

아이폰 입문

아이폰 iPhone은 현재 가장 특색 있는 게임 플랫폼 중 하나이다. 터치 화면, 아이튠즈 iTunes와의 통합, Objective-C를 사용하는 프로그래밍, 적은 개발비용, 쉬운 제품 출시 등은 매우 낯설지만 새로운 개발 기회를 약속한다. 모바일 폰 시장에 가장 늦게 진입한 아이폰은 주목할 만한 점유율을 보이고 있고, 모토로라, 삼성, LG 같은 제조사의 모방기기 열풍을 이끌어내고 있다.

프로그래머로서 판매량이나 시장 점유율에 감흥이 없을수도 있지만, 전체적으로 아이폰의 생존 능력에는 흥미를 가져야 할 것이다. 사람들이 아이폰을 가지고 있지 않다면 프로그래머가 만든 게임을 아무도 사지 않을 것이다. 좋은 소식은 2009년 경제 침체 국면에도 불구하고, 아이폰은 계속해서 팔렸다는 것이다.

아이폰을 시작하기 위해서는 무료 애플 개발자 계정을 받아야 한다. 그 다음에는 맥Mac에 아이폰 SDK를 다운받는다. SDK에는 Xcode IDE와 화면을 구성하는 인터페이스 빌더 Interface Builder 도구가 포함돼 있다. 애플은 윈도우즈를 비롯한 맥이 아닌 플랫폼에는 개발도구를 제공하지 않으므로 반드시 맥이 있어야 한다. 그리고 아이폰 API는 Objective-C를 사용해야 하므로 이에 대해 잘 모른다면 관련 입문서를 읽어야 한다. 이 장에서는 이 작업 모두를 단계별로 설명할 것이다.

1.1 애플 개발자 계정과 SDK 다운로드

아이폰 개발환경 구축의 첫 단계는 애플 개발자 계정을 등록하는 것이다. 계정 가입은 무료이며 이를 통해 애플의 온라인 문서, 튜토리얼 비디오, SDK 다운로드에 접근할 수 있다.

1. <http://developer.apple.com/iphone/>에 접속한다.
2. 등록 Register 링크를 클릭하고 Start Now를 클릭한다. new Apple ID 만들기를 옵션을 선택하거나 아이튠즈나 앱스토어 App Store 계정에서 쓰던 Apple ID로 로그인한다.
3. 등록을 하면 아이폰 개발 센터에 로그인할 수 있다.
4. 애플에서 무료 SDK를 다운로드하는 링크를 이메일로 이미 보냈거나, 웹사이트의 링크를 통해 SDK 다운로드를 선택할 수 있을 것이다. SDK 다운로드 패키지 안에는 이미 Xcode가 포함되어 있으므로 따로 Xcode를 받을 필요가 없다는 점을 알아두자. 그리고 SDK에는 최신 버전의 Xcode가 포함되어 있다.
5. 다운로드하고 SDK를 설치하면, 하드 드라이브의 */Developer/Applications* 폴더에서 Xcode와 인터페이스 빌더를 찾을 수 있다. 이 두 애플리케이션을 빠르게 실행하기 위해 검색기능 Spotlight을 이용하여 Xcode와 인터페이스 빌더를 찾을 수도 있다.

무료 개발자 계정으로 애플리케이션을 만들고 맥의 시뮬레이터를 통해 애플리케이션을 실행할 수 있다. 그러나 애플리케이션을 아이폰에서 테스트하려면 유료 개발자 프로그램에 가입해야 한다. 1년 단위로 내는 비용이 크지 않기 때문에 지갑이 얇은 개인 개발자일지라도 그리 부담되지는 않을 것이다.

1. <http://developer.apple.com/iphone/program/apply.html>에 접속한다.
2. 스탠다드 프로그램과 엔터프라이즈 프로그램이란 두 옵션이 있다. 일반적인 목적으로 프로그램을 만든다면 아마 엔터프라이즈 프로그램은 필요하지 않을 것이다. 설명을 모두 읽었다면 엔터프라이즈 프로그램이 회사에서 내부적으로 사용할 프로그램을 만들기 위한 것임을 알 수 있을 것이다. 만일 제작한 게임을 앱스토어를 통해 판매할 계획이라면, 스탠다드 프로그램이

옳은 선택이 될 것이다.

3. Enroll Now를 선택하고 필요할 경우 로그인한다.

4. 이제 다른 선택이 가능하다. 개인이나 혹은 회사로 등록하는 것이다. 개인을 선택하면, 개발과 테스트 단계에서 다른 이들에게 애플리케이션을 배포하는 데에 필요한 다른 프로그래머나 품질 보증 멤버를 계정에 추가할 수 없을 것이다. 만약 회사를 선택한다면, 회사에 대한 자세한 정보 제공을 요청받을 것이다.

5. 적당한 정보를 선택하면서 'Thank you for submitting your enrollment (등록해주셔서 감사합니다)'라는 문구가 적힌 화면을 만날 때까지 웹사이트를 통해 계속 진행한다. 이제 애플의 이메일을 기다린다(도착할 때까지 대략 한 달 정도 걸릴 것이다).



유료 개발자 프로그램 가입은 애플이 활성화하는 동안 아이폰 OS와 SDK의 최신 버전 베타 릴리즈에 접근할 수 있게 해준다.

실제 필요할 때 쓸 수 있도록 가능한 빨리 유료 개발자 계정 등록을 해두는 것이 좋다.

1.1.1 애플리케이션 번들

Xcode를 사용하여 애플리케이션을 만들 때 마지막 결과를 애플리케이션 번들(Application Bundle)이라 부른다. 맥 OS X와 아이폰에서 애플리케이션 번들은 실행파일과 실행하는 데에 필요한 리소스를 지닌 특수한 타입의 디렉터리다. 이것은 애플리케이션을 나타내는 아이콘과 애플리케이션에 대한 특수한 정보를 지닌 파일, 애플리케이션이 사용하는 이미지나 사운드를 포함한다.



파인더(Finder)에서 애플리케이션 번들은 단순히 애플리케이션 아이콘으로 나타난다. 우클릭이나 컨트롤-클릭을 하고 메뉴의 View Package Contents를 선택하면 안에 무엇이 들어있는지 볼 수 있다.

아이폰에서는 이렇게 할 수 없지만 아이폰 시뮬레이터는 아이폰 애플리케이션을 찾을 수 있다. 아이폰 SDK를 설치했다면 검색 기능을 사용하여 *MobileSafari.app* 파일을 찾을 수 있다. 이 파일을 파인더에서 열고(맥에서 실행하지는 말고) 패키지 콘텐츠를 보라.

전형적인 아이폰 애플리케이션 번들은 다음 구조를 지닌다.

Executable

(필수) 컴파일하여 실행가능한 코드이다. 보통 애플리케이션과 같은 이름을 지닌다.

*MobileSafari.app*에서 이 파일의 이름은 *MobileSafari*이다.

Info.plist

(필수) 키-값의 쌍으로 이루어진 형태인 속성들의 컬렉션이다. 애플리케이션에 대한 중요한 정보를 기록한다. 이 중에서 주목할 만한 속성은 애플리케이션의 이름, 버전 숫자, 유니크 ID 숫자 등이다. 이 파일들은 바이너리 포맷을 사용하여 텍스트 에디터에서 읽을 수 없지만, */Developer/Applications/Utilities*에 있는 Property List Editor를 사용하면 볼 수 있다.

icon.png

(필수) 아이폰의 홈 화면에서 애플리케이션을 표현하는 57×57픽셀의 아이콘이다. 광택 버튼 효과는 이미지의 윗부분에 자동으로 추가되므로 균일한 색으로 한다.

Various resources

(선택) 실행파일처럼 이미지, 사운드, 바이너리 데이터 같은 애플리케이션에서 사용하는 모든 일반적인 리소스 파일들은 같은 폴더에 놓인다. 아이폰 애플리케이션 번들에 존재하는 유일한 하위 폴더는 각 언어 리소스의 묶이다.

Localization folders

(선택) 애플리케이션이 여러 언어를 지원한다면 번들에 각 언어를 담은 리소스를 포함하는 서브 폴더를 추가한다. 폴더 이름은 언어 이름이나 ISO 언어 약자 뒤에 *.lproj*를 붙인다. 예를 들면 *English.lproj*, *French.lproj*, *German.lproj*, *uk.lproj*는 각각 영어, 프랑스어, 독일어, UK 영국식 영어를 포함한다.

Settings.bundle

(선택) 제작한 애플리케이션의 사용자 환경 옵션을 아이폰의 설정 애플리케이션에서 제공하려면 이 파일을 만들어야 한다.

Icon-Settings.png

(선택) *Settings.bundle* 파일을 추가했다면 이 이미지는 설정 애플리케이션에서 해당 애플리케이션을 표현하는 데에 쓰인다. 이미지는 29×29 픽셀이어야 하지만 이미지를 추가하지 않으면 *Icon.png* 이미지를 자동으로 바꿔 사용한다.

MainWindow.nib

(선택) 인터페이스 빌더 애플리케이션이 만드는 *MainWindow.nib*는 애플리케이션이 시작할 때처럼 애플리케이션을 그리는 데에 필요한 코드와 리소스를 포함한다. 더 많은 *.nib* 파일을 이후에 읽어올 수 있지만, 이 파일이 메모리에서 항상 첫 위치에 있다.

Default.png

(선택) 이 이미지는 애플리케이션이 *MainWindow.nib* 파일을 로딩하는 동안 보여준다. 아이폰에서는 480×320 픽셀인 전체 화면 크기여야 한다. 애플리케이션 로딩을 마치는 동안 유저에게 이 이미지를 보여주면 더 적은 시간이 걸린 것처럼 보일 것이다.

iTunesArtwork

(선택) 애플리케이션을 앱스토어 밖에서 배포하면, 아이튠즈를 이용하여 핸드폰에 로드하는 동안 애플리케이션을 나타내는 데에 이 아트웍^{artwork}을 사용한다. 자세한 것은 이후에 다루도록 하겠다.

다음 절에서 보겠지만, 애플리케이션을 만들 때에 Xcode와 인터페이스 빌더로 이 파일들 대부분을 만들 것이다.

