여 라벨에 문자열을 표시하는 코드를 추가한다.

```
- (IBAction)buttonTouched {
    self.statusLabel.text = @"Hello";
}
```

애플리케이션을 컴파일하고 실행해보자. 버튼을 터치하면 라벨에 "Hello"라는 문자열이 표시될 것이다.

S1.5 요약

Xcode의 어시스턴트 에디터를 사용하면 아웃렛과 액션을 쉽게 구현할 수 있다. 이번 장에서는 어시스턴트 에디터를 사용하여 아웃렛과 액션의 연결을 설정하는 것에 대해 알아보았다. 아웃렛과 액션을 구현하기 위해 이 장에서 배운 것처럼 어시스턴트 에디터를 사용할 수도 있고 기존의 방식대로 직접 코딩을 할 수도 있다. 두 가지 방법이 있으므로 독자의 취향에 따라 방법을 선택하면 된다.

Special 2

Xcode 스토리보드로 iOS 5 아이폰 탭 바 애플리케이션 만들기

18장에서 간단한 스토리보드 기반의 애플리케이션을 만들어보았다. 이제 약간 복잡한 스토리보드 예제를 만들어보자.

여기까지는 사용자에게 하나의 뷰만 보여주는 애플리케이션을 다루었다. 실제로는 사용자의 요청에 맞춰 다양한 내용을 보여주는 애플리케이션이 필요하다. 이를 위해서는 여러 개의 뷰(콘텐트 뷰(content view)라고도 함)와 하나의 뷰로부터 다른 뷰로 옮겨갈 수 있는 메커니즘(mechanism)이 필요하다. 이러한 메커니즘 중 하나가 UINavigationBar 혹은 UITabBar 컴포넌트다. 이번 장에서는 탭 바를 이용하여 다중 뷰 애플리케이션을 구현하는 방법에 대해 알아보자.

S2.1 탭 바 개요

UITabBar 컴포넌트는 일반적으로 화면의 아랫부분에 위치하며, 여러 가지 다른 내용의 뷰를 사용자들이 선택할 수 있도록 문자열이나 부가적인 아이콘들을 표시한다. 탭 바 애플리케이션의 예로는 아이폰의 음악이나 전화 프로그램이 있다. 음악 애플리케이션의 경우 재생목록, 아티스트, 노래와 앨범이라는 탭 바를 가지고 있다. 사용자의 선택에 따라 다른 뷰가 화면에 표시된다.

S2.2 멀티뷰 애플리케이션의 뷰 컨트롤러 이해하기

이전 장들에서 모델-뷰-컨트롤러 개념에 대해 알아보았으며, 각각의 뷰는 해당하는 뷰 컨트롤러를 가지고 있다는 것을 배웠다(10장 "**아이폰 iOS 5 애플리케이션 개발 아키텍처 개요**"를 참고하자). 멀티 뷰 애플리케이션에서 각각의 콘텐트 뷰는 사용자와의 상호작용 및 화면 표시를 위해 역시 각각의 뷰 컨트롤러를 갖는다. 그렇지만 멀티 뷰 애플리케이션은 이외에도 추가적인 컨트롤러가 더 필요하다.

멀티뷰 애플리케이션은 화면 컨트롤이 필요하다. 이는 사용자들이 하나의 뷰에서 다른 뷰로 이동할 때 사용되며, 흔히 탭 또는 내비게이션 바의 형태를 갖는다. 이들 컴포넌트들은 뷰(view)이기도 하며, 또한 뷰 컨트롤러(view controller)가 필요하다. 멀티뷰 애플리케이션에서 이들은 root controller라고 불리며, 사용자에게 어떤 뷰가 표시되고 있는지를 제어하는 역할을 맡고 있다. 개발자 입장에서는 UIViewController 클래스를 서브클래싱함으로써 새로운 root controller를 만들 수도 있지만, UIKit의 UITabBarController나 UINavigationController 클래스를 사용하는 것이 일반적이다.

어떻게 구현을 하든 root controller는 애플리케이션이 실행될 때 로드되는 최초의 컨트롤러다. 로드된 후에는 사용자에게 보여지는 첫 번째 뷰를 표시해야 하며, 이후 사용자의 요청에 의해 적절한 뷰를 화면에 표시하거나 사라지게 해야 한다.

이번 장은 탭 바 기반의 애플리케이션에 대해 다루기 때문에 root controller로 UITabBarController의 인스턴스를 사용하기로 하자.

S2.3 탭 바 예제 애플리케이션 설정하기

예제를 만들기 위해 처음 해야 할 일은 새로운 Xcode 프로젝트를 만드는 것이다. 이를 위해 Xcode를 실행하고 Create a new Xcode project 옵션을 선택한다.

Xcode에 의해 제공되는 여러 가지 템플릿 중에 Tabbed Application 템플릿이 있다. 이 템플릿을 선택하면 미리 설정되어 있는 두 개의 콘텐트 뷰를 갖는 탭 바 애플리케이션을 만들어준다. 이번 장의 예제를 위해 이 템플릿을 사용할 수 있지만, 이 템플릿을 사용하면 너무 쉽게 탭 바 애플리케이션을 만들어주기 때문에 스토리보드를 사용하여 복잡한

애플리케이션을 개발하는 데 필요한 기본적인 기술들을 익힐 수 없다는 단점이 있다. 그러므로 탭 바 애플리케이션을 위한 템플릿이 있다는 정도만 알아두고 이번 장의 예제는 Single View Application 템플릿을 이용하여 만들어보자.

Single View Application 템플릿을 선택하고 Next를 클릭하여 다음으로 진행하자. 다음 화면에서 product name과 클래스 접두사로 TabBar를 입력하고, Device Family가 iPhone으로 되어 있는지 확인하고 Use Storyboard와 Use Automatic Reference Counting 옵션을 사용함으로 설정한다. 마지막 화면까지 진행하고 프로젝트 파일이 저장될 적절한 위치를 선택한 후 Create를 클릭한다.

S2.4 프로젝트 파일 리뷰하기

프로젝트를 만드는 과정을 통해 여러 가지의 옵션을 선택하게 되고, 이에 기반하여 Xcode는 여러 가지 파일을 생성한다. 단일 뷰 컨트롤러 기반 애플리케이션에 필요한 파일인 TabBarViewController.m과 TabBarViewController.h 파일이 생성되었을 것이다. Use Storyboard를 선택하였으므로 MainStoryboard.storyboard 파일 역시 생성되었다.

S2.5 시작 뷰 컨트롤러 이름 바꾸기

다음 과정은 Xcode에 의해 만들어진 TabBarViewController 클래스를 적절한 이름으로 변경한다. 이 뷰 컨트롤러는 사용자가 탭 바의 첫 번째 탭을 터치했을 때 화면에 표시되는 뷰를 표시하므로 이 클래스의 이름을 Tab1ViewController로 변경하자. 이를 수행하기 위해 TabBarViewController.h 파일을 선택하고 편집 화면에서 마우스를 사용하여 TabBarViewController 클래스 이름을 하이라이트한다. 클래스 이름이 하이라이트된 상태에서 Edit ➞ Refactor ➞ Rename… 메뉴 옵션을 선택한다.

입력 화면에서 Tab1ViewController를 입력하고 Preview를 클릭한다. 미리보기 화면에서 Save를 선택하고 스냅샷 옵션에서 Disable을 선택한다.

S2.6 두 번째 콘텐트 뷰를 위한 뷰 컨트롤러 추가하기

이번 장에서 만들 예제는 각각의 뷰를 가지고 있는 두 개의 탭이 있는 탭 바 기반 애플리케이션이다. Xcode가 이미 첫 번째 뷰 컨트롤러를 만들었으며, 이의 이름을 변경하였다. 그러므로 다음 과정은 두 번째 뷰를 위한 뷰 컨트롤러를 추가하는 것이다. 이를 위해 File ➞ New ➞ File... 메뉴를 선택하고 입력 화면에서 Objective-C class를 선택한다. Next를 클릭하고 새로운 클래스의 이름을 Tab2ViewController로 하고, Subclass 메뉴에서 UIViewController를 선택한다. Next를 클릭하여 생성 과정을 진행한다.

S2.7 스토리보드에 탭 바 컨트롤러 추가하기

앞에서 설명했듯이 탭 바 기반의 인터페이스에서 뷰 컨트롤러들 간의 전환은 탭 바 컨트롤러가 담당한다. 그러므로 스토리보드에 탭 바 컨트롤러를 추가하는 것이 필요하다. Xcode 프로젝트 내비게이터 화면에서 MainStoryBoard.storyboard 파일을 선택하고, 그림 S2-1과 같이 Xcode에 의해 만들어진 오직 한 개의 뷰 컨트롤러만 갖고 있음을 확인한다.

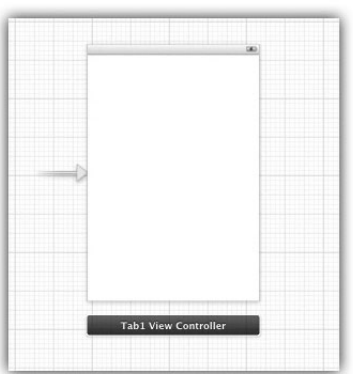

그림 S2-1

탭 바 컨트롤러를 스토리보드에 추가하기 위해 스토리보드 디자인 영역에서 Tab1View Controller를 선택하고, Editor ➞ Embed In ➞ Tab Bar Controller 메뉴를 선택한다. 그림 S2-2와 같이 탭 바 컨트롤러가 스토리보드에 표시될 것이다.

