ADC
Apple Developer Connection
**Recommended Title**

# *Learning*

한글판

# Cocoa

O'REILLY®

*Apple Computer, Inc.*

# Learning Cocoa

# 목차

# 이 책을 읽기 전에

우리가 어떤 지식을 습득하려면, 어디서부터 시작할 것인가를 먼저 결정해야 한다. 이것이 결정되어야, 기본 용어, 경계 및 기법의 개념, 사물이 조화를 이루는 방법과 그 같은 방법이 어떻게 가능한지에 대한 개념을 파악할 수 있다. 이 책은 Cocoa 응용 프로그램 개발 방법을 배우고자 하는 이들을 위해 이 같은 출발점을 제공한다.

이 책은 Cocoa 프로그래밍을 간편하게 사용하는 방법을 소개한다. 우리는 이 책을 읽는 사람들이 Cocoa를 즐겁게 공부할 수 있도록 노력했다. 그래서, 이 책을 다 읽고 나면, Cocoa를 사용하여 중요한 응용 프로그램을 보다 능숙하게 개발할 수 있다. 또한, 애플의 개발 환경을 훨씬 명확하게 정의하여, 개발자가 원하는 프로그래밍 환경을 만들 수 있다.

이 책에 수록된 예제를 따라하기 위해 풍부한 프로그래밍 경험이 필요하지는 않지만, C 프로그래밍 언어에 대한 경험은 도움이 될 것이다. 각 사례의 코드는 개발자가 간단하게 입력할 수 있도록 텍스트에 포함되어 있으며, 이 예제들은 다음에 열거되어 있는 사이트들에서 얻을 수 있다. 사용자가 Java 또는 Smalltalk 등 객체 지향 프로그래밍 언어와 친숙하다면, 이 책에서 사용된 Objective-C에 금방 적응할 수 있을 것이다.

이 책의 튜토리얼을 완성하기 위해 Mac OS X에서의 프로그래밍 경험은 필요하지 않으며, 애플이 Mac OS X에 수록한 풍부한 개발자 문서를 읽어보면 된다. `/Developer/Documentation`에서 Mac OS X 시스템 아키텍쳐, 개발자 툴, 릴리즈 등 상세한 정보를 찾아볼 수 있다. 대부분의 Cocoa 프로그래밍 문서는 `/Developer/Documentation/Cocoa`에 있으며, 여기서 클래스, 프로토콜, 함수, 유형, 상수에 대한 내용이 수록된 Cocoa API 참조 문서 및 Objective-C 언어를 사용한 프로그래밍 정보를 얻을 수 있다.

또한, Apple 도움말을 통해 이 도큐멘테이션에 액세스할 수 있다. Help 메뉴에서 Developer Help Center를 선택한 후, Cocoa를 선택하면 된다.

이 책에 수록된 프로그래밍 예제는 Apple 웹 사이트에서 찾아볼 수 있다.

*http://developer.apple.com/techpubs/macosx/Cocoa/CocoaTopics.html*

O' Reilly 사이트는

*http://www.oreilly.com/catalog/learncocoa이다*.

이 책의 주요 목적은 Cocoa 사용 방식을 교육하는 데에 있지만, 프로그래밍에 관심이 있는 이들이 재미있게 Cocoa를 배울 수 있도록 하는데 중점을 두었다.

# *이 책의 구성*

Cocoa 배우기는 3부로 구성되어 있다.

## *제1부, Cocoa 개요*

*Cocoa 개요*는 Cocoa 프레임워크를 소개하고, 응용 프로그램 프레임워크가 제공하는 상위 단계 기능에 관해 설명한다. 또한, Cocoa 프로그래밍에서 사용되는 객체 지향 프로그래밍, Objective-C 언어 및 개발 툴을 간단하게 소개한다.

제1장, *Cocoa 소개*는 Cocoa 역사, Mac OS X 프로그래밍 환경에서의 Cocoa, 그리고 Cocoa API를 구성하는 프레임워크 및 클래스를 소개한다.

제2장, *객체 지향 프로그래밍*은 절차 프로그래밍과 비교해서 OOP 장점을 명확히 규명한다. 또한, 제1장에서 소개된 Cocoa 프레임워크를 효과적으로 사용하기 위해 사용자가 알아야만 하는 용어 및 핵심 개념에 대한 개요를 제공한다.

제3장, *Objective-C 입문*은 Objective-C 프로그래밍 언어의 기본에 관해 설명한다.

제4장, *개발 툴*은 Cocoa 개발에서 사용되는 프로젝트 빌더 및 인터페이스 빌더, 주요 툴에 관해 설명한다. 또한, Mac OS X에서 응용 프로그램의 구축, 디버깅 및 성능 개선에 도움을 줄 수 있는 수 많은 툴과 유틸리티에 관해 설명한다.

## *제2부, 단일 윈도우 응용 프로그램*

*단일 윈도우 응용 프로그램*은 사용자가 Cocoa 프로그래밍의 기본 요소를 파악할 수 있도록 단순한 튜토리얼에서 시작한다. 그리고, 점점 더 복잡해지는 응용 프로그램 사례를 제공한다. 사용자가 1개의 튜토리얼에서 습득한 기법과 개념은 다음에 나오는 더욱 진보된 기법과 개념의 초석이 된다.

제5장, *Hello World*는 기존의 Hello World 응용 프로그램을 Cocoa용으로 생성하기 위해 Mac OS X 개발 환경을 사용하는 방법을 제시하는 간단한 튜토리얼이다.

제6장, *중요한 Cocoa 패러다임*은 여러 개의 작은 튜토리얼을 통해 Cocoa 응용 프로그램 구축에 사용되는 가장 기본적인 디자인 패턴을 소개한다.

제7장, *Currency Converter 튜토리얼*은 단순한 응용 프로그램을 구축하는 방법을 처음부터 끝까지 보여 줌으로써, Cocoa 응용 프로그램 개발의 완전한 워크플로우를 경험할 수 있도록 해준다. 또한, Model-View-Controller라 불리는 일반적인 객체 지향 설계 파라다임을 사용하는 방법을 설명한다.

제8장, *이벤트 처리*는 사용자와 프로그램이 발생시키는 이벤트를 중점적으로 다루며, Cocoa에서 이벤트를 인터셉트, 처리 및 조정하는 방법을 설명한다.

제9장, *데이터 기능성*은 단순한 튜토리얼 응용 프로그램을 사용하여 가로 세로로 데이터를 출력하는 사용자 인터페이스 객체, 즉 테이블 뷰를 소개한다. 이 장의 두 번째 부분에서는 사용자가 데이터를 저장 장치에 저장할 수 있도록 응용 프로그램을 확장하는 방법을 설명한다.

제10장, *Travel Advisor 튜토리얼*은 지금까지 습득한 모든 기법을 단일 응용 프로그램으로 조합하는 대규모 튜토리얼이다. Travel Advisor 는 사용자가 여행하는 다양한 나라들에 관한 여행 관련 정보를 관리하기 위해 사용된 양식 기반 응용 프로그램이다.

## *제3부, 멀티 윈도우 응용 프로그램*

*멀티 윈도우 응용 프로그램*은 Cocoa의 멀티 도큐멘트 아키텍쳐를 사용하여 복잡한 멀티 도큐멘트 기반 응용 프로그램을 구축하는 방법을 제공하는 (주요 개념에 관한 내용이 들어간) 확장된 튜토리얼을 사용한다. 사용자는 To Do라 불리는 응용 프로그램을 디자인하고, 코드화하여, 캘린더의 특정한 날에 접근할 수 있으며, 약속이나 스케쥴을 입력할 수 있다.

제11장, *Cocoa의 멀티플 도큐멘트 아키텍쳐*는 작업 개발자가 멀티 도큐멘트 응용 프로그램 구현을 위해 수행해야 하는 작업 내용을 과감히 단순화시킨 Cocoa의 강력한 아키텍쳐에 관해 설명한다.

제12장, *To Do: 기본 원리*는 Cocoa의 다중 도큐멘트 아키텍쳐를 사용한 To Do 응용 프로그램의 디자인과 초기 생성에 관해 설명한다.

제13장, *To Do: 확장*은 제12장에서 생성된 응용 프로그램에 향상된 기능을 추가하는 과정을 설명한다. 사용자는 알람 및 만료 날짜 재조정등 다양한 To Do 속성을 설정하기 위해 Mac OS X 정보 윈도우를 추가하게 된다.

제14장, *To Do: 마무리 작업*은 최종적으로 기능을 추가하여 To Do 응용 프로그램을 마무리 짓는다. 사용자는 컴파일러를 최적화 옵션을 켜고, 응용 프로그램 및 도큐멘트 아이콘을 추가시킨 뒤, 버전 정보와 응용 프로그램 시그너쳐 등을 포함시킨 빌드 설정값을 구성하여 응용 프로그램을 배치할 준비를 한다.

부록A, *Cocoa에서 드로잉하기*는 Cocoa 및 Cocoa 그래픽(Quartz) API를 사용한 드로잉을 소개한다.

# 규약

다음은 이 책에서 사용되는 인쇄상의 규약이다.

*이탤릭체*는 URL을 나타내거나 새로운 용어를 소개할 때 사용된다.

`Constant width`는 파일명, 데이터 유형, 디렉토리 및 경로명, 함수, 상수, 변수 및 `repeat`같은 흐름 제어 문장, 커맨드 라인의 출력 및 코드 사례를 나타낼 때 사용된다.

**`Constant width 볼드체`**는 사용자 입력을 보여줄 때 사용된다.

# 문의 방법

이 책에 실린 모든 내용은 수 많은 시험과 검증을 통해 작성되었다. 그러나, 기능이 변경되었거나, 일부 오류가 발견될 수 도 있다. 이 책에서잘못된 내용을 발견했거나, 향후 버전에 대한 아이디어가 있으면, 다음 주소로 연락하길 바란다.

O' Reilly & Associates, Inc.
101 Morris Street.
Sebastopol, CA 95472.
(800) 998-9938 (미국 또는 캐나다).
(707) 829-0515 (국제/현지).
(707) 829-0104 (fax)

또한, 이메일로 의견을 보낼 수 있다. 메일링 리스트에 가입하거나 카다로그를 요청하려면, 다음 주소로 이메일을 전송한다.

*info@oreilly.com*

이 책에 대한 기술적인 의문 사항이나 의견은 다음 주소로 이메일을 전송한다.

*bookquestions@oreilly.com*

이 책의 예제, 정오표, 개정판의 계획등은 다음 사이트에서 얻을 수 있다.

*http://www.oreilly.com/catalog/learncocoa*

이 책에 관한 상세한 내용은 O' Reilly 웹 사이트를 참조한다.

*http://www.oreilly.com*

# I

# *Cocoa 개요*

# 1

# *Cocoa 소개*

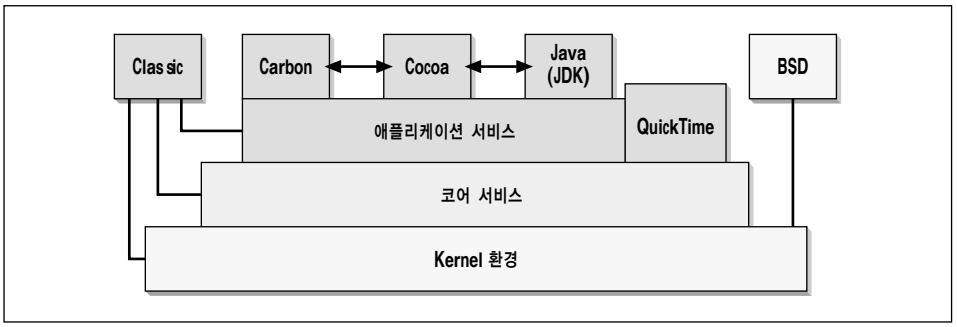
Cocoa는 광범위한 소프트웨어 컴포넌트 라이브러리로 Mac OS X에서 운용되는 응용 프로그램을 구축하기 위해 사용된다. 또한, 개발자가 있는 그대로 사용하거나 개발자의 필요에 따라 재구성할 수 있는 응용 프로그램 구축 블록의 집합이라 볼 수 있다. 이 장은 블록 구축에 대한 상위 단계의 개요를 제공하며, 블록이 어떻게 구성되어 있는지와 어떤 기능을 제공하는지를 설명한다.

새롭게 배우려는 사용자는 Cocoa를 익히는 데 얼마간의 시간이 소요될 수 있지만, 일단 사용방법을 익히고 나면, 응용 프로그램 개발 작업이 그야말로 간단하고, 흥미로워질 것이다. Cocoa 프레임워크는 "무료"로 기본적인 응용 프로그램 기능을 제공하기 때문에 시간과 에너지를 들이지 않고, 새로운 기술 구축 방법을 터득할 수 있다.

Cocoa는 실제로 매킨토시와 거의 동일한 역사를 가지고 있는데, 이는 1987년에 도입된 NeXTSTEP(OpenStep으로 명칭이 바뀜)에 기반하기 때문이다. OpenStep은 수 많은 릴리즈를 거듭할수록 진화하였으며, 수 많은 기업들이 개발과 배치 환경으로 OpenStep을 채택하였고, 각종 매체에서 열화와 같은 반응을 얻었다. 또한, 시장에서 수년동안 선두자리를 지키고 있는 디자인을 기반으로 확고한 기술력을 지속적으로 유지하고 있으며, 끊임없이 향상을 거듭하고 있다.

Cocoa를 사용하여 다양한 응용 프로그램을 개발하는 방법을 잘 알기 위해 Cocoa의 기능과 프레임워크를 살펴보면 다음과 같다. <그림 1-1>은 Mac OS X 시스템 소프트웨어의 일반적인 구조를 나타낸다.

Cocoa는 Mac OS X을 위한 주요 응용 프로그램 환경중의 하나이다. Cocoa의 객체 지향 API는 자바와 Objective-C로 구성된 응용 프로그램을 용이하게 개발할 수 있도록 한다. Cocoa는 공유 객체 라이브러리의 통합된 집합체, 즉 *프레임워크(frameworks)*이며, 런타임 시스템 및 개발 환경으로써 다음 사항을 수행한다.




<그림 1-1> Cocoa는 Mac OS X 시스템 소프트웨어의 일부이다

- 시스템 리소스로의 액세스를 조정하고, 프로그램이 다른 어드레스 공간을 침범하지 않도록 함으로써 핵심 운영 제체 내부 작동으로부터 프로그램 분리.
- 일반적으로 프로그램 제작에 필요한 모든 인프라 제공.
- 객체 지향성의 장점을 프로그램 개발에 적용.

Cocoa는 프로그램과 핵심 운영 시스템을 용이하게 조정 및 운용하는 객체 계층이라 할 수 있다. 따라서, 프로그램의 안정성, 성능 및 신뢰성은 이 같은 특성을 지닌 기본적인 핵심 운영 시스템에 따라 결정된다.

# Cocoa 기능

이 섹션은 Cocoa를 사용하여 응용 프로그램을 개발할 때 이용할 수 있는 상위 단계 기능의 일부를 설명한다.

## 이미지 및 프린트 모델

Mac OS X용 이미지 및 프린팅 모델은 Adobe사의 PDF(Portable Document Format)이다. 이전 Mac OS 버전과는 달리 화면에 출력된 내용을 보거나 인쇄하는것에 동일한 메카니즘을 사용하였다. 개발자는 화면과 PostScript 기반 장비로 출력을 보내기 위해 더 이상 코드를 작성할 필요가 없어졌다. ColorSync 및 QuickDraw GX 타이포그래피 등 Apple사의 최고의 그래픽 기술도 핵심 그래픽에 포함되었다.

## 멀티미디어

Cocoa 응용 프로그램은 가장 강력한 기술인 QuickTime과 OpenGL을 사용할 수 있다.

## QuickTime

Mac OS X은 최신 버전의 QuickTime과 함께 출시되었다. QuickTime은 비디오, 사운드, 애니메이션, 그래픽, 텍스트, 음악, 360도 가상 현실을 조작, 향상 및 저장하기 위해 사용하는 강력한 멀티미디어 기술이다.

QuickTime 스트리밍은 사용자가 산업 표준 프로토콜인 RTP(Real-Time Transport Protocol) 및 RTSP(Real-Time Streaming Protocol)를 사용하여 생방송 및 VOD(Video-on-demand) 무비를 볼 수 있도록 한다. 사용자는 스트리밍 생방송, 녹화된 영화 또는 이들의 혼합된 내용을 볼 수 있다. 브로드캐스트는 유니캐스트(일대일 전송) 또는 멀티캐스트(일대다 전송)일 수 있다.

## OpenGL

Mac OS X은 3D 그래픽을 위한 시스템 API 및 라이브러리로서 애플의 최적화된 구현체인 OpenGL을 포함한다. OpenGL은 포터블 3D 그래픽 응용 프로그램 개발을 위한 산업 표준이다. OpenGL은 오늘날 가장 광범위하게 채택된 그래픽 API 표준중의 하나이므로, OpenGL로 작성된 코드는 다른 플랫폼으로의 이식성이 뛰어나다. OpenGL은 게임, 애니메이션, CAD/CAM, 의료 영상 및 2, 3D 영상을 시각화하기 위해 풍부하고, 강력한 프레임워크를 필요로 하는 기타 응용 프로그램을 위해 특별히 설계되었다. OpenGL의 Mac OS X 버전은 고품질 그래픽 이미지를 뛰어난 성능으로 생성한다.

OpenGL은 텍스처 매핑, 숨겨진 표면 제거, 알파 블렌딩(투명성), 앤티앨리어싱, 픽셀 조작, 뷰잉 및 모델링 변형, 대기 효과(안개, 스모그 및 아지랭이) 및 기타 특수 효과가 포함된 강력하고 광범위한 이미지 기능을 제공한다. 각 OpenGL 명령어는 드로잉 액션을 지시하고, 특수 효과를 발생시킨다. 개발자는 효과를 반복하기 위해 이 같은 명령어 리스트를 만들 수 있다. OpenGL은 각 운영 체제의 윈도우 특성에 독립적이지만, 특수 “글루(glue)” 루틴은 운영 체제의 윈도우 환경에서 OpenGL이 사용될 수 있도록 구현되었다.

## 인터넷

Apple은 Mac OS X에 인터넷을 위한 최고의 기술력을 제공하고자 한다. 이 개발 플랫폼은 인터넷 기반 메일, 메시징, 디렉토리 및 보안 서비스를 위한 API를 갖추고 있다. Cocoa는 HTML 렌더링 기능도 내장되어있다.

## 로컬라이제이션 및 국제화

잘 설계된 로컬라이제이션 아키텍쳐 및 유니코드 지원 체계가 프레임워크에 구축되었기 때문에 사용자는 Cocoa 응용 프로그램을 간단하게 로컬라이즈할 수 있다. 이 아키텍쳐에서 사용자 인터페이스 요소는 실행 코드와는 분리되어 저장된다.

따라서, 다양한 지역에 적합한 단일 코드 기반을 가질 수 있다. 개발자는 하나의 응용 프로그램에 다중 로컬라이제이션을 번들함으로써 응용 프로그램의 전반적인 크기를 크게 줄일 수 있다. 로컬라이제이션 번들은 기존 응용 프로그램에서 쉽게 추가 또는 삭제할 수 있으므로 새로운 로컬라이제이션은 업데이트를 통해 배포가 가능하게 된다.

Cocoa는 기본 문자 세트로서 유니코드 2.0을 사용하므로 응용 프로그램은 간단하게 전 세계의 모든 언어를 처리할 수 있다. 유니코드의 보급은 수 많은 문자 인코딩의 문제를 감소시킨다. Cocoa는 비유니코드 텍스트 처리를 지원하기 위해 유니코드와 현재 사용되고 있는 다양한 문자 세트를 번역할 수 있는 API를 제공한다.

Apple의 개발 환경은 여러 방식으로 로컬라이제이션을 지원한다. 즉, 어떤 파일이(어떤 언어로) 로컬라이즈되어야 하는지를 쉽게 식별할 수 있는 방안을 제공한다. 또한, 특정 환경에 맞춰 설계된 사용자 인터페이스를 생성할 수 있도록 한다.

## 텍스트 및 폰트

Cocoa는 고성능을 요하는 텍스트 집약 응용 프로그램에 그대로 적용할 수 있는 강력한 텍스트 서비스를 제공한다. 이 같은 서비스는 가상 메모리 공간만큼 텍스트 버퍼를 지원할 수 있으며, 커닝, 합자(Ligature), 탭 포맷 및 룰러를 제공한다. 텍스트 시스템은 임베디드 그래픽과 기타 인라인 어태치먼트도 지원한다.

Cocoa는 Type1 PostScript, Type3 PostScript, Type42 PostScript 및 TrueType(TrueType GX의 타이포그래피 능력 포함)을 포함한 다양한 폰트 형식을 지원한다. Mac OS X의 목표는 사용자가 원하는 폰트 형식으로 간편하게 작업할 수 있도록 개방형 폰트 아키텍쳐를 구축하는 것이다.

## 컴포넌트 기술

개발 환경으로서의 Cocoa의 장점은 재사용이 가능한 컴포넌트를 조합하여 신속하고 용이하게 프로그램을 개발할 수 있다는 것이다. 개발자는 적합한 프로그래밍 툴과 간단한 작업으로 다른 개발자가 사용할 수 있도록 일괄적인 처리 및 배포가 가능한 Cocoa 컴포넌트를 구축할 수 있다. 이러한 컴포넌트 기술로 응용 프로그램 뿐만 아니라 다른 것들도 개발할 수 있다. 개발자는 Cocoa 및 Apple 개발 툴로 다음 사항을 개발할 수 있다 .

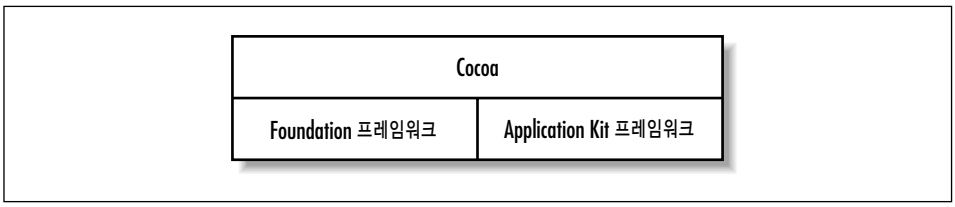
- 개발자가 프레임워크 API 기반 코드를 이용하여 프로그램을 개발하기 위해 사용할 수 있는 프레임워크.
- 프로그램이 동적으로 로딩할 수 있는 실행 가능 코드 및 관련 리소스가 포함된 번들.

- Mac OS X 개발 툴을 사용하여 사용자 인터페이스로 드래그 앤 드롭할 수 있는 커스텀 사용자 인터페이스 객체를 내장한 팔레트.

개발자는 Cocoa의 컴포넌트 아키텍쳐로 응용 프로그램에 대한 확장 및 플러그인을 간단하게 개발 및 배포할 수 있다.

# *Cocoa 프레임워크*

Cocoa는 <그림 1-2>에서 보여진 바와 같이 Foundation(`Foundation.framework`) 및 Application Kit(`AppKit.framework`)같은 2개의 객체 지향 프레임워크를 제공한다.




*<그림 1-2> Cocoa 프레임워크*

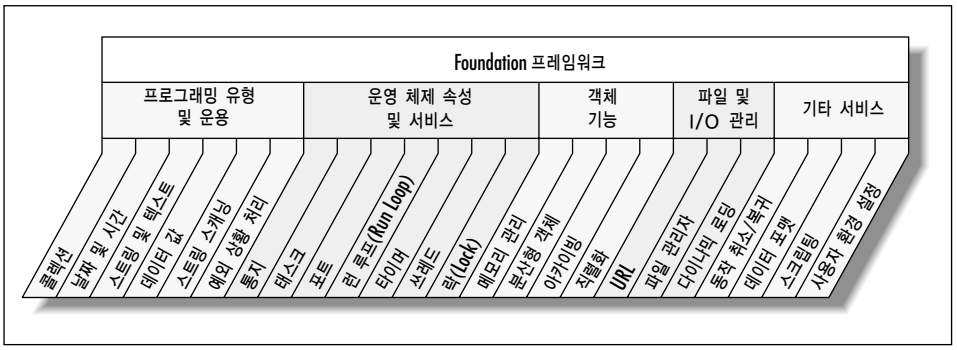
Foundation의 클래스는 Cocoa의 기초를 이루는 객체 및 기능성을 제공한다. Application Kit의 클래스는 윈도우, 버튼 같은 사용자 인터페이스에서 볼 수 있는 객체 및 동작을 제공하며, 마우스 클릭 및 키 누르기 같은 기능을 처리한다. Application Kit는 Foundation에 직접적으로 의존한다.

## *Foundation*

Foundation 프레임워크는 Objective-C 클래스의 기저 계층을 정의한다. 또한, 유용한 원시 객체 클래스를 제공할 뿐만 아니라 Objective-C 언어(제3장, *Objective-C 입문*에서 상세하게 설명)에서 다뤄지지 않는 기능성을 규정한 여러 패러다임을 제공한다. Foundation 프레임워크는 다음과 같은 목표를 염두에 두고 설계되었다.

- 기본적인 유틸리티 클래스 제공.
- 메모리 관리와 같은 패러다임에 일관된 규정을 제공하여 소프트웨어 개발을 용이하게 함.
- 유니코드 스트링, 객체 지속성 및 파일 관리 지원.

Foundation 프레임워크는 루트 객체 클래스, 스트링 및 바이트 행렬 같은 기본적인 데이터 유형을 나타내는 클래스, 다른 객체를 저장하는 콜렉션(Collection) 클래스, 날짜등 시스템 정보를 나타내는 클래스, 통신 포트를 나타내는 클래스등으로 구성된다. <그림 1-3>은 Foundation이 제공하는 기능을 보여준다.




<그림 1-3> Foundation 프레임워크 기능

Foundation 프레임워크는 일반적인 상황에서의 혼동을 방지하고, 클래스 계층간의 일관된 레벨을 지원하기 위해 여러 패러다임을 제공한다. 이는 객체 소유권(객체를 처리할 책임이 어디에 있는지 여부)를 결정하는 등의 일부 표준 규약과, 콜렉션을 수치화하는 추상화 클래스를 통해 가능해진다. 이 패러다임은 API에서 특정적이고 예외적인 경우를 감소시키고, 사용자가 다양한 객체의 동일한 메카니즘을 재활용하여 더욱 효과적으로 코드화 할 수 있도록 한다.

다음 섹션은 Foundation 프레임워크에서 중요한 사항에 관해 설명한다.

## 프로그래밍 유형 및 운용

Cocoa의 Foundation 프레임워크는 스트링, 배열, 딕셔네리 및 숫자를 포함해 수 많은 기본적인 데이터 유형을 제공한다. Foundation은 여러 클래스를 제공하는데, 이 같은 클래스의 용도는 다른 객체의 콜렉션 클래스를 확보하는 데 있다. 이 책은 이 같은 데이터 유형에 관해 상세한 정보를 제공한다.