1.1.2 Xcode와 인터페이스 빌더

Xcode에 익숙하지 않다면 처음에는 새 IDE를 배우는 것이 다소 부담스러울 수는 있겠지만 이것은 아이폰 개발 작업에 반드시 필요한 일이다. IDE를 다루는 데에 익숙해지면 Xcode의 편리함을 알 수 있을 것이다. 산업 표준 IDE에 필요한 모든 기능을 가지고 있기 때문에 컴파일 에러가 난 곳으로 바로 이동할 수 있고, API 메소드를 자동으로 완성하거나 SDK 레퍼런스를 통합할 수 있다.

이 외에도 Xcode는 기기에서의 디버깅, 모든 기능을 갖춘 아이폰 시뮬레이터, 유용한 프로젝트 마법사, 리팩토링 도구, 버전관리 기능의 직접적인 통합을 제공한다.

Xcode 프로젝트는 아이폰 애플리케이션을 만드는 데에 필요한 코드, 리소스, 인증서, 환경설정 모두를 포함하고 있다. 이 환경을 알기 위해 Xcode IDE를 열어 다음 단계를 따라 전형적인 'Hello World' 애플리케이션을 만들어 보자.

1. Xcode를 연다.
2. File → New Project를 선택한다.
3. 열린 다이얼로그에서 iPhone OS를 선택하고, View-Based Application을 선택한 다음 Choose를 클릭한다. 그림 1-1을 참조한다.

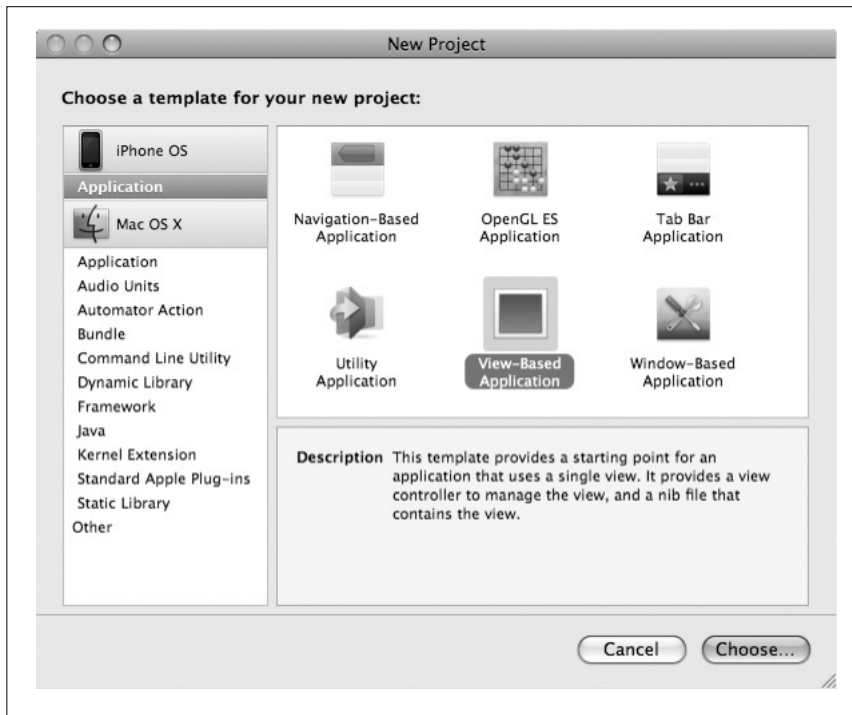


그림 1-1 View-Based Application을 선택

4. project를 'Hello World' 라 이름 짓고 Save를 클릭한다.
5. 이 단계에 이르면, 빌드하고 실행할 수 있다(툴바의 Build와 Go 아이콘을 클릭

한다). 시뮬레이터에서 실행하면 그림 1-2처럼 HelloWorld 애플리케이션은 텅 빈 회색 화면만을 보여준다.



그림 1-2 시뮬레이터의 텅 빈 애플리케이션

아직 별로 재미있지는 않을 것이다. 이 ‘Hello World’ 애플리케이션을 만들기 전에 프로젝트를 만들 때 만들어진 파일들에 대해 알아보자.

HelloWorldAppDelegate.m, HelloWorldAppDelegate.h

이 파일들에 담긴 클래스는 애플리케이션의 시작점이 되는 메인 코드라 볼 수 있다. *app delegate*는 메인 창과 메인 뷰 컨트롤러를 제어하고 화면을 구성하는 역할을 한다.

HelloWorldViewController.m, HelloWorldViewController.h

이 파일들의 클래스는 메인 뷰를 지니고 있는데, 아주 재미없는 회색 화면만을 보여준다. 곧 ‘Hello World’ 라고 말하도록 고칠 것이다.

MainWindow.xib

이 인터페이스 빌더 파일은 프로젝트를 컴파일하면 애플리케이션 번들에 위치하는 *.nib* 파일이다. 로드할 때 *app delegate*를 만들고, 메인 창과 뷰 컨트롤러를 로드한다.

HelloWorldViewController.xib

이 파일은 *HelloWorldViewController*의 뷰에 쓰는 디자인을 정한다.



NIB는 NeXTSTEP Interface Builder를 나타내고, XIB는 Xcode Interface Builder를 나타낸다. NIB 파일은 컴파일된 난해한 바이너리 파일이고, XIB 파일은 사람이 읽을 수 있는 XML 파일이다. 앞에서 말했듯이 Xcode는 XIB 파일을 NIB 파일로 컴파일한다. XIB 포맷은 소스 관리 차원에서 프로젝트에 NIB 파일을 합치는 문제를 풀기 위해 만들어졌다. 바이너리 파일보다 XML 파일을 훨씬 쉽게 구별할 수 있기 때문이다.

이제 ‘Hello World’ 텍스트를 그린다. 여러 가지 방법을 통해 이를 실행할 수 있다.

- *HelloWorldViewController.m*에 직접 코드를 적어서 Cocoa UILabel을 추가한다.
- 인터페이스 빌더에서 *HelloWorldViewController.xib*에 Cocoa UILabel을 추가한다.
- UIView의 서브클래스를 정의하고, drawRect에서 Quartz 글꼴 렌더링을 이용한다.
- OpenGL ES에서 텍스처 매핑(texture-mapped) 글꼴을 만든다.

먼저 첫 번째 방법을 사용해보자. *HelloWorldViewController.m*에 코드를 적어서 UILabel을 추가하는 것이다. *HelloWorldViewController.m* 안에는 이미 코드를 추가하기 좋은 viewDidLoad라는 이름의 스텝 메소드가 있다. 이 메소드는 .nib 파일의 로딩이 끝나고 화면을 그리기 전에 호출된다.

1. *HelloWorldViewController.m*의 viewDidLoad 함수를 다음의 내용으로 바꾼다. 이 함수는 기본적으로 주석처리되어 있기 때문에 /*부터 */까지의 부분을 지워야 한다.

```
- (void) viewDidLoad {
    [super viewDidLoad];
    // Cocoa UIKit을 사용해서 "Hello World"를 그린다.
    // 화면의 크기를 알아낸다.
    int w = self.view.frame.size.width;
    int h = self.view.frame.size.height;

    // 레이블을 만들고 임의로 (100,50)이라는 크기로 설정한다.
    // "Hello World" 문자열을 충분히 담을 크기이다.
    UILabel* label = [[UILabel alloc] initWithFrame:CGRectMake(0, 0, 100, 50)];
    // 레이블을 화면 중앙에 놓는다.
```



```

label.center = CGPointMake(w/2, h/2);
// 기본적으로 좌측정렬이기 때문에 문자열을 레이블의 가운데로 정렬시킨다.
label.textAlignment = NSTextAlignmentCenter;
// 기본 배경색이 흰색이므로 레이블의 배경을 따로 그리지 않는다.
label.backgroundColor = [UIColor clearColor];
label.text = @"Hello world!";
// 레이블을 그리기 위해 뷰에 레이블을 추가한다.
[self.view addSubview:label];
// 레이블에 메모리를 할당하였으므로 여기에서 레이블의 메모리를 해제한다.
[label release];
}

```

2. 프로젝트를 빌드하고 실행한다. 이전에 실행한 시뮬레이터를 아직 종료하지 않았다면 바로 종료한다. 이제 앱은 그림 1-3처럼 텍스트를 보여줄 것이다.



그림 1-3 'Hello world!' 텍스트가 보인다

이번에는 두 번째 방법을 사용해보자. 먼저 인터페이스 빌더를 이용하여 *HelloWorld-ViewController.xib*에 UILabel을 추가해본다. 앞 예제를 따라 했다면 바꾼 부분들을 되돌려야 한다.