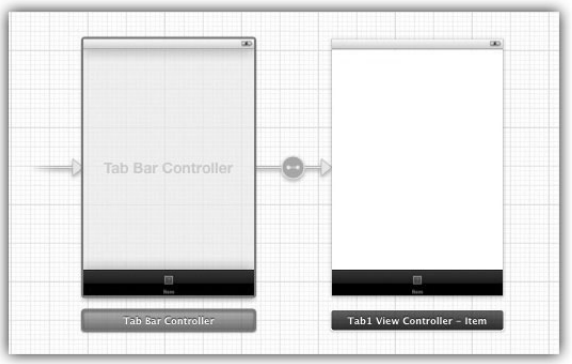

그림 S2-2

탭 바 컨트롤러를 추가하였으므로 이제 앞에서 만든 Tab2ViewController 클래스에 해당하는 뷰 컨트롤러를 스토리보드에 추가해보자.

S2.8 스토리보드에 두 번째 뷰 컨트롤러 추가하기

두번째 뷰 컨트롤러를 스토리보드에 추가하기 위해 오브젝트 라이브러리에서(View ➞ Utilities ➞ Show Object Library) 뷰 컨트롤러를 스토리보드로 드래그 앤 드롭한다. 스토리보드 화면에 새로운 뷰 컨트롤러가 추가되면 이를 선택하고 Identity Inspector 화면을 표시한다(View ➞ Utilities ➞ Show Identity Inspector). 인스펙터 화면에서 클래스 설정을 UIViewController에서 Tab2ViewController로 변경한다.

이제 Tab2ViewController에 해당하는 스토리보드의 뷰 컨트롤러와 탭 바 컨트롤러와의 관계를 설정해야 한다. 이를 위해 스토리보드 화면에서 탭 바 컨트롤러 오브젝트를 Ctrl-클릭한 후 Tab2ViewController로 드래그한다. 마우스를 릴리즈하면서 그림 S2-3과 같이 Relationship-viewControllers 메뉴를 선택한다. 이는 Tab2ViewController를 탭 바 컨트롤러 오브젝트의 viewControllers 속성에 추가함으로써 탭 내비게이션에 포함시킨다.

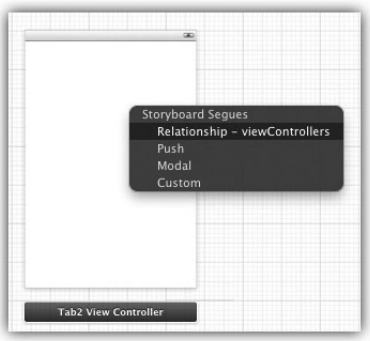

그림 S2-3

이제 스토리보드의 디자인 화면은 하나의 탭 바 컨트롤러와 이에 연결된 Tab1View Controller 및 Tab2ViewController로 구성되어 있다. 스토리보드 디자인 화면은 다음의 그림과 같이 표시될 것이다.

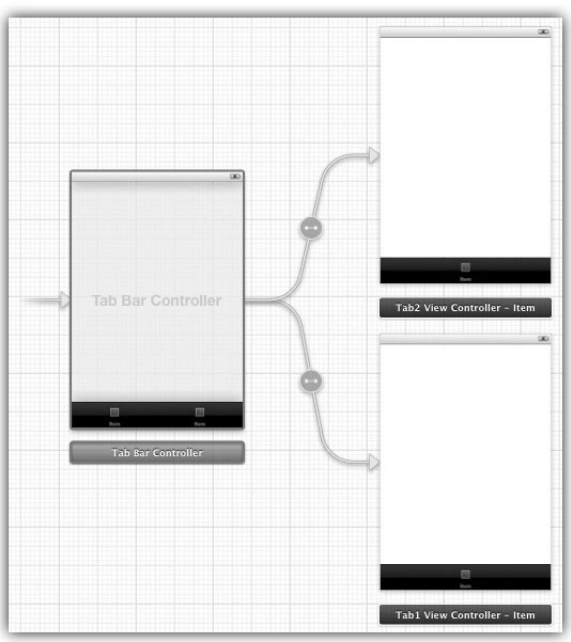

그림 S2-4

이제 애플리케이션을 완성하기 위해 탭 바 아이템을 설정하고 두 개의 뷰 컨트롤러의 유저 인터페이스를 디자인하자.

S2.9 뷰 컨트롤러 유저 인터페이스 디자인하기

두 개의 뷰 컨트롤러를 다르게 보이도록 뷰에 라벨을 추가하고 바탕색을 바꿀 것이다. 화면을 축소한 상태라면 스토리보드 화면의 아래 편 오른쪽의 컨트롤을 이용하여 적당한 크기로 확대하자. Tab1ViewController 오브젝트의 뷰를 선택한다. Attribute Inspector 화면(View ➞ Utilities ➞ Show Attribute Inspector)에서 Background 라벨 옆의 흰색 사각형을 클릭하고 Colors 화면에서 붉은 색을 선택한다. 오브젝트 라이브러리에서 라벨 오브젝트를 선택하고 붉은 색 뷰의 가운데로 드래그 앤 드롭한다. 라벨을 더블 클릭하여 Screen One이라는 문자열을 입력한다.

작업이 완료되면 Tab1ViewController 스토리보드 화면은 그림 S2-5와 같이 표시될 것이다.

그림 S2-5

위의 과정을 반복하여 Tab2ViewController 뷰의 바탕색을 연두색으로 바꾸고 Screen Two라는 라벨을 추가한다.

S2.10 탭 바 아이템 구성하기

두 개의 뷰 컨트롤러의 아래쪽에 위치한 탭 바는 현재 "Item"이라는 문자열을 표시하고 있다. 또한 물음표를 포함한 작은 아이콘은 현재 할당된 이미지가 없다는 것을 의미한다. 이제 애플리케이션을 완성하기 위해 이 문제들을 해결하자. 먼저 Tab1ViewController의 탭 바의 "Item"을 더블 클릭하여 이를 Screen One으로 변경한다. 이 과정을 반복하여 Tab2ViewController의 탭 바 아이템의 문자열을 Screen Two로 변경한다.

만약 탭 바 아이템에 사용할 수 있는 아이콘 파일을 이미 가지고 있으면 이것을 사용해도 좋다. 혹시 없다면 다음의 URL에서 예제 아이콘 파일을 다운로드받아 사용하자.

URL http://www.ebookfrenzy.com/code/tabbaricons.zip

아이콘 파일에는 first.png와 second.png라는 이름의 두 개의 PNG 형식의 아이콘 이미지가 있다. 아이콘 파일을 파인더 윈도우에서 선택하고 Xcode 프로젝트 내비게이터 화면의 Supporting Files로 드래그 앤 드롭한다. Tab1ViewController의 탭 바의 아이콘을 클릭하고, Attribute Inspector 화면에서 Image의 드롭다운 메뉴를 클릭하여 first.png를 이미지 파일로 선택한다.

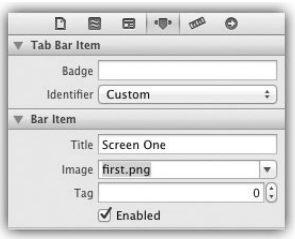

그림 S2-6

동일한 과정을 반복하여 Tab2ViewController의 이미지 파일을 second.png로 설정한다.

S2.11 애플리케이션 빌드 및 실행하기

이제 예제 애플리케이션이 완성되었다. 컴파일하고 실행해보자. Xcode 툴바의 Run 버튼을 클릭하여 애플리케이션이 컴파일되고 iOS 시뮬레이터에서 실행되기를 기다리자. 애플리케이션이 실행되면 Tab1ViewController가 활성화된 상태에서 화면 아래쪽에 두 개의 탭 아이템이 있는 탭 바가 표시될 것이다. Screen Two 탭을 선택하면 Tab2View Controller의 뷰로 화면이 전환될 것이다.

그림 S2-7

S2.12 요약

Xcode의 스토리보드를 이용하면 애플리케이션에 손쉽게 탭 바 기반의 내비게이션 기능을 추가할 수 있다. 아마 이번 장에서 만든 예제 프로젝트에서 가장 놀라운 점은 단 한 줄의 오브젝티브-C 코드도 작성하지 않았다는 점일 것이다.

Special 3

iOS 5 테이블 뷰와 Xcode 스토리보드 개요

아이폰의 iOS를 많이 다루어보았다면 UIKit의 테이블 뷰 오브젝트에 대해 익숙할 것이다. 테이블 뷰는 많은 iOS 5 아이폰 애플리케이션에서 사용되는 내비게이션 시스템이 기반이 된다. 예를 들면, 아이폰의 메일과 설정 애플리케이션은 테이블 뷰를 많이 사용하고 있는데, 이를 통해 사용자에게 목록 형태의 정보를 제공하고 특정 리스트 아이템을 선택하였을 때 세부사항이 표시되도록 하고 있다.

테이블 뷰는 iOS의 유저 인터페이스 개발에서 복잡한 것들 중 하나다. 이러한 점을 인지한 애플은 Xcode의 스토리보드 기능을 이용하여 테이블 뷰를 구현하는 새로운 방식을 소개하였다.

이번 장에서는 UITableView 클래스의 개요를 소개하고 스토리보드를 이용하여 테이블 뷰를 구현하는 방법에 대해 알아볼 것이다. 앞으로 몇 개의 장을 통해서 테이블 뷰와 관련한 스토리보드의 사용법을 알아보기 위해 예제 프로젝트를 만들어볼 것이다.