Foundation은 파싱 및 예외 상태 처리 클래스 뿐만 아니라 바이트 교환 기능도 제공한다.

***콜렉션(Collection)*** 콜렉션은 논리적인 방식으로 데이터를 구성 및 검색할 수 있도록 한다. 이 같은 콜렉션 클래스는 수 많은 메카니즘을 통해 내용을 저장하고 검색할 수 있기 때문에 상당히 유용하다.

- 배열(Array)은 제로 베이스 인덱싱을 통해 순서대로 객체를 저장 및 검색한다.
- 딕셔네리(Dictionary)는 키-값 쌍을 이용하여 신속하게 객체를 저장 및 검색한다. 예를 들면, "TextColor" 키는 빨간색을 나타내는 색상 객체와 연결될 수 있다.

- 세트(Set)는 서로 다른 요소들의 순서 없는 집합체이다.
- 셀 수 있는 세트는(동일한) 듀플리케이트 요소를 수용할 수 있는 세트이다. 이 같은 듀플리케이트는 카운터로 추적한다. 멤버 검사의 속도가 중요하다면, 세트를 사용한다.

콜렉션 객체는 데이터를 저장하기 위한 중요한 방법을 제시한다. 파일 시스템에 콜렉션 객체를 저장(또는 보관)할 경우, 구성 객체도 저장된다.

콜렉션 클래스는 수정 가능 버전 및 수정 불가능 버전을 제공한다. 수정 가능한 버전의 클래스는 콜렉션 객체가 생성된 이후에도 객체를 프로그램적으로 추가 및 삭제할 수 있다. 수정 불가능 버전은 이 같은 기능을 수행하지 않는다.

모든 Cocoa 콜렉션은 동적으로 증가할 수 있다.

콜렉션 클래스는 제6장, *중요한 Cocoa 패러다임*에서 상세하게 다뤄진다.

***날짜 및 시간*** 날짜 및 시간 클래스는 시간적 차이 산출, 원하는 포맷으로 날짜 및 시간 출력, 장소에 따른 날짜 및 시간 조정등의 방법을 제공한다.

***스트링 및 텍스트*** 스트링 객체는 문자열을 나타낸다. 스프레드 시트의 입력 라벨에서 워드프로세서 도큐멘트까지 응용 프로그램의 대부분의 텍스트는 스트링을 사용한다. 특히, NSStrings(또는, 스트링 객체)은 친숙한 C 프로그래밍 언어 유형인 `char *`를 대신한다. 스트링 객체는 ASCII 문자 세트가 제공하는 좁은 범위의 문자가 아닌 유니코드 문자를 수용한다. 따라서, 중국어, 아라비아어 및 다양한 언어를 표현할 수 있다.

스트링 클래스는 수정 가능 및 수정 불가능 스트링을 생성하고, 서브스트링 검색, 스트링 비교 및 연결등 스트링을 운용하는 API를 제공한다.

***데이터 및 값*** 데이터 객체는 바이트 버퍼를 위한 객체 지향 랩퍼이다. 또한, 데이터 객체는 단순히 할당된 버퍼(즉, 임베드 포인터가 없는 데이터)가 Foundation 객체의 동작 기능을 수행할 수 있도록 한다. 일반적으로, 데이터 객체는 데이터 저장용으로 사용되며, 응용 프로그램간의 복사 및 이동이 가능한 데이터를 포함한 응용 프로그램에서 유용하다.

데이터 객체는 어떤 크기의 데이터도 랩(Wrap)할 수 있다. 데이터 크기가 메모리 페이지보다 클 경우, 가상 메모리 관리를 사용한다. 또한, 데이터 객체는 데이터가 어떻게 할당되었는지와 상관없이 기존 데이터를 랩할 수 있다. 객체는 데이터 정보(유형 등의)를 포함하지 않는다. 데이터 사용 방법의 결정 여부는 클라이언트에 달려있다. 유형이 있는 데이터의 경우엔 값 객체를 사용한다.

값 객체는 단일 데이터 아이템을 위한 단순한 컨테이너이다. 또한, 포인터, 스트럭쳐 및 객체 주소 뿐만 아니라 정수, 부동소숫점 및 문자 등 스칼라 타입을 포함한다. 콜렉션은 객체를 구성 요소로 사용하기 때문에, 이런 데이터 유형을 콜렉션에서 사용하기 위해서는 값 객체를 이용하면 된다.

***스트링 스캐닝*** 스트링 스캐닝은 문자열을 해석하여 숫자와 스트링 값으로 변환해 준다. 스캐너는 스트링에 대한 요구가 들어올 때마다 그 스트링의 처음부터 끝까지 순서대로 문자를 스캔한다.

오류가 발생하여 특정 부분을 재검색하거나 특정 갯수의 문자를 건너뛰기 위해 스캔 도중 스캔 위치를 변경할 수 있다. 또한, 특정 문자를 건너뛰거나, 대소문자를 구분 또는 무시하도록 스캐너를 설정할 수도 있다.

***예외처리*** 예외 상황은 프로그램이 비정상적으로 실행되는 특수 상황을 의미한다. 각각의 응용 프로그램은 다양한 이유로 예외가 발생할 수 있다. 예를 들어, 어떤 응용 프로그램은 쓰기 방지된 디렉토리에 파일을 저장하는 것을 예외로 해석할 수 있다. 이 예외 상황이라는 것은 어떤 의미에서 보면 오류가 발생한 것과 같다. Foundation은 예외를 처리하고, 예외에 관한 정보를 얻을 수 있는 기능을 제공한다.

## *운영 체제 요소 및 서비스*

Foundation은 락(Lock), 쓰레드, 타이머와 같은 핵심 운영 시스템 기능에 액세스하기 위한 클래스를 제공한다. 이 서비스는 응용 프로그램이 실행되는 강력한 환경을 제공하기 위해 상호 운용된다.

***통지*** 통지 관련 클래스는 응용 프로그램내에서 발생하는 변경에 대한 통지를 송출하기 위해 시스템을 구현한다. 어느 객체나 통지를 지정 및 발송할 수 있으며, 다른 어느 객체나 자신을 그 통지의 관찰자로 등록할 수 있다. 통지에 관한 상세한 내용은 제8장, *이벤트 처리*를 참조한다.

***쓰레드*** 쓰레드(Threads)는 실행 쓰레드를 제어한다. 독자적인 실행 쓰레드에서 메소드를 운용하고자 하거나 현재의 쓰레드를 종료하거나 지연시키고자 할 경우 쓰레드를 사용한다.

쓰레드는 실행 가능한 유니트이다. 태스크는 1개 이상의 쓰레드로 구성된다. 각각의 쓰레드는 독자적인 실행 스택을 내장하고 있으며, 독립적으로 입력/출력 기능을 한다. 모든 쓰레드는 가상 메모리 주소 공간 및 태스크의 통신권을 공유한다. 쓰레드가 구동되면, 기존 쓰레드에서 분리되며, 새로운 쓰레드는 독립적으로 운용한다. 즉, 기존 쓰레드는 새로운 쓰레드의 상태를 파악하지 못한다. 같은 태스크에 포함된 쓰레드라도, 각각의 쓰레드는 멀티 프로세서 시스템에서는 다른 CPU에서 수행될 수 있다.

***락*** 락(Locks)은 동일한 응용 프로그램내에서 실행되는 다양한 쓰레드의 운용을 조정하기 위해 사용된다. 또한, 항상 단 한개의 쓰레드만이 보호된 리소스에 접근하도록 함으로써, 응용 프로그램의 전역 데이터에 대한 접근을 조절하거나 코드의 크리티컬 섹션을 보호하기 위해 락을 사용한다.

***태스크*** 한 프로그램은 태스크(Tasks)를 이용하여 서브프로세스로서 다른 프로그램을 실행시키고, 그 프로그램의 실행을 감시할 수 있다. 태스크는 개별적인 실행 가능 요소를 생성한다. 또한, 태스크는 자신을 생성한 프로세스와 메모리 공간을 공유하지 않는다는 점에서 쓰레드와 차이가 있다.

***포트*** 포트(Ports)는 주로 다른 쓰레드나 태스크에 상주하고 있는 다른 포트로(부터)의 통신 채널을 나타낸다. 분산형 객체 시스템은 포트를 사용하여 쓰레드나 태스크 간에 메세지를 전송한다.

***런 루프*** 런 루프(Run loops)는 입력 소스를 관리하는 객체를 위한 프로그램 인터페이스이다. 또한, 윈도우 시스템,포트, 타이머 및 기타 연결물에서 마우스 및 키보드 이벤트 같은 소스에 대한 입력을 처리한다.

응용 프로그램은 Foundation으로 인해 런 루프를 생성하거나 관리할 필요가 없다. 각 쓰레드에는 자동적으로 생성된 런 루프가 있다. 각 프로세스는 디폴트 쓰레드로 시작하여, 디폴트 런 루프를 가진다. Application Kit는 메인 쓰레드의 런 루프 운용을 처리한다. 그러나, 다른 쓰레드의 런 루프는 수동으로 처리해야 한다.

***타이머*** 타이머를 사용하면 일정 간격으로 객체에 메시지를 전송할 수 있다. 예를 들면, 일정한 시간이 경과되면 타이머를 발생시켜 윈도우에 업데이트 명령을 할 수 있다. 타이머는 제13장, *To Do: 확장*에서 상세하게 다뤄진다.

## *객체 기능*

Foundation은 객체를 생성하고 파괴하는 것 뿐만 아니라, 저장하고 분산 환경에서 공유하는 등의 객체 관리 기능을 제공한다.

***메모리 관리*** 메모리 관리는 객체가 더 이상 필요하지 않을 때 적절하게 객체를 삭제한다. 이 메카니즘은 객체 소유권 방식에 따라 결정되며, 해제가 표시된 객체를 자동적으로 추적하여, 현재의 런 루프가 종료되면 객체를 삭제한다. 성공적인 Cocoa 응용 프로그램을 개발하는 데에는 메모리 관리에 대한 이해가 중요하다. 상세한 내용은 제6장에서 설명한다.

***분산형 객체*** 분산형 객체(Distributed objects)는 프로세스간 메시징 솔루션이다. 이 메카니즘은 Cocoa 응용 프로그램이 생성한 1개 이상의 객체가 다른 응용 프로그램에서 사용될 수 있도록 한다.

***직렬화 및 아카이빙*** 직렬기(Serializer)는 객체가 갖고 있는 데이터를 아키텍쳐 독립적인 포맷으로 표현함으로써 애플리케이션 간의 데이터 공유를 가능하게 한다. 코더는 한 단계 더 나아가 데이터와 함께 클래스 정보를 저장함으로써 아카이빙과 분산을 가능하게 한다.

아카이빙(Archiving)은 파일에 인코딩된 객체와 데이터를 저장한다. 분산은 다양한 프로세스, 쓰레드 또는 장치간에 인코딩된 객체 데이터를 전송하는 것을 의미한다.

아카이빙에 관한 내용은 제9장, *데이터 기능성*에서 상세하게 설명한다.

### *파일 및 입출력(I/O) 관리*

파일 시스템 및 입출력 관리 기능은 URL 처리, 파일 관리, 코드 및 로컬라이즈된 리소스의 동적인 로딩 등으로 구성된다.

***URL 처리*** URL과 그 URL이 참조하는 리소스는 Foundation을 통해 액세스할 수 있다. Foundation은 RFC 1808 및 1738에 명시된 대로 URL을 인식한다.

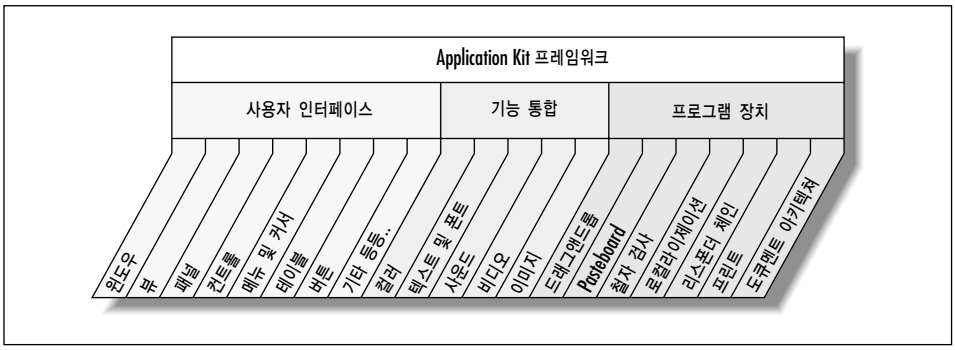
파일을 참조하는데 URL이 사용될 수 있으며, 실제로도 이 방법이 애용된다. 파일에서 데이터를 읽거나 쓸 수 있는 Cocoa 객체는 파일 참조 방법으로 경로명 뿐만 아니라 URL도 받아들인다.

***기타 서비스*** Foundation 프레임워크는 사용자 환경설정, 명령의 취소와 복귀, 데이터 포맷을 관리하는 능력을 제공한다. Cocoa 응용 프로그램은 AppleScript 명령어에 반응할 수 있다.

## *Application Kit*

Application Kit는 이벤트에 반응하는 그래픽 사용자 인터페이스 구현에 필요한 모든 객체(윈도우,패널, 버튼, 메뉴, 스크롤러 및 텍스트 필드)를 포함한 프레임워크이다. Application Kit는 화면에 효과적으로 드로잉을 하고, 하드웨어 장치 및 스크린 버퍼와 통신하며, 드로잉하기 전에 화면 영역을 제거하고, 보이지 않은 부분을 표시하지 않음으로써, 개발자를 위해 세세한 부분까지 처리를 해준다. <그림 1-4>는 Application Kit의 클래스 계층을 보여준다.

사용자는 Application Kit의 1백개가 넘는 클래스가 어렵게 느껴질 수 있다. 그러나, 대부분의 Application Kit 클래스는 간접적으로 사용되는 지원 클래스이다. 다음은 Application Kit의 구조적인 중요 사항을 간략하게 설명한다.




<그림 1-4> Application Kit 프레임워크 기능

## 사용자 인터페이스

사용자 인터페이스는 사용자가 응용 프로그램과 상호작용하는 수단이다. 개발자는 Application Kit이 제공하는 윈도우, 대화 상자, 메뉴, 팝업리스트 및 기타 조절 장치를 생성하고 관리할 수 있다. 이 책에 수록된 튜토리얼에서 수 많은 사용자 인터페이스 요소를 사용하게 될 것이다.

***윈도우*** 윈도우의 2가지 주요 기능은 뷰가 배치될 수 있는 영역을 제공하고, 마우스와 키보드 액션으로 사용자가 발생시킨 이벤트를 받아 들여 해당 뷰로 배포하는 것이다.

일반적으로, Interface Builder를 사용하여 윈도우를 설정한다. Interface Builder는 수 많은 비쥬얼 및 동작 속성을 설정하고, 뷰를 배치한다. 윈도우를 통해 수행하는 프로그램 작업은 주로 화면 불러오기 및 닫기, 윈도우 타이틀과 같은 동적 속성 변경, 끝내기, 확대하기, 크기 조절과 같은 윈도우 액션을 감시하는 것이다.

***뷰*** 뷰는 윈도우에 표시되는 모든 객체에 대한 추상적인 표현방식이라 할 수 있다. 뷰는 드로잉, 프린팅 및 이벤트 처리를 위한 구조를 제공한다. 뷰는 윈도우내에서 계층화된 서브뷰의 형태로 정렬된다.

***패널*** 패널(Panel)은 윈도우의 일종으로 일시적, 포괄적 또는 긴급한 정보를 출력하기 위해 사용한다. 예를 들어, 오류 메시지를 표시하거나 예외적인 상황에 응답이 필요한 경우 윈도우보다는 패널이 적합하다. Application Kit는 도큐멘트를 저장, 열기 및 인쇄하기 위해 사용되는 Save, Open 및 Print 등 일반적인 패널을 구현한다. 이 같은 패널을 사용하여 사용자에게 일관된 룩 앤드 필을 제공한다.

***컨트롤*** Cocoa는 응용 프로그램의 일부를 조절하기 위해 그래픽적으로 조정할 수 있는 버튼, 슬라이더, 브라우저 등 사용자 인터페이스 객체를 제공한다. 각각의 용도는 개발자에게 달려있다.

일반적으로, Interface Builder를 사용하여 인터페이스를 배치한다. 다음 리스트는 Application Kit가 제공하는 몇 개의 도구에 관한 설명이다.

- **메뉴 및 커서**의 생김새와 클래스는 응용 프로그램이 사용자에게 출력하는 메뉴 및 커서의 생김새와 동작을 정의한다.
- **테이블**은 행과 열 양식으로 데이터를 표시한다. 테이블은 각 행이 각각의 레코드에 해당하고, 각 열이 레코드 속성을 나타내는 데이터베이스 레코드를 표시하는 데 이상적이다. 사용자는 개별 셀을 편집하고, 세로열을 재배열할 수 있다.
- **버튼**은 클릭했을 때 메시지를 다른 객체로 전송해 주는 사용자 인터페이스 객체이다. Push 버튼 같은 단일 버튼이나 스위치 또는 Radio 버튼같은 관련 버튼 그룹을 만들 수 있다.

Cocoa는 이러한 도구뿐만 아니라 시트, 슬라이더, 드라우어 및 툴 바를 제공한다.

## 기능 통합

Application Kit는 컬러, 폰트 및 출력을 통합 및 관리하는 방법(이 같은 기능을 위해 대화 상자까지도)을 제공한다.

***텍스트 및 폰트*** 텍스트 필드는 단순한 편집 텍스트 필드를 구현한다. 텍스트 뷰는 더 큰 텍스트를 위해 좀 더 포괄적인 편집 기능을 제공한다. 폰트 패널은 사용자가 텍스트 뷰에 출력된 폰트를 원하는 대로 조절할 수 있도록 한다.

***이미지*** 이미지는 그래픽 데이터를 캡슐화한다. 따라서, 디스크에 저장되었거나 화면에 출력된 이미지에 간단하고, 효과적으로 액세스할 수 있다. Cocoa는 다양한 포맷의 이미지를 처리하기 때문에 개발자가 굳이 생성 방법을 알 필요는 없다.

***컬러*** 컬러 및 컬러 뷰를 나타내는 다양한 클래스가 컬러를 지원한다. 여기에는 맞춤형 컬러를 포함해 풍부한 컬러 포맷과 표현 방식이 있다. Cocoa의 컬러 클래스는 다양한 컬러 공간을 자동적으로 처리한다. 컬러 지원 클래스는 사용자가 컬러를 선택 및 적용할 수 있도록 패널과 뷰를 정의하고 나타내준다. 예를 들면, 사용자는 컬러 패널에서 컬러 웰로 컬러를 드래그할 수 있으며, 개발자는 표준 컬러 패널을 확장할 수도 있다.

## 장치

Application Kit은 사용자가 Mac OS X의 응용 프로그램에 기대하는 모든 기능을 활용할 수 있는 강력한 응용 프로그램을 개발자가 만들 수 있도록 다른 수 많은 장치를 제공한다.

***도큐멘트 아키텍처*** 워드 프로세싱 및 스프레드시트 응용 프로그램과 같은 도큐멘트 기반 응용 프로그램은 오늘날의 일반적인 응용 프로그램 유형이다. 워드 프로세서 및 스프레드시트 응용 프로그램은 가장 잘 알려진 도큐멘트 기반 응용 프로그램이다.

다양한 Application Kit 클래스는 상기 절에 수록된 기능을 구현하기 위해 개발자가 수행해야 하는 작업을 단순화한 도큐멘트 기반 응용 프로그램용 아키텍처를 제공한다. 이 같은 클래스는 응용 프로그램의 도큐멘트 작성, 저장, 실행 및 관리 작업을 분배하고, 조정한다. 도큐멘트 아키텍처는 제11장, *Cocoa의 멀티플 도큐멘트 아키텍처*에서 상세하게 설명한다.

***리스폰더 체인*** 리스폰더 체인(Responder chain)은 사용자 이벤트에 응답하는 객체의 순서를 나열한 리스트이다. 사용자가 마우스를 클릭하거나, 키를 누를 경우, 이벤트가 발생하고, Application Kit는 "응답"할 수 있는 객체를 순서대로 검색한다. 응용 프로그램의 컨트롤 흐름을 파악하려면, 제8장을 참조한다.

***다른 응용 프로그램과의 데이터 공유*** Pasteboard는 응용 프로그램에서 복사된 데이터의 저장장소이며, 이 같은 데이터가 어떤 응용 프로그램에서든 사용될 수 있도록 한다. 또한, 오려두기-베껴두기-붙이기 및 드래그 앤 드롭 기능을 구현한다.

***드래그 앤 드롭*** 최소한의 프로그래밍만으로도 객체를 어느 곳에서나 드래그 앤 드롭할 수 있도록 한다. Application Kit는 마우스를 트래킹하고, 끌어온 이미지를 출력하는 것과 같은 모든 세부사항을 처리한다.

***프린트*** 프린트 클래스는 응용 프로그램이 윈도우 및 뷰에 출력하는 정보를 인쇄할 수 있도록 한다. 또한, 뷰를 PDF 방식으로 표현할 수도 있다.

***파일 시스템 액세스하기*** 파일 랩퍼(wrapper)는 디스크의 파일이나 디렉토리에 해당하며, 다른 응용 프로그램으로 출력, 변경하거나 다른 응용 프로그램으로 전송할 수 있도록 파일의 내용을 메모리에 담아 둔다. 또한, 파일을 끌어오거나 첨부 도큐멘트로 파일을 표현하기 위한 아이콘을 제공한다. 파일 관리자는 파일 및 디렉토리에 액세스하고, 이들 수를 세기 위해 사용된다. Open 및 Save 패널은 파일 시스템에 편리하고, 친숙한 사용자 인터페이스를 제공한다.

***철자 검사*** 철자 서버는 개발자가 철자 검사 서비스를 설정하고, 다른 응용 프로그램에 제공할 수 있도록 한다. 응용 프로그램을 철자 검사 서비스에 연결하려면 철자 검사기를 사용한다. Cocoa의 텍스트 시스템은 자동적으로 철자 검사 기능을 통합한다.

***로컬라이제이션*** 응용 프로그램을 1개 이상의 국가에서 사용할 경우, 리소스를 해당 언어, 국가 또는 문화권에 맞춰 "로컬라이즈"해야 할 수도 있다. 예를 들면, 응용 프로그램에는 문자열, 아이콘, 사용자 인터페이스 정의 또는 컨텍스트 도움말에 대한 개별적인 일본어, 영어, 불어 및 독일어 버전이 필요할 수도 있다. 특정 언어에 적합한 리소스는 번들 디렉토리(`.lproj` 확장자를 가진 디렉토리)에 함께 저장된다. 그러면 사용자 환경 설정에 알맞는 로컬라이즈된 리소스가 응용 프로그램이 구동될 때 동적으로 로딩된다.

# 2

# 객체 지향 프로그래밍

Mac OS 개발자가 Cocoa 프로그램 개발을 시작할 때 경험할 가장 큰 변화는 개발 도구가 아니라 객체 지향 프로그래밍(OOP)을 최대한 활용하기 위해 필요한 사고방식의 전환이다. 개발자는 절차 및 데이터 측면을 고려하기보다는 객체, 즉 데이터를 동작시키는 절차와 데이터를 내장하고 있는 개별적인 프로그램 유니트 측면을 고려해야 한다.

개발자가 객체 지향 프로그래밍의 기본 개념을 파악하지 않으면, Cocoa를 개발할 수 없다. 이 같은 프로그램 접근법을 처음 접한 개발자는 객체 지향 프로그래밍이 낯설게 느껴지지만, 약간만 배우고 나면, 공통적으로 "맞아, 이렇게 하는구나"라고 이해하게 될 것이다. 이 장은 Objective-C의 관점에서 객체 지향 프로그래밍의 개요를 제공한다. OOP 및 Objective-C에 관한 상세한 내용은 `/Developer/Documentation/Cocoa`의 *Inside Cocoa: Object-Oriented Programming and the Objective-C Language*를 참조한다.

객체와 프로그램하는 방법을 배우려면, 처음에는 약간의 노력을 필요로 하지만, 얼마 지나지 않아 객체 지향 프로그램이 자연스럽고, 우아하고 강력하게 느껴지기 시작할 것이다. 응용 프로그램은 Cocoa 프레임워크의 풍부한 기능성으로 인해 간단하게 개발할 수 있으며, "무료"로 무수한 응용 프로그램 기능을 이용할 수 있다. 특히 Cocoa 객체를 이용한 "프로그래밍"은 개발자를 수 많은 반복적인 코딩 작업에서 벗어나게 하여 생산성을 증가시킨다. 따라서, 창조적인 작업을 수행할 수 있는 더 많은 시간을 확보할 수 있다.

# 객체 지향 프로그래밍의 장점

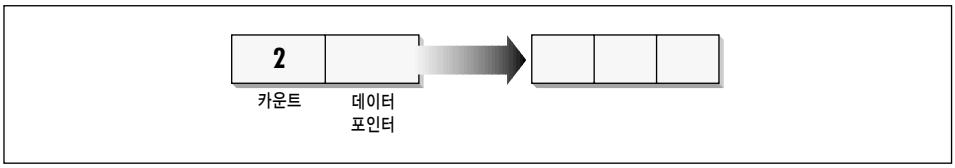
객체 지향성은 소프트웨어 측면에서 산업 혁명에 해당한다고 할 수 있다. 공장들은 처음부터 모든 제품을 만들어 내는 아니라 부품으로 제품을 조립한다. 이와 마찬가지로, 객체 지향성은 프로그래머가 *객체*라 불리는 소프트웨어 컴포넌트를 재사용하여 복잡한 소프트웨어를 구축할 수 있도록 한다. 특히, 객체는 다음과 같은 여러 장점을 가지고 있다.

- **신뢰도 향상.** 객체 지향성은 복잡한 소프트웨어 프로젝트를 작고, 독립적인 모듈화된 객체로 세분함으로써,소프트웨어 프로젝트의 일부에 가해지는 변경이 소프트웨어의 다른 부분에 역으로 영향을 미치지 않음을 보장해 준다. 이 같은 작은 각각의 객체는 충분히 검증된 코드 모듈이므로, 소프트웨어의 전반적인 신뢰도는 증가한다.
- **유지보수 용이성.** 객체가 모듈화되어, 작아졌기 때문에(프로젝트의 전반적인 코드 크기 면에서) 코드내 버그를 쉽게 발견할 수 있다. 개발자는 응용 프로그램의 다른 부분에 장애를 일으키지 않고, 객체 구현을 변경할 수 있다.
- **재사용으로 생산성 향상.** 객체 지향성의 주요 장점은 재사용을 할 수 있다는 것이다. 1개의 객체는 여러 응용 프로그램으로 통합될 수 있다. 개발자는 기존의 클래스에서 기능을 상속하고 그 클래스의 동작을 변경하거나 확장하는 새로운 코드를 추가하여 새롭고, 특화된 객체를 생성할 수 있다. 이 기법은 새로운 클래스가 개발자의 응용 프로그램에 적합한 동작이나 로직만을 구현하기 때문에 코딩을 감소시키고, 신뢰도를 향상시킨다. 기본적인 기능은 Cocoa 프레임워크 객체에서 얻어진다.