1. Xcode의 프로젝트 파일 목록에서 *HelloWorldViewController.xib* 파일을 더블클릭하여 인터페이스 빌더에서 연다.
2. 뷰 오브젝트를 더블클릭하여 수정하기 시작한다(그림 1-4 참조).

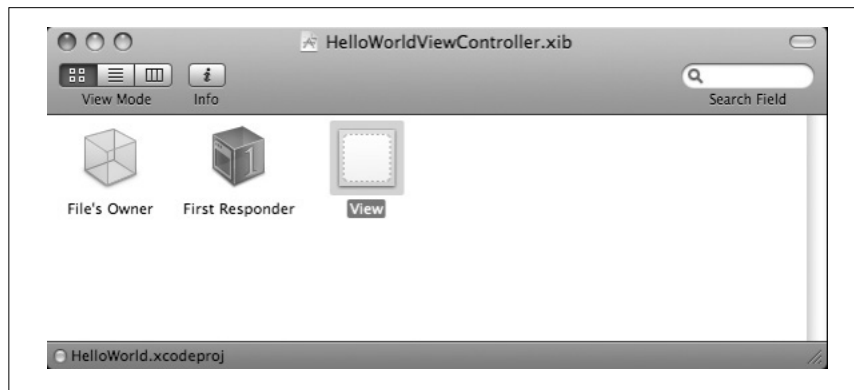


그림 1-4 뷰 오브젝트를 더블클릭하여 파일을 수정한다

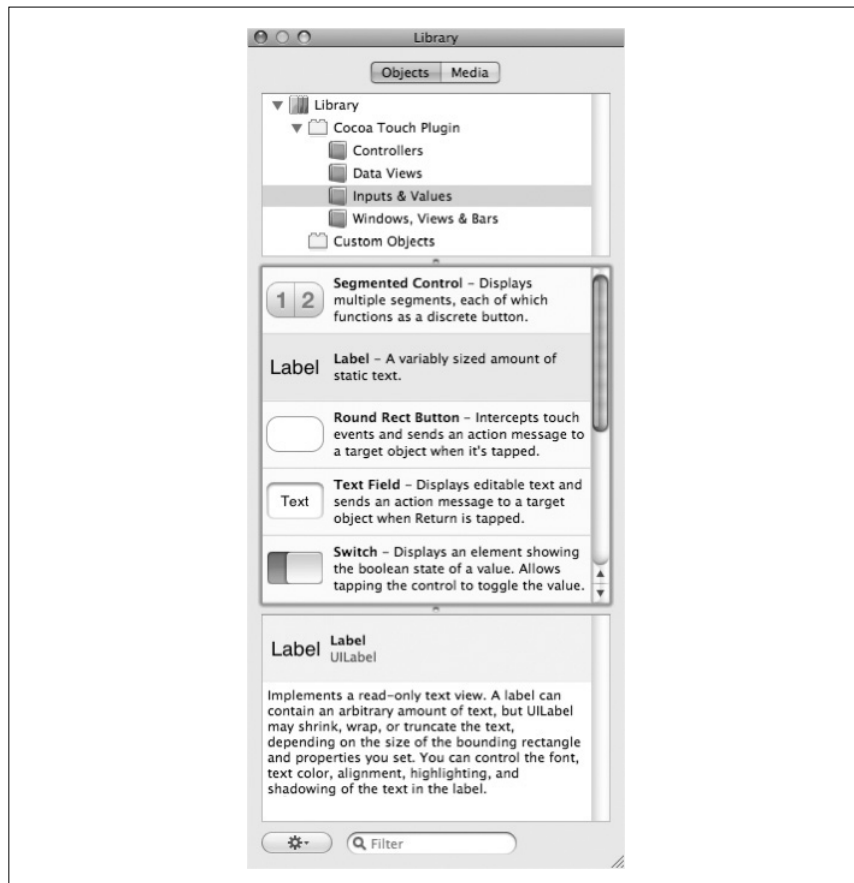


그림 1-5 Objects Library에서 Label 선택한다

3. Library panel(열려 있지 않다면 Tools → Library Panel을 선택하여 panel을 연다)에서 Label 오브젝트를 찾는다. 그림 1-5를 보라.
4. 그림 1-6처럼 label을 View 수정 창으로 드래그한다.
5. 'Hello World' 를 보여주기 위해 새 label을 더블클릭하고 텍스트를 수정한다. Attributes Inspector(Tools → Attributes Inspector)에서도 수정할 수 있다.
6. Label Size Inspector(Tools → Size Inspector)에서 Placement 버튼을 둘 다 클릭하여 label을 그림 1-7처럼 수평과 수직으로 가운데에 위치하게 한다.

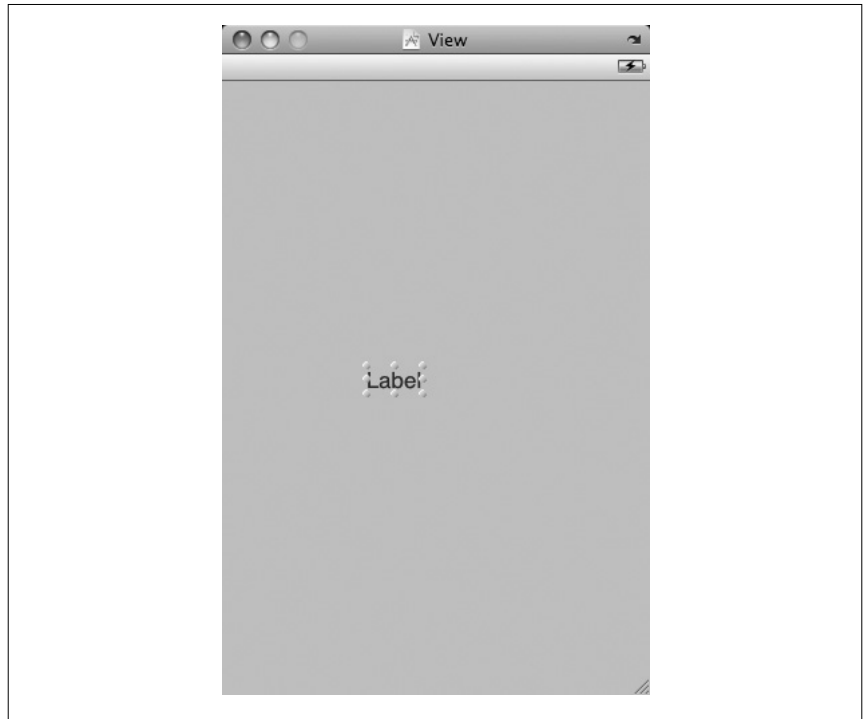


그림 1-6 View 수정 창에 새로운 label을 추가한다.

7. .xib 파일을 저장하고 Xcode로 돌아온다. 애플리케이션을 빌드하면 그림 1-8처럼 애플리케이션이 실행될 때 보이는 모습을 만드는 .nib 파일을 업데이트한다.

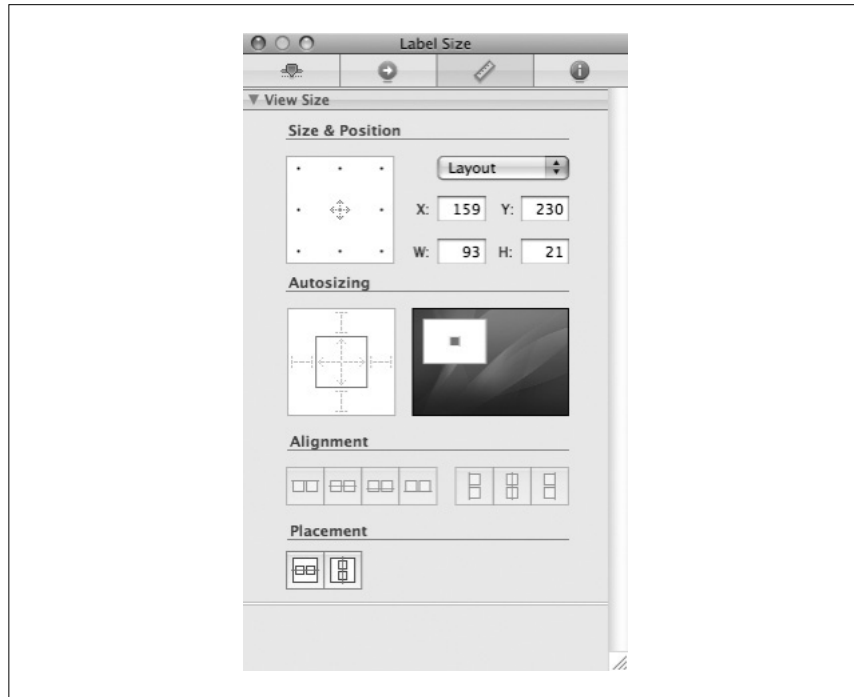


그림 1-7 Label 오브젝트 속성 수정



그림 1-8 Label 오브젝트로 만든 'Hello World!' 텍스트

8. *.nib* 파일이 로드되면 UILabel을 만들어 보여준다. 코드가 만들어진 label을 읽거나 수정하기 위해 접근하려면, 코드의 IBOutlet과 연결한다. *HelloWorldViewController.h*에서 HelloWorldViewController의 스텝^{stub} 정의를 다음과 같이 바꾼다.

```
@interface HelloWorldViewController : UIViewController {
    IBOutlet UILabel* myLabel;
}
```

IBOutlet은 뷰의 오브젝트를 코드에서 처리할 수 있게 한다는 것을 인터페이스 빌더에 알리는 코드 태그이다.

9. *.nib*의 label을 아웃렛에 연결하기 위해 인터페이스 빌더의 *HelloWorldViewController.xib*를 연 다음 Connections Inspector(Tools → Connections Inspector)를 연다. 그 다음에 label을 클릭하여 선택하고, label의 New Referencing Outlet을 File의 Owner 오브젝트에 드래그한 후 놓고, 팝업 메뉴에 보이는 'myLabel' 텍스트를 클릭한다(그림 1-9부터 1-11까지를 보자). Referencing Outlet을 설정했기 때문에 두 오브젝트는 *.nib*가 로드되면 서로 연결될 것이다. 다시 말해 *HelloWorldViewController.nib*가 애플리케이션에 로드되면, myLabel 변수가 UILabel을 가리키도록 연결된다는 것이다.