S3.1 테이블 뷰 개요

테이블 뷰는 사용자에게 목록의 형식으로 데이터를 표시하며, UIKit 프레임워크의 UITableView 클래스를 사용한다. 데이터는 행(row)에 표시되며, 각 행의 내용은

UITableViewCell 오브젝트의 형태로 구현된다. 각 테이블 셀은 텍스트 라벨(textLabel), 서브타이틀(detailedTextLabel)과 이미지(imageView)로 표현될 수 있다. 보다 복잡한 형태의 셀은 셀에 서브뷰를 추가하거나 UITableViewCell을 서브클래싱하여 여러분의 독자적인 기능을 구현하여 만들 수 있다.

S3.2 정적 혹은 동적 테이블 뷰

Xcode의 스토리보드를 사용하여 테이블 뷰를 구현할 때 정적(static) 혹은 동적(dynamic) 테이블의 차이에 대해 이해하는 것이 중요하다. 정적 테이블은 고정된 숫자의 행이 테이블에 표시될 때 유용하다. 예를 들면, 애플리케이션의 설정 페이지는 미리 정의된 선택 항목으로 구성되므로 정적 테이블이라고 볼 수 있다.

동적 테이블(혹은 prototype-based 테이블)은 가변적인 숫자의 행을 테이블에 표시한다. 스토리보드 에디터에서 프로토타입 테이블 셀을 디자인한 후 이를 실행 중에 복제하여 동적 테이블 뷰에 사용할 수 있다.

S3.3 테이블 뷰 델리게이트와 데이터소스

애플리케이션의 각 테이블 뷰는 델리게이트와 연관된 데이터소스(dataSource)를 가지고 있다. 데이터소스는 타이틀 정보를 정의하고, 몇 개의 데이터를 표시할지 결정하고, 몇 개의 영역으로 나누어지며, 표시될 셀 오브젝트의 테이블 뷰를 제공하는 기능을 하는 기본적인 메서드들을 정의하는 UITableViewDataSource 프로토콜을 구현해야 한다. 델리게이트는 UITableViewDelegate 프로토콜을 구현해야 하며, 표시되는 모습에 대한 제어, 사용자의 터치에 대한 인식, 행의 높이, 들여쓰기, 행 삭제 및 수정 기능 등을 제공한다.

S3.4 테이블 뷰 스타일

테이블 뷰는 plain과 grouped 스타일로 설정할 수 있다. grouped 스타일에서는 행들이 둥근 모서리의 사각형으로 표시되는 영역으로 묶여서 표현된다. 그림 S3-1은 grouped 스타일의 테이블 뷰의 모습이다.

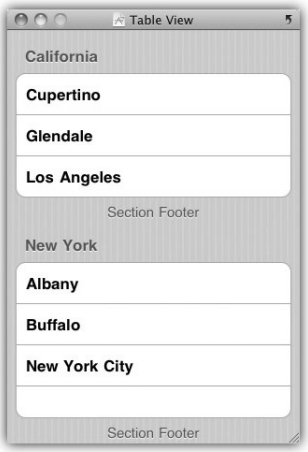

그림 S3-1

plain 스타일에서는 영역 구분 없이 표시되며, 화면의 폭을 최대한 사용한다.

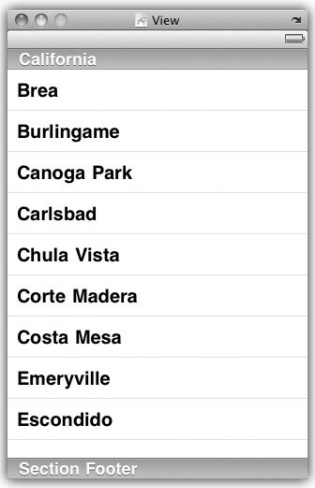

그림 S3-2

plain 스타일의 테이블 뷰는 인덱스되어질 수 있으므로 문자 혹은 숫자 정렬 같은 특별한 기준에 의해 그룹으로 구성될 수 있다.

S3.5 테이블 뷰 셀 스타일

테이블 뷰의 스타일 외에 각각의 테이블 셀에 대한 스타일을 정의할 수 있다. iOS 5 SDK에서는 현재 다음과 같은 4가지의 셀 스타일을 지원한다.

- UITableViewCellStyleDefault – 왼쪽으로 정렬된 검정색의 labelText만 표시됨.
- UITableViewCellStyleSubtitle – 왼쪽으로 정렬된 검정색의 labelText, 아래쪽에 회색의 작은 폰트로 detailLabelText가 표시됨.
- UITableViewCellStyleValue1 – 왼쪽으로 정렬된 검정색의 labelText, 파란색의 작은 폰트의 detailLabelText가 오른쪽으로 정렬되어 같은 줄에 표시됨.
- UITableViewCellStyleValue2 – 왼편에 오른쪽으로 정렬된 파란색의 labelText, 오른편에 왼쪽으로 정렬된 검은색의 detailedLabelText가 표시됨.

S3.6 요약

테이블 뷰가 애플리케이션에서 데이터를 표시하고 뷰 내비게이션을 제공하는 일반적인 메커니즘을 제공하지만 구현하기 복잡한 면이 있었다. 그렇지만 이제 Xcode의 스토리보드가 소개됨으로써 테이블 뷰의 많은 기능을 시각적인 디자인을 통해 구현하고 코딩은 최소화할 수 있게 되었다. 표시될 데이터의 성격에 따라 정적 혹은 동적 테이블 뷰를 구현하자.

Special 4

Xcode 스토리보드로 프로토타입 테이블 뷰 셀을 가진 동적 테이블 뷰 만들기

Xcode 스토리보드의 강력한 기능 중 한 가지는 프로토타입 테이블 셀을 이용하여 테이블 뷰를 구현하는 것이다. 이를 통해 개발자는 라벨, 이미지 등 테이블 셀에 표시될 사용자 인터페이스 요소들을 눈으로 확인하며 디자인할 수 있으며, 애플리케이션 실행 시 필요에 의해 이를 복제하여 테이블 뷰에 사용할 수 있다. 스토리보드가 소개되기 전에는 많은 시행착오를 거치며 코딩을 해야 이를 구현할 수 있었다.

이번 장에서는 예제를 통해 프로토타입 셀을 사용하여 동적 테이블 뷰를 구성하는 방법에 대해 알아보자. 다음 장에서는 테이블 뷰의 내비게이션과 스토리보드를 이용하여 화면 간에 데이터를 전달하는 법을 구현해보겠다.

S4.1 예제 프로젝트 생성하기

Xcode를 시작하고 single view 애플리케이션을 생성한다. 프로젝트와 클래스 접두사를 TableViewStory로 설정한다.

프로젝트 내비게이터 화면을 통해 생성된 파일을 확인해보자. Xcode가 TableView StoryViewController라는 이름의 뷰 컨트롤러 서브클래스를 만들었다. 또한 Main Storyboard.storyboard 파일을 선택하여 이 뷰 컨트롤러가 스토리보드에도 존재함을 확인하자.

스토리보드 기반의 테이블 뷰 애플리케이션 개발에 대해 완벽하게 이해하기 위해 Xcode에 의해 만들어진 뷰 컨트롤러를 제거하고 시작하자. 스토리보드 화면에서 Table View Story View Controller 항목을 선택하여 파란색으로 표시되었음을 확인하고 키보드의 Delete 키를 누른다. 다음은 프로젝트 내비게이터 화면에서 TableViewStoryView Controller.m과 TableViewStoryViewController.h 파일을 선택하고 삭제한다. 결과화면에서 파일들을 삭제하는 옵션을 선택한다.

이제 기본적인 앱 델리게이트 코드 파일과 하나의 스토리보드 파일을 가진 프로젝트가 되었다. 스토리보드를 이용하여 UITableView와 UITableViewCell 클래스 기반의 아이폰 애플리케이션을 만들어보자.

S4.2 스토리보드에 TableView 컨트롤러 추가하기

사용자가 애플리케이션을 실행하면 차 목록을 가진 테이블 뷰가 표시될 것이다. 각 테이블 뷰 셀은 차 제조사, 모델과 해당하는 이미지로 구성된다. 이를 위해 Table View Controller 인스턴스를 스토리보드 파일에 추가해야 한다. MainStoryboard.storyboard 파일을 선택하고 Xcode 윈도우의 중앙에 스토리보드 화면이 표시되도록 한다. 오브젝트 라이브러리 화면에서(View ➞ Utilities ➞ Show Object Library) Table View Controller 오브젝트를 그림 S4–1과 같이 스토리보드 화면에 드래그 앤 드롭한다.

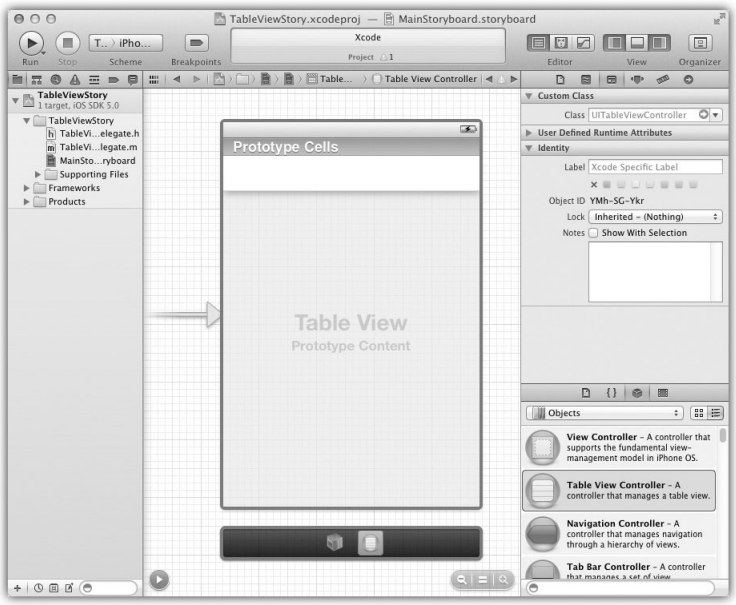

그림 S4-1

이제 스토리보드는 테이블 뷰 컨트롤러의 인스턴스를 가지게 되었다. 이 인스턴스에는 테이블에 표시될 셀을 구성할 수 있는 프로토타입 테이블 뷰 셀이 있다. 현재는 기본적인 UITableViewCell과 UITableViewController 클래스로 구성되어 있다. 추가적인 기능의 제공을 위해 UITableViewCell과 UITableViewController의 서브클래스로 선언할 필요가 있다. 이를 위해 우선 이들 서브클래스 자체를 만들어야 한다.