객체 지향 프로그래밍은 크고, 복잡한 프로그램에서 장점이 발휘된다. 그러나, 이같은 장점은 상당히 간단한 사례를 통해서도 증명될 수 있다.

응용 프로그램은 순차적 프로그래밍 기법으로 데이터를 직접 조작한다. <그림 2-1>은 이 기법을 통한 문제점을 보여주며, 카운트 변수 및 데이터 포인터를 구성하는 데이터 구조를 제공한다. 응용 프로그램이 데이터를 직접 조정할 경우엔, 불일치성을 제공할 수 있다. 즉, 데이터에 항목을 추가했을 때, 카운트는 항목이 증가되었음을 망각한다. 카운트 변수는 실제적으로 3개의 데이터 요소가 있음에도 불구하고, 2개의 데이터 요소가 있다고 알려준다. 따라서, 구조가 일관성이 없으며, 신뢰도가 낮다.

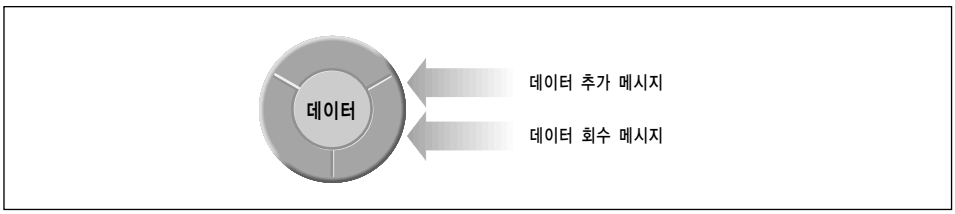
순차적 프로그램 기법의 또 다른 문제점은 응용 프로그램의 모든 부분이 데이터 구조에 관해 알고 있어야 한다는 것이다. 데이터 요소가 정적으로 할당된 배열에서 동적으로 할당된 링크 리스트로 변경되면, 그 리스트의 요소에 접근, 추가 또는 삭제하는 응용 프로그램의 모든 부분에 영향을 준다.




<그림 2-1> 순차적 프로그램에서 데이터 조작

순차적 프로그래밍 기법에서는 응용 프로그램이 커질수록 프로시저와 데이터간의 상호작용 구조는 더욱 복잡해진다. 데이터 구조의 단순한 변경도 수 많은 절차와 코드, 더 나아가서 많은 소스 파일에 영향을 준다. 따라서, 응용 프로그램을 유지하고, 개선시켜야 하는 개발자들에게 악몽이 될 수 밖에 없다. 또한, 한 함수이 의존하고 있는 데이터를 다른 함수이 부주의하게 변경할 경우, 발견하기 힘든 심각한 버그를 발생시킨다.

객체 지향 프로그래밍 패러다임을 사용하면 응용 프로그램이 직접 데이터 구조를 조작하지 않는다. 대신, 그 작업을 특정 객체에 일임한다. 응용 프로그램이 직접 데이터에 액세스하지 않기 때문에 불일치성을 제공하지 않는다. <그림 2-2>는 OOP의 이 같은 기능을 보여준다.



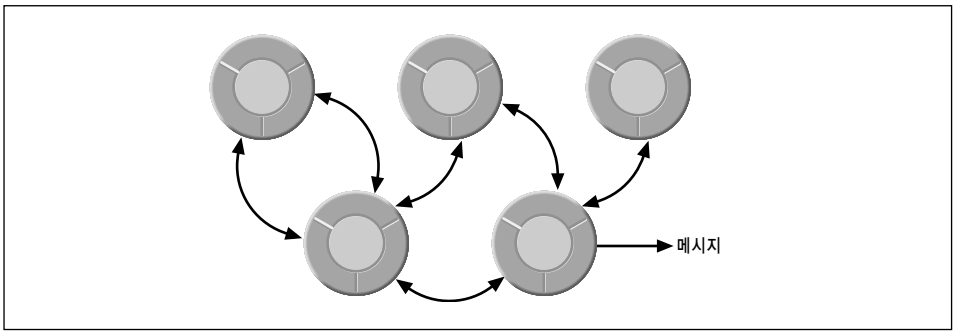

<그림 2-2> 객체 지향 프로그램의 데이터 조정

응용 프로그램의 다른 부분에 영향을 주지 않으면서 객체의 구현을 변경할 수 있다. 예를 들면, 데이터 저장 방식은 성능을 최적화하기 위해 변경될 수 있다. 객체가 동일한 메시지에 응답하면, 응용 프로그램의 다른 부분은 그 객체의 내부 구현에 영향을 받지 않는다.

## 객체의 프로그래밍

객체 지향 프로그래밍은 데이터와 함수를 구성하는 방법 이상을 제공한다. 이는 소프트웨어 엔지니어로 하여금 우리가 우리 주변의 사물을 다루는 방식과 비슷한(통상의 프로그램보다 더 비슷한) 모델을 사용하여 복잡한 프로그램에 대한 해결책을 계획하고 구축할 수 있도록 한다. 프로그램 구조의 객체 지향 모델은 역할과 관계를 명확하게 하여 문제 해결을 단순화한다.

개발자는 객체 지향 프로그램을 명확한 동작 및 특성이 있는 객체의 네트워크로 간주할 수 있다. 객체는 <그림 2-3>에서 보여진 바와 같이 메시지를 통해 상호작용한다.




<그림 2-3> 객체 지향 응용 프로그램의 객체 네트워크

네트워크상의 수 많은 객체는 다양한 역할을 수행한다. 일부는 사용자 인터페이스의 그래픽 요소에 해당한다. 응용 프로그램의 윈도우는 버튼, 메뉴 또는 텍스트 디스플레이처럼 개별적인 객체로 나타난다.

또한, 응용 프로그램은 각각의 객체에 다양한 책임 영역을 부여하면서 인터페이스에서 직접 확인할 수 없는 기능을 객체에 할당한다. 객체가 사용자 인터페이스 객체 및 산술 객체간의 상호작용을 조정하면서 데이터 출력 및 전송을 관리하는 동안, 다른 객체는 특정 산술 작업을 수행할 수 있다.

객체가 정의되면, 프로그램 개발은 그 객체들이 서로 통신할 수 있는 연결을 만들어 주는 것이 주된 작업이다. 이 작업은 개발자의 조각들과 Cocoa 프레임워크가 제공하는 조각들을 맞춰주는 것이다. 객체 지향 프로그램을 작성하는 작업의 많은 부분은 단순히 시스템이 생성한 메시지에 응답하는 방법을 구현하는 것이다.

# 기본적인 객체 지향 개념

객체 지향 프로그래밍의 본질을 파악하려면 몇 개의 개념만을 이해하면 된다. 이 섹션은 Objective-C 구현 사항 및 주요 개념을 제공한다.

## 객체

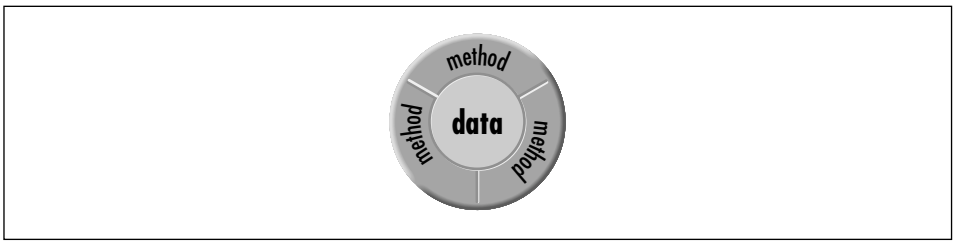
인스턴스 변수 객체는 데이터를 운용하는 절차와 그 데이터를 결합한 개별적인 프로그램 단위이다. 객체의 데이터는 *인스턴스 변수*에 있다. 객체의 인스턴스 변수에 있는 데이터에 영향을 주거나 이 데이터를 사용하는 함수를 *메소드(method)*라고 한다.

프로그램의 객체는 물리적 세계의 사물처럼 두드러진 특성과 동작을 가지고 있다. 프로그램 객체는 종종 실제 사물을 기반으로 모델링된다.

예를 들면, 버튼 같은 객체는 스테레오 장비 및 전화 같은 컨트롤 장치의 버튼과 같은 역할을 수행한다. 버튼 객체는 "실제" 버튼을 본따 화면상에 출력되고, 사용자의 액션(action)에 따라 응답하기 위해 데이터 및 코드를 내장한다.

## 캡슐화

프로시저가 코드를 분류하는 것처럼 객체는 코드 및 데이터를 분류한다. 이는 데이터를 조작하는 프로시저로 그 데이터를 효과적으로 포위하면서 데이터를 캡슐화한다. <그림 2-4>는 객체 데이터 및 메소드의 관계를 보여준다.




*<그림 2-4> 객체*

이 그림은 젤리 도넛이나 타이어처럼 보이지만, 객체의 본질적인 특성인 데이터 캡슐화를 나타내고 있다. 다른 객체나 외부 코드는 캡슐화된 데이터에 직접 액세스할 수 없으며, 데이터를 요구하는 객체에 메시지를 전송해야 한다.

일반적으로, 객체는 *블랙 박스*로 간주되는데, 이는 프로그램이 객체 변수에 직접 액세스할 수 없다는 것을 의미한다. 실제로, 프로그램은 객체가 기능을 수행하기 위해 어떤 변수를 가지고 있는지 알 필요가 없다. 대신, 프로그램은 메소드를 통해서만 객체에 액세스한다. 메소드는 어떤 측면에서 보면 객체의 인스턴스 변수를 보호할 뿐만 아니라 이에 대한 액세스를 조정하면서 데이터를 포위하고 있다.

객체는 객체 지향 응용 프로그램의 기본적인 구축 블록이다. 각 객체는 장애가 발생한 영역에서 각자의 역할을 수행하면서 프로그램이 필요로 하는 기능의 특정 영역을 캡슐화한다. 객체의 메소드는 이 같은 기능에 대한 인터페이스를 제공한다. 예를 들면, 데이터베이스 레코드를 나타내는 객체는 데이터를 저장하고, 그 데이터에 액세스하기 위한 명확한 방식을 제공한다.

객체 지향 프로그램은 이러한 모듈화 방식을 이용하여 특정 데이터 및 특정 작업을 위한 개별적인 객체로 분리될 수 있다. 프로그램 개발 팀은 각 팀원이 맡은 특정 기능을 위한 가장 효과적인 방법으로 데이터 구조와 코드를 구현하는 한편 개별 객체에 대한 인터페이스를 합의함으로써 팀원간의 책임 영역을 간단하게 구분해낼 수 있다.

## 클래스

응용 프로그램에 네트워크된 객체의 일부는 다른 유형일 수도 이고, 같은 유형일 수도 있다. 동일한 유형의 객체는 동일한 클래스에 속해있다. *클래스*는 자신의 인스턴스, 즉 객체를 생성하는 프로그램 엔티티이다. 클래스는 인스턴스의 구조 및 인터페이스를 정의하고, 동작을 지정한다.

새로운 종류의 객체가 필요한 경우, 새로운 클래스를 정의한다. 클래스 정의는 객체 유형을 정의하는 것과 같다. 클래스는 그 클래스에 속하는 모든 객체가 가질 데이터 구조와, 그 객체가 메시지 응답에 사용할 메소드를 지정한다. 하나의 클래스 정의로 원하는 수 만큼의 객체를 생성할 수 있다. 이런 면에서 클래스는 특정한 종류의 객체를 제작하는 공장과도 같다.

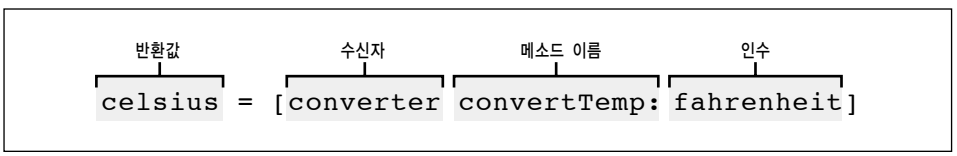
코드 라인 수 측면에서 보면, 객체 지향 프로그램은 대부분 클래스 정의로 이루어져 있다. 프로그램이 작업을 수행하기 위해 사용하는 객체는 런타임시 클래스 정의로부터 생성된다.

그러나, 클래스는 객체를 제작하는 "공장" 이상의 기능을 제공한다. 즉, 클래스는 객체처럼 메소드가 주어질 수도 있고, (일부 객체 지향 언어의 경우) 변수를 가질 수도 있다. 클래스 변수는 인스턴스 변수와는 달리 새로운 클래스의 인스턴스가 생성될 때 마다 생성되지는 않는다. 수 많은 인스턴스가 존재한다하더라도 전체 클래스를 위해 단 1개의 변수만이 있을 뿐이다. 일반적으로, 클래스 메소드는 다양한 옵션을 통해 인스턴스를 생성하기 위해 사용된다.

## 메소드 및 메시지

메소드는 객체를 위해 클래스가 구현하는 프로시저이다.(클래스 메소드의 경우, 특정 인스턴스에 국한되지 않는 기능을 제공하기 위해 구현된다) 메소드는 Public 또는 Private 메소드가 있다. Public 메소드는 클래스 헤더 파일에 선언되는 반면에 Private 메소드는 클래스의 구현 파일에만 나타난다.

*메시지*는 객체가 메소드를 발생시킬 수 있도록 하는 메카니즘이다. 메시지는 객체가 기능을 수행하거나 값을 리턴하도록 요청한다. Objective-C에서 메시지 표현방식은 <그림 2-5>에서 보여진 바와 같이 대괄호를 사용한다.




*<그림 2-5> Objective-C 메시지 표현방식*

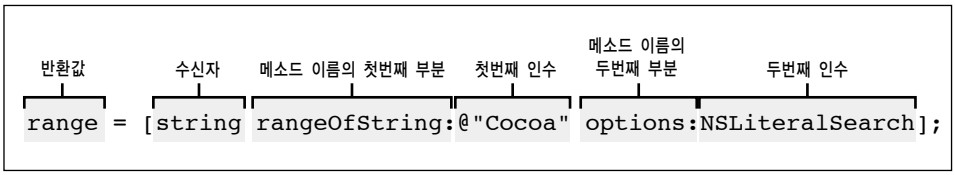
<그림 2-5>에서 `converter`는 메시지를 수신하는 객체인 *수신자(receiver)*의 역할을 한다. 수신자의 오른쪽이 메시지이다. 메시지는 메소드 이름과 메소드가 필요로 하는 인수로 구성된다.

converter가 수신한 메시지는 화씨에서 섭씨로 온도를 전환하여, 그 값을 전송하라는 내용을 보여준다. 이 사례를 보면, converter가 전환 결과를 전송하고, 결과는 변수 이름인 celcius에 저장된다.

Objective-C의 모든 메시지 인수는 라벨로 식별한다. 인수는 콜론으로 끝나는 키워드 다음에 오며, 메소드 명칭의 일부이다. 키워드당 1개의 인수를 허용한다. 메소드가 NSString의rangeOfString:options:와 같이 1개 이상의 인수를 가질 경우, 명칭은 세분되어 인수를 수용한다.

```
range = [string rangeOfString:@"Cocoa" options:NSLiteralSearch];
```

<그림 2-6>은 분석한 내용을 보여준다.




*<그림 2-6> 다중 인수를 사용한 Objective-C 메시지 표현방식*

종종, 메시지는 메시지의 송신자에게 값을 리턴한다. 반환값(returned value)은 해당 유형의 변수가 수신해야 한다. 위의 예에서 변수 range는 NSRange 유형이어야 한다. 값을 돌려보낸 메시지는 중첩될 수 있다. 한 메시지 표현 방식을 다른 표현 방식에 포함시킴으로써, 개발자는 반환값을 위해 변수를 선언할 필요 없이 그 값을 인수나 리시버로 사용할 수 있다.

다음의 코드는 2개의 다른 메시지를 중첩하고 있으며, 각 메시지는 인수로 사용되는 값을 리턴한다. 맨 안쪽에 있는 메시지를 먼저 분석하고, 다음 메시지를 분석한다. 그리고, 3번째 메시지를 전송하고, 값을 돌려보낸 뒤 newString:에 저장한다.

```
newString = [stringOne stringByAppendingString:
        [NSString substringFromRange:[stringTwo rangeOfString:
        @"Cocoa" options:NSAnchoredSearch]]];
```

## *다형성 및 동적 바인딩*

메시지의 용도는 메소드 호출이지만, 메시지는 함수 호출과는 같지 않다. 객체는 자신에게 정의되었거나 자신이 상속한 메소드만을 "인지"한다. 메소드가 동일한 명칭을 가지고 있다 하더라도 다른 객체의 메소드와 혼동하지 않는다.

각각의 객체는 명칭으로 고유의 심볼을 인식하는 프로그램 영역인 *이름영역*을 가진 개별적인 단위이다. C 함수내 지역 변수가 프로그램의 다른 부분과 분리되어 있는 것과 마찬가지로 객체의 변수 및 메소드도 분리되어 있다.

따라서, 2개의 다른 객체가 메소드에 대한 동일한 이름을 가지고 있다면, 2개의 객체는 동일한 메시지를 수신하지만, 다른 방법으로 각각 응답한다. 1개의 메시지가 다른 리시버에서 다른 동작을 발생시킬 수 있는 것을 *다형성(polymorphism)*이라 지칭한다.

다형성이 응용 프로그램 개발자에게 제공하는 잇점은 상당하다. 코드의 단순함을 유지하면서 프로그램의 유연성을 개선시킬 수 있다. 다형성을 사용하면, 코드를 작성할 때 객체에 대해 알지 못하더라도 다양한 객체에 영향을 줄 수 있는 코드를 작성할 수 있다. <예제 2-1>과 <예제 2-2>는 다양한 유형의 도형 객체를 디스플레이하는 일반적인 루틴을 구현하는 2가지 가능한 방법을 대조한 것이다. 첫 번째 사례는 다형성을 지원하지 않는 C에서 쓰여진 것이다.

*<예제 2-1> 다형성을 사용하지 않은 디스플레이 함수*

```
void displayShape(Shape ob)
{
    // Figure out what kind of shape we have.
    switch (ob->type) {
        case SQUARE:
            // Display a square.
            displaySquare(ob);
            break;
        case TRIANGLE:
            // Display a triangle.
            displayTriangle(ob);
            break;
        default:
            printf("Unknown Shape");
    }
}
```

<예제 2-1>은 코드에서 각 Shape 유형에 특수한 경우가 있다는 것을 보여준다. 새로운 Shape 객체를 프로그램에 추가하려면 디스플레이 과정을 변경해야 한다. 이 사례를 다형성을 지원하는 Objective-C에서 쓰여진 <예제 2-2>와 비교해본다.

*<예제 2-2> 다형성을 지원하는 디스플레이 함수*

```
- (void) displayShape:(Shape *)ob
{
    // Send the display message to the shape object.
    [ob display];
}
```

<예제 2-2>에서 보여진 바와 같이, 다형성을 사용하면 **displayShape:** 루틴은 어떤 종류의 객체를 처리해야 할지를 알 필요가 없다. 개발자는 Objective-C(및 기타 객체 지향 언어)에서 디스플레이 메시지를 디스플레이 메소드를 구현하는 어떤 객체에든 전송할 수 있으며, 객체는 독자적인 방법으로 메시지를 처리한다.

이 같은 사례는 또한 다형성에서 상속(상속)의 역할을 강조한다. 서브클래스는 특화된 동작을 획득하기 위해 슈퍼클래스와 동일한 명칭의 메소드(즉, 메소드를 오버라이드한다)를 구현한다.

*동적 바인딩(Dynamic binding)*은 다형성보다 훨씬 유용하다. 이는 메시지를 수신할 객체와 수신될 메시지가 실행중에 결정될 수 있다는 것을 의미한다. 동적 바인딩은 베껴두기 또는 붙이기 같은 사용자 명령어 하나가 많은 수의 사용자 인터페이스 객체에 적용될 수 있는 그래픽한 사용자 구동 환경에서 특히 중요하다.

런타임 프로세스는 동적 바인딩을 사용하여 메시지의 수신자에 적합한 메소드 구현 방식을 찾는다. 그리고 나서 이 구현을 호출하여, 수신자의 데이터 구조를 전달한다. 이 메카니즘은 사용자의 선택이나 액션에 응답하는 프로그램을 쉽게 만들 수 있도록 한다.

예를 들면, 수신자와 메소드 이름과 같은 메시지 표현은 사용자 액션에 따라 값이 결정되는 변수가 될 수 있다. 현재의 선택을 제어하는 어떤 객체에도 오려두기, 베껴두기나 붙이기 같은 메뉴 명령어를 간단한 메시지 표현만으로 전달할 수 있다. 동적 바인딩은 응용 프로그램을 구축할 때 고려하지 않았던 새로운 객체까지도 처리할 수 있도록 한다.

다형화 및 동적 바인딩은 동적 유형 및 내관과 같은 2개의 기능에 따라 결정된다. Objective-C 언어에서는 객체를 `id`라는 데이터 유형으로 포괄적으로 식별할 수 있다. 이 유형은 객체와 그 데이터 구조(즉 인스턴스 변수)에 대한 포인터를 정의하는데, 그 데이터 구조에는, 루트 클래스 NSObject로부터 상속한 유형인, 그 객체의 클래스에 대한 포인터를 포함하고 있다. 이를 이용하면 코드에서 객체를 특정 클래스 유형으로 정의할 필요가 없다. 객체 클래스는 내관을 통해 런타임에서 결정될 수 있다.

*내관(Introspection)*은 객체(`id`로 정의된 유형을 비롯하여)가 자신의 클래스 유형과 기타 특성을 런타임에 드러낼 수 있다는 것을 의미한다. 객체의 상속 관계, 응답할 수 있는 메소드 및 그에 부합한 프로토콜 등을 확인할 수 있는 몇몇 내관 메소드가 제공된다.

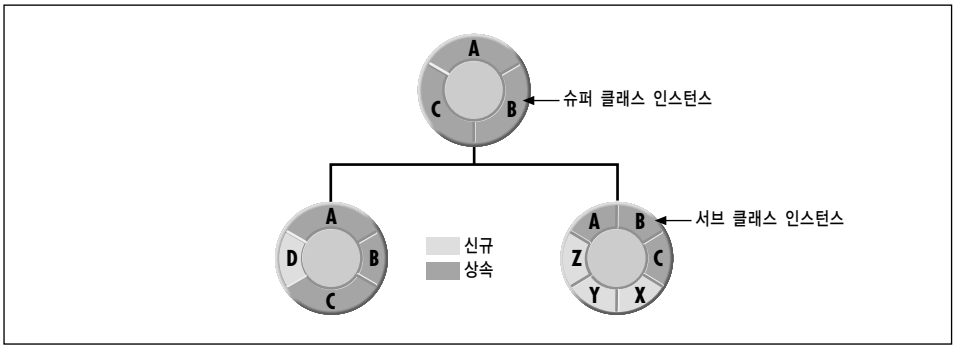
## 상속

상속은 객체 지향 프로그램의 가장 강력한 기능 중 하나이다. 사람들이 조상으로부터 특성을 물려받는 것과 마찬가지로 클래스의 인스턴스는 "조상" 클래스에서 속성과 동작을 물려받는다. 객체의 인스턴스 변수 및 메소드는 이를 생성한 클래스 뿐만 아니라 그 클래스가 물려받은 기존의 모든 클래스에서 파생된다.

Objective-C 클래스는 상속 기능으로 인해 모든 메소드 및 변수를 정의할 필요가 없어진다. 개발자가 원하는 모든 기능을 수행하는 클래스가 있지만, 추가로 기능이 필요할 경우, 기존 클래스에서 물려받은 새로운 클래스를 정의할 수 있다.

새로운 클래스는 원래 클래스의 *서브클래스(subclass)*로 규정한다. 이 클래스는 *슈퍼클래스(superclass)*에서 기능을 상속받은 클래스를 의미한다.

새로운 클래스를 생성하는 작업은 특수화와 관련된다. 새로운 클래스는 슈퍼클래스의 모든 동작을 물려받기 때문에, 개발자가 원하는 동작을 생성하기 위해 클래스를 재구현할 필요가 없다. 서브클래스는 새로운 메소드와 그 메소드에 필요한 변수를 추가함으로써 물려받은 동작을 확장한다. 슈퍼클래스를 위해 규정했거나 슈퍼클래스에서 물려받은 모든 메소드 및 변수는 서브클래스가 전수한다. 서브클래스는 슈퍼클래스의 구현과는 다른 동작을 획득하기 위해 메소드를 재구현하면서 물려받은 메소드를 오버라이드하여 슈퍼클래스 동작을 변경할 수 있다. <그림 2-7>을 참조한다.

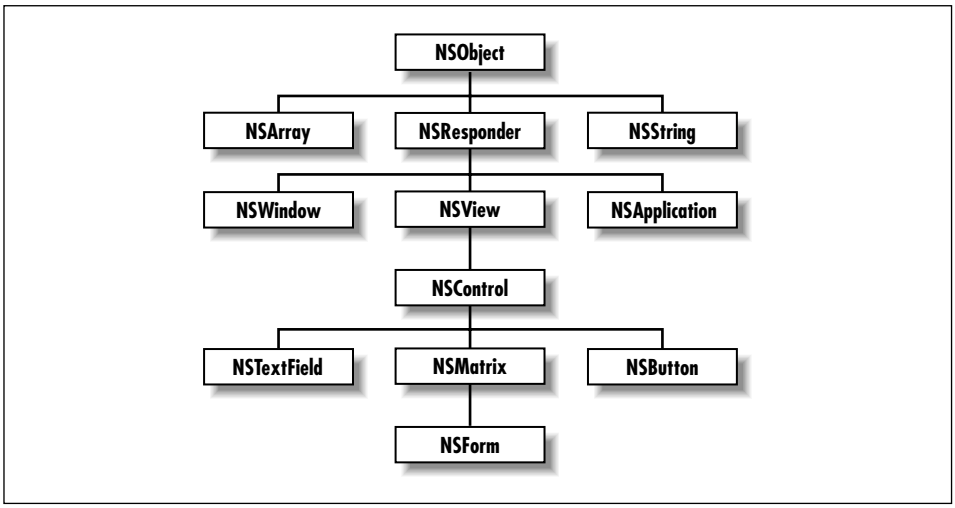



*<그림2-7> 상속*

클래스에는 여러 개의 서브클래스가 존재할 수 있지만, 슈퍼클래스는 단 1개가 존재한다. 이는 클래스가 슈퍼클래스가 없는 상층부의 1개의 클래스, 즉 루트클래스가 있는 분기 계층에 정렬되어 있기 때문이다. <그림 2-8>은 Cocoa 클래스 계층의 대표적인 부분을 나타낸다.