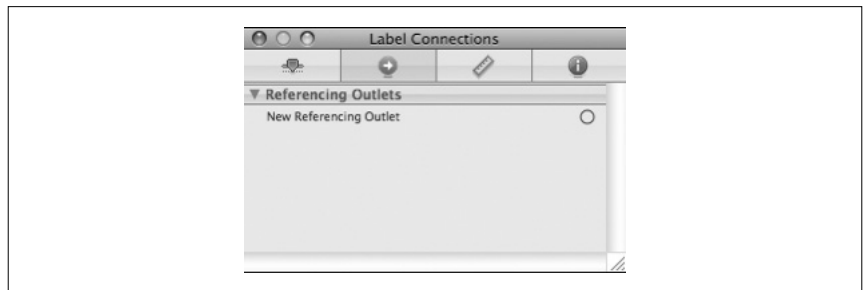


그림 1-9 Label 연결 창

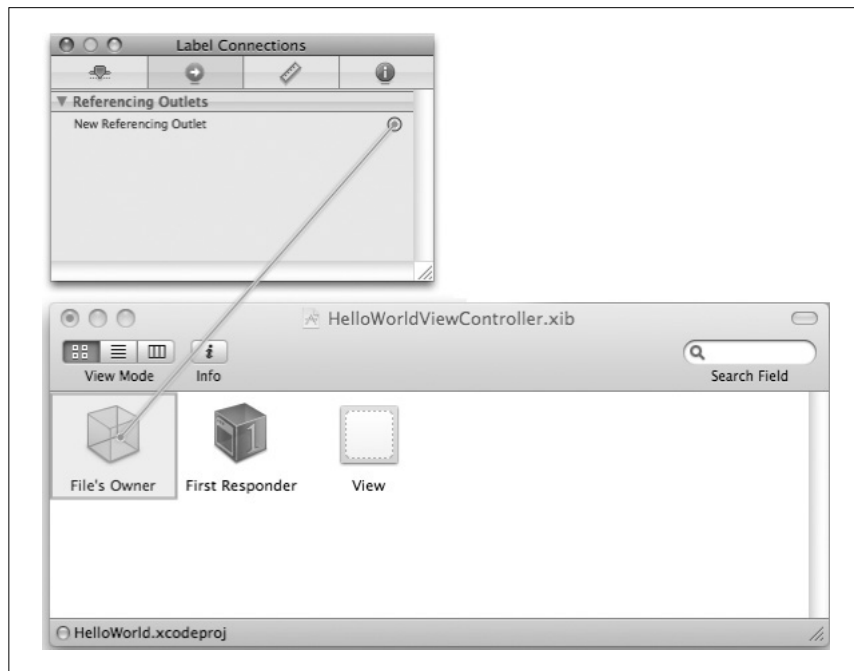


그림 1-10 New Referencing Outlet을 클릭하고 File's Owner로 드래그하기

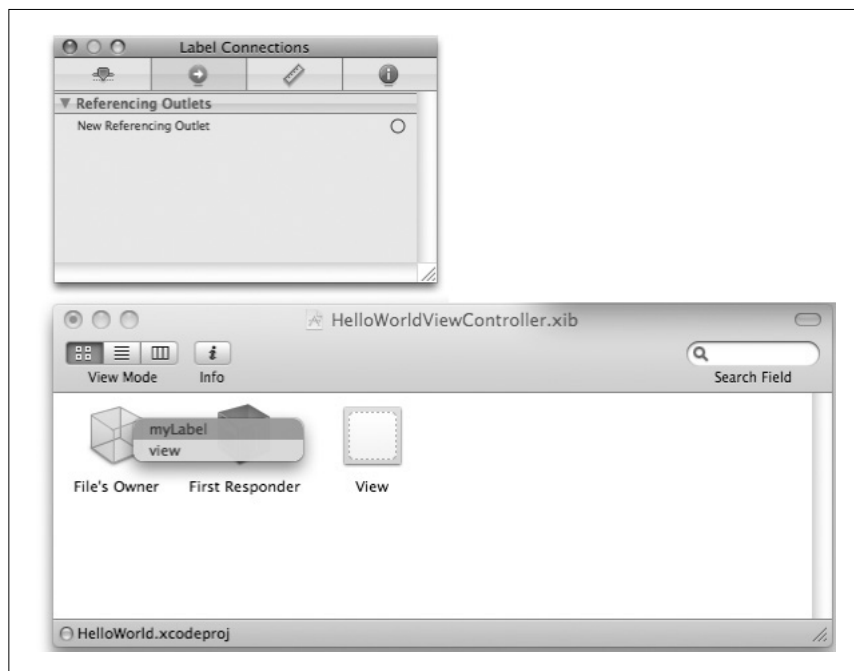


그림 1-11 팝업 창에서 'myLabel' 선택

10. *HelloWorldViewController.xib*를 저장하고 인터페이스 빌더를 종료한다.

이제 코드는 `myLabel`을 통해 `label`에 접근할 수 있다. 이 연결 과정은 인터페이스 빌더에서 아웃렛, 액션, 딜리게이트, 데이터 소스를 명시하기 위해 자주 쓰인다. 인터페이스에 보이는 요소들과 코드 사이의 모든 소통은 인터페이스 빌더 연결을 통해 만들어진다.

1.1.3 뷰와 컨트롤러

아이폰 SDK에서 `UIView` 클래스는 `View` 오브젝트를 나타낸다. 뷰(View)는 드로잉(Drawing)과 애니메이션, 이벤트 핸들링, 서브 뷰 관리를 담당하는 화면의 사각영역이다. 아이폰에서 아이팟 애플리케이션을 보면 상단의 네비게이션 바, 하단의 탭 바, 가운데 콘텐츠 영역의 세 부분으로 분리되어 있는 뷰를 볼 수 있다.

새 프로젝트 마법사에서 뷰 기반 애플리케이션(view-based application)을 만들 때에는 하나의 뷰 컨트롤러(Controller)와 하나의 뷰를 가지고 시작한다. 애플리케이션이 더 복잡해지면 보다 많은 뷰를 필요로 하게 된다. 예를 들어 게임은 지금 하나의 뷰를 가지고 있겠지만, 메인 메뉴나 설정 화면, 온라인 최고 점수 화면 등을 위한 뷰가 추가될 수 있다.

뷰들이 한 코드의 많은 부분을 공유한다면, 그 뷰들의 코드를 공유하는 뷰 컨트롤러에 추가하는 것이 합리적이다. 앞의 예에서 메인 메뉴와 설정은 같은 뷰 컨트롤러를 쓰면 좋았겠지만, 메인 게임 스테이트와 최고 점수 화면을 각각 고유의 뷰 컨트롤러에 넣었다. 그 둘을 모두 하나의 뷰 컨트롤러에 넣거나 각각 다른 뷰 컨트롤러에 넣어도 안 되는 것은 아니지만 순수하게 구조적인 면을 고려한 것이다.

1.1.3.1 새 뷰와 뷰 컨트롤러를 기존의 창에 추가하기

다음 과정을 거쳐 새 뷰와 뷰 컨트롤러를 기존의 창에 추가한다.

1. Xcode에서 `UIViewController`를 확장하는 새 클래스를 만든다. 이를 `TestViewController`라 하자.
2. `File` → `New File`을 선택하고 `User Interfaces` 부분에서 `View XIB`를 골라서 그림 1-12처럼 새 XIB 파일을 만든다.



그림 1-12 새 View XIB 파일 추가하기

3. 인터페이스 빌더에서 File의 Owner icon을 선택한다. Identity Inspector에서 Class Identity를 TestViewController로 설정한다. Identity Inspector 창이 안 열려 있으면 Tools → Identity Inspector를 통해 열 수 있다. 그림 1-13을 보라.
4. 앞의 Hello World 예제에서 했던 것과 같은 방법으로 그림 1-14처럼 View's Referencing Outlet을 File's Owner view로 설정한다.
5. TestViewController 인스턴스를 코드에 추가하고 IBOutlet을 적는다. 이를 app delegate class인 AppDelegate 안에 넣을 것이다. AppDelegate.h에서 새 코드는 이와 같은 모양이 될 것이다.

```
IBOutlet TestViewController* testViewController;
```

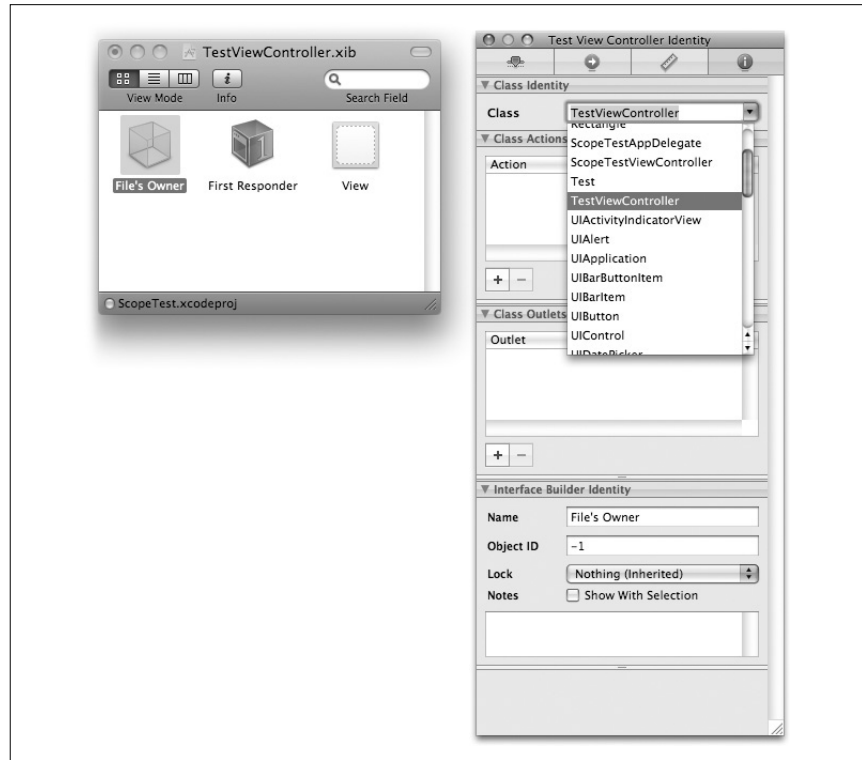



그림 1-13 클래스 이름 수정

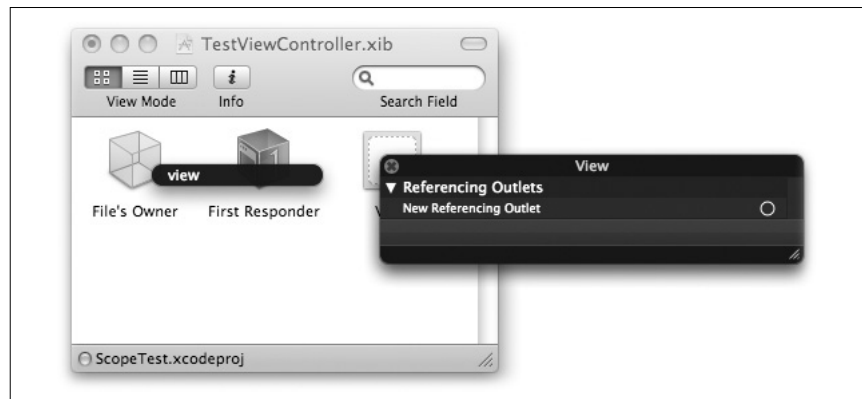


그림 1-14 File's Owner 오브젝트에 연결하기

6. 메인 창의 XIB 파일을 수정하고 Library 창에서 *MainWindow.xib* 창으로 드래그하여 **ViewController** 오브젝트를 추가한다. 그림 1-15와 1-16에서 보듯이 Library → Cocoa Touch Plugin → Controllers의 Library에서 **ViewController** 오브젝트를 찾을 수 있다.