S4.3 UITableViewController와 UITableViewCell 서브클래스 만들기

스토리보드의 테이블 뷰 컨트롤러의 인스턴스를 CarTableViewController라는 이름의 UITableViewController의 서브클래스로 선언하려고 한다. 아직 프로젝트에 이 서브클래스가 존재하지 않으므로 일단 이를 만들어야 한다. 이를 위해 File ➡ New ➡ File... 메뉴 옵션을 선택하고 새로운 Objective-C class를 선택한다. Next를 클릭하고 클래스의 이름을 CarTableViewController, UITableViewController의 서브클래스로 설정한

다. Targeted for iPad와 With XIB for user interface 옵션이 선택되지 않음을 확인하고 Next, Create를 클릭한다.

스토리보드에 추가된 테이블 뷰 컨트롤러에는 Xcode에 의해 프로토타입 테이블 셀이 추가되어 있다. 이번 장의 뒷부분에서 두 개의 라벨과 이미지 뷰 오브젝트를 이 셀에 추가할 것이다. 클래스의 기능을 확장하기 위해 다시 한 번 서브클래스를 생성한다. File ➞ New ➞ File... 메뉴 옵션을 선택하고 Objective-C class를 선택한 후 Next를 클릭한다. 계속되는 화면에서 새로운 클래스의 이름을 CarTableViewCell, UITableViewCell의 서브클래스로 설정하여 클래스를 생성한다.

이제 스토리보드의 항목들을 이들 서브클래스의 인스턴스로 구성해야 한다. MainStoryboard.storyboard 파일을 선택하고 Table View Controller를 선택하여 파란색으로 하이라이트되게 한다. Identity Inspector 화면에서(View ➞ Utilities ➞ Show Identity Inspector) 클래스 드롭다운 메뉴를 사용하여 그림 S4-2와 같이 클래스를 UITableViewController에서 CarTableViewController로 변경한다.

그림 S4-2

같은 방식으로 테이블 뷰 컨트롤러 화면의 프로토타입 테이블 셀을 선택하고, 클래스를 UITableViewCell에서 새롭게 만든 서브클래스인 CarTableViewCell로 변경한다.

서브클래스 생성과 스토리보드의 오브젝트와의 연결을 완료하였으므로 다음 과정은 프로토타입 셀을 디자인한다.

S4.4 셀 재사용 식별자 선언하기

이번 장의 뒷부분에서 프로토타입 테이블 셀을 복제하는 코드를 추가할 것이다. 이를 위해 reuse identifier가 필요하다. 스토리보드에서 프로토타입 테이블 셀을 선택하고 Attributes Inspector를 화면에 표시한다. 인스펙터에서 Identifier 항목을 carTableCell로 변경한다.

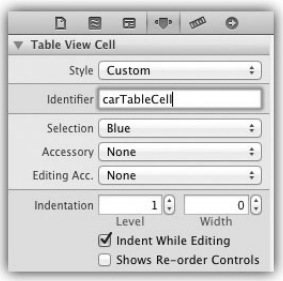

그림 S4-3

S4.5 스토리보드 UITableView 프로토타입 셀 디자인하기

테이블 뷰는 여러 개의 셀들로 구성되어진다. 각각의 셀은 UITableViewCell 클래스 혹은 이것의 서브클래스의 인스턴스다. 스토리보드의 유용한 기능 중 한 가지는 개발자가 시각적으로 테이블 셀에 표시될 유저 인터페이스 요소를 디자인하고 이를 실행 중에 사용할 수 있다는 점이다. 이번 장의 예제에서는 각각의 테이블 셀은 이미지 뷰와 CarTableViewCell 서브클래스에 선언될 아웃렛과 연결된 두 개의 라벨을 표시한다. 인터페이스 빌더와 유사하게 스토리보드도 오브젝트 라이브러리의 컴포넌트를 스토리보드 화면에 드래그 앤 드롭하여 사용한다. 그렇지만 화면을 확대하는 기능은 스토리보드만 지원한다. 이 기능을 사용하기 위해 화면 오른편 아래쪽에 있는 컨트롤을 이용하며, 스토리보드 화면을 확대한 후 두 개의 라벨과 이미지 뷰를 프로토타입 테이블 셀로 드래그 앤 드롭한다. 위치 및 크기를 조절하여 그림 S4-4처럼 표시한다. 라벨은 오른편 경계까지 확장하여 충분한 문자열이 표시될 수 있게 한다.

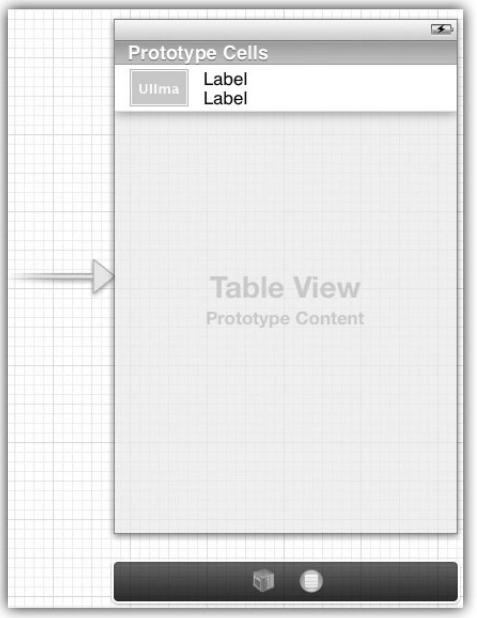

그림 S4-4

애플리케이션의 테이블 뷰에서 사용될 셀을 구성하였으므로 이제 테이블 뷰와 셀의 서브클래스를 수정한다.

S4.6 CarTableViewCell 클래스 수정하기

스토리보드에서 두 개의 라벨과 하나의 이미지 뷰를 추가하였다. 이를 코드에서 사용하기 위해 세 개의 아웃렛을 선언하고 이들 아웃렛을 스토리보드 화면의 오브젝트에 연결하자. 이를 위해 CarTableViewCell.h 파일을 선택하고 다음과 같이 세 개의 아웃렛 프로퍼티를 추가한다.

```
#import <UIKit/UIKit.h>

@interface CarTableViewCell : UITableViewCell
@property (nonatomic, strong) IBOutlet UIImageView *carImage;
@property (nonatomic, strong) IBOutlet UILabel *makeLabel;
@property (nonatomic, strong) IBOutlet UILabel *modelLabel;
@end
```

프로퍼티를 선언하였으므로 CarTableViewCell.m 구현파일에 synthesize 명령어를 추가한다.

```
#import "CarTableViewCell.h"

@implementation CarTableViewCell
@synthesize makeLabel = _makeLabel;
@synthesize modelLabel = _modelLabel;
@synthesize carImage = _carImage;
.
.
@end
```

이제 아웃렛과 유저 인터페이스의 오브젝트를 연결해야 한다. 스토리보드 파일에서 프로토타입 테이블 셀의 흰색 바탕에서 Ctrl-클릭하고 파란색 선을 그림 S4-5와 같이 위쪽의 라벨로 드래그한다.

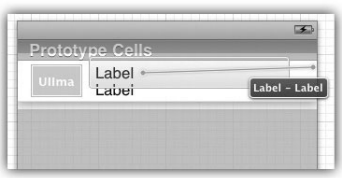

그림 S4-5

마우스를 릴리즈하고 결과 메뉴에서 makeLabel을 선택하여 아웃렛과 연결을 설정한다.

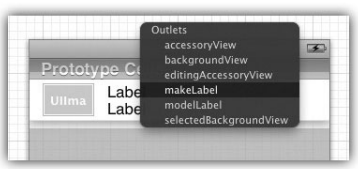

그림 S4-6

위의 과정을 반복하여 CarModels와 carImage 아웃렛을 두 번째 라벨과 이미지 뷰 오

브젝트에 연결한다.