NSObject는 대부분의 Objective-C 클래스 계층에서 루트 클래스이다. 다른 클래스는 메시지를 처리하고 다른 객체와 상호작용하며, 객체로 동작할 수 있는 기본적인 기능을 NSObject에서 상속받는다. 루트 클래스는 객체의 생성, 초기화, 삭제, 내관 및 저장을 위한 기본틀을 생성한다.

앞서 언급했듯이, 슈퍼클래스는 개발자가 필요로 하는 거의 모든 동작을 제공하기 때문에 그 슈퍼클래스로부터 서브클래스를 생성하는 일이 빈번하다. 그러나, 기존의 클래스에는 없는 자신만의 고유한 기능을 가진 서브클래스를 만들 수도 있다. 객체의 기본적인 동작이외에 특수한 동작을 상속할 필요가 없는 새로운 클래스를 정의하려면, NSObject 클래스의 서브클래스를 생성하면 된다. NSObject의 서브클래스는 일반적인 객체 특성으로 인해 Cocoa 응용 프로그램에서 매우 광범위하게 사용되며, 주로 특정 응용 프로그램에 필요한 기능이나 산술 기능을 수행한다.




<그림 2-8> Objective-C의 루트 클래스

상속은 여러 클래스의 공통적인 기능을 하나의 클래서 정의로 손쉽게 묶을 수 있도록 한다. 예를 들면, 화면에서 드로잉 기능을 하는 모든 객체(버튼, 슬라이더, 텍스트 디스플레이 또는 점 그래프 이미지를 드로잉하는 여부)는 어떤 윈도우의 어느 위치에 드로잉하는지를 알고 있어야 한다. 또한, 드로잉을 위한 적절한 시기와 사용자 액션에 응답해야 할 시기를 파악해야 한다. 모든 세부사항을 처리하는 코드는 단일 클래스 정의(Application Kit의 NSView 클래스)의 일부이다. 버튼, 슬라이더 또는 텍스트 디스플레이를 드로잉하는 특정 작업은 서브클래스로 일임할 수 있다.

이 같은 기능의 통합은 응용 프로그램에 사용되어야 하는 코드의 구성을 단순화하며, 복잡한 작업을 수행하는 객체를 용이하게 설정할 수 있도록 한다. 각 서브 클래스는 슈퍼클래스와 다르게 동작하는 부문만을 구현하며, 이미 수행된 작업을 재구현할 필요는 없다.

더욱이, 계층적인 설계는 더욱 견고한 코드를 보장한다. NSView와 같이 널리 쓰이고 테스트된 클래스를 기반으로 하여 구축됨으로써 서브클래스는 검증된 기능을 상속하게 된다. 서브클래스를 위한 새로운 코드는 고유한 동작의 구현에 국한되므로, 그 코드를 테스트하고 디버깅하는 작업은 더욱 수월하다.

어떤 클래스는 새로운 서브클래스에 대한 슈퍼클래스가 될 수 있다. 따라서, Cocoa가 제공하거나 개발자가 생성하거나 타 벤더가 지원하는 모든 클래스는 상속을 통해 간단하게 확장될 수 있다.

# 3

# *Objective-C 입문*

Objective-C 언어는 객체 지향 프로그래밍을 가능케 하는 특수 문법 및 런타임 확장을 바탕으로 한 ANSI C의 수퍼세트이다. Objective-C 문법은 단순하지만 강력한 기법이다. 개발자는 표준 C를 Objective-C 코드와 결합할 수 있다. 프로그래머가 C 및 객체 지향 프로그래밍 기법과 친숙하다면, 가정에서 수일내에 Objective-C에 간단하게 적응할 수 있다.

이 장은 2개의 섹션으로 구성된다. 첫번째 섹션은 추가된 기본적인 언어에 대한 개요를 설명하고, 두번째 섹션은 자주 사용되는 언어 기능을 설명한다.

상세한 내용은 `/Developer/Documentation/Cocoa`의 *Inside Cocoa: Object-Oriented Programming and the Objective-C Language*를 참조한다.

## *언어의 개요*

Objective-C는 소수의 구문을 C언어에 추가하고, 런타임 시스템과 효과적으로 상호작용하기 위해 사용되는 규정을 정의한다

### *메시지*

메시지는 대괄호를 사용하여 표현한다.

```
[receiver message]
```

수신자는 다음 중 하나이다.

- 변수 또는 객체를 계산한 표현(`self` 변수 포함)

- 클래스명(클래스 객체를 나타냄)
- super(메소드 구현에 대한 대안적인 검색을 나타냄)

메시지는 메소드 이름과 그 메소드에 전달되는 인수들을 나열한 것이다.

## 기정의된 유형

Objective-C에서 사용하는 주요 유형은 헤더 파일 objc/objc.h:에 정의되어 있다.

| 유형 | 정의 |
|---|---|
| id | 객체(데이터 구조 포인터) |
| Class | 클래스 객체 (클래스 데이터 구조 포인터) |
| SEL | 메소드명을 식별하는 컴파일러 할당 코드, 셀렉터 |
| IMP | id를 돌려보내는 메소드 구현 포인터 |
| BOOL | Boolean 값, YES 또는 NO |

id는 어떤 종류의 객체, 클래스 또는 인스턴스를 유형짓기 위해서든 사용될 수 있다. 또한, 클래스의 인스턴스를 정적으로 유형짓기 위해 클래스명을 타입명으로 사용할 수 있다. 정적으로 유형지어진 인스턴스는 해당 클래스나 상위 클래스의 인스턴스 포인터로 선언된다.

objc.h 헤더 파일은 유용한 개념을 정의한다.

| 타입 | 정의 |
|---|---|
| nil | null 객체 포인터, (id)0 |
| Nil | null 클래스 포인터, (Class)0 |

## 전처리기 지시문

전처리기(preproessor)는 새로운 표기법을 인식한다.

| 표기법 | 정의 |
|---|---|
| #import | 헤더 파일을 임포트. 이 지시문은 동일한 파일을 한 번 이상 포함하지 않는다는 것을 제외하고는 #include와 동일하다. |
| // | 라인 마지막까지 지속되는 주석을 시작. |

## 컴파일러 지시문

컴파일러 지시문은 @로 시작한다. 다음 지시문은 클래스, 카테고리 및 프로토콜을 선언하고, 정의하기 위해 사용된다.

| 지시문 | 정의 |
|---|---|
| @interface | 클래스 또는 카테고리 인터페이스의 선언 시작 |
| @implementation | 클래스 또는 카테고리의 정의 시작 |
| @protocol | 정형 프로토콜의 선언 시작 |
| @end | 클래스, 카테고리 또는 프로토콜의 선언/정의 종료 |

다음의 상호 배타적인 지시문은 인스턴스 변수의 가시성을 명시한다.

| 지시문 | 정의 |
|---|---|
| @private | 인스턴스 변수 범위를 이를 선언한 클래스로 제한 |
| @protected | 인스턴스 변수 범위를 클래스 선언 및 상속으로 제한 |
| @public | 인스턴스 변수 범위의 제한을 해제 |

초기 설정은 @protected이다.

추가로, 특정 용도의 지시문은 다음과 같다.

| 지시문 | 정의 |
|---|---|
| @class | 다른 곳에서 정의된 클래스명을 선언. |
| @selector(*method*) | *method*를 식별하는 컴파일러 셀렉터를 반환. |
| @protocol(*name*) | *name* 프로토콜(프로토콜 클래스의 인스턴스)을 반환. 포워드 선언의 경우, (@protocol은 선언에 대한 (*name*)이 없어도 유효하다.) |
| @encode(*spec*) | *spec*의 유형 구조를 인코딩하는 문자열을 산출. |
| @defs(*classname*) | *classname* 인스턴스의 내부 데이터 구조를 산출. |

## 클래스

새로운 클래스는 @interface 지시문을 사용하여 선언한다. 슈퍼클래스를 위해 인터페이스 파일을 임포트해야 한다.

```
#import "ItsSuperclass.h"

@interface ClassName : ItsSuperclass < protocol list >
{
    instance variable declarations
}
method declarations
@end
```

컴파일러 지시문과 클래스명을 제외하고는 모든 것은 선택적이다. 콜론과 슈퍼클래스명을 생략한 경우, 클래스는 새로운 루트 클래스로 선언된다. 프로토콜이 나열된 경우, 그 프로토콜이 선언된 헤더 파일도 임포트해야 한다.

클래스 정의를 포함하고 있는 파일은 자신의 인터페이스를 임포트한다.

```
#import "ClassName.h"

@implementation ClassName
method definitions
@end
```

## *카테고리*

개발자는 카테고리명으로 되어있는 인터페이스 파일에 메소드를 선언하고, 같은 이름의 구현파일에 이를 정의하여 메소드를 클래스에 추가할 수 있다. 카테고리명은 메소드가 새로운 클래스가 아닌 다른 곳에 선언된 클래스에 추가되었다는 것을 나타낸다.

카테고리는 서브클래스의 대안일 수 있다. 기본 클래스를 확장하기 위해 서브클래스를 정의하는 대신, 카테고리를 통해 그 클래스에 메소드를 직접 추가할 수 있다. 예를 들면, 카테고리를 NSArray 및 다른 Cocoa 클래스에 추가할 수 있다. 서브클래스의 경우와 같이 클래스를 확장하기 위해 소스 코드를 필요로 하지 않는다.

카테고리에 추가된 메소드는 클래스 타입의 일부가 된다. 예를 들어, NSArray 클래스에 카테고리로 추가된 메소드는, 컴파일러가 NSArray 인스턴스에 있는 기본 메소드로 간주할 것이다. NSArray 클래스의 서브클래스에 추가된 메소드는 NSArray 유형에 포함되지 않는다.(정적으로 유형을 정의해야만 컴파일러가 객체의 클래스를 알 수 있으므로, 이는 정적으로 유형이 정의된 객체의 경우에만 해당된다)

카테고리 메소드는 클래스에 설정된 메소드가 정상적으로 수행할 수 있는 모든 작업을 이행한다. 런타임시, 별다른 차이점은 없다. 카테고리에서 클래스에 추가된 메소드는 다른 메소드처럼 클래스의 모든 서브클래스에서 상속한다.

카테고리는 클래스와 동일한 방법으로 선언한다. 클래스를 선언한 인터페이스 파일을 임포트해야 한다.

```
#import "ClassName.h"

@interface ClassName ( CategoryName ) < protocol list >
method declarations
@end
```

프로토콜 목록과 메소드 선언은 생략 가능하다. 그 프로토콜이 선언되어 있는 헤더 파일을 임포트해야 한다.

클래스 정의와 마찬가지로 카테고리 정의를 포함하고 있는 파일은 자신의 인터페이스를 임포트한다.

```
#import "CategoryName.h"

@implementation ClassName ( CategoryName )
method definitions
@end
```

## 프로토콜

클래스 및 카테고리 인터페이스는 특정 클래스와 관련된 메소드(주로 그 클래스가 구현한 메소드)를 선언한다. 반면에, 비정형과 정형 프로토콜은 특정 클래스와 관련되지 않은, 1개 또는 그 이상의 클래스가 구현한 메소드를 선언한다.

*프로토콜(protocol)*은 클래스 정의와 분리된 메소드 선언 리스트이다. 예를 들면, 마우스에 대한 사용자 액션을 보고하는 메소드는 프로토콜로 정의될 수 있다.

```
-(void)mouseDown:(NSEvent *)theEvent;
-(void)mouseDragged:(NSEvent *)theEvent;
-(void)mouseUp:(NSEvent *)theEvent;
```

마우스 이벤트에 응답하고자 하는 클래스는 이 프로토콜을 채택하여 메소드를 구현할 수 있을 것이다.

프로토콜은 클래스 및 카테고리에서 사용할 수 없는 방법으로 메소드를 사용할 수 있도록 클래스 계층의 종속성에서 메소드 선언을 분리한다. 프로토콜은 다른 곳에서 구현된(또는 구현되었을) 메소드를 나열하지만, 그 메소드를 구현한 클래스가 무엇인지는 중요하지 않다. 중요한 것은 특정 클래스가 프로토콜에 부합하는가, 즉 프로토콜이 선언한 메소드에 대한 구현이 그 클래스에 있는가이다. 따라서, 객체는 동일한 클래스에서 상속된 기능으로 인한 유사성에 기반하는 것이 아니라 동일한 프로토콜에 부합한 유사성에 기반하여 그룹화될 수 있다. 상속 계층과 관련이 없는 여러 클래스가, 동일한 프로토콜에 부합하기 때문에 유사한 그룹으로 분류될 수도 있다.

프로토콜은 객체 지향 디자인에서 중요한 역할을 할 수 있다. 특히 프로젝트가 수많은 개발자에 의해 진행되거나, 다른 프로젝트에서 개발된 객체들을 통합할 경우에 그러하다. Cocoa 소프트웨어는 Objective-C 메시지를 통해 프로세스간 통신을 지원하기 위해 프로토콜을 많이 사용한다.

그러나, Objective-C 프로그램이 프로토콜을 사용할 필요가 없다. 클래스 정의 및 메시지 표현방식과는 달리 프로토콜은 옵션 기능이다. Cocoa 프레임워크의 일부는 프로토콜을 사용하며, 또 다른 일부는 사용하지 않는다. 사용 여부는 작업 내용에 따라 달라진다.

정형 프로토콜은 @protocol을 사용하여 선언한다.

```
@protocol ProtocolName
< protocol list >
method declarations
@end
```

통합된 프로토콜 목록과 메소드 선언은 생략 가능하다. 프로토콜은 통합된 프로토콜을 선언한 헤더 파일을 임포트해야 한다.

소스 코드내에서 프로토콜은 괄호로 프로토콜명을 묶은 형태의 @protocol() 지시문을 사용하여 지칭한다.

꺽쇠 괄호 <...>로 묶인 프로토콜명은 다음을 수행한다.

- 프로토콜 선언에서, 다른 프로토콜 통합 (이전 참조)
- 클래스 또는 카테고리 선언에서 공식적으로 프로토콜 채택("클래스" 및 "카테고리" 섹션 참조)
- 유형 규격에서 프로토콜에 부합한 객체로 유형을 제한 타입 자격자

프로토콜 선언에서 이 같은 유형의 자격자는 프로토콜 선언 내에서 원격 메시징을 지원한다.

| 유형 자격자 | 정의 |
|---|---|
| oneway | 이 메소드는 비동기식 메시지용이며, 유효한 리턴값이 없다 |
| in | 이 인수는 정보를 원격 수신자에 넘겨준다. |
| out | 이 인수는 참조에 의해 반환된 정보를 받는다. |
| inout | 인수는 정보를 주고, 받는다. |
| bycopy | 프록시가 아닌 객체의 카피를 넘겨주거나 돌려받아야 한다. |
| byref | 카피가 아닌 객체의 레퍼런스를 넘겨주거나 돌려받아야 한다. |

## 메소드 선언

다음 규정은 메소드 선언에서 사용한다.

- A + (플러스), 클래스 메소드 선언 앞에 위치한다.
- A - (마이너스), 인스턴스 메소드 선언 앞에 위치한다.
- 인수는 콜론(:) 뒤에 온다. 일반적으로 인수를 설명하는 라벨은 콜론 앞에 위치한다. 라벨과 콜론은 메소드 명의 일부로 간주된다.
- 인수 및 리턴 값의 유형은 C 언어의 타입캐스팅 문법을 사용하여 선언한다.
- 메소드를 위한 디폴트 리턴 및 인수 타입은 int가 아닌 id이다.(그러나, unsigned 뒤에 아무 것도 없는 경우에는 항상 unsigned int를 의미한다)

## 메소드 구현

각 메소드 구현은 2개의 숨겨진 인수를 제공한다.

- 수신 객체(self)
- 메소드용 셀렉터(_cmd)

구현 내에서, self 및 super는 모두 수신 객체를 지칭한다. 구현에서 상속된 메소드만이 메시지 응답하여 수행되어야 한다는 의미로, super는 메시지의 리시버로서 self를 대신한다.

일반적으로, 다른 유효 리턴을 갖지 않은 메소드는 void를 리턴한다.

## 이름 정의 규정

Objective-C 소스 코드를 포함한 파일은 이름에 .m 확장자를 갖는다. 클래스 및 카테고리 인터페이스를 선언하거나 프로토콜을 선언한 파일은 헤더파일 고유의 .h 확장자가 있다.

일반적으로, 클래스, 카테고리 및 프로토콜 명칭은 대문자로 시작하며, 메소드명과 인스턴스 변수는 소문자로 시작한다. 인스턴스 값을 갖는 변수명도 소문자로 시작한다.

Objective-C에서는 다양한 용도로 사용되는 동일한 명칭은 상호 충돌하지 않는다. 클래스내 명칭은 제한없이 할당될 수 있다.

- A 클래스는 다른 클래스의 메소드와 동일한 이름을 사용한 메소드를 선언할 수 있다.
- A 클래스는 다른 클래스의 변수와 동일한 이름을 사용한 인스턴스 변수를 선언할 수 있다.
- 인스턴스 메소드는 클래스 메소드와 동일한 이름을 가질 수 있다.
- 메소드는 인스턴스 변수와 동일한 이름을 가질 수 있다.

또한, 동일한 클래스의 프로토콜 및 카테고리에서 이름영역을 보장하고 있다.

- 프로토콜은 클래스, 카테고리 또는 기타 사항과 동일한 이름을 가질 수 있다.
- 한 클래스의 카테고리는 다른 클래스의 카테고리와 동일한 이름을 가질 수 있다.

그러나, 클래스 이름은 전역 변수 및 기정의된 유형과 동일한 이름 영역 내에 존재한다. 프로그램에는 클래스와 동일한 이름을 사용한 전역 변수가 존재할 수 없다.

# 실행중인 Objective-C

이 섹션에서는 가장 자주 사용되는 Objective-C 언어의 면모를 다룬다. 언어의 구성이 실제로 응용 프로그램의 관점에서 어떻게 사용되는지를 설명하기 위해, 적절한 곳에서는 코드 예제를 이용한다. 내용을 완전히 이해하지 못하더라도 염려하지 않아도 된다. Objective-C는 사례를 통해 쉽게 이해할 수 있으며, 뒷 장의 튜토리얼을 통해 이를 충분히 이해할만한 시간을 가질 수 있다.

## 선언

객체는 정적 또는 동적으로 선언할 후 있다. 정적 유형 객체는 클래스 포인터로 선언된다.

```
NSString *mystring;
```

정적 유형은 컴파일시의 유형 검사를 더 잘 하도록 하며, 코드를 더욱 쉽게 이해할 수 있도록 한다. 상기 사례에서 mystring은 NSString의 인스턴스일 수도 있으며, NSString에서 상속한 클래스의 인스턴스일 수도 있다.

동적 유형 객체는 id로 선언된다.

```
id myObject;
```

동적 유형 객체의 클래스는 런타임시 결정되므로, 개발자는 클래스에 관한 지식이 없어도 코드에서 객체를 참조할 수 있다. 객체가 다형화 또는 동적 바인딩과 관련될 경우, 이 같은 방법으로 객체를 유형화한다.

인스턴스 메소드의 선언은 마이너스 표시(-)로 시작한다. 마이너스 표시 다음의 공백은 생략 가능하다.

```
- (NSString *)countryName;
```

메소드가 반환하는 값의 유형은 마이너스 표시(또는 플러스 표시)와 메소드명 사이에 괄호로 묶는다. 반환값이 없는 메소드는 void의 리턴 타입을 가져야 한다.

메소드 인수의 유형은 괄호로 묶고, 인수의 키워드와 인수 사이에 배치한다.

```
- (id)initWithName:(NSString *)name andType:(int)type;
```

모든 선언은 세미콜론으로 마무리한다.

인스턴스 변수의 가시성은 기본적으로 보호되어, 그 변수를 선언한 클래스나 그 서브클래스의 객체에서만 직접 접근이 가능하다. 인스턴스 변수를 비공개(선언된 클래스내에서만 액세스 가능)로 하려면, 선언앞에 `@private`를 삽입한다.

## 메시지 및 메소드 구현

메시지 표현 방식은 수신 객체를 지칭하는 변수와 호출하고자 하는 메소드의 이름으로 구성된다. Objective-C에서 표현 방식은 대괄호로 묶는다.

```
[anObject doSomethingWithArg:anArgument];
```

표준 C에서와 같이, 세미콜론으로 마무리한다.

메소드 호출로부터 값이 돌려보내지는 경우가 흔히 있다. 이 값을 수신하려면 할당문의 왼쪽에 해당 유형의 변수를 두어야 한다.(또는 C에서와 같이 반환값을 완전히 무시할 수도 있다)

```
int result = [anObj calcTotal];
```

한 메시지 표현방식 안에 또다른 메시지 표현방식이 들어갈 수 있다. 이 예제는 NSForm 객체의 윈도우를 확보하여, 반환된 NSWindow 객체를 다른 메시지의 수신자로 만든다.

```
[[form window] makeKeyAndOrderFront:self];
```

메소드는 함수와 같은 구조로 되어 있다. 메소드의 완전한 선언 뒤에, 중괄호로 묶인 구현 코드가 나온다.

`nil`을 사용하여 `null` 객체를 명시한다. 이는 `null` 포인터와 유사하다. 일부 Cocoa 메소드는 `nil`을 인수로 받아들이지 않는다.

메소드는 `self` 및 `super`와 같은 2개의 식별자를 유용하게 참조할 수 있다. 객체는 자신을 참조하기 위해 이 식별자를 사용한다. 객체가 메시지를 전송할 때 이 같은 식별자를 사용하면 메소드 구현 파일 검색에 영향을 준다. `self`라는 키워드는 전송자의 클래스에서 검색을 시작한다. `super`는 전송자의 슈퍼클래스에서 검색을 시작한다. 예를 들면,

```
[self redraw];
```

객체는 객체 자신의 `redraw` 메소드를 호출한다.

일반적으로 super는, 서브클래스가 새로운 동작을 구현하기 위해 메소드를 오버라이드하였지만 수퍼클래스의 메소드를 수행할 필요가 있을 때에만 사용된다. 예를 들면,

```
- (id)init {
    [super init];
    [self setDate: [NSCalendarDate date]];
    return self;
}
```

객체가 특정 서브클래스를 초기화하기 전에 슈퍼클래스의 init 메소드를 발생시킨다.

개발자는 클래스 인스턴스의 인스턴스 변수에 직접 액세스할 수 있다. 그러나, 성능이 매우 중요한 경우를 제외하고는 직접적인 접근 보다는 접근 메소드를 권장한다.

## 클래스 정의하기

클래스는 2개의 부분으로 정의한다. 한 부분은 인스턴스 변수 및 인터페이스(주로 그 클래스에 속한 객체로 전송된 메시지가 호출할 수 있는 메소드)를 선언한다. 다른 부분에서는 이 같은 메소드를 실제로 구현한다. 인터페이스는 공개적이다. 구현은 비공개로 행해지며, 인터페이스나 클래스가 사용된 방법에 영향을 주지 않고, 변경될 수 있다.

클래스 정의의 기본적인 절차는 제7장, *Currency Converter 튜토리얼*을 참조한다. 그러나, 이 섹션에서는 클래스를 정의할 때 기억해야 할 규정과 요점사항을 보충적으로 설명한다.

- 일반적으로, 클래스의 public 인터페이스는 클래스의 이름인 헤더파일(.h 확장자 사용)에 선언된다. 이 헤더 파일은 클래스를 사용하는 프로그램으로 임포트될 수 있다.
- 일반적으로, 클래스를 구현하는 코드는 클래스명을 제공하고, .m 확장자를 갖는 파일에 저장된다. 이 코드는 프레임워크, 동적 공유 라이브러리, 정적 라이브러리 또는 구현 파일의 형태로 클래스를 내장한 프로그램이 컴파일될 때 사용된다.
- 메소드 선언 및 구현은 마이너스 표시(–)나 플러스 표시(+)로 시작한다. 마이너스 표시는 메소드가 클래스의 인스턴스에서 사용된다는 것을 나타내며, 플러스 표시는 클래스 객체가 사용하는 메소드임을 의미한다.
- 메소드 정의는 함수 정의와 상당히 유사하다. 메소드는 메시지에 응답할 뿐만 아니라, 함수가 다른 함수를 호출하는 것과 마찬가지로 자체적으로 메시지를 호출한다.

- 메소드 구현시, 그 메소드가 정의된 클래스에 속한 객체에 한하여, 그 객체의 인스턴스 변수를 직접 참조할 수 있다. 변수에 접근하거나 객체의 데이터 구조를 넘겨주기 위해 추가 문법이 필요하지는 않다. 모든 것이 숨겨진 채로 존재한다.
- 메소드는 수신 객체를 **self**로 지칭할 수 있다. 이 변수는 객체가 자신의 메소드 정의 내에서 객체 자신에게 메시지를 보낼 수 있게끔 해 준다.

## 메소드 오버라이드하기

서브클래스는 새로운 기능을 추가할 수 있을 뿐만 아니라, 상속한 메소드를 새로운 구현으로 대체할 수도 있다. 특별한 문법이 필요하지 않으며, 단순히 그 메소드를 재구현하기만 하면 된다.

메소드를 오버라이드하면, 객체가 수신할 수 있는 메시지의 종류를 변경하는 것이 아니라, 메시지에 응답하기 위해 사용될 메소드의 구현을 변경한다. 앞서 언급했듯이, 각 클래스가 독자적인 버전의 메소드를 구현할 수 있는 능력을 *다형성*이라 한다. 특정 클래스에서 메소드를 오버라이드하는 작업은 슈퍼클래스의 메소드 동작에 영향을 주지는 않는다.

상속한 메소드를 완전히 대체하는 대신 확장하는 것이 가능하다. 메소드를 확장하려면, 새로운 구현파일에서 메소드를 오버라이드하면서 슈퍼클래스의 동일한 메소드를 호출한다. 이 호출은 Objective-C 언어의 특별한 수신자인 **super**로 메시지를 보냄으로써 발생한다. **super**는 현 클래스에서 정의된 메소드가 아닌 상속된 메소드가 작업을 수행해야 함을 의미한다.