그림 1-15 Library로부터 뷰 컨트롤러 추가하기

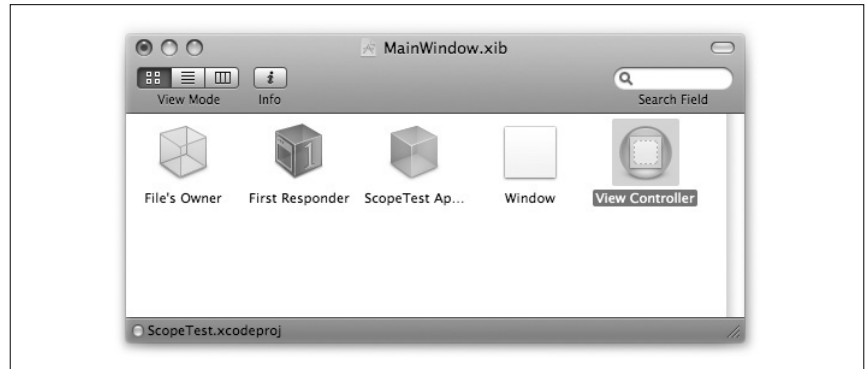


그림 1-16 뷰 컨트롤러 오브젝트 수정하기

7. 클래스의 이름을 바꾸고 NIB가 'TestViewController'를 로드하도록 바꾸기 위해 이 오브젝트를 수정하자. 먼저 *MainWindow.xib* 창의 View-Controller 오브젝트를 선택하고 Identity Inspector 창에서 클래스 이름을 수정한다. 다음에는 같은 창의 Attributes 버튼을 선택하여 Identity Inspector에서 Attributes Inspector로 바꾼다. 마지막으로, Attributes Inspector에서 NIB 이름을 수정한다. 그림 1-17과 1-18을 참조한다.
8. 이 오브젝트의 Referencing Outlet을 이전에 만든 TestAppDelegate의 IBOutlet인 testViewController에 연결한다. 그림 1-19를 보라.
9. 다음 코드를 사용하여 화면을 보여주기 전에 view 오브젝트를 Window 오브젝트에 추가한다.

```
[window addSubview:testViewController.view];
```

10. View 오브젝트가 Window 오브젝트에 추가되면 호출하여 활성화한다.

```
[window bringSubviewToFront: testViewController.view];
```

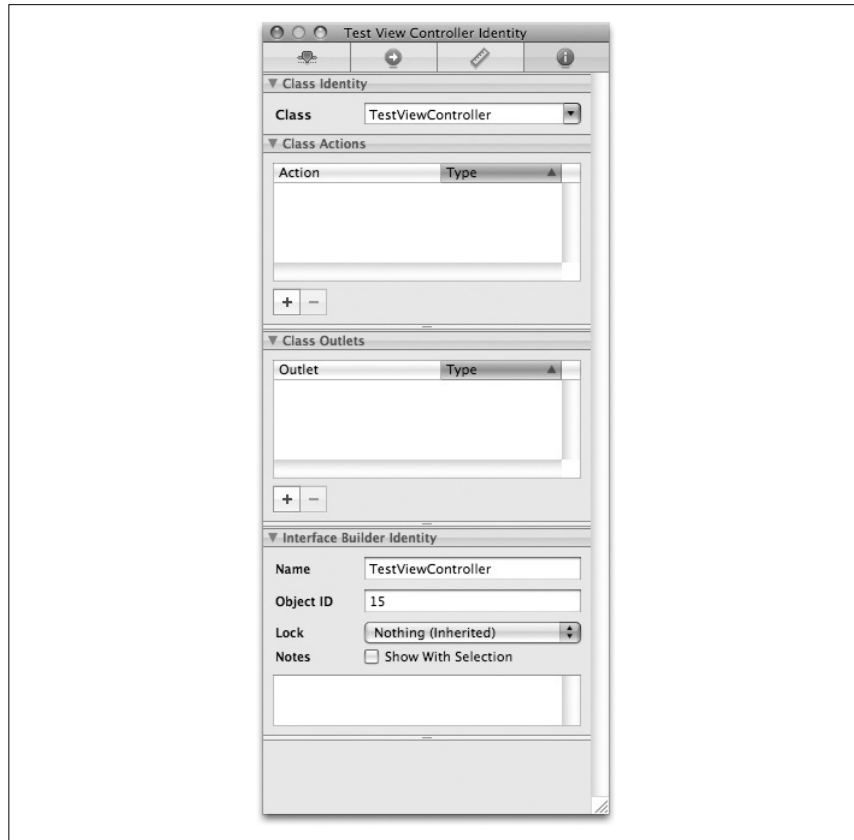


그림 1-17 클래스 이름 수정



그림 1-18 NIB 이름 수정

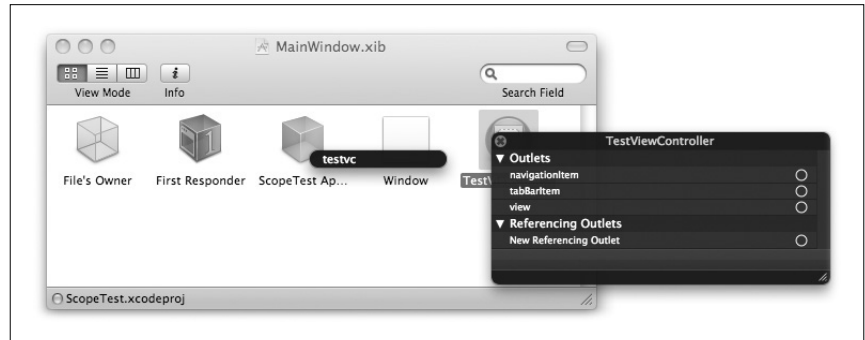


그림 1-19 TestViewController에 연결하기

어느 때건 오직 하나의 뷰만 활성화할 수 있다는 점을 기억하자. `bringSubviewToFront`를 호출하면 리스폰더 체인에서 현재 활성 중인 뷰는 제거된다. 그 뷰는 `bringSubviewToFront`를 다시 호출할 때까지 전면에 나타나거나 입력 이벤트를 받지 않을 것이다.

1.1.3.2 새 뷰를 기존의 뷰 컨트롤러에 추가하기

이 과정을 따라서 기존의 뷰 컨트롤러에 새 뷰를 추가한다.

1. Xcode의 뷰 컨트롤러의 `.h` 파일을 열고 새 뷰를 위해 `IBOutlet UIView` 포인터를 추가한다. 이름은 `secondView`라 짓는다. 코드는 이와 같은 모습이 될 것이다.

```
IBOutlet UIView* secondView;
```

2. 인터페이스 빌더에서 뷰 컨트롤러의 XIB 파일 창을 열고 Library로부터 `UIView` 오브젝트를 추가한다. 그림 1-20처럼 `UIView`는 Library → Cocoa Touch Plugin → Windows, Views & Bars 아래에서 찾을 수 있다.
3. `UIView`의 Referencing Outlet을 이전에 만든 `IBOutlet`인 `secondView`에 설정한다.
4. 뷰 컨트롤러 XIB 파일 창의 `UIView` 오브젝트에 이름을 붙일 수 있다.
5. 뷰 컨트롤러의 어딘가에 있는 새 뷰로 바꾸기 위해 이를 호출한다.

```
self.view = secondView;
```

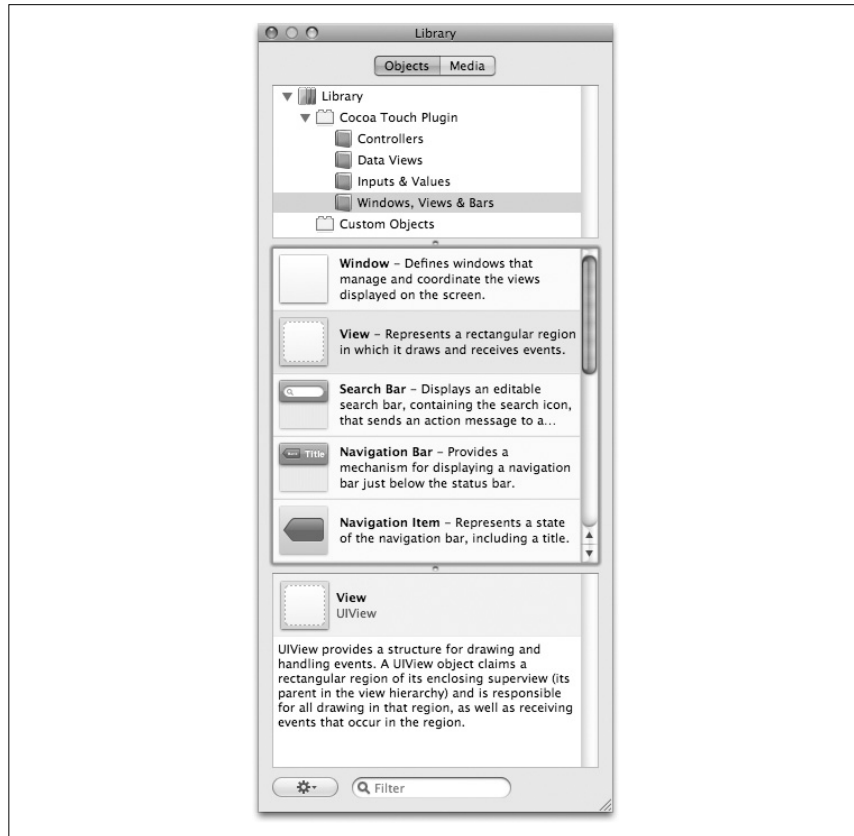


그림 1-20 Library로부터 뷰 오브젝트 추가하기

여기에서 뷰 컨트롤러의 view 속성을 덮어 쓰기 때문에, 다시 돌아올 필요가 있는 경우에는 이전 뷰를 참조할 레퍼런스를 가지도록 해야 한다. 일반적으로 각 UIView를 고유의 IBOutlet에 넣어야 하며, 추가로 view 속성에 초기 시작 UIView를 할당해야 한다. 어느 UIView이건 하나 이상의 레퍼런스 아웃렛을 가질 수 있다. 따라서 초기 뷰는 initialView라는 이름의 아웃렛과 view를 가질 것이다.