S4.7 테이블 뷰 데이터소스 만들기

동적 테이블 뷰는 셀에 표시될 데이터를 제공하는 데이터소스를 필요로 한다. 기본적으로 Xcode는 스토리보드의 테이블 뷰 컨트롤러의 데이터소스로 CarTableViewController를 지정한다. 그러므로 이 클래스에서 예제 애플리케이션에 필요한 간단한 데이터 모델을 구성할 수 있다. 먼저 CarTableViewController.h 파일에 배열들을 프로퍼티로 선언한다.

```
#import <UIKit/UIKit.h>

@interface CarTableViewController : UITableViewController

@property (nonatomic, strong) NSArray *carImages;
@property (nonatomic, strong) NSArray *carMakes;
@property (nonatomic, strong) NSArray *carModels;
@end
```

프로퍼티로 선언하였으므로 CarTableViewController.m 파일에 synthesize 명령어를 추가한다. 애플리케이션이 실행된 후 배열들이 적절한 데이터를 가지고 있도록 viewDidLoad: 메서드를 구현한다. CarTableViewController.m 파일을 선택하고 다음과 같이 코드를 수정한다. 코드에서 CarTableViewCell 인스턴스를 처리하기 때문에 CarTableViewCell.h 파일을 임포트한다.

```
#import "CarTableViewController.h"
#import "CarTableViewCell.h"

@implementation CarTableViewController
@synthesize carMakes = _carMakes;
@synthesize carModels = _carModels;
@synthesize carImages = _carImages;
.
.
```

```
.
- (void)viewDidLoad
{
    [super viewDidLoad];
    self.carMakes = [[NSArray alloc]
                     initWithObjects:@"Chevy",
                     @"BMW",
                     @"Toyota",
                     @"Volvo",
                     @"Smart", nil];

    self.carModels = [[NSArray alloc]
                      initWithObjects:@"Volt",
                      @"Mini",
                      @"Venza",
                      @"S60",
                      @"Fortwo", nil];

    self.carImages = [[NSArray alloc]
          initWithObjects:@"chevy_volt.jpg",
                      @"mini_clubman.jpg",
                      @"toyota_venza.jpg",
                      @"volvo_s60.jpg",
                      @"smart_fortwo.jpg", nil];
}
```

어떠한 클래스가 테이블 뷰 컨트롤러의 데이터소스가 되기 위해서는 몇 가지 메서드를 반드시 구현해야 한다. 이 메서드들은 테이블 뷰 오브젝트가 테이블에 대한 정보를 얻거나 또는 표시될 테이블 셀 오브젝트에 대한 정보를 얻기 위해 호출된다. CarTableView Controller 클래스를 만들 때 이 클래스를 UITableViewController 클래스의 서브클래스로 지정하였다.

따라서 Xcode는 CarTableViewController.m 파일에 데이터 소스에 필요한 메서드들의 템플릿을 만들어 놓았다. 이 데이터소스 메서드를 찾기 위해 #pragma mark – Table view data source 표시가 보일 때까지 파일을 스크롤 다운한다. 첫 번째 템플릿 메서드는 numberOfSectionsInTableView:이며, 이 메서드는 테이블에서 사용하는 영역의 개수를 리턴한다. 이번 장의 예제에서는 하나의 영역만 사용할 것이므로 1을 리턴한다

(#warning 라인도 삭제한다).

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    // Return the number of sections.
    return 1;
}
```

다음 필요한 메서드는 표시될 행의 개수를 리턴해야 한다. 이는 carModels 배열의 아이템 개수와 같으므로 다음과 같이 수정한다.

```
- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{
    // Return the number of rows in the section.
    return [self.carModels count];
}
```

이 코드에서는 carModels 배열 오브젝트의 count 메서드를 사용하여 배열의 아이템 개수를 얻어 이를 리턴한다.

수정해야 할 마지막 메서드는 cellForRowAtIndexPath:다. 테이블 뷰 컨트롤러가 새로운 셀을 화면에 표시하기 위해 표시해야 할 행의 번호를 인자로 넘겨주며 이 메서드를 호출한다. CarTableViewCell 클래스의 새로운 인스턴스를 만드는 것 역시 이 메서드의 역할이다(혹은 이미 만들어져 있는 인스턴스를 재사용한다). 그리고 전달받은 인덱스 값을 기반으로 데이터 배열에서 적당한 제조사, 모델 및 이미지 파일의 이름을 찾는다. 그 다음 이들 값을 CarTableViewCell 오브젝트의 해당하는 아웃렛에 설정한다. 템플릿 코드를 삭제하고 다음과 같이 코드를 수정한다.

```
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"carTableCell";
```

```
    CarTableViewCell *cell = [tableView
          dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[CarTableViewCell alloc]
          initWithStyle:UITableViewCellStyleDefault
          reuseIdentifier:CellIdentifier];
    }

    // Configure the cell...
    cell.makeLabel.text = [self.carMakes
           objectAtIndex: [indexPath row]];

    cell.modelLabel.text = [self.carModels
           objectAtIndex:[indexPath row]];

    UIImage *carPhoto = [UIImage imageNamed:
           [self.carImages objectAtIndex: [indexPath row]]];

    cell.carImage.image = carPhoto;

    return cell;
}
```

잠깐 이 코드가 무슨 일을 하는지 살펴보자. 이 코드는 재사용 식별자(reuse identifier)를 생성하는 것으로 시작된다. 이 재사용 식별자는 스토리보드의 CarTableViewCell 클래스에서 할당하였으며, 예제에서는 carTableCell을 사용한다.

```
static NSString *CellIdentifier = @"carTableCell";
```

다음의 코드가 실행된다.

```
    CarTableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[CarTableViewCell alloc]
          initWithStyle:UITableViewCellStyleDefault
          reuseIdentifier:CellIdentifier];
    }
```

이 코드는 dequeueReusableCellWithIdentifier: 메서드를 호출하여 재사용이 가능한 이미 만들어진 셀이 있는지 확인한다. 만약 재사용이 가능한 셀이 없으면 CarTable ViewCell 클래스의 새로운 인스턴스를 생성하여 사용한다. 새로운 셀을 사용하든 혹은 기존 셀을 재사용하든 CarTableViewCell 클래스에 추가한 아웃렛을 사용하여 차 제조사와 모델을 설정한다. 그 다음 코드는 현재 선택된 차의 이미지에 해당하는 UIImage 오브젝트를 만들고 이를 이미지 뷰 아웃렛에 할당한다. 마지막으로 메서드는 셀 오브젝트를 테이블 뷰에게 리턴한다.

```
// Configure the cell...
    cell.makeLabel.text = [self.carMakes objectAtIndex: [indexPath row]];

    cell.modelLabel.text = [self.carModels
           objectAtIndex:[indexPath row]];

    UIImage *carPhoto = [UIImage imageNamed:
           [self.carImages objectAtIndex: [indexPath row]]];

    cell.carImage.image = carPhoto;