## 객체 생성

클래스의 주요 기능은 그 클래스가 정의하는 유형의 새로운 객체를 생성하는 것이다. 예를 들면, NSButton 클래스는 새로운 NSButton 객체를 생성하고, NSArray 클래스는 새로운 NSArrays를 생성한다. 객체는 런타임시 2단계의 과정을 거쳐 생성된다. 우선 새로운 객체의 인스턴스 변수를 위해 메모리를 할당하고, 그 다음 이 변수를 초기화한다. **init** 메시지는 동일한 코드 라인에 있는 **alloc** 이나 **allocWithZone:** 메시지와 함께 하며, **alloc** 메시지의 수신자는 클래스이다.

```
TheClass *newObject = [[TheClass alloc] init];
```

**alloc** 메소드는 수신 클래스의 새로운 인스턴스를 위한 메모리를 동적으로 할당하여 새로운 객체를 반환한다. **init** 메시지용 수신자는 **alloc**이 동적으로 할당한 새로운 객체이다. 객체는 초기화되기전까지는 사용할 준비가 되어 있지 않다. 객체는 단 한번만 초기화되어야 한다. NSObject 클래스에 있는 **init** 메소드 버전은 초기화를 수행하지 않는다. 단지, **self**를 돌려보낸다.

새로운 객체는 할당 및 초기화 이후, 독자적인 변수를 보유하여 완전한 기능을 하는 클래스의 일원이 된다. **newObject**는 메시지를 수신하고, 인스턴스 변수에 값을 저장할 수 있다. 객체를 더 필요로 할 경우, 동일한 방법으로 동일한 클래스 정의에서 객체를 생성한다.

**init** 메소드의 서브클래스 버전은 정상적으로 초기화된 이후 새로운 객체인(**self**)를 돌려보내야 한다. 초기화될 수 없다면 메소드는 객체를 해제하여 **nil**을 돌려보내야 한다. 어떤 경우엔, **init** 메소드는 새로운 객체를 해제하고, 대체물을 돌려보낼 수 있다. 따라서, 프로그램은 **init**이 돌려보낸 객체를 사용해야 하고, **alloc** 또는 **allocWithZone:**이 돌려보낸 객체를 반드시 사용할 필요는 없다.

모든 클래스는 **init** 메소드가 클래스에서 완전한 기능을 하는 인스턴스를 돌려보내거나 예외 상황을 제기할 수 있다는 것을 보장해야 한다. **init**의 서브클래스 버전은 **super**에 메시지를 보냄으로써, 상속받은 클래스의 초기화 코드를 통합해야 한다.

```
- init
{
    if (self = [super init]) {
    /* class-specific initialization goes here */
    }
    return self;
}
```

**super**로 보내지는 메시지는 메소드에 추가된 초기화 코드 앞에 위치해야 한다. 이는 초기화가 상속받은 순서대로 수행되어야 함을 의미한다.

## *지정 초기화기*

서브클래스는 종종 특정값을 설정할 수 있도록 추가 인수를 갖는 **init...** 메소드를 정의한다. 메소드에 인수가 많을수록, 개발자가 초기화된 객체의 특성을 자유롭게 결정할 수 있다. 클래스는 다양한 수의 인수를 갖고 있는 여러 개의 **init...** 메소드를 보유한다. 예를 들면,

```
- init;
- initArg:(int)tag;
- initArg:(in0t)tag arg:(struct info *)data;
```

이들 중 적어도 1개의 메소드, 보통 가장 많은 인수를 가진 메소드는 상위 계층 클래스의 초기화를 통합하기 위해 **super**로의 메시지를 포함하는 것이 관례이다. 이 메소드는 클래스의 *지정 초기화기 (designated initializer)*라 지칭한다.

클래스에서 정의된 다른 **init...** 메소드는 **self**로의 메시지를 통해 지정 초기화기를 직간접으로 호출한다. 이러한 방식으로 모든 **init...** 메소드는 연결된다. 예를 들면,

```
- init
{
    return [self initArg:-1];
}

- initArg:(int)tag
{
    return [self initArg:tag arg:NULL];
}

- initArg:(int)tag arg:(struct info *)data
{
    [super init. . .];
    /* class-specific initialization goes here */
}
```

이 사례는 **initArg:arg:** 메소드가 클래스의 지정 초기화기라는 것을 보여준다. 서브클래스의 초기화가 필요하면, 자신의 지정 초기화기를 정의해야 한다. 이 메소드는 그 슈퍼클래스의 지정 초기화기를 수행하도록 **super**로 메시지를 전송하는 것으로 시작해야 한다.

예를 들어, 앞서 설명한 3개의 메소드가 B 클래스에서 정의되었다고 가정해본다. B의 서브클래스인 C 클래스는 지정 초기화기가 있어야 한다.

```
- initArg:(int)tag arg:(struct info *)data arg:anObject
{
   [super initArg:tag arg:data];
   /* class-specific initialization goes here */
}
```

상속된 **init...** 메소드가 성공적으로 서브클래스의 인스턴스를 초기화하려면, 모든 메소드가 새로운 지정 초기화기를 직간접으로 호출하도록 해야 한다. 이 작업을 수행하려면, 서브클래스는 슈퍼클래스의 지정 초기화기만을 오버라이드해야 한다. 예를 들면, C 클래스는 지정 초기화기뿐만 아니라 이 메소드도 구현한다.

```
- initArg:(int)tag arg:(struct info *)data
{
    return [self initArg:tag arg:data arg:nil];
}
```

이는 B 클래스에서 상속된 3개의 메소드가 모두 C 클래스의 인스턴스에서도 동작할 수 있음을 보장한다. 종종 서브클래스의 지정 초기화기는 슈퍼클래스의 지정 초기화기를 오버라이드하기도 한다. 이 경우, 서브클래스는 하나의 **init...** 메소드만을 구현하면 된다.

이 방식은 `init...`의 직접적인 연결을 유지하며, 새로운 메소드와 상속된 모든 `init...` 메소드가 정상적으로 초기화된 객체를 돌려보낼 수 있도록 한다. 또한, 슈퍼클래스 메소드를 수행하기 위해 서브클래스 메소드가 메시지를 전송하고(`super`로), 다음에 그 슈퍼클래스가 서브 클래스 메소드를 수행하기 위해 메시지를 전송하는 경우(`self`로)와 같은 무한 루프의 가능성을 차단한다.

`init` 메소드는 NSObject 클래스의 지정 초기화기이다. 초기화를 수행하는 서브클래스는 상기 설명한 대로 그 메소드를 오버라이드해야 한다.

# 4

# 개발 툴

Apple은 Mac OS X에서 Cocoa를 위한 강력하고, 통합적인 크로스플랫폼 개발 환경을 갖추고 있다. 이 개발환경은 Mac OS X의 프레임워크, 서브시스템, 라이브러리, 컴포넌트, 그리고 기타 리소스로부터 최대의 생산성을 이끌어낼 수 있는 응용 프로그램과 툴로 구성된다.

Cocoa로 응용 프로그램을 개발하려면, 프로그래밍 언어를 선택해야 한다. 개발자는 C, C++, Java 또는 Objective-C로 프로그램의 전체 또는 일부를 작성할 수 있다. 자바에서 Cocoa 클래스를 서브클래스로 분류할 수 있으며, "순수" 자바와 Cocoa 객체를 조합하여 코드를 작성할 수 있다. C++로부터 Cocoa API에 접근할 수 없다는 것을 명심한다.

본 장에서 제공하는 자료를 비롯해, 컴파일러, 디버거, 그리고 기타 툴을 포괄하는 참조 문서는 `/Developer/Documentation/DeveloperTools`에서 찾아볼 수 있다.

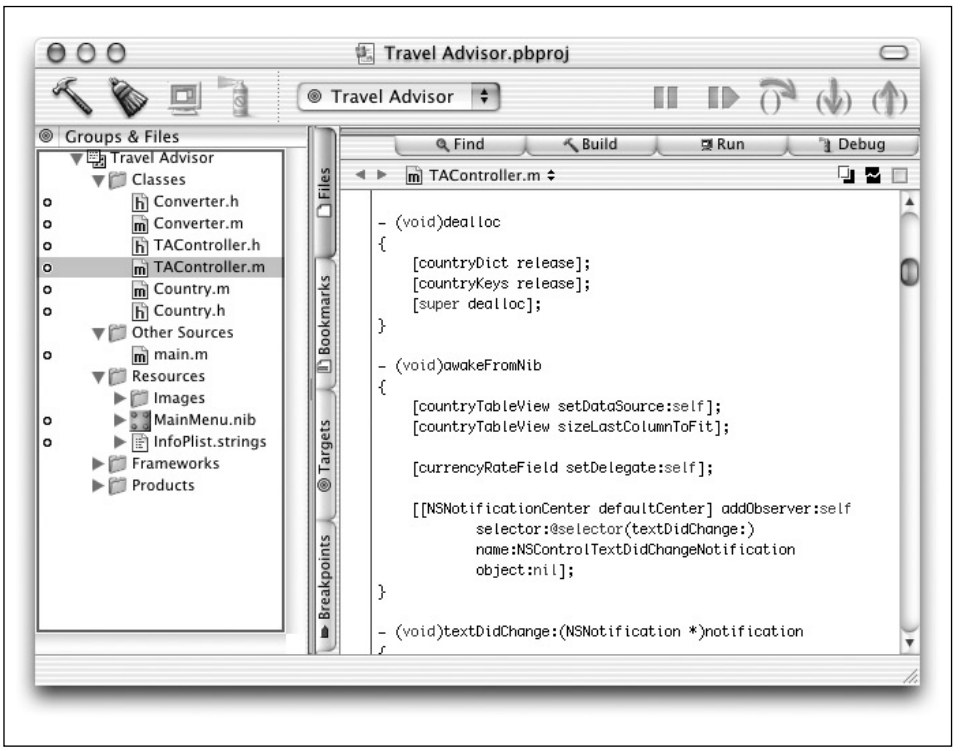
## *Project Builder*

Project Builder는 소프트웨어 개발 프로젝트를 관리하고, 개발 과정을 조정하며, 효율성을 높인 응용 프로그램이다. Project Builder의 주요 기능은 다음과 같다.

- 프로젝트 브라우저
- 완전한 기능을 갖춘 코드 편집기
- 언어 이해 심볼 인식
- 정교한 프로젝트 검색 기능
- 헤더 파일과 도큐멘테이션 액세스

- CVS 소스 제어 관리 시스템과의 빌트인 통합
- 맞춤형 구축
- 그래픽 소스 레벨 디버거

Project Builder의 메인 윈도우는 <그림 4-1>과 같다.




*<그림 4-1> Project Builder의 메인 윈도우*

## 기능 하이라이트

Project Builder로 코드를 작성할 경우, 원하는 대로 "Workbench" 툴을 갖출 수 있다.

### 구분문자 검사

각괄호(왼쪽 또는 오른쪽)를 더블 클릭하면, 각괄호에 들어 있는 코드(구분문자 포함)가 강조된다. 이와 유사한 방법으로, 알맞는 각괄호를 배치하려면 메시지에 있는 대괄호를 더블 클릭하면 되고, 코드를 선택하려면 둥근 괄호에 있는 문자를 더블 클릭하면 된다. 알맞는 구분문자가 없다면 Project Builder는 경고 신호를 울린다.

### *Emacs 키 바인딩*

Emacs는 유닉스 플랫폼에서 코드를 작성하기 위해 사용되는 대표적인 편집기이다. Project Builder의 코드 편집기에서 가장 일반적인 Emacs 명령어들을 사용할 수 있다. 예를 들어, 한 화면 위로 이동(Control-V), 한 단어 앞으로 이동(Meta-F), 커서 앞에 나오는 단어 삭제(Meta-D), 앞줄 삭제(Control-K), 그리고 마지막으로 삭제된 텍스트 삽입(Control-Y)과 같은 명령어들이 있다. 어떤 Emacs 명령어들은 매킨토시의 키 바인딩과 상반될 수 있다. Option 키나 Shift-Control 키를 Emacs의 Control이나 Meta키로 전환하려면 Project Builder의 코드 편집기를 사용하여 변경할 수 있다. Option(Meta)키 바인딩은 특수 문자를 작성하기 위해 사용되기 때문에 Mac OS X에서는 기본적으로 동작하는 키는 아니다. 이 같은 기능 사용에 관한 내용은 Project Builder의 문서를 참조한다.

### *Find*

응용 프로그램을 개발할 경우, Project Builder를 통해 여러 방식으로 Cocoa API에 관한 정보를 얻을 수 있다.

- Find 패인(pane)은 프로젝트에서 클래스, 메소드, 함수, 상수, 그리고 다른 심볼의 정의를 찾아준다. Find 패인은 프로젝트 인덱싱을 기반으로 하기 때문에, 찾기는 신속하고, 철저하게 이루어지며, 해당 코드로 직접 연결된다. 찾기는 또한 텍스트나 일반 표현방식을 기반으로 할 수 있다.
- Project Find를 사용하고 난 후, 결과에 Cocoa 심볼이 포함되어 있으면, 관심있는 심볼 옆의 북 아이콘을 클릭함으로써, 심볼을 설명한 참조 도큐멘테이션에 쉽게 액세스할 수 있다.
- Project Builder에서 Cocoa의 프레임워크와 관련된 헤더 파일과 문서를 이용할 수 있다. Application Kit와 Foundation 프레임워크는 Cocoa 프로젝트를 위해 기본적으로 포함되어 있다.

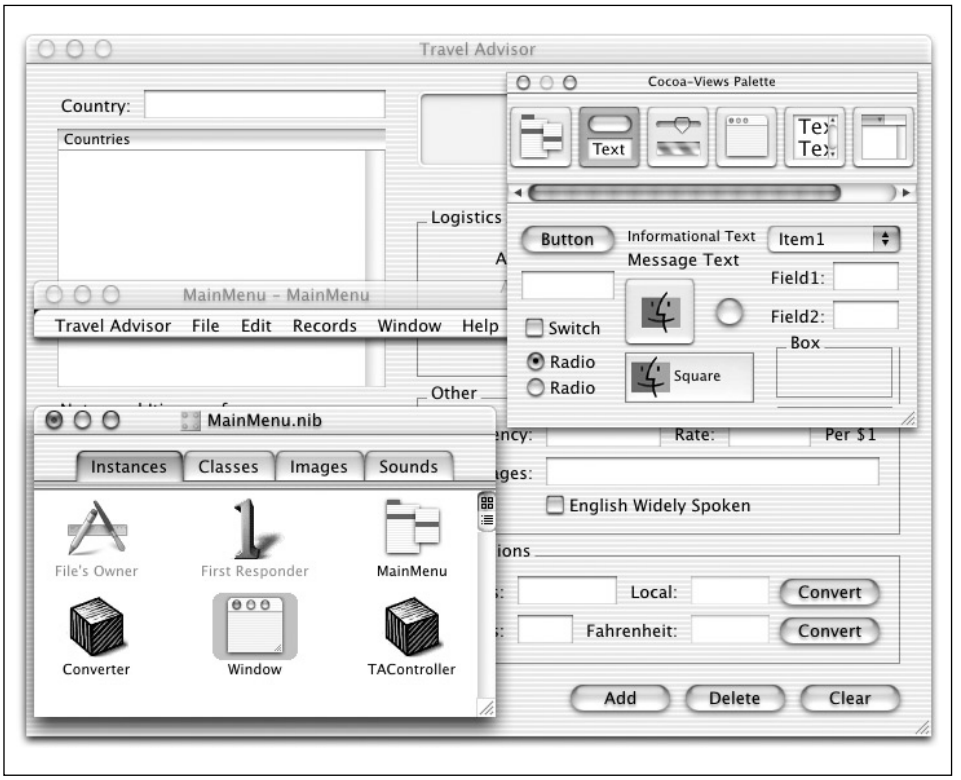
### *유연한 빌드 시스템*

Project Builder는 응용 프로그램의 GUI나 커맨드 라인을 사용하여 응용 프로그램을 구축할 수 있도록 상당히 유연한 빌드 시스템을 갖추고 있다. Project Builder는 다양한 종류의 프로젝트(응용 프로그램을 비롯해, 팔레트, 프레임워크, 번들)를 간단하게 생성할 수 있다. 이들 기능 사용에 관한 추가적인 내용은 Project Builder의 온라인 도움말 시스템(Help 메뉴에서 이용 가능)을 참조한다.

## *Interface Builder*

Interface Builder는 응용 프로그램의 인터페이스를 간단하게 만들 수 있도록 한다. 팔레트에서 객체를 드래그하여 만들고자 하는 그래픽 사용자 인터페이스에 드롭하면 된다.(<그림 4-2> 참조) Info 윈도우를 통해 객체 속성들을 설정하고, 메시지를 전송할 수 있도록 응용 프로그램의 다른 객체에 연결할 수 있다.

Interface Builder는 커스텀 클래스의 정의를 지원하며, 코드를 컴파일하지 않고서도 인터페이스를 시험할 수 있도록 한다.




*<그림 4-2> Interface Builder*

Interface Builder의 기본적인 팔레트에는 다양한 Application Kit 객체들이 갖춰져 있다. 다른 팔레트에는 그 외 프레임워크, 타 객체, 그리고 맞춤형 컴파일 객체들이 구성되어 있다. 또한, 동적 팔레트에 비컴파일 객체들을 저장할 수 있다. Interface Builder는 사용자 인터페이스 요소들(연결 포함)을 객체(컴파일된 실행 파일과 고정적으로 연결되지 않은)로써 저장하고, 복원한다.

# 기타 개발 툴

Cocoa 개발 과정에서 사용할 수 있는 개발 툴은 Project Builder와 Interface Builder를 비롯하여 다른 응용 프로그램들이 있다. 그래픽 사용자 인터페이스의 특징을 갖는 개발 툴은 <표 4-1>에 나타나 있다. 설치된 곳이 명시되지 않은 응용 프로그램은 `/Developer/Applications`에 설치되어있다.

<표 4-1> 개발 툴

| 이름 | 설명 |
|---|---|
| FileMerge | 2개의 파일이나 2개의 디렉토리에 담긴 내용을 시각적으로 비교한다. 예를 들어, 같은 소스 코드 파일 버전이나 2개의 프로젝트 디렉토리간의 차이점을 알아보기 위해 FileMerge를 사용할 수 있다. 또한, 바뀐 내용을 합치기 위해 FileMerge를 사용할 수 있다. |
| icns Browser | Mac OS X 아이콘 파일의 전체 내용을 보여준다. |
| IconComposer | 소스 아트로부터 Mac OS X 아이콘 파일을 만든다. |
| IORegistryExplorer | 시스템 입출력 레지스트리를 계층적으로 보여준다. |
| JavaBrowser | 자바 클래스 계층과 도큐멘테이션을 보여준다. |
| MallocDebug | 응용 프로그램의 동적 메모리 사용을 측정하고, 메모리 누설을 찾아내며, 응용 프로그램에 할당된 모든 메모리를 분석하고, 제공된 시간이후 할당된 메모리를 측정한다. |
| MRJAppBuilder | 실행 자바를 Mac OS X에 적합한 더블 클릭이 가능한 응용 프로그램으로 전환한다. |
| ObjectAlloc | 실행되고 있는 응용 프로그램을 위해 모든 객체 할당(Cocoa와 Core Foundation)을 추적하여 보여준다. 객체를 할당한 뒤, 콜 스택과 객체 리스트를 볼 수 있도록 한다. |
| OpenGL Info | OpenGL 렌더러 속성을 보여준다. |
| PackageMaker | Mac OS X 인스톨러 패키지를 생성한다. |
| PEFViewer | PEF(Preferred Executable Format)의 내용을 보여준다. |
| Pixie | 화면 객체를 구성하고 있는 픽셀을 정확하게 볼 수 있도록 커서 아래 화면영역을 확대한다. 화면을 1~12배로 확대 조절할 수 있다. |
| PropertyListEditor | 속성 리스트(`.plist`) 파일을 열어, 보여주거나 변경할 수 있도록 한다. |
| QuartzDebug | 시스템인 모든 윈도우 리스트를 보여준다. 윈도우 서버에 의해 업데이트될 때 화면에서 노란색으로 반짝거리는 Quartz 디버깅 모드를 작동시킨다. |
| Sampler | 사용자 지정 주기동안 프로그램의 콜 스택을 샘플링하여 응용 프로그램의 성능을 분석한다. |

# 유용한 커맨드 라인 툴

Apple은 컴파일, 디버깅, 성능 분석 등을 위해 여러 커맨드라인 툴을 만들거나 변경해왔다. <표 4-2>에 유용한 툴이 나와있다. Manpage 시스템을 사용하여 좀더 자세한 정보를 얻을 수 있다. 툴은 `/usr/bin` directory에 모두 위치한다.

<표 4-2> 커맨드 라인 개발 툴

| 이름 | 설명 |
|---|---|
| cc | C, Objective-C 및 C++ 소스 코드 파일을 컴파일한다. |
| gdb | Apple이 Objective-C와 C++을 지원하기 위해 확장한 C를 위한 소스 단계의 심볼 디버거이다. |
| gnumake | 종속성 정보를 기반으로 프로그래밍 프로젝트의 제품을 구축한다. |
| as | 조합; 어셈블리 코드를 객체 코드로 변환한다. |
| default | 사용자가 지정한 디폴트를 읽고, 작성하며, 찾아서 삭제한다. 디폴트 시스템은 응용 프로그램이 실행되지 않을 때 지속하는 사용자 환경설정을 기록한다. 사용자가 응용 프로그램의 Preference 패널에 디폴트를 지정하면, NSUserDefaults 메소드는 디폴트를 작성하기 위해 사용된다. |
| nibtool | Interface Builder nib 파일의 내용을 읽는다. 클래스, 계층, 객체, 커넥션, 그리고 로컬라이즈된 스트링을 출력한다. |
| libtool | 단일 또는 다중 아키텍처로 지정된 객체 빈 파일에서 정적 또는 동적 라이브러리를 생성한다. |
| otool | 객체 파일이나 라이브러리의 지정된 부분을 보여준다. |
| nm | 지정된 객체 파일이나 파일의 심볼 테이블을 전부 또는 일부분 보여준다. |
| fixPrecomps | 각각의 주요 프레임워크를 위해 미리 컴파일된 헤더파일을 만들거나 제공한다. |
| strip | 조합되었거나 링크되어 있는 출력물에 첨부된 심볼 테이블을 삭제하거나 변경한다. |

Mac OS X 개발 환경에서는 이 장에서 설명한 수 많은 툴이 사용되지만, 이 책에서는 Project Builder와 Interface Builder의 사용에 전적으로 중점을 둔다. 컴파일러나 링커같은 일부 툴은 Project Builder를 통해 간접적으로 생성된다. 대부분의 다른 툴은 Cocoa 응용 프로그램을 구축하기 위해 반드시 필요한 것은 아니다. 그러나, 디버깅과 성능 분석 툴인 ObjectAlloc, QuartzDebug, Sampler는 응용 프로그램의 내부 작업을 이해하는 데 상당히 유용하다. 튜토리얼 구축 과정에서 이 툴들을 사용하여 마음껏 시험해보길 바란다.

# II

# 단일 윈도우 응용 프로그램

# 5

# *Hello World*

이 장에서는 Hello World 프로그램을 만들기 위한 훌륭한 튜토리얼을 제공한다. 제공되는 튜토리얼은 Hello World 프로그램을 구축하는 프로그램을 만들기 위해 제4장, *개발 툴*에서 설명한 Project Builder 응용 프로그램을 사용하는 방법을 보여준다. Hello World는 뭔가를 인지할 수 있는(1개의 스트링이 화면에 출력되는 것처럼) 아주 단순한 작업 프로그램이다. Hello world 프로그램을 운용하면 개발 환경이 정상적으로 작동하고 있다는 것을 확인할 수 있다.

## *프로젝트 만들기*

Cocoa 프로그램의 가장 단순한 유형은 Foundation 툴이다. 이 프로그램 유형은 Foundation 프레임워크만을 사용하므로, 그래픽 사용자 인터페이스(GUI)가 갖춰져 있지 않다. GUI를 갖추지 않은 Cocoa 프로그램을 *Tool*로 지칭함으로써, GUI를 갖춘 Cocoa 프로그램을 Application 라고 지칭하는 것과 명확하게 구분한다. Application Kit Framework(제1장, *Cocoa 소개*에서 상세히 설명)를 사용하는 Cocoa 응용 프로그램과는 달리, Foundation 툴은 Project Builder와 커맨드라인에서만 실행될 수 있다.

Foundation 툴은 Mac OS X BSD 환경에서 상당히 강력한 커맨드라인 응용 프로그램을 신속하게 개발하기 위한 훌륭한 방안이다. Foundation 툴은 표준 C 프로그램과 상당히 유사하다. 그러나,Foundation 툴은 Foundation 프레임워크와 연결되어 있으므로, 표준 Objective-C 구조를 비롯해 Foundation 클래스의 모든 기능을 이용할 수 있다.

이 섹션에서는 "Hello World!"를 출력하는 아주 간단한 기본 툴을 만들기 위한 필요한 몇 가지 단계를 제공한다.

## Project Builder 열기

Project Builder로 응용 프로그램을 구축하기 전에, 먼저 응용 프로그램을 구동해야 한다.

1. `/Developer/Applications`에 있는 Project Builder를 찾는다.
2. 아이콘을 더블 클릭한다.

먼저, Project Builder를 구동하면, 응용 프로그램의 환경을 설정하기 위한 Assistant가 나타난다.

1. Assistant의 Welcome 페이지에서 Next를 클릭한다.
2. 빌드 프로덕트의 위치를 선택하고 Next를 클릭한다. Assistant의 첫 페이지에서는 빌드 프로덕트(예를 들어, 실행 파일이나 라이브러리)와 중간 빌드 파일(예를 들어, 객체파일)을 어느 곳에 저장할 것인지 묻는 질문이 나타난다. 기본적으로 빌드 프로덕트와 빌드 파일은 현 프로젝트의 빌드 디렉토리에 저장되어 있지만, 원하는 디렉토리에 저장할수도 있다.
3. 편집 환경설정을 선택하고, Finish를 클릭한다. 두번째와 마지막 Assistant 페이지에서는 새 텍스트 파일을 독립된 윈도우에서 열 것인지 Project Builder의 단일 윈도우 코드 편집기에서 열것인지에 대한 질문이 나타난다.