1.1.3.3 프록시 오브젝트

NIB 안의 오브젝트들은 File의 Owner 클래스 안의 속성들에 접근할 수 있다. NIB 밖에서 속성들에 접근하려면 Proxy 오브젝트를 사용해야 한다. Proxy는 연관된 클래스 이름과 프록시 이름을 지니지만 연관된 클래스의 인스턴스를 만들지는

않는다. NIB 파일을 초기화할 때 Proxy 인스턴스에 자신을 넘겨주어야 한다.

```
// 메인 윈도 NIB에서 뷰 컨트롤러를 자동으로
// 가져오지 않고, 수동으로 가져와서 프록시 오브젝트를 구성할 수 있다.
viewController = [[HelloWorldViewController alloc] init];
// viewController가 HelloWorldViewController와
// HelloWorldAppDelegate의 IBAction에 접근할 수 있어야 한다.
// File's Owner는 View Controller가 되고,
// AppDelegateProxy는 app delegate [self]가 될 것이다.
NSDictionary* proxies = [NSDictionary
    dictionaryWithObject:self forKey:@"AppDelegate"];
// 프록시 오브젝트 이름을 app delegate 인스턴스에 연결한다.
NSDictionary* options = [NSDictionary
    dictionaryWithObject:proxies
    forKey:UIInibProxiedObjectsKey];
// 프록시로 옵션을 만든다.
[[NSBundle mainBundle] loadNibNamed:@"HelloWorldViewController"
    owner:viewController options:options];
```

1.2 기기에 올리기

개발자 프로그램을 구입하고 가입절차가 끝나면, Xcode를 이용해서 “Hello World” 예제 같은 애플리케이션을 아이폰에 올릴 수 있다. 하지만 그전에 해결할 어려운 일들이 아직 남아 있다.

아이폰은 보안 플랫폼이기에 기기에서 코드를 실행하기 전에 해야 할 일들이 잔뜩 있다. 빌드에 서명을 하고, 서명은 애플에 인증을 받아야 하고, 기기는 서명한 애플리케이션을 올릴 준비가 되어 있어야 한다.

서명과 인증 관리, 프로파일 작성은 애플 웹사이트의 프로그램 포탈을 통해 온라인으로 이루어진다. 이미 프로그램을 구매한 개발자라면 <http://developer.apple.com/iphone/manage/overview/index.action>에서 할 수 있다.

포탈에서는 서명 인증 요청(Certificate Signing Requests, CSRs), 배포 프로파일 작성, 기기 ID 추가 등의 각 주제에 맞는 단계별 가이드를 제공하지만, 절차가 복잡하기 때문에 먼저 전반적인 개념에 대해 알아보도록 하자.

1.2.1 인증과 프로파일

애플리케이션을 전화기나 앱스토어에 출시하려면 서명한 프로파일이 2개 필요하다. 하나는 애플리케이션을 개발하고 테스트하는 동안 사용하는 개발자 프로파일이고, 다른 하나는 개발을 끝낸 애플리케이션을 퍼블리시^{publish}하기 위해 앱스토어에 보낼 때 사용하는 배포 프로파일이다. 회사에서 설정할 때는 수석 프로그래머가 개발자 프로파일을 사용하고, 제품 매니저는 배포 프로파일을 사용할 것이다.

프로파일들을 만들려면 공개 키^{Public Key}와 개인 키^{Private Key}를 가진 인증서를 만들어야 한다. 이 인증서는 애플의 서명을 받아야 사용할 수 있기 때문에 제일 먼저 할 일은 맥에서 키체인 접근^{Keychain Access}을 사용하여 애플의 개발자 프로그램에 보낼 인증서 의뢰 요청^{Certificate Submission Request}을 만드는 것이다. 애플에서 요청을 확인하고 수락하면 애플 개발자 포털 웹사이트에서 인증서를 다운받을 수 있다.

애플이 서명한 인증서를 받은 다음에는 App ID, UDID와 WWDR 인증서가 필요하다.

1.2.1.1 App ID

프로파일과 애플리케이션 번들에 사용하는 App ID는 개발자 포털에서 만든다. App ID는 'com.게임회사이름.게임이름'의 형태를 취한다. 이것을 리버스 도메인 표기라고 부르는데, 자바와 C# 프로그래머들은 패키지 이름 포맷과 같기에 익숙할 것이다.

한 개발자가 여러 개의 애플리케이션을 만들 때 와일드카드 포맷을 사용해서 애플리케이션들에 하나의 App ID를 사용할 수도 있다. 와일드카드를 사용한 App ID는 'com.게임회사.*'의 형태를 취한다.

App ID를 개발자 포털에 기록하면 ID의 앞에 영문과 숫자로 이루어진 10개의 문자가 추가된다. 이는 애플에서 내부적으로 사용하는 것이니 무시해도 상관없다.

개발자 포털에서 App ID를 만든 후에, 애플리케이션 번들에서 *Info.plist* 파일을 열고 *bundle identifier key*의 기본 값을 App ID 값으로 바꾼다. 포털에서 App ID를 만들 때 와일드 카드를 사용했다면 *Info.plist* 파일을 수정할 때 *를 게임 이름으로 바꿔야 한다.

1.2.1.2 UDID

UDID는 프로그램을 올릴 아이폰 기기에서 얻는 40글자의 16진수 문자열인데, 기기를 맥에 연결하고 아이튠즈나 Xcode Organizer 창을 열면 알 수 있다.



UDID는 개발자 프로파일에만 쓰이고 배포 프로파일에는 쓰이지 않는다.

1.2.1.3 WWDR 인증서

애플의 Worlde Wide Developer Relations^{WWDR} 인증서는 애플 사이트의 <http://developer.apple.com/certificationauthority/AppleWWDRCA.cer>에서 구할 수 있다. 인증서를 다운로드하여 설치하면 빌드를 인증할 때 사용할 수 있다. WWDR 인증서는 개발 인증서를 애플과 연결하여 애플리케이션의 신뢰 체인을 완성한다.

1.2.1.4 프로파일 설치

이 셋을 갖추고 나면 개발자 포탈에서 준비^{Provisioning} 프로파일을 다운로드할 수 있다. 이 파일은 `.mobileprovision` 확장자를 가진다. 다운로드한 다음에 더블클릭을 하면 프로파일을 설치한다. 이 프로파일은 Xcode의 Organizer에 넣어야 하고, 아이폰을 컴퓨터에 연결하면 Xcode의 Organizer에 나타나게 된다. 이때 프로파일이 기기에 설치된 프로파일 목록에 있는지 꼭 확인해야 한다.

애플리케이션은 개발자 프로파일의 목록에 있는 기기에서만 실행할 수 있다는 것을 기억하자.

1.2.2 Xcode 설정

모든 인증서, 키, 프로파일 설치가 완료되면 아이폰에서 실행할 애플리케이션을 컴파일하는 Xcode를 설정할 준비가 된 것이다.

Project → Edit Project Settings → Build → Code Signing Identity → Any iPhone OS Device에서 코드 서명 프로파일을 설정한다. 그림 1-21을 보라.

provisioning 프로파일이 드롭-다운 목록에 나타나지 않으면 서명 체인의 어느 부분을 빠뜨린 것이다.

- 개인 키
- 서명 인증서
- WWDR 인증서
- provisioning 프로파일

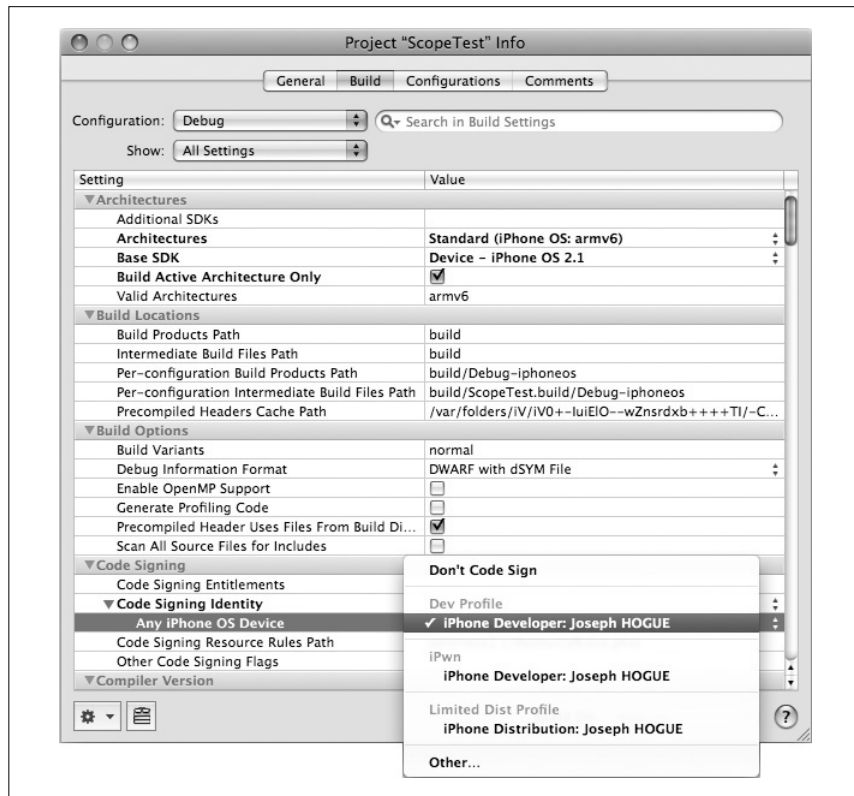


그림 1-21 코드 서명 프로파일 설정

bundle identifier를 provisioning profile의 App ID name에 맞도록 바꾼다. *Info.plist*가 바뀐 후에 Build Clean 메뉴 옵션을 선택한다. 하지만 변화는 빌드에 반영되지 않을 것이다. 그림 1-22를 보자.

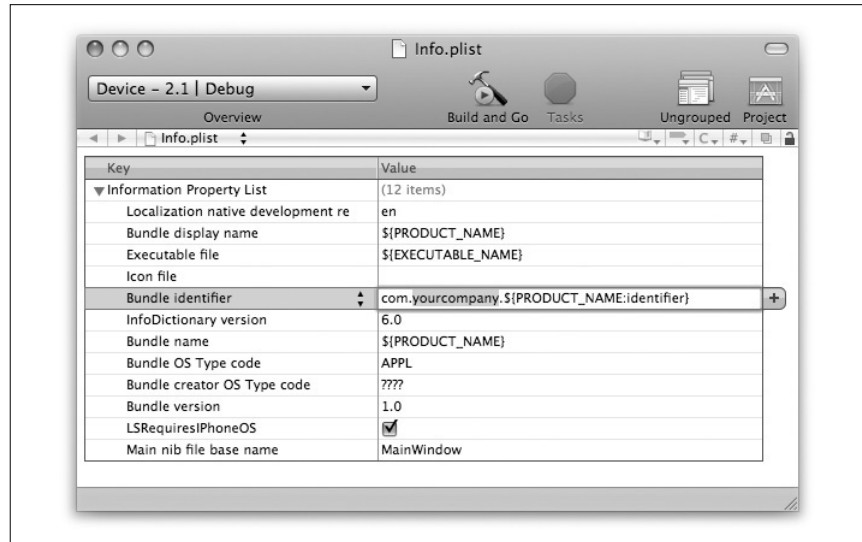


그림 1-22 Info.plist의 App ID를 수정

기기를 위해 빌드하도록 그림 1-23처럼 Xcode를 설정한다.