    return cell;
```

S4.8 이미지 파일 다운로드 및 추가하기

애플리케이션을 수행하기 위해서는 코드에서 필요한 이미지 파일을 프로젝트에 추가하여야 한다. 이미지 파일들은 다음 URL에서 다운로드받을 수 있다.

URL http://www.ebookfrenzy.com/code/carImages.zip

파일을 다운로드한 후 압축을 풀고 파인더에서 해당 파일을 Xcode 프로젝트 내비게이터 화면의 Supporting Files로 드래그 앤 드롭한다.

S4.9 애플리케이션 컴파일 및 실행하기

이제 스토리보드 작업과 코드 수정을 완료하였으므로 애플리케이션을 실행하기 위해 Xcode 툴바의 Run 버튼을 클릭한다. 컴파일이 완료되면 그림 S4-7과 같이 애플리케이션이 iOS 시뮬레이터에서 실행될 것이다.

테이블 뷰는 여러 개의 프로토타입 테이블 뷰 셀을 가지고 화면에 표시된다. 각각의 셀은 차에 대한 정보와 사진을 표시하기 위해 아웃렛을 사용하여 사용자화하였다. 다음 장인 "**Xcode 스토리보드를 이용한 테이블뷰 내비게이션 구현하기**"에서는 스토리보드를 이용하여 테이블의 행을 선택하면 상세정보가 화면에 표시되는 내비게이션 기능을 추가해 보자.

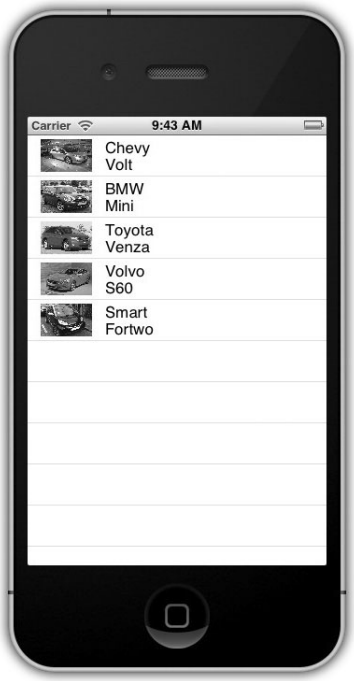

그림 S4-7

S4.10 요약

Xcode의 스토리보드 기능은 테이블 뷰 기반의 인터페이스를 갖는 아이폰 애플리케이션을 매우 쉽게 개발할 수 있게 한다. 기억할 점은 테이블 뷰 셀의 모양을 시각적으로 디자인하고 이를 실행 시 자동으로 복제하여 표시하는 기능이라 할 수 있다.

Special 5

Xcode 스토리보드를 이용한 테이블뷰 내비게이션 구현하기

이번 장에서는 앞 장에서 만든 예제 애플리케이션을 확장하여 스토리보드를 이용한 테이블 뷰 내비게이션 기능을 추가하자. 즉, 앞 장에서 만든 차에 관한 예제 애플리케이션을 수정하여 차 정보가 표시되는 테이블 뷰의 행을 선택했을 때 이 차에 대한 상세정보를 보여주는 새로운 화면을 표시할 것이다. 이를 구현하기 위해 스토리보드를 이용하여 서로 다른 화면 사이의 데이터를 전달하는 방법에 대해서도 알아볼 것이다.

S5.1 내비게이션 컨트롤러 이해하기

내비게이션 기반의 애플리케이션은 사용자에게 정보를 제공하기 위해 계층구조를 사용한다. 보편적으로 이러한 애플리케이션은 내비게이션 바(UINavigationBar)와 여러 가지의 테이블 기반의 뷰(UITableView)를 가진다. 테이블 목록에서 특정 항목을 선택하면 그 항목에 대한 정보를 담은 뷰가 표시된다. 내비게이션 바는 현재 표시되는 내용을 표현하는 제목과 사용자가 이전 뷰로 돌아갈 수 있는 버튼을 제공한다. 아이폰의 메일이나 음악 애플리케이션을 생각해보자.

내비게이션 기반의 애플리케이션을 만들 때 기본이 되는 컴포넌트는 내비게이션 컨트롤러다. 여기에 부가적으로 뷰와 해당하는 뷰 컨트롤러를 가지고 있는 화면으로 구성된다. 내비게이션 컨트롤러는 이들 뷰 컨트롤러의 스택을 관리한다. 새로운 뷰가 표시될 때 이

뷰는 내비게이션 컨트롤러의 스택으로 푸시되고 현재 활성화된 컨트롤러가 된다. 내비게이션 컨트롤러는 내비게이션 바와 "back" 버튼을 표시한다. 사용자가 버튼을 선택하여 이전 화면으로 돌아가면, 뷰 컨트롤러는 스택에서 팝(pop)되고 아래의 뷰 컨트롤러가 최상위 컨트롤러가 되면서 현재 활성화된 컨트롤러의 역할을 한다.

애플리케이션이 실행될 때 화면에 표시되는 첫 번째 뷰 컨트롤러를 루트 뷰 컨트롤러(root view controller)라 부른다. 루트 뷰 컨트롤러는 내비게이션 컨트롤러 스택에서 팝될 수 없다.

S5.2 스토리보드에 새로운 화면 추가하기

이번 장의 예제에서는 스토리보드를 이용하여 두 번째 화면으로 보여질 뷰 컨트롤러를 추가한다. 앞 장에서 만든 TableViewStory 프로젝트를 Xcode에서 로드하자.

프로젝트가 로드되었으면 새로운 뷰 컨트롤러를 추가하자. File ➞ New ➞ File... 메뉴를 선택하고, Objective-C class 옵션을 선택하여 새로운 UIViewController의 서브클래스를 추가한다. 옵션 화면에서 UIViewController의 서브클래스임을 확인하고 이름을 CarDetailViewController로 설정한다. Targeted for iPad와 With XIB file for user interface 옵션은 모두 사용하지 않는다.

다음으로, 프로젝트 내비게이터에서 MainStoryboard.storyboard 파일을 선택하여 스토리보드 화면이 표시되게 한다. 오브젝트 라이브러리에서 View Controller를 선택하고 그림 S5-1처럼 기존의 뷰 컨트롤러의 오른편으로 드래그 앤 드롭한다. 새로운 뷰 컨트롤러가 추가되면 이를 선택하고, identity inspector(View ➞ Utilities ➞ Show Identity Inspector)를 표시한 후 클래스 설정을 UIViewController에서 CarDetailViewController로 변경한다.

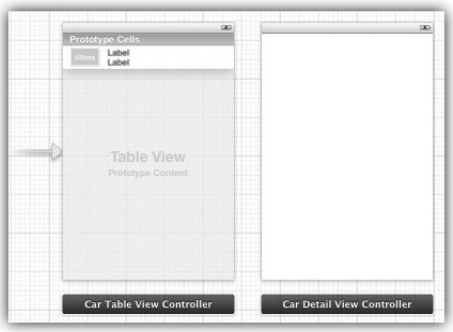

그림 S5-1

상세정보 화면이 추가되고 새롭게 만든 서브클래스로 지정되었다.

S5.3 내비게이션 컨트롤러 추가하기

애플리케이션이 완성되면 테이블 뷰에서 행을 선택할 때 이에 대한 상세정보를 가진 뷰 컨트롤러를 표시하는 segue를 호출할 것이다. 상세정보 화면은 사용자의 선택에 의해 테이블 뷰로 돌아가는 segue를 실행하는 버튼을 가지고 있을 것이다. 이러한 기능은 내비게이션 컨트롤러를 스토리보드에 추가함으로써 가능하게 할 수 있다. 내비게이션 컨트롤러를 추가하기 위해 스토리보드 화면에서 Car Table View Controller를 선택하여 파란색으로 표시되게 한 후 Xcode의 Editor ➞ Embed In ➞ Navigation Controller 메뉴를 선택한다. 이 과정을 수행하면 스토리보드 화면이 그림 S5-2와 같이 표시될 것이다.

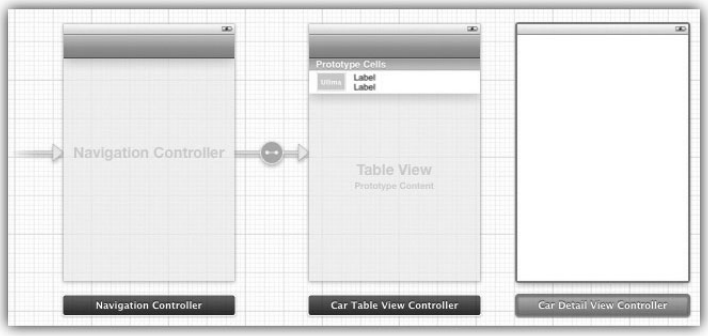

그림 S5-2

S5.4 스토리보드 segue 설정하기

사용자가 테이블 뷰의 행을 선택하면 선택된 차에 대한 상세정보를 표시하는 뷰 컨트롤러를 표시하는 segue가 호출되어야 한다. 이 설정을 위해 Car Table View Controller의 prototype cell을 Ctrl-클릭한 후 Car Detail View Controller로 드래그한다. 마우스 버튼을 놓고 결과 메뉴에서 Push 옵션을 선택한다(스토리보드를 확대하여 수행한다). 스토리보드에 테이블 뷰와 뷰 컨트롤러 사이의 segue 연결이 표시될 것이다. 코드에서 이 segue를 참조하기 위해 segue의 식별자가 필요하다. Car Table View Controller와 Car Detail View Controller 사이의 segue 연결을 선택하고, Attribute Inspector(View ➞ Utilities ➞ Show Attribute Inspector)를 표시하고 Identifier 값을 ShowCarDetails로 변경한다.

또한 화면에 툴바가 표시되었을 것이다. 이 툴바를 더블 클릭하고 각각 "Cars", "Car Details"로 변경한다.

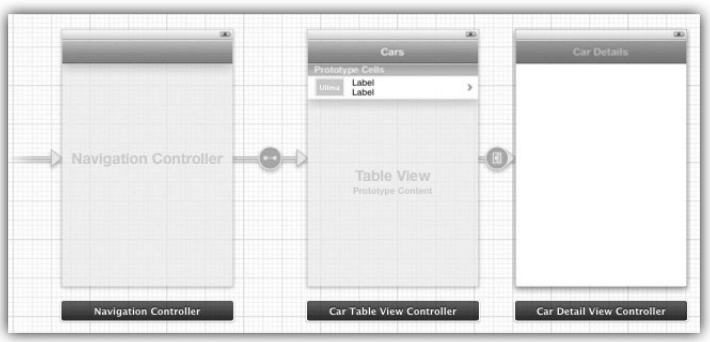

그림 S5-3

애플리케이션을 빌드하고 실행시켜 보자. 테이블 뷰의 행을 선택하면 새로운 뷰 컨트롤러가 표시될 것이며, 테이블 뷰로 다시 돌아갈 수 있는 버튼이 있는 툴바가 보일 것이다. 이제 CarDetailViewController 클래스를 수정하여 선택된 차에 대한 상세정보를 표시해보자.

S5.5 CarDetailViewController 클래스 수정하기

예제 애플리케이션은 선택된 차에 대한 제조사, 모델 정보를 사진과 함께 상세정보 화면에 표시한다. 이를 위해 클래스는 두 개의 라벨 및 UIImageView 오브젝트에 대한 아웃렛이 필요하다.

아웃렛 이외에도 클래스는 차에 대한 정보를 저장하고 있는 내부 데이터 모델이 필요하다. 테이블 뷰 컨트롤러는 차가 선택되면 이들 정보를 갱신하고 화면을 전환하는 segue를 호출한다. 간단하게 구현하기 위해 데이터 모델은 NSArray 오브젝트를 사용하자. CarDetailViewController.h 파일을 선택하고 다음과 같이 파일을 수정한다.

```
#import <UIKit/UIKit.h>

@interface CarDetailViewController : UIViewController

@property (strong, nonatomic) NSArray *carDetailModel;
@property (strong, nonatomic) IBOutlet UILabel *makeLabel;
@property (strong, nonatomic) IBOutlet UILabel *modelLabel;
@property (strong, nonatomic) IBOutlet UIImageView *imageView;
@end
```

다음으로, CarDetailViewController.m 파일을 수정하여 해당하는 @synthesize 명령어를 추가한다. 뷰가 화면에 표시될 때 선언한 아웃렛을 통해 데이터 모델의 정보로 업데이트되어야 한다. 이를 위해 다음과 같이 viewDidLoad: 메서드를 수정한다.