## New Project Command 선택하기

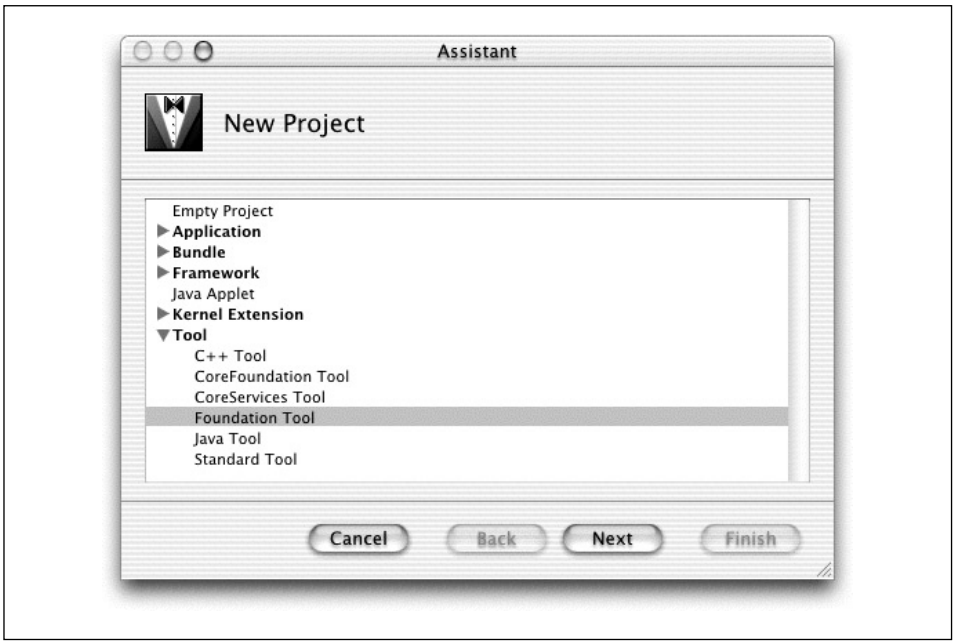
Project Builder를 처음으로 구동하면, Project Builder의 메뉴바가 나타난다. 프로젝트를 만들려면, File 메뉴에서 New Project를 선택한다. 그러면, <그림 5-1>에서와 같이 New Project Assistant가 나타난다. New Project Assistant에서는 New Project를 만들기 위해 간단한 몇 단계를 수행한다.

### 프로젝트 유형 선택하기

이 책에서는 Cocoa 응용 프로그램을 구축하는 작업에 중점을 두지만, Project Builder는 Carbon과 Java 응용 프로그램을 비롯해 Mac OS X 커널 확장과 프레임워크에 이르기까지 다른 유형의 프로젝트에 사용될 수 있다. Assistant에서 프로젝트 탬플리트를 선택하면, 응용 프로그램을 구축하는데 유용하게 사용할 수 있다.

1. 탬플리트 리스트에서 Foundation Tool을 선택하고, Next를 클릭한다. New Project Assistant에서 새 프로젝트의 이름을 지정하고, 저장할 위치를 파일 시스템에서 선택한다. <그림 5-2>를 참조한다.
2. Name 필드에 **`Hello World`**를 입력한다.

3. 기본적으로 제공되는 프로젝트의 저장 위치(홈 디렉토리)를 사용하지 않으려면, Set 버튼을 클릭한 뒤, 파일 시스템 브라우저를 사용하여, 저장하고자 하는 디렉토리를 찾는다.
4. Finish를 클릭한다.




*<그림 5-1> Project Builder의 New Project Assistant*

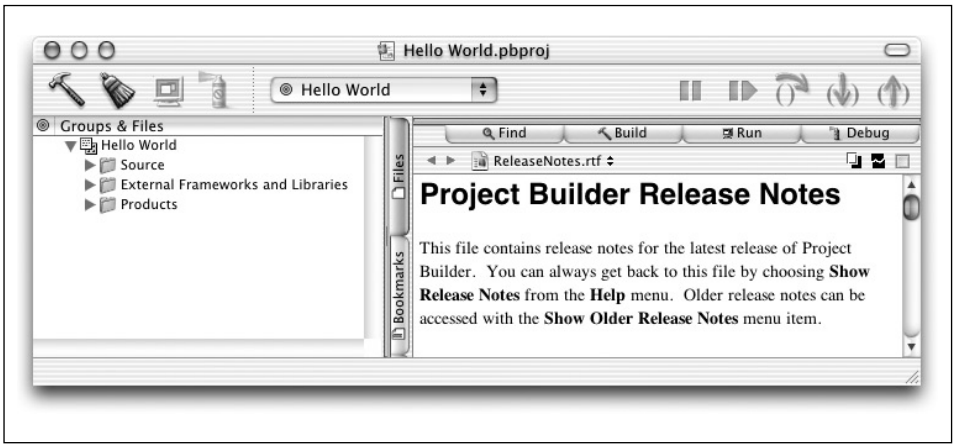
## 메인 윈도우

작업이 완료되면, Project Builder는 메인 프로젝트 윈도우를 제공한다. <그림 5-3>을 참조한다.

메인 프로젝트 윈도우는 응용 프로그램 개발에 필요한 모든 툴을 갖추고 있다. 이 윈도우를 통해 다음과 같은 작업을 수행할 수 있다.

- 소스 파일 보기, 편집 및 구성하기
- 빌드 시스템 불러오기
- 디버거 설정값 변경, 브레이크포인트 설정, 그리고 코드를 실행
- 프로젝트 설정값 변경
- 시스템 헤더와 소스 파일 찾기
- 도큐멘테이션에 액세스하기 위해 Help Viewer를 불러오기

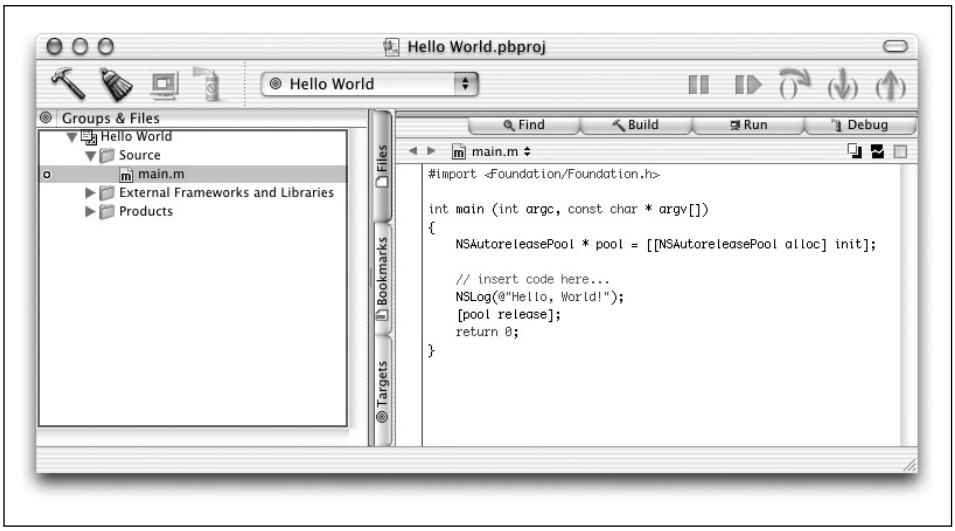

<그림 5-2> Project Builder의 프로젝트 이름 지정하기




<그림 5-3> Hello World의 메인 윈도우

## 응용 프로그램 구축하기

1. Project Builder 메인 윈도우의 Groups & Files 리스트에서 Source 왼쪽에 있는 삼각형을 클릭한다.
2. `main.m`을 클릭한다. 그러면, <그림 5-4>에서와 같이 코드 편집기에서 파일 내용을 볼 수 있다.




<그림 5-4> Project Builder의 Groups & Files 리스트

main.m 구현 파일은 응용 프로그램을 위한 시작점(Entry point)을 가지고 있다. Foundation Tool의 프로젝트 템플리트는 "Hello World"를 Project Builder의 Run 패인에 출력하는 기본적인 main 함수를 제공한다. NSAutoreleasePool의 함수는 다음 장에서 다룰 예정이므로, 현재로써는 염려하지 않아도 된다.

```
int main (int argc, const char *argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    // insert code here
    NSLog(@"Hello, World!");

    [pool release];
    return 0;
}
```

3. NSLog 함수는 C 언어의 printf와 상당히 유사한 기능을 한다. 그러나 NSLog은 C 스트링 대신에 NSString 객체를 취한다는 차이점이 있다. Apple의 Objective-C 컴파일러는 따옴표에 문자를 넣어 NSString 객체를 생성하는 @"" 지시문을 제공한다.
4. 메인 윈도우에서 Build 버튼을 클릭한다.

이제, 빌드 과정이 시작된다. 과정이 진행됨에 따라 빌드에 관한 상세한 정보를 알아보기 위해 Project Builder의 Build 패인을 연다. Project Builder가 빌드를 완료하면(진행 중에 오류가 발생하지 않아야 함), 프로젝트 윈도우의 하단 좌측에 Build Succeeded가 나타난다.

축하합니다! 이로써, 첫 번째 Cocoa 응용프로그램 개발에 성공했다. 이제, 응용 프로그램을 구동해 보려면 Project Builder에서 Run 버튼을 클릭하면 된다.

응용 프로그램이 구동하면, Project Builder 메인 윈도우의 Run 패인은 NSLog 함수의 출력물을 보여주기 위해 확장된다. 다음과 같은 스트링이 나타나야 한다.

```
Nov 10 15:56:12 Hello World[256] Hello, World!
```

NSLog는 스트링을 비롯해, 현재 시간과 날짜, 프로그램 이름, 프로그램의 PID(Process ID) 번호를 출력한다.

# 6

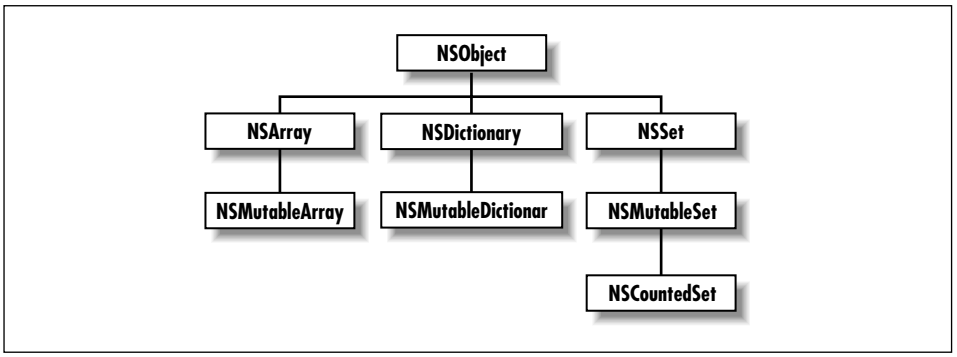
# 중요한 Cocoa 패러다임

이 장에서는 Cocoa 응용 프로그램의 기본적인 구축 블록을 나타내는 몇 몇 프로그래밍 원리에 대해 알아보고자 한다. 먼저, 다른 객체들을 수용할 목적으로 사용되는 객체들, 즉 Cocoa의 콜렉션 클래스를 사용하는 방법에 관해 설명한다. 다음은 Cocoa의 그래픽 사용자 인터페이스를 구축하는 기본 원리에 관해 설명한다. Interface Builder를 사용하는 방법을 배우고, 이 강력한 툴이 Project Builder와 어떻게 상호작용하는지를 파악함으로써, 응용 프로그램을 신속하게 표준화하고, 구축할 수 있다. 마지막으로, 응용 프로그램에서 효과적으로 리소스를 사용할 수 있도록 Cocoa에서 객체 소유권과 처리에 관한 복잡한 사항을 자세하게 설명한다.

위 주제들은 Cocoa 프레임워크를 디자인하는데 있어 핵심적인 사항일 뿐만 아니라, 관련된 구조적 개념이 객체 지향 프로그래밍을 처음으로 접한 개발자에게는 낯설게 느껴질 수 있기 때문에 각별한 관심을 모으기 위해 선택하였다. 이들 디자인 패턴을 연구하다보면 Objective-C와 같은 동적인 언어와 함께 사용할 수 있는 몇몇의 강력한 디자인 접근법을 이해할 수 있으며, Cocoa 개발자로서의 자세도 터득할 수 있다. 관련 원리를 파악하면, Cocoa 툴과 프레임워크를 더욱 효과적으로 사용할 수 있으며, 이 툴을 다루기 위해 필요한 경험을 쌓을 수 있다.

## Cocoa의 콜렉션 클래스

Cocoa의 Foundation 프레임워크에 있는 여러 클래스들은 객체들(문자 그대로, 객체를 언급)을 수용할 목적으로 객체를 생성한다. 이 클래스들을 *콜렉션 클래스*라 한다. 가장 일반적으로 사용되는 2개의 콜렉션 클래스는 NSArray와 NSDictionary이다. NSArray는 제로 베이스 인덱싱을 통해 순서대로 객체를 저장하고, 찾는다. NSDictionary는 키와 값을 사용하여 객체들을 저장하고, 찾는다. <그림 6-1>에서와 같이 이들 콜렉션 클래스는 Cocoa 응용 프로그램 개발에서 아주 유용하게 사용된다.




<그림 6-1> Cocoa 콜렉션 클래스

콜렉션 클래스에는 수정 가능과 수정 불가능의 두 가지 유형이 있다. 수정 불가능 버전(예를 들어, NSArray와 NSDictionary)은 콜렉션이 생성되었지만, 더 이상의 변경이 허용되지 않을 때 아이템을 추가할 수 있도록 한다. 수정 가능 버전(NSMutableArray, NSMutableDictionary)은 콜렉션 객체가 만들어진 이후에 프로그램적으로 객체를 추가하고, 삭제할 수 있도록 한다.

모든 콜렉션 객체들은 개발자가 특정 외부 속성을 만족시키는 값(contained value)에 액세스할 수 있도록 한다. 일반적으로, *키(key)*라는 속성은 콜렉션 유형이 시행하는 구성 방식에 따라 차이가 있다. 예를 들어, 어레이용 키는 콜렉션내에 위치를 지정하는 정수이다. 그러나, 딕셔네리(키라는 용어가 일반적인 의미 이상을 가진다)는 값을 찾는 키로 동작하기 위해 임의 값을 허용한다.

콜렉션 클래스의 역할은 키를 사용하여 저장 및 검색기능을 수행하고, 수용하고 있는 객체들을 조작하는데 있다. 콜렉션 클래스는 수용하고 있는 객체들에 대한 다양한 기능을 수행하기 위해 메소드를 구현한다. 콜렉션 객체라고 해서 모든 기능을 수행하지는 않는다. 일반적으로, 콜렉션 객체는 다음과 같은 기능을 수행한다.

- 다른 객체 콜렉션을 비롯해 파일과 URL로부터 초기 내용을 찾는다.
- 내용을 추가, 삭제, 배치 및 분류한다.
- 다른 콜렉션 객체와 내용을 비교한다.
- 내용을 열거한다.
- 메시지를 내용에 전송한다.
- 디스크 파일에 내용을 저장하고,(NSArchiver/NSUnarchiver의 도움으로) 나중에 찾는다.

Cocoa 응용 프로그램을 구축할 때 콜렉션 객체를 자주 사용하므로 콜렉션 객체가 동작하는 방법에 대한 기본 원리를 이해하는 것은 중요하다.

콜렉션 클래스를 연구하다보면 Objective-C와 Project Builder를 실행할 기회를 접할 수 있다.

## NSArray로 처리하기

이 섹션에서는 NSMutableArray 클래스의 인스턴스를 만들어, 실행할 수 있도록 제5장, *Hello World* 응용 프로그램을 변경하는 방법을 설명한다.

1. Hello World 프로젝트를 연다.
2. main.m을 열어, Autorelease Pool의 선언 뒤, 그리고 NSLog 앞에 NSMutableArray 선언을 추가한다.

```
MSMutableArray *myArray;
```

3. 새로운 어레이를 만들기 위해 코드를 추가한다.

```
myArray = [[NSMutableArray alloc] init];
```

4. 어레이 객체의 내용을 출력하기 위해 코드를 추가한다. NSLog 호출에서, %@은 객체에 의해 교체된다. %@은 printf 호출에서와 같은 동작을 한다. 이 경우, %@은 객체값인 NSString에 대한 확장자라 볼 수 있다. 어레이 항목에 관한 정보를 출력할 수 있도록 NSLog은 자동으로 description 메시지를 어레이로 전송한다.

```
NSLog(@"Array description: %@ item.\n", myArray);
```

5. 어레이를 해제하기 위해 코드를 추가한다. 이 문장은 어레이가 안전하게 헤제될 수 있도록 어레이를 마무리한 실행시간을 나타낸다.

```
[myArray release];
```

main 함수는 <예제 6-1>과 같아야 한다.

*<예제 6-1> 수정 가능 어레이 만들기*

```
int main (int argc, const char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    NSMutableArray *myArray;

    NSLog(@"Hello World: This is a Foundation Tool.\n");

    myArray = [[NSMutableArray alloc] init];

    NSLog(@"Array description: %@ items.\n", myArray);

    [myArray release];
    [pool release];
    return 0;
}
```

응용 프로그램을 구축하여 실행한다. Run 패인에서 다음과 같은 새로운 스트링이 나타나야 한다.

```
Nov 10 15:56:12 HelloWorld[256] Array description: () items.
```

현재 어레이는 비어있지만 아이템을 추가하면, description이 괄호안에 나타난다.

NSMutableArray를 다루면 몇 분이 소요된다. 다음을 수행한다.

- 스트링을 추가한다. 어레이는 보유되지만, 자신의 멤버 객체를 복사하지는 않는다. 자세한 내용은 이 장의 뒷 부분에 수록된 "응용 프로그램 구축 및 디버그"를 참조한다.

```
[myArray addObject: @"here's a string!"];
```

- 수 많은 아이템이 어떻게 어레이에 들어 있는지 확인한다.

```
int itemCount;
itemCount = [myArray count];
```

- 어레이로부터 객체를 얻는다.

```
NSString *myString;
myString = [myArray objectAtIndex: 0];
```

- 객체를 삽입한다.

```
[myArray insertObject: @"Hello2" atIndex: 0];
```

- 아이템을 교환한다.

```
[myArray exchangeObjectAtIndex: 0 withObjectAtIndex: 1];
```

본 컨테이너 클래스의 모든 메소드에 관한 문서는 **/Developer/Documentation/Cocoa**에서 NSArray와 NSMutableArray의 클래스 규격을 참조한다.

## *NSDictionary로 처리하기*

딕셔네리(NSDictionary 또는 NSMutableDictionary 클래스의 객체)는 해싱(hashing) 기반 콜렉션이다. 해싱 기반 콜렉션의 키는 임의 값으로 프로그램이 정의한 데이터의 일부이다. 키는 일반적으로 스트링(NSString 객체)이지만, 어느 객체나 키가 될 수 있다. 키로 사용되는 객체는 NSCopying 프로토콜을 준수해야 하며, **isEqual:** 메시지에 응답해야 한다. 딕셔네리의 키는 개념적인 측면에서 값과 함께 콜렉션에 수용되어야 한다는 점에서 다른 콜렉션의 객체와 다르다. 딕셔네리는 주로 사용자 인터페이스의 텍스트 필드에서 추출된 값처럼 라벨로 분류될 수 있는 데이터를 저장하고, 구성하는 데 유용하게 사용된다.

Cocoa에서 딕셔네리는 딕셔네리의 특정값에 액세스하기 위해 사용된 키가 딕셔네리에 추가되었거나 딕셔네리로부터 삭제된 값과 동일하다는 점에서 어레이와 다르다.(특정 키에 연결된 값이 교체되거나 삭제될 때까지) 어레이에서 특정 값을 찾기 위해 사용된 키(인덱스)는 값이 어레이에 삽입되거나 어레이로부터 삭제될 때 변경될 수 있다. 딕셔네리는 어레이와 달리 값을 순서대로 배치하지 않는다. 나중에, 값을 편리하게 검색하려면, 키-값 쌍의 키가 상수(또는 상수로 취급되어야 함)여야 한다. 키가 딕셔네리에서 값을 배치하기 위해 사용된 후에 변경되면, 값을 검색할 수 없다. 딕셔네리의 키는 한 세트로 구성된다. 즉, 키는 딕셔네리에서 고유하게 사용될 수 있도록 보장된다.

## *딕셔네리로 처리하기*

이 섹션에서는 NSMutableDictionary 클래스의 인스턴스를 만들어 실행할 수 있도록 Hello World 응용프로그램을 다시 한번 변경하는 방법에 관해 설명한다.

1. Hello World 프로젝트를 열어(이미 열려있지 않다면), 편집하기 위해 `main.m`을 불러온다.
2. NSMutableDictionary의 지역 변수에 대한 선언을 추가한다.

```
NSMutableDictionary *myDict;
```

3. 새로운 딕셔네리를 만들기 위해 코드를 추가한다.

```
myDict = [[NSMutableDictionary alloc] init];
```

4. 딕셔네리 객체의 내용을 출력하기 위해 코드를 추가한다.

```
NSLog(@"Dict description: %@ items.\n", myDict);
```

5. 딕셔네리를 해제하기 위해 코드를 추가한다.

```
[myDict release];
```

NSMutableArray를 다루면 몇 분이 소요된다. 다음을 수행한다.

- 키-값쌍을 추가한다.

```
[myDict setObject:@"stringOne" forKey:@"keyOne"];
```

- 키를 사용하여 딕셔네리로부터 객체를 얻는다.

```
myString = [myDict objectForKey:@"keyOne];
```

- 여러 키-값 쌍을 딕셔네리에 추가하여, 모든 키의 어레이를 얻는다. 모두 출력하기 위해 루프를 사용한다.

```
keyArray = [myDict allKeys];
for (index = 0; index < [keyArray count]; index++)
    NSLog(@"Dictionary key at index %i is: %@.\n", index,
        [keyArray objectAtIndex: index]);
```

컨테이너 클래스의 모든 메소드에 관한 문서는 NSArray와 NSMutableArray의 클래스 규격을 참조한다.

# Cocoa에서 그래픽 사용자 인터페이스 만들기

이 섹션에서는 그래픽 사용자 인터페이스로 응용 프로그램을 구축하는 과정을 단순화하고, 가속화하기위해 Cocoa와 Interface Builder가 어떻게 결합하는지를 설명한다. 다음과 같은 내용을 설명하고자 한다.

- **윈도우.** 윈도우는 응용 프로그램이 컨트롤, 필드, 텍스트, 그래픽 등을 보여주는 화면 영역(일반적으로 직사각형)이다.
- **Nib 파일.** Nib 파일은 Interface Builder에 의해 만들어진 파일로 사용자 인터페이스 객체를 비롯해 윈도우를 포함한다.
- **아웃렛.** 아웃렛은 객체가 다른 객체에 메시지를 전송할 수 있도록 Interface Builder에 의해 생성된 특수 인스턴스 변수이다.

마지막으로, 현재 시간과 날짜를 포함한 단일 텍스트 필드를 가진 윈도우를 제공하는 Cocoa 응용 프로그램을 개발함으로써 배운 것을 적용할 수 있다.

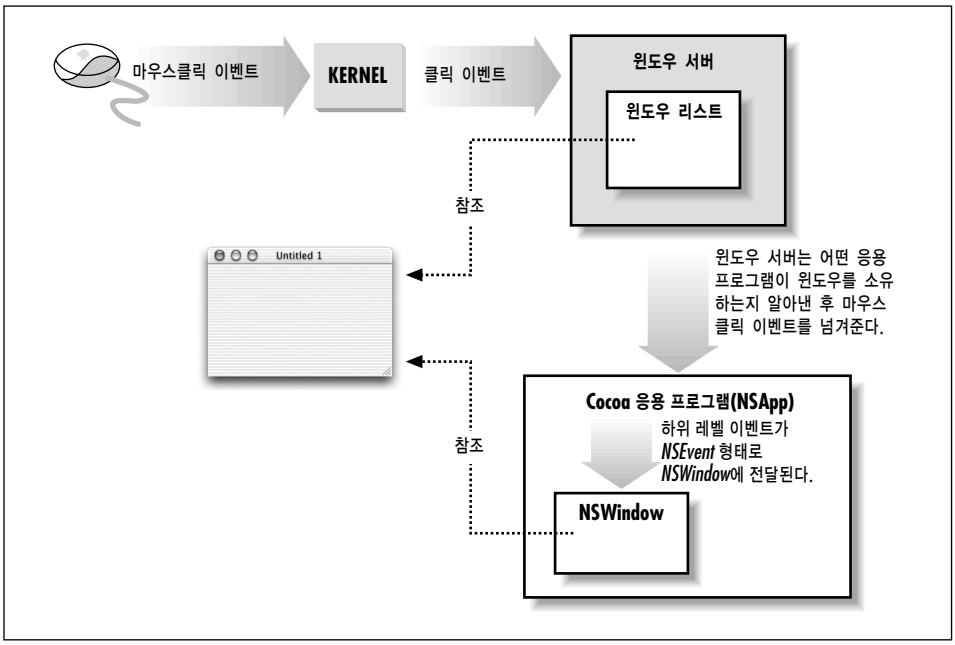
## Cocoa 윈도우

Cocoa의 윈도우는 마이크로 소프트 윈도우나 Mac OS 9의 윈도우처럼 다른 사용자 환경에 있는 윈도우와 상당히 유사하다. 윈도우는 화면 주위로 이동할 수 있으며, 신문처럼 층층히 쌓을 수도 있다. 대표적인 Cocoa의 윈도우에는 타이틀 바, 컨텐트 영역, 그리고 여러 컨트롤 객체들이 있다.

### NSWindow와 윈도우 서버

기본적인 윈도우 이외에도 수 많은 사용자 인터페이스 객체들을 윈도우라 할 수 있다. 메뉴, 팝업 메뉴, 스크롤링 리스트도 윈도우로 분류되며, 대화 상자, Alert, Info 윈도우, 툴 팔렛트도 윈도우라 한다. 사실, 화면으로 불러올 수 있는 객체라면 무엇이든지 윈도우에 나타나야 한다. 그러나, 최종 사용자는 이들을 "윈도우"라고 인식하거나 취급하지 않는다.

2개의 상호작용 시스템이 Cocoa 윈도우를 생성하고, 관리한다. 한편, 1개의 윈도우는 윈도우 서버에 의해 만들어진다. 윈도우 서버는 Quartz 그래픽을 사용하여 윈도우를 불러오고, 크기를 조정하며, 숨기거나 이동시키기 위해 Quartz의 내부 윈도우 관리 포션을 사용하는 프로세스이다. <그림 6-2>에서와 같이 윈도우 서버는 사용자 이벤트(마우스 클릭)를 감지하여, 응용 프로그램으로 전송한다.