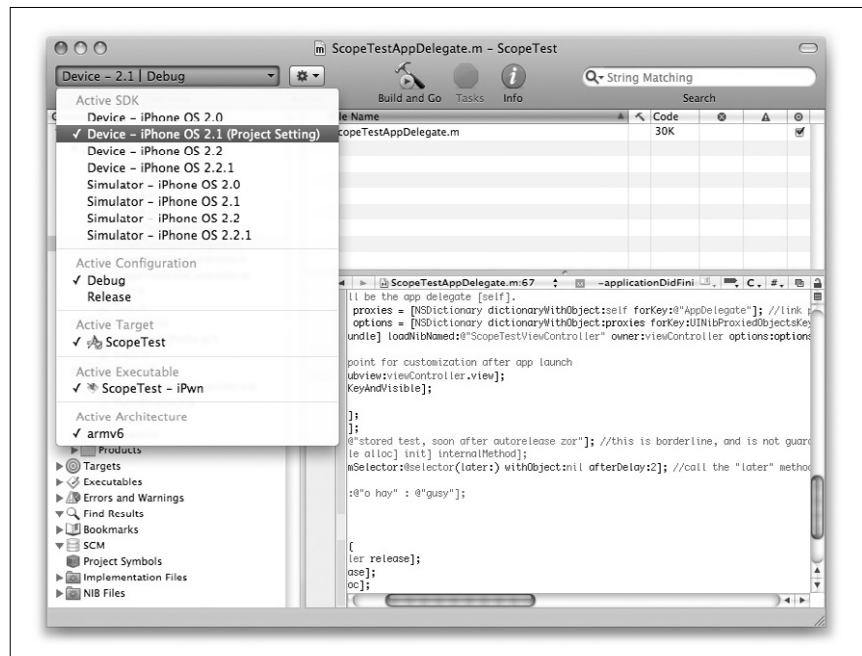


그림 1-23 빌드 대상에 맞는 기기를 설정

모든 과정이 잘 이루어졌다면 애플리케이션을 빌드하고 기기에서 실행할 수 있을 것이다. 빌드가 진행되는 동안 Xcode는 앱을 기기에 설치할 것이다.

무언가 잘못되었다면 유용한 에러 메시지는 볼 수 없고 잘못되고 있다는 의미를 담고 있는 ‘ApplicationVerificationFailed’를 보게 될 것이다. 다음은 에러의 원인이 될 수 있는 흔한 유형의 목록이다.

bundle identifier가 오자나 공백을 포함한다

bundle identifier는 App ID에 맞추어야 한다. 번들 identifier를 수정한 후에 Build → Clean을 선택한다.

인증 체인의 일부를 찾을 수 없다

WWDR 인증서는 잊기가 쉽다.

인증서가 만료됐다

모든 것이 잘 됐는데 갑자기 문제에 직면한다면 인증서가 만료됐을 것이다.

때로는 아이폰을 리부팅하고 다시 연결하는 것만으로도 문제가 해결되곤 한다.

1.3 Objective-C 입문

아이폰 API는 Objective-C 언어를 사용한다. Objective-C는 스몰토크와 C언어의 원리를 응용한 것이다. 발상은 업계의 표준인 C 언어의 힘과 스몰토크 언어의 OOP 접근을 합치는 것이었다. 이로 인해 처음엔 혼란스럽겠지만 몇 가지 개념을 이해하면 그 구조를 이해하기 어렵지 않을 것이다.

먼저 알아야 할 것은 Objective-C가 C를 포함한다는 점이다. C로 작성한 것은 모두 Objective-C 컴파일러를 이용해 컴파일할 수 있다. C 스타일 구조, 함수 호출, 메모리 할당, 포인터 모두 가능하다. 여기에 보통 C 언어 문법의 기본 이외에 Objective-C는 스레드 동기화, 트라이-캐치 블록, 가비지 컬렉션 같은 고유의 기능 몇 가지가 추가되었다. 그리고 주 개선점이자 핵심은 객체지향 기반이라는 것이다.

1.3.1 클래스

Objective-C의 클래스는 자바나 C++의 클래스와 개념적으로 매우 비슷하다. 클래스^{class}는 멤버 변수^{member variable}라 부르는 데이터와 메소드^{method}라 부르는 함수를 포함하는 오브젝트이다. 자바와 C++에서와 같이 클래스는 사용하기 전에 정의된다 (Objective-C에서는 런타임에 클래스가 만들어질 수 있다). 클래스는 `@interface` 키워드를 사용하여 정의한다. C++와 비슷하게 Objective-C 클래스는 `.h` 파일에 정의를 하지만, 구현은 `.m` 파일에 분리하여 해야 한다. 또한 C++로 하는 것처럼 `@public`이나 `@private` 키워드를 사용하여 따라오는 줄의 보호 문법 단계를 표시한다. 그러나 Objective-C는 C++나 자바와 다르기 때문에 메소드를 클래스의 종괄호 바깥에 정의한다. 메소드는 바로 앞의 클래스와 연관이 되고 `@end` 키워드로 끝난다.

다음 코드는 `Animal` 클래스를 만든다.

```
@interface Animal
{
    // 멤버 변수는 여기에 둔다.

    @private
    int foo;
}
// 클래스 메소드는 종괄호 바깥인 여기에 둔다.

- (int) bar: (double) input1;
@end
```

메소드를 정의하는 재미있는 방법을 기억하자. 보통 함수는 `-`로 시작을 하고(`+`는 클래스 메소드에 사용을 한다. 곧 다룰 것이다), 괄호 안에 리턴 타입을 적고 이어서 함수의 이름을 적는다. 함수가 파라미터를 가지면 콜론 뒤에 각 파라미터 설명을 적는다. 파라미터 타입은 괄호 안에 적는다. 그리고 파라미터 이름을 적는다(파라미터 이름 대신 함수 이름을 사용하는 첫 번째 파라미터는 제외한다).

여기에 C++로 작성한 같은 클래스가 있다. 마지막에는 세미콜론이 필요하다는 것에 주의하자.

```
class Animal
{
    // 멤버 변수는 여기에 둔다.
private:
    int foo;
public:
    int bar( double input1 );
};
```

자바로 한번 더 해보자.

```
class Animal
{
    // 멤버 변수와 메소드는 여기에 둔다.
private int foo;
public int bar( double input1 ) {}
}
```

세 언어 모두 //로 한 줄 주석을 사용한다.

그리고 자바와 C++처럼 클래스는 다른 클래스를 확장^{extend} 또는 상속^{inherit}할 수 있다. 거의 모든 Objective-C 클래스가 상속하는 공통 베이스 클래스는 후에 설명할 NSObject이다. 상속 문법은 public과 private이 부모 클래스의 앞에 사용되지 않는 것을 제외하면 C++와 비슷하다.

```
@interface Animal : NSObject
{
}

// Objective-C 상속
@end

class Animal : public NSObject
{
    // C++ 상속
}
;

class Animal extends NSObject
{
    // Java 상속
}
```

자바처럼 Objective-C도 여러 클래스를 상속할 수 없다는 점을 주의하는 것이 중요하다. NSObject뿐만 아니라 게임의 특수한 이벤트들에 응답할 수 있는 GameEventListener 클래스도 같이 상속하는 Animal 클래스를 만들고 싶다고 해보자. C++에서는 GameEventListener를 추상 베이스 클래스로 정의하고 둘 다 상속할 수 있다. 그러나 자바에서는 GameEventListener를 인터페이스 클래스로 정의해야 하고, Objective-C에서는 @protocol로 정의할 것이다. Objective-C의 프로토콜은 절대적으로 virtual이어야 한다는 점만 제외하고 C++에서의 추상 베이스 클래스와 비슷하다. 어떠한 멤버 변수나 메소드 구현을 가져서는 안 된다. 자바의 인터페이스처럼 Objective-C의 클래스는 필요한 만큼 많은 프로토콜을 구현할 것이다.

C++ 구현은 이렇게 한다.

```
class GameEventListener
{
    // 멤버 변수, 메소드, 가상 메소드는 여기에 온다.
private:
    bool active;
public:
    bool isListening(); // active가 true일 때 true를 리턴한다.
    virtual void handleEvent( int event ) = 0;
};
class Animal : public NSObject, public GameEventListener
{
    // 멤버 변수, 메소드, 오버라이드 메소드가 여기에 온다.
    // GameEventListener의 모든 순수 virtual 함수는 반드시 구현해야 한다.
private:
    int foo;

public:
    int bar( double input1 );
    void handleEvent( int event );
    // isListening()은 구현되어 있으므로 오버라이드할 필요가 없다.
};
```

여기에 자바로 같은 구현을 했다.

```
interface GameEventListener
{
    // 인터페이스는 멤버 변수를 포함하지 않는다.
    // 메소드 정의는 여기에, 하지만 아직 구현은 하지 않는다.
```

```

        public bool isListening( );
        public void handleEvent( int event );
    }

class Animal extends NSObject implements GameEventListener
{
    // 멤버 변수, 메소드, 오버라이드 메소드는 여기에 온다.
    private int foo;
    private bool active; // 여기서 꼭 정의해야 한다.

    public int bar( double input1 ) {}
    public bool isListening( ) {}
    public void handleEvent( int event ) {}
}

```

자바 인터페이스 클래스는 **Objective-C**에서 프로토콜 클래스라 부른다. 그리고 **@protocol**을 사용하여 정의한다.

```

@protocol GameEventListener
    // 프로토콜은 멤버 변수를 포함하지 않고 종괄호도 사용하지 않는다.
    // 메소드 정의는 여기에, 하지만 아직 구현은 하지 않는다.
    - (BOOL) isListening;
    - (void) handleEvent: (int) event;
@end

@interface Animal : NSObject <GameEventListener>
{
    // 멤버 변수는 여기에 둔다.
    @private
    int foo;
    BOOL active;
}

    // 메소드와 오버라이드 메소드는 여기에 둔다.
    - (int) bar: (double) input1;
    - (BOOL) isListening;
    - (void) handleEvent: (int) event;
@end

```



C++의 템플릿 클래스나 자바의 제네릭을 경험해 본 프로그래머라면 여기서 혼란스러울 수 있다. Objective-C의 프로토콜 사용 문법이 C++의 템플릿이나 자바의 제네릭 문법과 비슷하기는 하지만 같은 것은 아니다.