```
#import "CarDetailViewController.h"

@implementation CarDetailViewController
@synthesize makeLabel = _makeLabel;
@synthesize modelLabel = _modelLabel;
@synthesize imageView = _imageView;
@synthesize carDetailModel = _carDetailModel;

- (void)viewDidLoad
{
```

```
    [super viewDidLoad];

    self.makeLabel.text = [self.carDetailModel objectAtIndex:0];
    self.modelLabel.text = [self.carDetailModel objectAtIndex:1];
    self.imageView.image = [UIImage imageNamed:
        [self.carDetailModel objectAtIndex:2]];
}
```

아웃렛을 선언하였으므로 이제 상세정보를 표시하는 사용자 인터페이스를 디자인하고 아웃렛을 연결하자. 스토리보드를 선택하고 확대 모드인 것을 확인한 후 오브젝트 라이브러리에서 필요한 항목을 드래그 앤 드롭하여 그림 S5-4처럼 디자인한다.

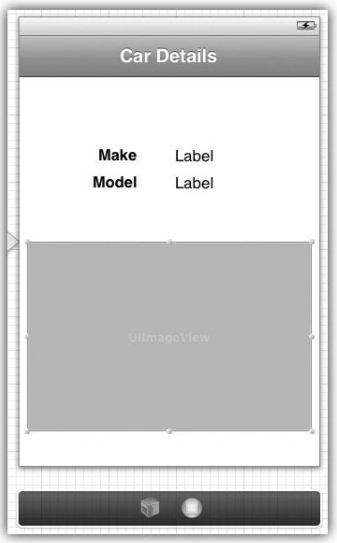

그림 S5-4

마우스를 화면 상단의 상태정보 바(배터리 정보가 표시된 회색 바)로 이동한다. 바를 Ctrl-클릭하고 그림 S5-5와 같이 "Make" 라벨의 오른편 라벨로 드래그한다. 마우스를 릴리즈하고 아웃렛 메뉴에서 makeLabel을 선택한다. 이 과정을 반복하여 modelLabel과 imageView 아웃렛을 연결한다.

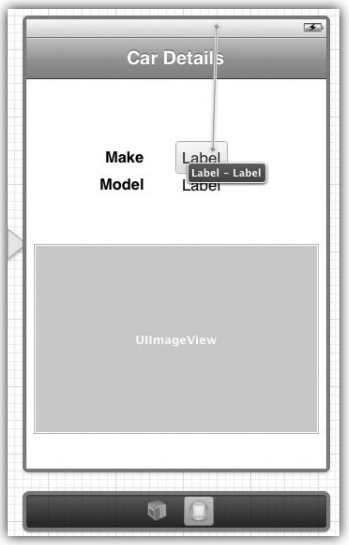

그림 S5-5

S5.6 prepareForSegue:를 사용하여 스토리보드 화면 간 데이터 전달하기

예제 프로젝트를 구현하는 마지막 단계는 사용자에 의해 테이블 뷰의 행이 선택되었을 때 해당하는 차의 세부정보로 CarDetailViewController 클래스의 데이터 모델이 업데이트되게 만드는 것이다. 18장 "**Xcode 스토리보딩 사용하기**"에서 다루었듯이 segue가 실행되기 전에 prepareForSegue: 메서드가 먼저 실행된다. 이 메서드에 화면이 전환될 때 데이터를 전달하는 코드를 추가하자. CarTableViewController.m 파일에 prepare ForSegue: 메서드를 추가하고 다음과 같이 수정한다. 코드에서 CarDetailViewController를 사용해야 하므로 CarDetailViewController.h 파일을 임포트한다.

```
#import "CarTableViewController.h"
#import "CarTableViewCell.h"

#import "CarDetailViewController.h"
.
```

```
.
.

-(void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    if ([[segue identifier] isEqualToString:@"ShowCarDetails"])
    {
        CarDetailViewController *detailViewController =
           [segue destinationViewController];

        NSIndexPath *myIndexPath = [self.tableView
           indexPathForSelectedRow];