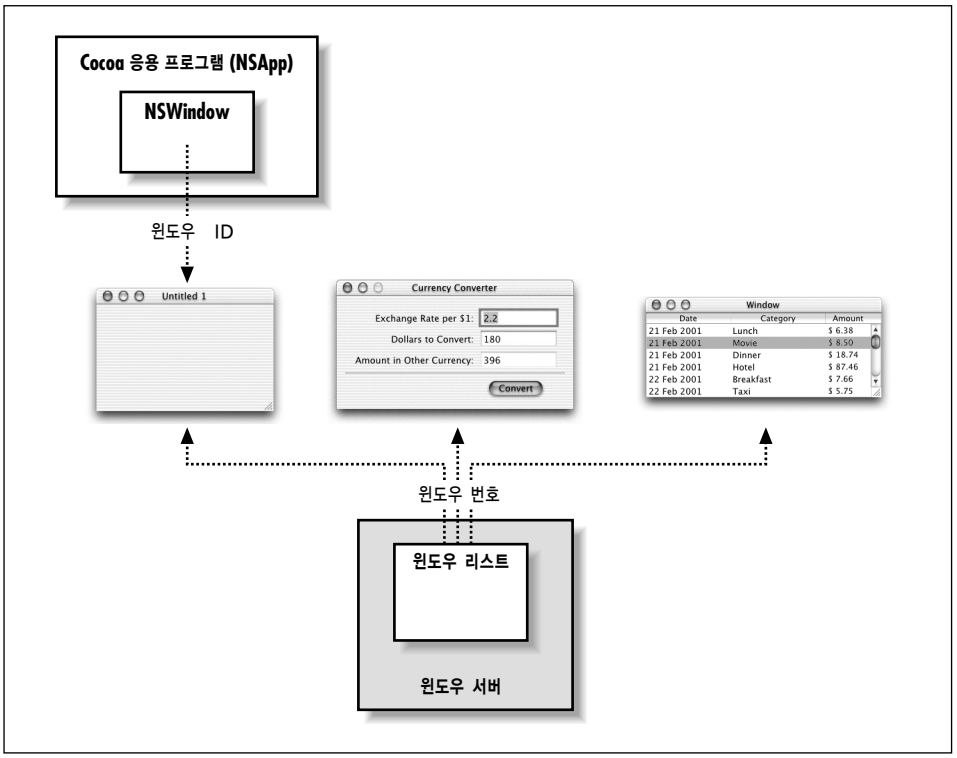



<그림 6-2> Cocoa와 윈도우 서버

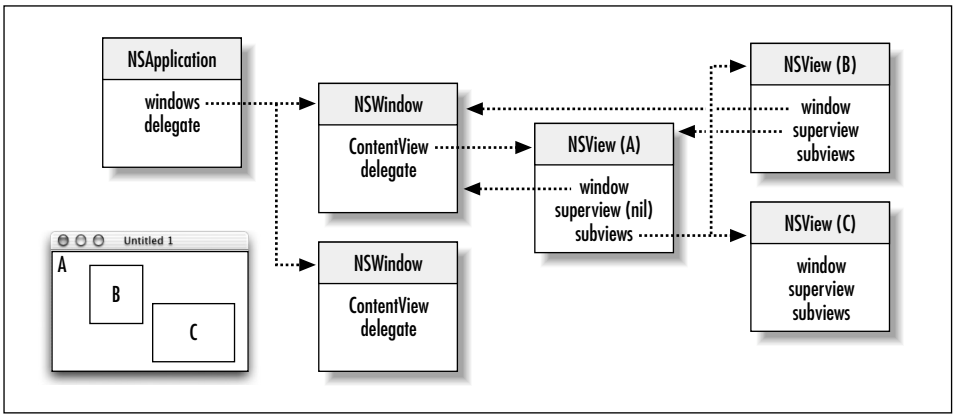
윈도우 서버가 만든 윈도우는 Application Kit(NSWindow 클래스의 인스턴스)에 의해 제공되는 객체와 쌍으로 이루어져 있다. Cocoa 프로그램에서 각각의 물리적 윈도우는 NSWindow 인스턴스(또는 서브클래스)의 관리를 받는다. NSWindow 객체를 만들 경우, 윈도우 서버는 NSWindow 객체가 관리하는 물리적 윈도우를 만든다. 윈도우 서버는 윈도우 번호에 의해 윈도우를, 자신의 식별자에 의해 NSWindow를 각각 참조한다. (<그림 6-3> 참조)

## 응용 프로그램, 윈도우 및 뷰

NSWindow 객체는 실행중인 Cocoa 응용 프로그램에서 NSApplication의 인스턴스와 응용 프로그램과 뷰 사이의 가운데 위치에 차지한다.(뷰는 사용자 이벤트를 스스로 불러오고, 감지할 수 있는 객체이다) NSApplication 객체는 자신의 윈도우 리스트를 가지고 있으며, 각각의 상태를 추적한다. 한편 각 NSWindow는 <그림 6-4>에서와 같이 자신의 윈도우를 비롯해, 뷰 계층을 관리한다.




<그림 6-3> NSWindow 객체와 윈도우 서버




<그림 6-4> 뷰 계층

이 계층의 상단에는 윈도우의 컨텐트 사각형에 들어가는 Content 뷰가 있다. Content 뷰는 Content 뷰 아래에 오는 다른 뷰(Content 뷰의 서브뷰)들을 둘러싼다. NSWindow는 이벤트를 계층에 있는 뷰로 분배하고, 뷰 사이의 좌표를 조절한다.

또 다른 사각형으로는 프레임 사각형이 있다. 이 프레임 사각형은 윈도우의 외부 경계면을 정의하고, 타이틀 바와 윈도우의 컨트롤을 갖추고 있다. Cocoa는 화면의 좌표계에 상응하는 윈도우의 위치를 정의하고, 윈도우의 뷰에 대한 기준 좌표계를 설정하기 위해 프레임 사각형의 하단 좌측의 코너를 사용한다. Carbon과 Classic 응용 프로그램은 좌표계의 좌측 상단 코너를 사용함으로써 프레임 사각형과 차이점을 보인다. 뷰는 이 기준 좌표계에서 변형된 좌표계에 나타난다.

### 키와 메인 윈도우

윈도우에는 무수한 특성이 있다. 윈도우는 온스크린이나 오프스크린이 될 수 있다. 온스크린 윈도우는 윈도우 서버에 의해 관리를 받는 계층에서 화면상에 층을 이룬다. 온스크린 윈도우는 키나 메인의 상태를 전달할 수도 있다.

키 윈도우는 응용 프로그램에서 키 누르기(key press)에 응답하며, 메뉴와 대화 상자로부터 주로 메시지를 수신한다. 일반적으로, 윈도우는 사용자가 키를 클릭할 때 키를 제공한다. 각 응용 프로그램은 단 1개의 키 윈도우만을 포함할 수 있다.

응용 프로그램은 종종 키 상태도 제공할 수 있는 1개의 메인 윈도우를 갖추고 있다. 메인 윈도우는 응용프로그램에 대한 사용자 액션에 중점을 둔다. 종종, 모달(modal) 키 윈도우(일반적으로 Font 대화상자나, Info 윈도우 같은 대화 상자)에서 사용자 액션은 메인 윈도우에 직적접인 영향을 준다.

## Nib 파일

nib 파일은 Interface Builder에 의해 생성된 객체 인스턴스의 아카이브이다. 시스템을 구축하고 있는 수 많은 사용자 인터페이스 제품과는 달리, nib 파일은 코드를 생성하지 않는다. nib 파일은 디스크에서 인코딩되고, 저장되는 진정한 객체이다. nib 파일에 있는 객체들은 Interface Builder의 그래픽 툴을 사용하여 만들어지고, 조작된다.

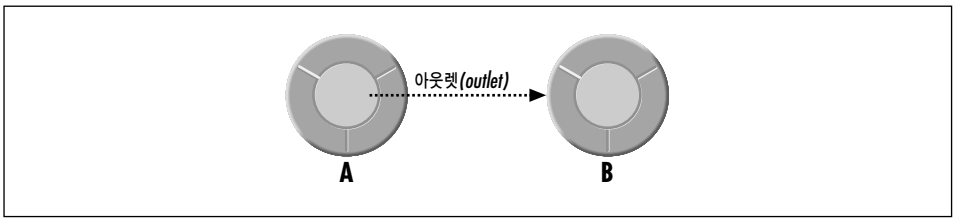
일반적으로, Nib 파일은 객체가 응용 프로그램에서 어떻게 상호연관되는지에 관한 정보를 비롯해, 관련 사용자 인터페이스 객체와 지원 리소스를 일괄적으로 그룹화한다. 그래픽 사용자 인터페이스를 사용하는 모든 응용 프로그램은 응용프로그램이 구동될 때 자동으로 로딩되는 적어도 1개의 nib 파일을 갖추고 있다. 일반적으로, 메인 nib 파일은 응용 프로그램 메뉴를 가지고 있는 반면에 보조 nib 파일은 사용자 인터페이스 객체가 있는 응용 프로그램 윈도우를 제공한다. 응용 프로그램의 인터페이스를 여러 nib 파일로 분리하는 경우에는 사용자 인터페이스의 일부분이 필요할 때만 로딩될 수 있다는 장점이 있다.

nib 파일은 아래에 나와 있는 항목을 1개 이상 제공한다.

- **아카이브된 객체.** 이들 객체는 디스크에 저장될 수 있고,(또는 네트워크 연결을 통해 전송됨) 추후 메모리에서 복구할 수 있는 방식으로 인코딩되어 있으며 객체 지향 객체 용어로 "플래튼된" 또는 "직렬화된" 객체라 한다. 아카이브된 객체는 객체 계층에서 객체의 크기, 위치, 상태등의 정보를 제공한다. 아카이브된 객체의 계층 상단에는 nib 파일을 소유한 실제 객체(일반적으로, 디스크로부터 nib 파일을 로딩한 객체)인 프록시 객체(파일의 소유자 객체)가 있다.
- **클래스 참조와 nib 파일.** Interface Builder는 팔레트에 구성할 수 있는 Cocoa 객체들과 다른 객체들을 저장한다. 그러나, Interface Builder는 코드에 액세스하지 않으므로, 커스텀 클래스의 인스턴스를 저장하는 방식을 인식하지 못한다. 이러한 클래스를 위해, Interface Builder는 클래스 정보를 첨부하여 프록시 객체를 저장한다.
- **커넥터 객체와 nib 파일.** Interface Builder는 객체 계층내 객체들이 어떻게 상호 연결되어 있는지에 관한 정보를 저장하기 위해 특수한 커넥터 객체를 만든다. 도큐멘트를 저장하면, 커넥터 객체는 자신과 연결된 객체와 함께 nib 파일에 저장된다.

## 아웃렛

아웃렛은 다른 객체에 대한 참조를 포함하고 있는 인스턴스 변수이다. 객체는 <그림 6-5>에서와 같이 아웃렛을 통해 메시지를 전송하여 응용 프로그램의 다른 객체와 통신할 수 있다.




*<그림 6-5> 메시지를 전송하기 위해 아웃렛을 사용하기*

아웃렛은 응용 프로그램에서 텍스트 필드와 버튼, 윈도우와 대화 상자, 커스텀 클래스의 인스턴스, 심지어 응용 프로그램 객체등 사용자 인터페이스의 어느 객체나 참조할 수 있다. 다른 인스턴스 변수들과 아웃렛이 구분되는 점은 Interface Builder와의 관계이다.

Interface Builder는 객체간에 연결 라인을 만들어 객체 값을 설정할 수 있도록 헤더 파일에서 아웃렛 선언을 "인식"할 수 있다.

Interface Builder에서 이와 같은 객체간의 관계를 명시하면 개발자가 손으로 초기화 코드를 작성하는 일을 방지할 수 있다. 아웃렛 이외에도 응용 프로그램에서 객체를 참조하는 방법들이 있다. 그러나, 아웃렛과 Interface Builder의 이와 같은 관계가 가장 편리하다고 볼 수 있다.

## *Cocoa Application Project 만들기*

이 섹션에서는 간단한 Cocoa 응용 프로그램을 만들기 위해 Interface Builder와 Project Builder를 사용하는 방법을 설명한다. 응용 프로그램을 실행하면, 응용 프로그램은 텍스트 필드를 포함하고 있는 1개의 윈도우를 제공하고, 텍스트 필드의 현재 시간과 날짜를 보여주기 위해 아웃렛을 사용한다.

1. Project Builder가 실행되어 있지 않으면, 실행시킨다.
2. New Project 커맨드를 선택한다. Project Builder는 <그림 6-6>과 같이 New Project Assistant를 제공한다.


*<그림 6-6> New Project Assistant*

3. 프로젝트 템플리트 리스트에서 Cocoa Application을 선택하고, Next를 클릭한다. Cocoa Application 템플리트는 Hello World에서 사용했던 Foundation Tool 탬플리트와 상당히 유사하며, GUI를 갖춘 응용 프로그램을 위해 시발점을 제공한다. Cocoa Application 템플리트는 윈도우와 메뉴 바를 갖춘 nib 파일을 기본적으로 지원한다.
4. Project Name 필드에 **`Nib Files`**을 입력하고, Finish를 클릭한다.

Project Builder는 프로젝트를 구성하기 위해 계층적 그룹을 사용한다. Cocoa 응용 프로그램을 위한 기본적인 그룹은 아래와 같다.

- **Classes.** 이 그룹은 처음에는 비어있으나, 프로젝트의 클래스용으로 구현(`.m`)과 헤더(`.h`) 파일을 수용하기 위해 사용된다.
- **Other sources.** 이 그룹은 초기 자원을 로딩하고, 응용 프로그램을 실행시키는 `main` 함수인 `main.m`을 포함한다.(이 파일은 변경하지 않아야 한다)
- **Resources.** 이 그룹은 `nib` 파일(`.nib`)과 응용 프로그램의 사용자 인터페이스를 명시하는 Other Resource를 포함한다.
- **Frameworks.** 이 그룹은 응용 프로그램이 시스템 서비스에 접근하기 위해 임포트하는 프레임워크(라이브러리와 유사)에 대한 참조를 포함한다.
- **Products.** 이 그룹은 프로젝트 빌드의 결과를 제공하고, 프로젝트에서 각 타켓에 의해 생성된 프로덕트의 참조를 자동으로 상주시킨다.

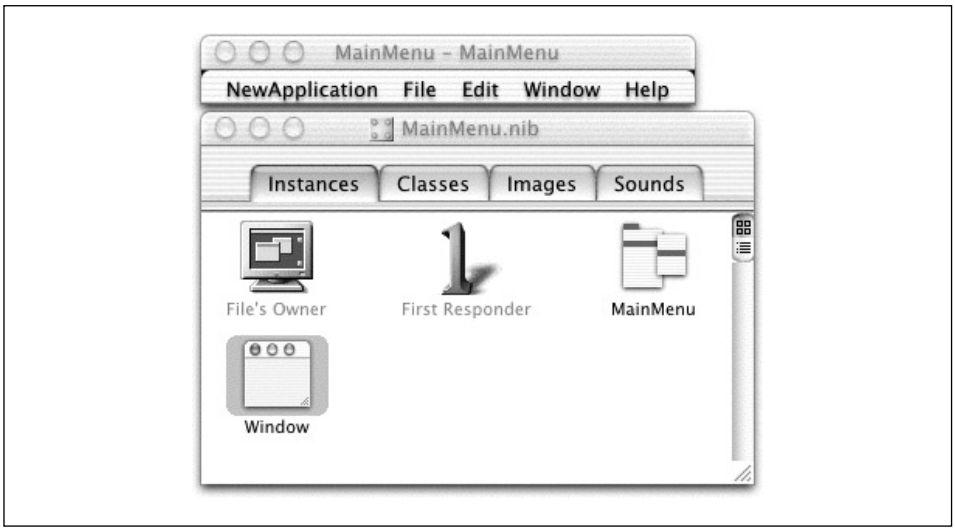
이들 그룹은 프로젝트의 온디스크 레이아웃이나 빌드 시스템이 파일을 처리하는 방식에 반드시 영향을 미치지는 않는다는 점에서 상당히 유연하다고 볼 수 있다. 탬플리트에 의해 만들어진 기본적인 그룹은 그 자체로 사용할 수 있으며, 개발자가 원하는 대로 재배열할 수도 있다.

## *Main Nib File 열기*

사용자 인터페이스를 구축하려면, Interface Builder에서 응용 프로그램의 메인 nib 파일을 열어야 한다.

1. 이전에 만들어둔 `Nib Files.pbproj`를 Finder에 배치하여 연다.
2. Project Builder의 메인 윈도우에서 Groups & Files 리스트의 Resource 그룹에 있는 `MainMenu.nib`를 더블 클릭한다. 그러면, Interface Builder가 실행(Interface Builder가 실행되지 않았다면)되고, nib 파일이 열린다.

Interface Builder가 nib 파일을 실행시키면 <그림 6-7>에서와 같이 기본적인 메뉴 바와 윈도우가 나타난다.

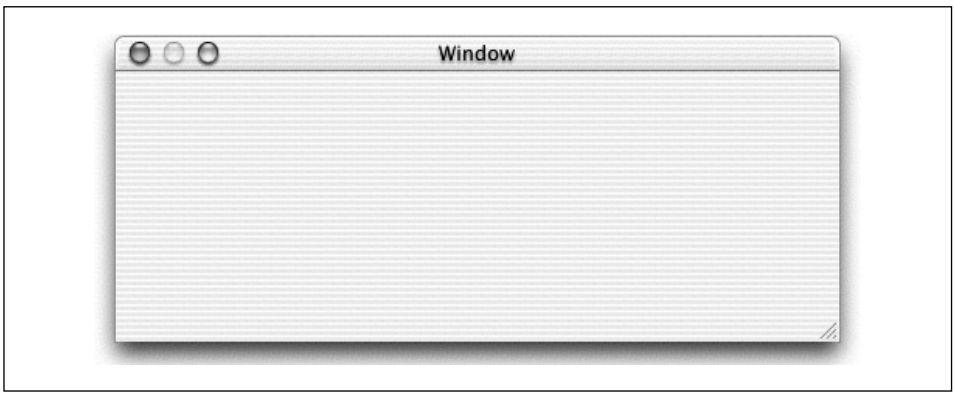



<그림 6-7> 새로운 Cocoa 응용 프로그램의 기본적인 메뉴 바와 nib 파일

## 윈도우를 이동시키고 크기를 조정하기

Interface Builder는 nib 파일에서 사용자 인터페이스 객체에 관한 모든 종류의 정보를 저장한다. 예를 들어, Interface Builder에서 윈도우의 크기를 조정하고, 이동시킴으로써 응용 프로그램의 메인 윈도우의 크기와 처음 위치를 정할 수 있다.

1. 윈도우의 타이틀 바를 드래그하여 화면의 상단 좌측 코너로 윈도우를 이동시킨다.
2. <그림 6-8>에서와 같이 윈도우의 하단 우측 코너에서 크기 조정 컨트롤을 사용하여 윈도우를 작게 만든다.

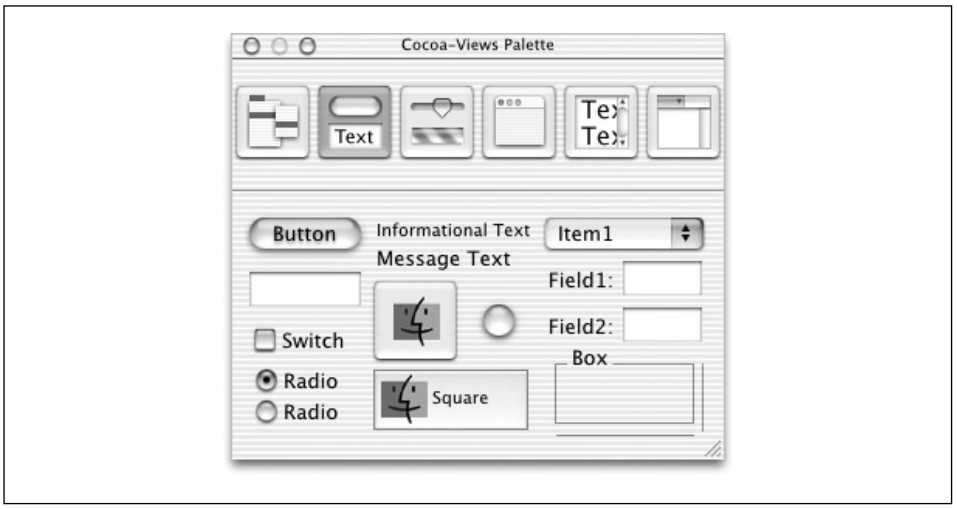



<그림 6-8> Cocoa 윈도우의 크기 조정 컨트롤

## Text Field 추가하기

이제, 텍스트 필드 객체를 응용 프로그램의 윈도우에 추가한다.

1. Cocoa 객체 윈도우의 상단 좌측에서 두번째 버튼을 클릭하면 Views 팔레트가 선택된다. Cocoa 팔레트 윈도우 <그림 6-9>가 나타나지 않으면, 앞으로 넘어가 Tools 메뉴에서 Palettes를 선택한다.



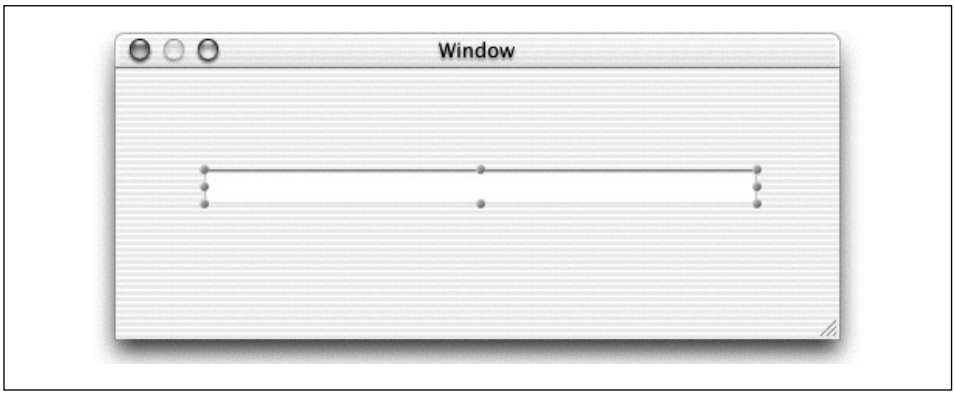

<그림 6-9> *Interface Builder의 Views Palette*

2. 텍스트 필드 객체를 윈도우로 드래그한다. 그러면, Views 팔레트 좌측의 Button 객체 아래에 텍스트 필드 객체가 놓인다.
3. <그림 6-10>에서와 같이 오른쪽 핸들을 잡고, 오른쪽으로 드래그하여 텍스트 필드가 넓어지도록 크기를 조정한다.

## Custom Subclass 만들기

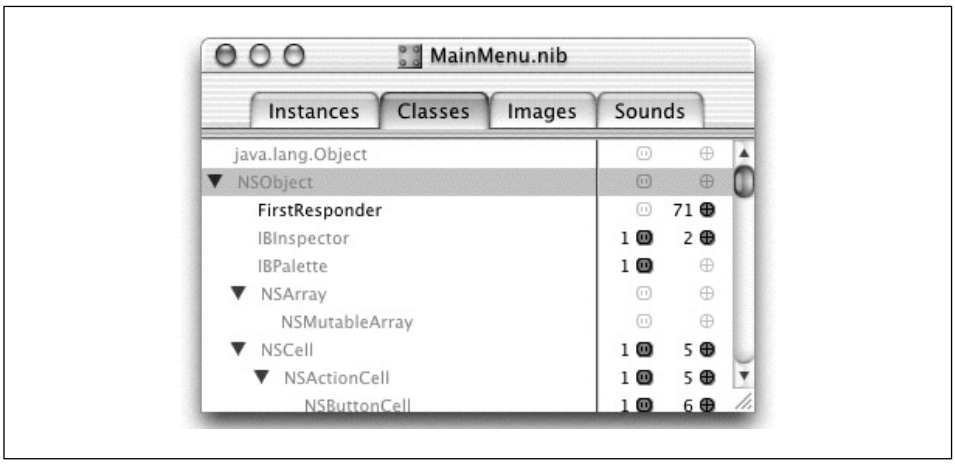
객체 지향 응용 프로그램은 사용자 인터페이스 객체로부터 사용자 입력에 반응하고, 어떤 동작을 수행하기 위한 역할(종종 역할을 다른 객체에 위임)을 담당하는 *controller* 클래스를 1개 이상 포함한다. 이 응용 프로그램에서 controller 클래스는 텍스트를 보여주기 위해 이를 텍스트 필드 객체에 전송하는 역할을 수행한다.

새로운 클래스를 정의하려면, `MainMenu.nib` 윈도우의 Classes 패인으로 가야한다. Classes 패널에서 해야 할 작업은 새로운 서브클래스가 상속할 슈퍼클래스를 선택하는 일이다.




<그림 6-10> 텍스트 필드 크기 조정하기

1. `MainMenu.nib` 윈도우에서 Classes 탭을 클릭하여 Classes 패인을 선택한다.
2. 새로운 서브클래스는 복잡한 동작을 상속할 필요는 없으므로, 클래스 리스트(<그림 6-11>에서와 같이 리스트 상단에 있는)에서 NSObject를 선택한다. NSObject는 Cocoa 객체로 동작하기 위해 필요한 모든 기능을 제공한다.




<그림 6-11> NSObject 서브클래스 만들기

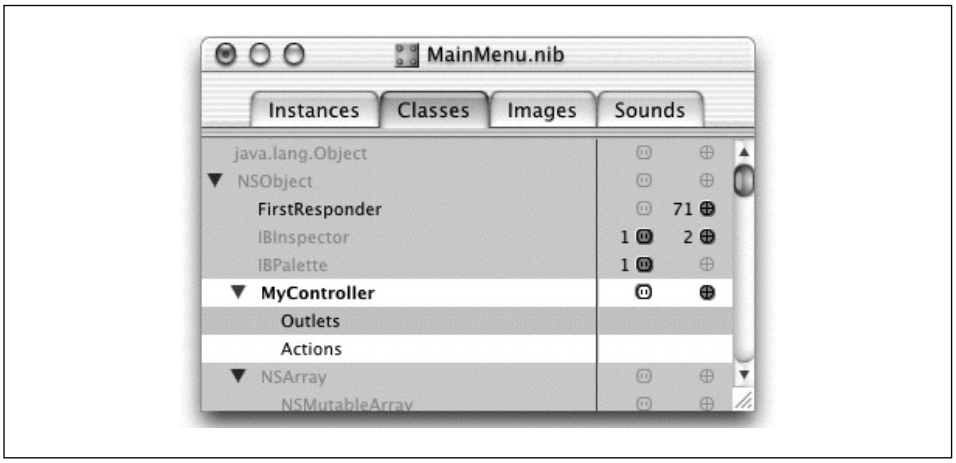
3. Classes 메뉴에서 Subclass를 선택한다.
4. MyObject 텍스트를 바꾸려면 **`MyController`**를 입력하고, Return을 누른다.

이제, 클래스가 nib 파일내 클래스 계층에 구축되었다.

## 클래스를 위한 아웃렛 정의하기

MyController는 메인 윈도우에 있는 텍스트 필드로 메시지를 전송하는 방법을 필요로 한다. 이 같은 용도의 아웃렛을 만들려면 Interface Builder를 사용한다.