앞에서 언급했듯이 함수 구현은 *.m* 파일에 분리된다. C++ 구현 예제는 이런 모양이 될 것이다.

```
bool GameEventListener::isListening()
{
    return active;
}

void Animal::bar( double input1 )
{
    // input1으로 할 일을 한다.
}

void Animal::handleEvent( int event )
{
    // event에 따라 할 일을 한다.
}
```

그리고 Objective-C 구현은 이런 모양이 될 것이다.

```
@implementation Animal

@synthesize foo;

- (void) bar: (double) input1
{
    // input1으로 할 일을 한다.
}

- (BOOL) isListening
{
    return self.active;
}

- (void) handleEvent: (int) event
{
    // event에 따라 할 일을 한다.
}

@end
```

1.3.2 인스턴스화

클래스를 정의했다면 인스턴스를 만들고 싶을 것이다.

C++와 자바에서 인스턴스는 다음과 같은 모양이 될 것이다.

```
Animal* beaver = new Animal();
```

반면에 Objective-C에서는 이렇게 한다.

```
Animal* beaver = [Animal alloc];
```

1.3.3 메시지

`alloc` 호출 주변에 대괄호를 사용한 것은 Objective-C의 재미있는 기능인 메시지 *message*라는 것이다. 그것은 Objective-C 오브젝트의 함수를 호출하는 기본적인 방법이다. 문법이 새롭겠지만 개념은 C++나 자바의 함수 호출과 같다. 앞의 예제에서 `Animal` 클래스의 `new` 함수를 호출한다. `new` 함수는 클래스의 인스턴스를 새로 할당하여 리턴한다.

예상대로 그 오브젝트의 함수를 지금 이렇게 호출할 수 있다.

```
BOOL listening = [beaver isListening];
```

C++에서는 이런 모양을 지닐 것이다.

```
bool listening = beaver -> isListening();
```

자바에서는 이런 모양을 지닐 것이다.

```
boolean listening = beaver.isListening();
```

파라미터 전달도 가능하다. 파라미터는 콜론으로 구분한다.

```
[beaver handleEvent: EVT_EXAMPLE];
```

C++에서는 이런 모양이 된다.

```
beaver -> handleEvent( EVT_EXAMPLE );
```

자바에서 다시 하면 다음과 같이 된다.

```
beaver.handleEvent( EVT_EXAMPLE );
```

메시지에 하나 추가되는 것은 키워드이다. 첫 번째 이후의 각 파라미터 앞에는 키워드가 붙는다. `Animal` 클래스가 찾을 음식의 양과 채소만 허용할지를 지정하는 `findFood`라는 함수를 가졌다면 이를 볼 수 있다.

```
[beaver findFood: 50 vegetablesOnly: true];
```

C++의 같은 함수는 이런 모양이다.

```
beaver -> findFood(50, true);
```

이것은 함수 호출이 더 많은 화면을 차지하도록 하지만, 읽기는 더 편해진다. 함수 정의를 찾아보거나 넘겨줄 값의 성격을 이해하기 위해 IDE의 코드 완성 기능에 의존할 필요가 없기 때문이다.

또한 파라미터가 있어야 할 곳에 다른 대괄호 세트를 넣어서 메시지를 중첩할 수 있다. `Animal` 클래스가 `BOOL isVegetarian` 함수를 가졌다면 이를 볼 수 있다.

```
[beaver findFood: 50 vegetablesOnly: [beaver isVegetarian]];
```

C++에서는 이런 모양이다.

```
beaver -> findFood(50, beaver -> isVegetarian());
```



C++나 자바의 `const` 또는 `factory` 메소드와 같은 것을 Objective-C에서 하려면 `@interface` 부분에서 메소드 이름 앞에 `+`를 사용한다. 이것은 클래스의 인스턴스 없이 클래스의 함수를 호출 가능하게 만든다.

예를 들어, 다음과 같다.

```
@interface Animal
+ (BOOL) canAnimalsTalk;
@end
```

다음의 방법으로 호출할 수도 있다.

```
[Animal canAnimalsTalk];
```

이를 Objective-C에서는 클래스 메소드라 부른다. 클래스의 인스턴스 대신에 클래스 자신이 직접 호출하기 때문이다. 이 방법으로 선언한 함수는 클래스의 멤버 변수를 사용할 수 없다.

1.3.4 멤버 변수

Objective-C는 클래스 멤버 변수에 접근하는 여러 방법을 제공한다. 첫 번째 방법은 메시지를 이용하는 것이다. 함수의 이름이 접근하려는 멤버 변수와 같을 때 사용한다. 이것을 *accessor* 함수라 부르고, C++나 자바의 `get`과 `set` 함수 작성과 비슷하다. 클래스의 `@implementation` 부분에서 `@synthesize` 키워드를 사용하고 `.h` 파일에서 정의할 때 `@property`를 사용하면 함수는 자동으로 만들어진다.

사용 문법은 다음과 같다.

```
int example = [beaver foo]; // int foo의 값을 리턴한다.
```

```
[beaver setFoo:10]; // int foo에 10을 넣는다.
```

자동으로 만들어진 *setter* 메소드의 이름은 `set` 뒤에 변수의 이름이 첫 글자가 대문자로 되어 이어짐을 기억하자. 같은 목적으로 다음처럼 `dot` 표기를 사용할 수도 있다.

```
int example = beaver.foo; // int foo의 값을 리턴한다.
```

```
beaver.foo = 10; // int foo에 10을 넣는다.
```



C++와 자바에서의 `this` 키워드처럼 Objective-C 클래스 메소드에서는 `self` 키워드를 사용한다.

1.3.5 메모리 관리

Objective-C는 C를 포함하기 때문에 `malloc`과 `free`에 관한 모든 규칙이 동일하게 적용된다. 아이폰에는 자바처럼 모든 오브젝트를 관리하는 가비지 컬렉터가

없다. 따라서 할당한 메모리를 적절히 해제하지 않으면 메모리 누수가 일어날 수 있다.

그러나 Objective-C는 베이스 클래스 NSObject 안에 레퍼런스 카운터를 구현한다. 그러므로 NSObject를 상속한 모든 클래스는 같은 레퍼런스 카운트 기능을 내부에 지닌다. 다시 말해 NSObject가 제공하는 `copy`, `new`, `retain`, `release`, `autorelease`, `alloc` 메소드를 사용할 수 있다는 것이다.

NSObject 서브 클래스 인스턴스가 `alloc` 메소드나 `new` 또는 `copy`를 이름 앞에 붙인 함수를 사용하여 만들어지면, 레퍼런스 카운트가 1인 상태로 만들어질 것이다. 이 값을 1 증가시키려면 `retain` 함수를 사용할 수 있고, 1 감소시키려면 `release`와 `autorelease` 함수를 사용할 수 있다. 레퍼런스 값이 0이 되면 오브젝트는 메모리에서 제거될 것이다.

오브젝트에 `retain`을 호출하고 `release`나 `autorelease`를 그 오브젝트에 호출하지 않을 경우, 레퍼런스 카운트가 0이 되지 않아서 메모리 누수가 생길 것이다.

1.3.6 생성자와 소멸자

C++처럼 Objective-C 클래스는 생성자와 소멸자 함수 개념을 지원한다. 그러나 약간 차이가 있다. 오브젝트가 할당될 때 프로그래머는 초기화 함수를 수동으로 호출해야 한다. 기본적으로 초기화 함수는 `init`이라는 이름을 지닌다.

다음 코드는 전형적인 Objective-C이다.

```
Animal* beaver = [[Animal alloc] init];
```

Objective-C 클래스가 메모리에서 제거될 때, 소멸자와 비슷한 `dealloc` 함수가 호출된다. 클래스에서 `dealloc` 함수를 오버라이드하면 슈퍼셋 클래스의 `dealloc` 메소드를 꼭 호출해야 한다. 그렇지 않으면 메모리 누수가 일어날 것이다.

```
-(void) dealloc {
    // 소멸자 코드를 여기에서 수행한다.
    [super dealloc];
}
```

1.3.7 인터페이스 빌더 통합

Objective-C와 인터페이스 빌더는 아이폰 앱 제작에 함께 사용되기 때문에, Objective-C 코드를 인터페이스 빌더 뷰에 연결하기 위해 IBOutlet과 IBAction 매크로를 포함시킨다.

UI 변수 앞에 IBOutlet을 놓고 클래스 메소드 앞에 IBAction을 놓으면, 인터페이스 빌더는 코드가 사용하는 변수와 함수들을 알게 된다.

```
@interface myWindow
{
    IBOutlet UIImageView *backgroundImage;
    IBOutlet UIWindow *window;
    IBOutlet UISwitch*soundToggle;
}
- (IBAction) playgame;
- (IBAction) toggleSound:(id) sender;
- (IBAction) handleEvent:(id) sender forEvent:(UIEvent*) event;
@end
```

컴파일 타임에 IBAction은 void로 바뀌고, IBOutlet은 간단히 제거된다. 이것들은 인터페이스 빌더에서 메소드와 변수들을 보이게 하기 위해 사용될 뿐 런타임 효과는 없다.

1.3.8 C++와 Objective-C 혼합

Objective-C는 C 언어의 슈퍼셋이기 때문에 코드 대부분을 C 언어로 작성할 수 있다. 그리고 Objective-C 컴파일러는 프로젝트에 C++ 또한 사용 가능하다. 같은 파일에서 C, C++, Objective-C 문법을 사용하는 것도 가능하다. C++ 구현을 포함한 파일은 확장자로 .m 대신에 .mm을 사용해야 한다.

1.4 결론

이 장에서는 Objective-C 언어의 기초 이해와 코드를 작성하고 컴파일하여 아이폰에서 실행하는 절차를 보여주었다. 언어와 API의 지식을 완전히 소화하기에 필요한 만큼 많은 튜토리얼을 읽어보길 권한다. 특히 <http://developer.apple.com>의 문서는 매우 유용한데, 애플 개발자 계정으로 접근하면 무료이다.

그러나 게임은 매우 복잡한 애플리케이션이기 때문에 게임 프로그래밍 이면의 이론적 이해 없이는 게임 개발이 매우 큰 도전이 될 것이다. 게임 프로그래밍을 처음 시작한다면, 게임을 만들기 전에 다음 장을 완전히 읽어보는 것이 큰 도움이 될 것이다.