        detailViewController.carDetailModel = [[NSArray alloc]
             initWithObjects: [self.carMakes
                objectAtIndex:[myIndexPath row]],
             [self.carModels objectAtIndex:[myIndexPath row]],
             [self.carImages objectAtIndex:[myIndexPath row]],
             nil];
    }
}
```

이 메서드에서 수행하는 첫 번째 작업은 실행될 segue가 ShowCarDetails segue인지 확인하는 것이다. 확인 후에 표시될 화면의 뷰 컨트롤러의 참조 값을 구한다(예제의 경우 CarDetailViewController 클래스). 다음은 테이블 뷰에서 선택된 행의 인덱스 값을 구한 후 이를 이용하여 CarDetailViewController 인스턴스의 carDataModel 배열 프로퍼티를 설정한다.

S5.7 애플리케이션 테스트하기

이제 애플리케이션을 컴파일하고 실행해보자. Xcode 툴바의 Run 버튼을 클릭하고 애플리케이션이 iOS 시뮬레이터에서 실행되기를 기다리자. 테이블에서 차를 선택하고 선택된 차에 대한 상세정보를 가진 새로운 화면이 표시되는 것을 확인한다.

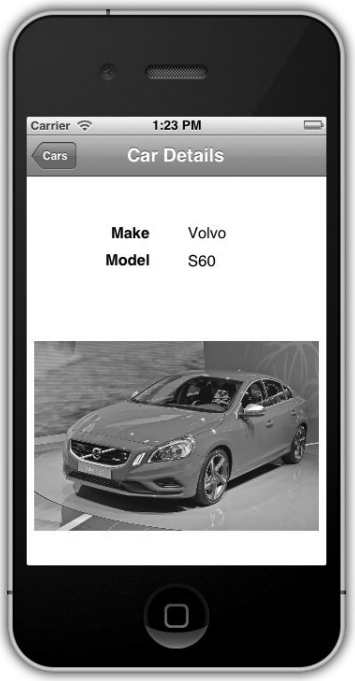

그림 S5-6

S5.8 요약

스토리보드를 사용하여 테이블 뷰 내비게이션을 구현할 때 중요한 요소는 segue의 사용과 화면 간의 데이터를 전달하는 것이다. 이번 장에서는 테이블 뷰에서 선택된 행을 기반으로 두 번째 화면을 표시하기 위해 segue를 사용하는 방법에 대해 알아보았다. 또한 prepareForSegue: 메서드를 사용하여 화면을 전환할 때 데이터를 전달하는 메커니즘에 대해서도 알아보았다.

Special 6

Xcode 스토리보드로 정적 테이블 뷰 만들기

앞 장에서는 스토리보드를 이용하여 동적인 테이블 뷰 iOS 5 아이폰 애플리케이션을 만들어보았다. Special 3장 "**iOS 5 테이블 뷰와 Xcode 스토리보드 개요**"에서 설명하였듯이 Xcode는 정적인 테이블 뷰를 만드는 옵션도 제공한다.

정적인 테이블 뷰는 행의 개수가 이미 정해져 있을 때 적당하다. 정적 테이블 뷰는 데이터 소스를 필요로 하지 않기에 쉽고 빠르게 구현할 수 있다.

이 장에서는 간단한 애플리케이션을 통해 Xcode의 스토리보드를 통해 정적 테이블 뷰를 구현하는 방법에 대해 알아본다.

S6.1 정적 테이블 프로젝트 개요

앞 장에서는 사용자에게 차의 목록을 표시하는 애플리케이션을 만들어보았다. 데이터 모델에 저장되어 있는 차의 개수에 따라 표시되는 행의 숫자가 달라져야 하므로 동적 테이블 뷰를 사용하였다. 목록에서 차를 선택하면 상세정보가 표시되는 두 번째 화면이 나타난다. 어떤 차가 선택되어도 상세정보는 항상 세 가지의 항목을 갖는다(차의 제조사, 모델 및 사진). 앞 장에서는 일반적인 뷰 컨트롤러와 UIView를 사용하였지만, 이러한 경우 정적 테이블 뷰를 사용할 수 있다.

이 장에서는 스토리보드를 이용하여 간단한 정적 테이블 뷰 애플리케이션을 구현해볼 것이다. 완성된 예제 애플리케이션은 앞 장에서 구현한 애플리케이션과 동일한 형태이지만 정적 테이블 뷰를 사용하여 구현될 것이다.

S6.2 프로젝트 만들기

앞 장에서는 Xcode에서 제공하는 Single View Application 템플릿을 이용하여 프로젝트를 만들었다. 그러므로 Xcode를 실행하고 StaticTable이라는 이름의 Single View Application을 만든다. 스토리보드 및 Automatic Reference Counting 옵션은 사용함으로 선택한다.

또한 Xcode에 의해 만들어지는 뷰 컨트롤러가 필요 없으므로 내비게이션 화면에서 StaticTableViewController.h와 StaticTableViewController.m 파일을 선택하고 삭제한다.

S6.3 테이블 뷰 컨트롤러 추가하기

예제 애플리케이션은 하나의 테이블 뷰 컨트롤러를 가질 것이다. 이를 위해 스토리보드에 새로운 화면을 추가하고 UITableViewController의 서브클래스 파일을 생성한다.

스토리보드 편집기가 활성화되어 있으면 오브젝트 라이브러리에서 Table View Controller 오브젝트를 스토리보드 캔버스로 드래그 앤 드롭한다. 화면이 추가되었으면 File ➞ New ➞ File... 메뉴 옵션에서 Objective-C class 추가를 선택한다. Next를 클릭하고, 옵션 화면에서 새로운 클래스의 이름을 StaticTableViewController로 입력하고 UITableViewController의 서브클래스로 설정한다. Targeted for iPad와 With XIB for user interface 옵션이 사용하지 않음으로 선택되었음을 확인하자.

MainStoryboard.storyboard 파일을 선택하고 Table View Controller 화면을 선택하여 파란색으로 하이라이트됨을 확인한다. Identity Inspector 화면(View ➞ Utilities ➞ Show Identity Inspector)을 표시하고 Class 드롭다운 메뉴를 선택하여 클래스를 UITableViewController에서 StaticTableViewController로 변경한다.

S6.4 테이블 뷰 콘텐트 타입 변경하기

기본적으로 Xcode는 테이블 뷰를 동적 테이블 뷰로 가정한다. 그러므로 화면에 Prototype Cell과 "Prototype Content"라는 표현이 보일 것이다. 테이블 뷰의 회색 영역을 클릭하고 Attribute Inspector 화면을 표시하자. Content 속성이 현재 Dynamic Prototypes로 되어 있을 것이므로 이를 Static Cells로 변경한다. 그림 S6-1과 같이 세 개의 행을 가진 정적 테이블이 화면에 표시될 것이다.

그림 S6-1

S6.5 정적 테이블 디자인하기

프로젝트에 정적 테이블 뷰를 추가하였으므로 인터페이스 빌더를 사용하여 테이블과 셀의 형태 및 콘텐트를 디자인한다. 먼저 테이블 뷰의 스타일을 변경한다. 스토리보드에서 테이블 뷰를 선택하고 Attribute Inspector 화면을 표시한 후 Style 속성을 Grouped, Sections 속성을 2로 변경한다. 이제 테이블 뷰는 세 개의 행을 가진 2개의 영역으로 표시될 것이다. 예제의 목적을 위해 위쪽 영역의 세 번째 행을 선택하고 키보드의 delete 키를 눌러 삭제한다. 원하는 테이블 뷰의 영역을 선택하고 Rows 속성을 변경함으로써

새로운 행을 추가할 수도 있다.

아래쪽 영역에서는 하나의 행만 필요하므로 Command-클릭으로 두 개의 행을 선택한 후 삭제한다.

약간 어둡게 표시된, 셀의 바로 윗부분을 클릭하여 위쪽 영역을 선택한다(스토리보드 에디터 윗부분의 툴바에서 Table View Section 항목을 선택, 혹은 Xcode 화면 가운데 부분의 계층 구조에서 선택 가능). Attributes Inspector에서 Header 속성을 Car Details로 변경한다. 아래 영역의 Header는 Car Photo로 변경한다.

아래 영역의 셀을 높이를 조절하여 다음의 그림처럼 테이블 뷰가 표시되도록 한다.

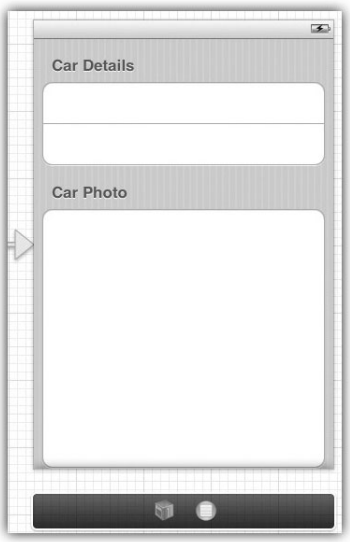

그림 S6-2

S6.6 테이블 셀에 항목 추가하기

오브젝트 라이브러리에서 Image View 오브젝트를 선택하고 아래 영역의 셀로 드래그 앤 드롭한 후 크기를 확대한다. 위쪽 영역에는 라벨을 드래그 앤 드롭하고 그림 S6-3과 같이 표시되도록 text 속성을 설정한다.

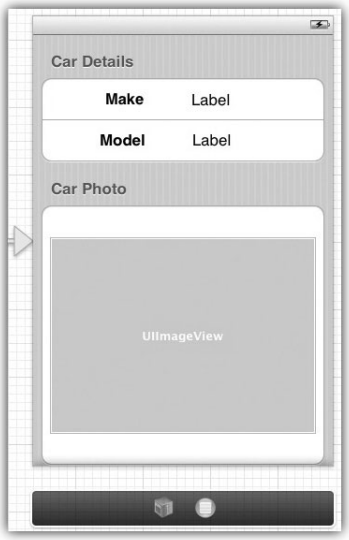

그림 S6-3

테이블 뷰에 대한 유저 인터페이스 디자인이 완료되었으므로 라벨과 이미지 뷰를 위한 아웃렛을 만든다. 물론 아웃렛은 StaticTableViewController 서브클래스에 선언된다.

Special 1장 "**Xcode 어시스턴트 에디터를 사용하여 아웃렛과 액션 만들기**"에서 배운 것을 활용하여 아웃렛을 만들자.

View ➞ Assistant Editor ➞ Show Assistant Editor 메뉴 옵션을 선택하거나 Xcode 에디터 툴바의 가운데 버튼을 클릭하여 어시스턴트 에디터를 표시한다. "Make" 라벨 오른편의 라벨을 선택하고 Ctrl-클릭한 후 그림 S6-4와 같이 어시스턴트 에디터의 @interface 영역으로 드래그한다.

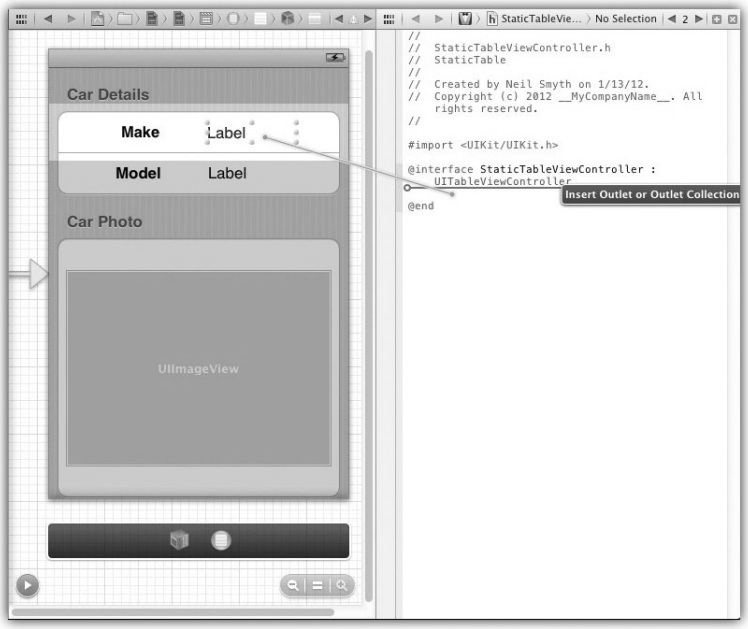

그림 S6-4

마우스를 릴리즈하면 그림 S6-5와 같은 대화창이 나타나 선언될 아웃렛의 정보를 표시한다. 변경할 기본값이 없으므로 Name 항목에 carMakeLabel을 입력하고 Connect 버튼을 클릭한다.

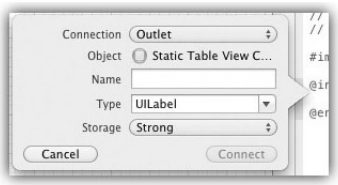

그림 S6-5

두 번째 라벨과 이미지 뷰의 연결을 위해 위의 과정을 반복한다. 아웃렛의 이름은 각각 carModelLabel과 carImageView로 설정한다. 아웃렛이 생성되고 연결되었으면 어시스턴트 에디터를 닫는다.

S6.7 StaticTableViewController 클래스 수정하기

예제를 위해 StaticTableViewController 클래스를 만들 때 UITableViewController의 서브클래스로 선언하였다. 따라서 Xcode는 테이블 뷰의 데이터 소스를 위한 몇 가지 템플릿 메서드를 추가하였다. 그렇지만 정적 테이블 뷰는 데이터 소스가 필요하지 않으므로 이들 템플릿 메서드를 삭제하도록 한다.

StaticTableViewController.m 파일을 선택하고 다음의 라인이 보일 때까지 스크롤 다운한다.

```
#pragma mark - Table view data source
```

클래스에서 데이터 소스를 삭제하기 위해 위의 표시부터 다음의 표시 사이의 메서드들을 삭제한다.

```
#pragma mark - Table view delegate
```

이제 라벨과 이미지 아웃렛을 설정하기만 하면 된다. 이 코드는 한 번만 수행하면 되므로 viewDidLoad: 메서드에 추가하자.

```
- (void)viewDidLoad
{

    [super viewDidLoad];

    self.carMakeLabel.text = @"Volvo";
    self.carModelLabel.text = @"S60";
    self.carImageView.image = [UIImage imageNamed: @"volvo_s60.jpg"];
}
```

파인더 윈도우에서 앞 장에서 사용한 Volvo_s60.jpg 파일을 찾아 프로젝트 내비게이터의 Supported Files 영역으로 드래그 앤 드롭한다.

S6.8 애플리케이션 빌드 및 실행하기

애플리케이션의 구현이 완료되었으므로 Xcode 툴바의 Run 버튼을 클릭하여 iOS 시뮬레이터에서 실행한다. 다음과 같이 정적 테이블이 표시될 것이다.

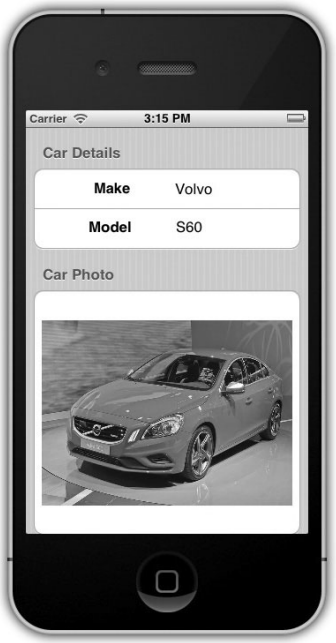

그림 S6-6

스토리보드와 테이블 뷰의 개념에 대한 설명을 하기 위해서 앞 장에서 구현한 Table ViewStory 애플리케이션을 정적 테이블 뷰를 이용하여 유사하게 구현해보았다.

S6.9 요약

동적 혹은 prototype 콘텐트 기반의 테이블 뷰는 데이터 소스에서 데이터를 가져와 표시하므로 가변적인 개수의 행을 표시하는 데 적당하다. 반면에 미리 표시할 항목의 개수가 정해져 있는 경우에는 정적 테이블 뷰를 사용하여 쉽게 구현할 수 있다. 이 장에서는 스토리보드를 이용하여 정적 테이블 뷰를 구현한 간단한 예제 애플리케이션을 만들어보았다.