1. `MainMenu.nib` 윈도우의 Classes 패인에서 MyController를 선택한다.
2. 클래스 오른쪽의 전자 아웃렛 아이콘을 클릭한다.(<그림 6-12> 참조)




*<그림 6-12> 아웃렛 추가하기*

3. Classes 메뉴로부터 Add Outlet을 선택한다.
4. 이 아웃렛의 이름을 `textField`라 정하고, Return을 누른다.

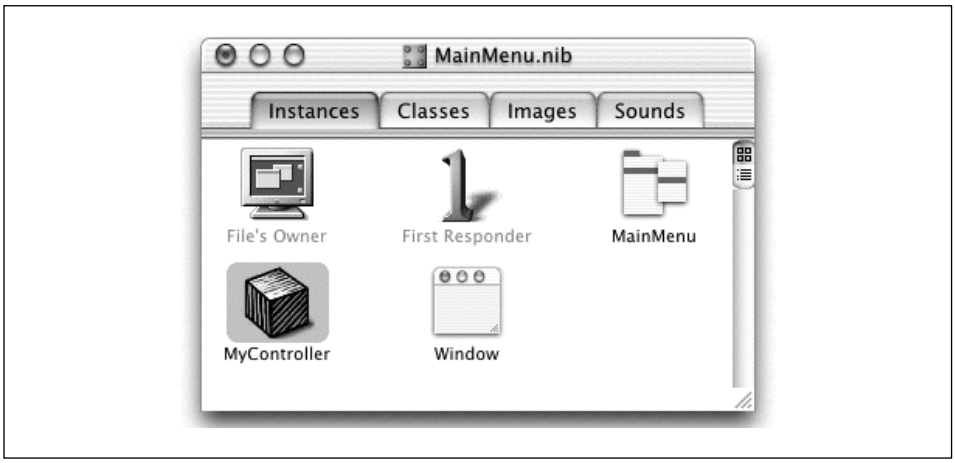
## Class의 인스턴스 만들기

Interface Builder에서 클래스를 정의하는 마지막 단계로는 클래스의 인스턴스를 만들어, 이 인스턴스를 nib 파일에 있는 다른 객체에 연결하는 것이다.

1. `MainMenu.nib` 윈도우의 Classes 패인에서 MyController를 선택한다.
2. Classes 메뉴에서 Instantiate를 선택한다.

클래스를 인스턴스화하면,(즉 클래스의 인스턴스를 만들면) Interface Builder는 Instances 패인으로 전환되며,(<그림 6-13> 참조) 클래스의 이름을 가진 새로운 인스턴스가 하이라이트된다.

사실, Instantiate 커맨드는 MyController의 진짜 인스턴스를 만들지 않는다. Instantiate 커맨드는 nib 파일에서 다른 객체로의 연결을 정의하기 위해 Interface Builder내에서 사용된 프록시 객체를 만드는 것이다. 응용 프로그램을 구동하고, nib 파일의 내용이 언아카이브될 때, 런타임 시스템은 MyController의 진짜 인스턴스를 만들고, nib 파일에서 다른 객체로의 연결을 구축하기 위해 프록시 객체를 사용한다.




<그림 6-13> Interface Builder의 Instances 패인

## 커스텀 클래스를 Interface로 연결하기

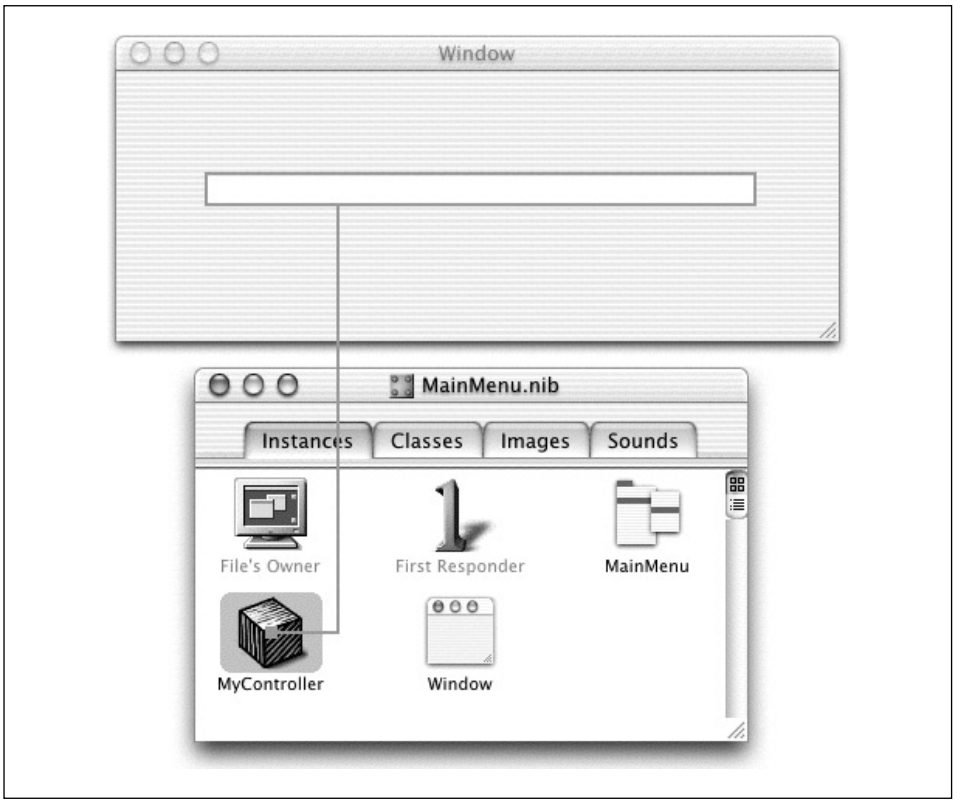
MyController의 프록시 인스턴스를 만들었으므로, 먼저 만들어 놓은 텍스트 필드와 클래스 간의 연결을 선언하기 위해 프록시 인스턴스를 사용한다.

1. `MainMenu.nib` 윈도우의 Instances 패인에서 MyController 인스턴스로부터 텍스트 필드로 연결 라인을 Control 키를 누른채 드래그한다. 텍스트 필드의 윤곽이 잡히면(<그림 6-14> 참조) 마우스 버튼을 해제한다.
2. <그림 6-15>에서와 같이 Interface Builder는 Custom Object Info 윈도우의 Connections 패인을 불러온다.
3. Connect 버튼을 클릭한다.
4. nib 파일을 저장한다.(Command-S)

이제 이 연결이 생성되어, 응용 프로그램에서 nib 파일이 로딩되면 런타임 시스템은 MyController의 `textField` 아웃렛(기억할 점은, 아웃렛은 인스턴스 변수일 뿐이다) 이 윈도우의 텍스트 필드 객체를 참조하기 위해 초기화되어야 한다는 것을 인식한다.

## Source Files 만들기

Interface Builder는 개발자가 만든(일부) 클래스 정의로부터 소스 코드 파일을 만든다. 이 파일들은 필수적인 Objective-C 지시문과 클래스 정의에 비해 적은 정보를 제공한다는 점에서 골격구조만을 이루고 있다. 그러므로, 코드를 작성하여 이 파일들을 보충해야 한다. 일부 인터페이스 구축 툴과는 달리, Interface Builder에 의해 만들어진 이 파일들은 인터페이스를 만들기 위한 코드를 제공하지 않는다. 이 모든 내용은 nib 파일에서 객체 아카이브에 저장된다.



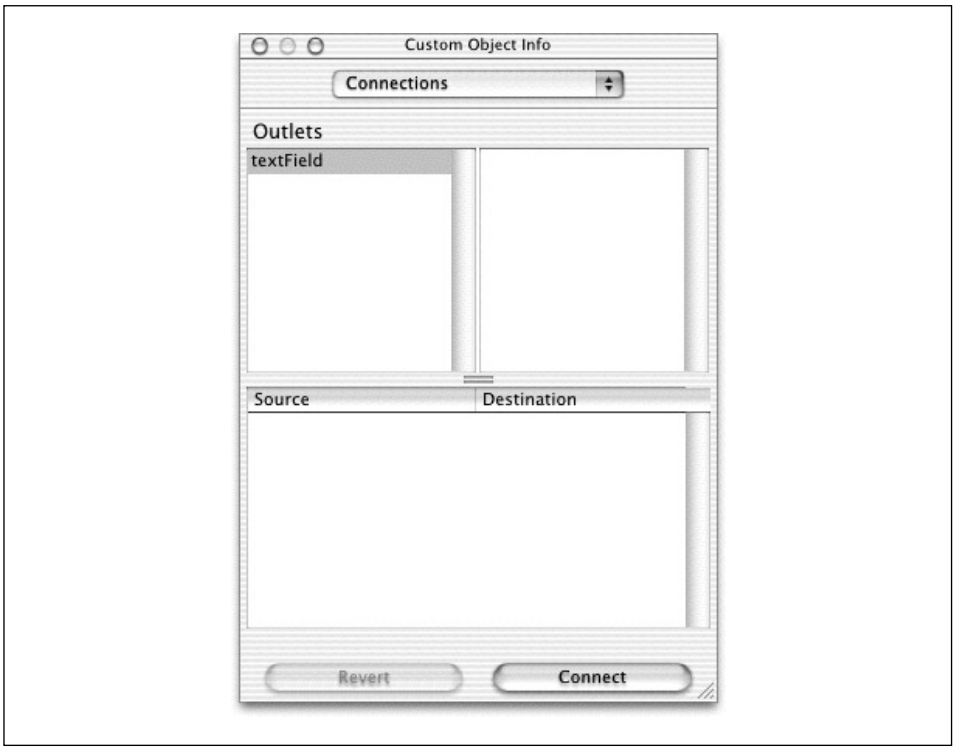

<그림 6-14> 인스턴스를 텍스트 필드에 연결하기

1. nib 파일윈도우의 Classes 패인으로 간다.
2. MyController 클래스를 선택한다.
3. Classes 메뉴에서 Create Files을 선택한다.

그러면, Interface Builder는 <그림 6-16>에서와 같은 대화 상자를 제공한다.

1. .h와 .m 파일 옆의 Create 세로열에서 체크 박스를 선택하였는지 확인한다.
2. Nib Files 옆의 체크 박스를 선택하였는지 확인한다.
3. Choose 버튼을 클릭한다.

이제, Nib Files 응용 프로그램에 대한 Interface Builder를 종료하고, Project Builder를 사용하여 응용 프로그램을 완성하도록 한다.



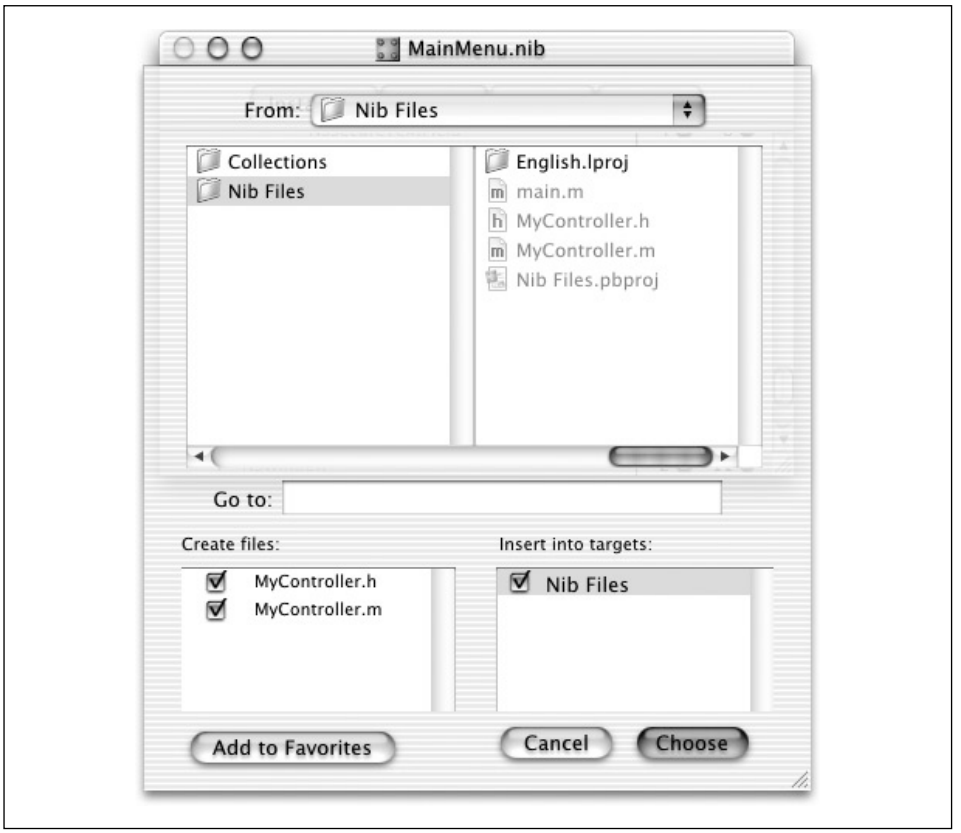

<그림 6-15> Custom Object Info 윈도우

## Project Builder에서 Header File 검토하기

Interface Builder가 소스 파일을 nib 파일 프로젝트에 추가하면, Project Builder는 기존의 그룹을 검토하여 새로운 파일이 어떤 그룹에 속하는지 알아낸다.Project Builder는 다른 소스 파일을 수용하고 있는 그룹에 소스파일을 추가하거나, 다른 소스 파일이 프로젝트에 없다면, 상위 레벨에 소스파일을 추가한다. Project Builder의 그룹은 무작위로 구성된 컨테이너이므로, 개발자가 원하는 대로 파일을 배열할 수 있다. 그러나, 이 파일들은 클래스의 인터페이스 및 구현이기 때문에 Classes 그룹에 배치되어야 한다.

1. Project Builder의 메인 윈도우를 활성화한다.
2. `MyController.h`와 `MyController.m`을 Classes 그룹으로 드래그한다.

리스트에서 파일명을 클릭하면 Project Builder 메인 윈도우의 우측에 있는 코드 편집기에 내용이 나타난다.




<그림 6-16> Create Files 대화 상자

## 정적으로 아웃렛 유형화하기

아웃렛 선언은 기본적으로 id 키워드를 사용하여 동적으로 유형화된다. id는 어느 객체나 유형화하기 위해 사용할 수 있다. 이는 객체의 클레스가 런타임시 결정된다는것을 의미한다. 동적으로 객체를 유형화할 필요가 없을 경우, 객체의 포인터로 아웃렛을 정적으로 유형화할 수 있다(또는, 유형화해야 한다). 정적으로 유형화하는 작업은 여분의 시간이 소요되지만, 프로그래밍의 실행을 원활하게 한다. 이 작업은 궁극적으로 디버깅 시간을 줄이면서 컴파일러로 하여금 유형 검사를 수행할 수 있도록 한다.

일반적인 아웃렛은 다음과 같이 선언된다.

```
IBOutlet id variableName;
```

선언을 변경한다.

```
MyController.h
```

다음과 같이 해석한다.

```
IBOutlet NSTextField *textField;
```

## MyController의 awakeFromNib 메소드 구현하기

응용 프로그램이 구동되면, NSApplicationMain 함수는 메인 nib 파일을 로딩한다. nib 파일이 완전히 열리고, nib 파일의 객체가 연결된 이후에 런타임 시스템은 로딩 과정이 완료되었다는 신호를 알리면서, nib 파일의 정보로부터 파생된 모든 객체에 awakeFromNib 메시지를 전송한다. 모든 객체의 아웃렛은 awakeFromNib이 호출되었을 때 초기화되어야 한다. 이로써, nib 파일의 객체들은 사용자나 나머지 응용 프로그램이 객체들과 상호작용하려는 시도를 하기 전에 필요로 하는 여분의 설정을 수행할 수 있다.

이 예제에서는 응용 프로그램의 메인 윈도우에 있는 텍스트 필드로 메시지를 출력하기 위해 MyController의 awakeFromNib 메소드를 사용한다.

MyController.h에서 awakeFromNib 메소드를 선언한다.

1. Project Builder 메인 윈도우의 Groups & Files 리스트에 있는 클래스 그룹에서 MyController.h를 선택한다.
2. 인스턴스 변수 선언과 @end 사이에 선언을 삽입한다.

```
@interface MyController : NSObject
{
    IBOutlet NSTextField *textField;
}
- (void)awakeFromNib;
@end
```

메소드 구현은 @implementation <class name>과 @end 사이의 MyController.m에서 이루어진다. 여기에 MyController의 awakeFromNib 메소드를 위한 코드를 추가한다.

1. Project Builder 메인 윈도우의 Classes 그룹에서 MyController.m을 선택한다.
2. awakeFromNib 코드를 삽입한다.

```
@implementation MyController
- (void)awakeFromNib
{
    [textField setObjectValue:[NSCalendarDate date]];
}
@end
```

awakeFromNib 메소드 구현은 단순히 현재 시간과 날짜를 텍스트 필드에 전송된다.

## Nib 파일을 구축, 디버그 및 실행하기

Project Builder에서 Build 버튼을 클릭하면, 컴파일을 조정하고, 실행 파일을 불러오는 절차를 링크하는 빌드 프로세스가 발생된다. 또한, 응용 프로그램 구축에 필요한 다른 작업도 수행한다.

빌드 프로세스는 객체의 소스 코드 파일을 전달하기 위해 컴파일러를 발생시킨다. 이 파일들(Objective-C, C++ 및 표준 C)을 컴파일하면 아키텍처를 위한 기계 가독형 객체 파일이나 빌드를 위해 지정된 아키텍처가 만들어진다.

빌드의 연결 단계에서 빌드 툴은 객체 파일 대신에 라이브러와 프레임워크를 링크에 전달하기 위해 링커를 실행한다. 프레임워크와 라이브러리는 어느 응용 프로그램에 의해서도 사용될 수 있는 미리 컴파일된 코드를 갖추고 있다. 연결 단계에서는 응용 프로그램의 실행 파일을 만들기 위해 라이브러리, 프레임워크와 객체 파일에 코드를 통합한다.

빌드 프로세스가 진행되는 동안, nib 파일, 사운드, 이미지, 그리고 다른 리소스는 프로젝트로부터 응용 프로그램 패키지의 로컬라이즈되었거나 로컬라이즈되지 않은 해당 장소에 복사된다. 응용 프로그램 패키지는 실행 응용 프로그램과 응용 프로그램을 실행하기 위해 필요한 리소스를 수용하고 있는 디렉토리이다. 이 디렉토리는 Finder에서 1개의 파일(응용 프로그램을 구동하기 위해 더블 클릭하면 됨)로 나타난다.

### 프로젝트 구축하기

Build 버튼을 클릭하면, 빌드 프로세스가 시작된다. Project Builder가 작업을 완료하면(작업 중 오류가 발생되지 않아야 함) 메인 윈도우의 좌측 하단에 Build Succeeded가 나타난다.

1. 소스 코드 파일과 프로젝트의 변경 사항을 저장한다.
2. 메인 윈도우에서 Build 버튼을 클릭한다.

### 프로젝트 디버그하기

물론, 프로젝트가 처음부터 완벽하게 구동되는 경우는 드물다. 프로젝트를 처음으로 구축하면, Project Builder는 어떤 오류를 잡아낼 수 있다. Project Builder의 오류 위치 기능을 확인해보려면, 미미한 버그를 코드에 입력해본다.

1. 오류를 만들기 위해 코드에서 세미콜론을 삭제한다.
2. 메인 윈도우에서 Build 버튼을 클릭한다.

3. 빌드 오류 브라우저에 나타나는 오류 통지 라인을 클릭한다.
4. 오류를 정정한다.
5. 프로젝트를 재구축한다.

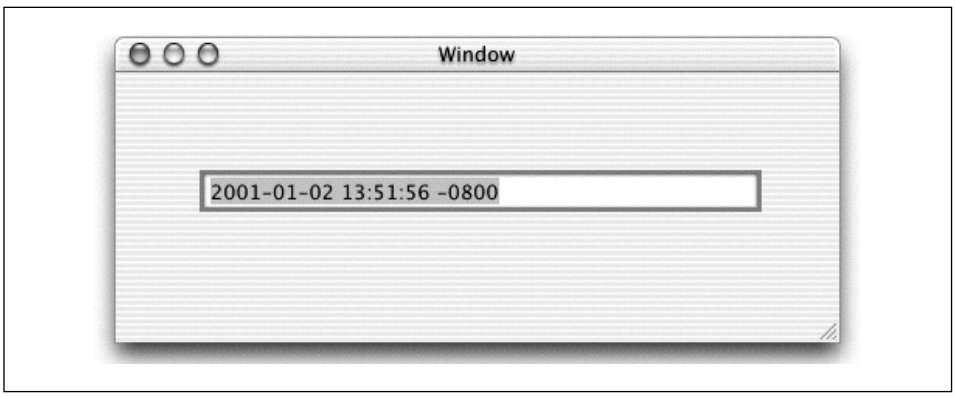
Cocoa 응용 프로그램의 개발과 디버깅이 다른 환경과 본질적으로 다른 한 가지 측면은 Interface Builder가 프로세스에서 수행하는 역할이다. 응용 프로그램의 수 많은 런타임 동작은 Interface Builder에서 만든 연결에 의해 결정되어지므로, 개발자는 코드뿐만 아니라 nib 파일도 점검해야 한다.

예를 들면, Nib Files 응용 프로그램을 운용할 때 장애가 발생하지 않는다면, 코드에 문제가 없어서 일 수 도 있고, Interface Builder에서 객체들을 정상적으로 연결하지 않았기 때문일 수 도 있다. 여기서 기억할 점은, 장애가 발생하면 먼저 장애의 원인에 대해 고심해야 한다. 그리고 난 후, 문제의 근원을 찾기 위해 Interface Builder를 점검할 것인지, 코드를 점검할 것인지 결정한다.

## *응용 프로그램 실행하기*

응용 프로그램을 구동하려면 Project Builder의 Run 버튼을 클릭한다.

원한다면, Finder에 응용 프로그램을 배치할 수 있다.(기본적인 위치를 변경하지 않는다면, Nib Files 프로젝트의 빌드 서브디렉토리에 배치되어 있음) Finder를 더블 클릭하여 실행시킨다. 이 경우, <그림 6-17>과 같은 화면이 나타나야 한다.

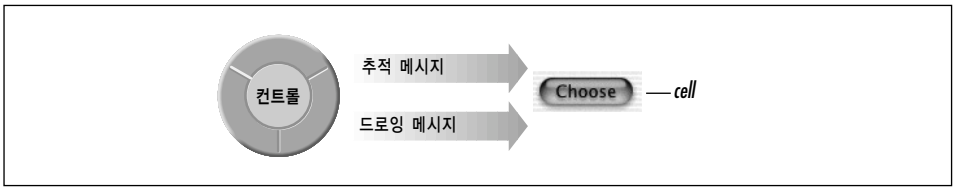



*<그림 6-17> Nib 파일의 최종 출력*

# 컨트롤, 셀 및 포매터

컨트롤과 셀(cell)은 버튼, 텍스트 필드, 슬라이더, 브라우저를 비롯한 Cocoa의 대다수 사용자 인터페이스의 외양과 동작 이면에 자리한다. 컨트롤과 셀은 상당히 다른 유형의 객체들이지만, 상호 밀접하게 작용한다.

컨트롤은 사용자의 의향을 응용 프로그램에 알려서, 발생된 상황을 컨트롤할 수 있도록 한다. 셀은 컨트롤내에 임베디드된 사각형 영역이다. 어떤 컨트롤은 표면을 동작 영역으로 분할하기 위한 방안으로 다양한 셀을 수용한다. 셀은 텍스트나 이미지로 내용을 표현할 수 있으며(때로는, 텍스트와 이미지 모두로 표현함), 독립적으로 사용자 액션에 반응할 수 있다. <그림 6-18>은 컨트롤과 셀과의 관계를 보여준다.




*<그림 6-18> 컨트롤과 셀*

컨트롤은 컨트롤 영역에서 사용자 이벤트(마우스 클릭이나 키스트로크)가 발생할때 언제 어디로 불러올것인지를 알려주고, 통보하기 위해 셀의 관리자로 동작한다. 셀과 컨트롤의 "중요성"을 고려해보면, 이러한 역할의 분할은 메모리를 보호하고, 응용 프로그램의 성능을 상당히 향상시킨다. 예를 들면, 버튼의 매트릭스는 수 많은 독립적인 컨트롤보다는 수 많은 셀을 가진 1개의 컨트롤로 구현될 수 있다.

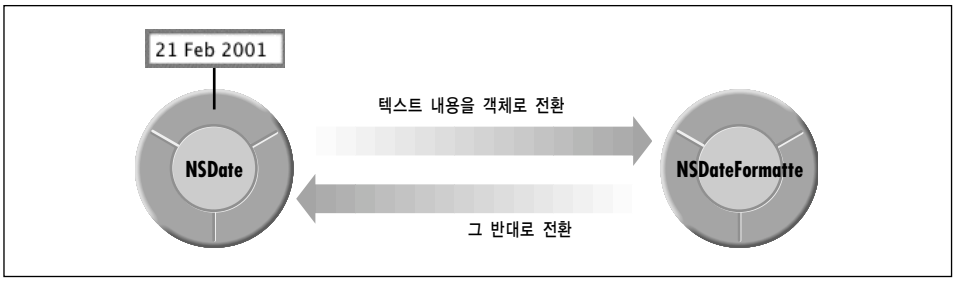
컨트롤은 자신과 연결된 셀을 가질 필요는 없지만, Interface Builder의 기본 팔레트에서 사용할 수 있는 대부분의 사용자 인터페이스 객체들은 셀과 컨트롤이 연결되어 있다. 심지어 단순한 버튼(프로그램적으로 만들어졌거나 Interface Builder에서 만들어진 버튼)도 컨트롤(NSButton 인스턴스)이 NSButtonCell와 연결되어 있다. 매트릭스와 같이 컨트롤에 있는 셀은 같은 크기여야 하지만, 클래스는 다를 수 있다. 테이블 뷰나 브라우저와 같은 더욱 복잡한 컨트롤은 셀의 다양한 크기와 유형을 결합할 수 있다. NSButton 같은 1개의 셀을 사용하는 대부분의 컨트롤은 편리한 메소드를 제공하므로, 개발자가 직접적으로 셀을 다룰 필요는 없다.

## 셀 및 포매터

셀의 내용을 고려할 때, 텍스트(NSString)와 이미지(NSImage)만을 고려하는 것은 당연하다. 내용은 무엇이든지 보여줄 수 있는 것으로 생각할 수 있다. 그러나 셀은 날짜(NSDate), 숫자(NSNumber), 커스텀 객체(즉, 전화 번호 객체)와 같은 다른 종류의 객체들을 수용할 수 있다.

응용 프로그램의 사용자 인터페이스를 더욱 훌륭하게 만들 수 있는 한 방법으로는 환율이나 숫자 데이터를 제공하는 필드의 항목을 구성하는 것이다. 필드는 10진법을 제공하고, 숫자를 특정 영역으로 제한하며, 통화 심볼을 나타내거나 특정 색상으로 음수값을 보여줄 수 있다.

포매터는 어떤 객체 값을 특정한 온스크린 표현방식으로 전환하는 객체이며, 사용자 인터페이스에서 포맷된 스트링을 표시된 객체로 전환할 수도 있다. 예를 들면, <그림 6-19>는 날짜 포매터가 NSDate 객체의 항목을 특정 스트링으로 전환하는 방법을 보여준다.




*<그림 6-19> 날짜 포매터*

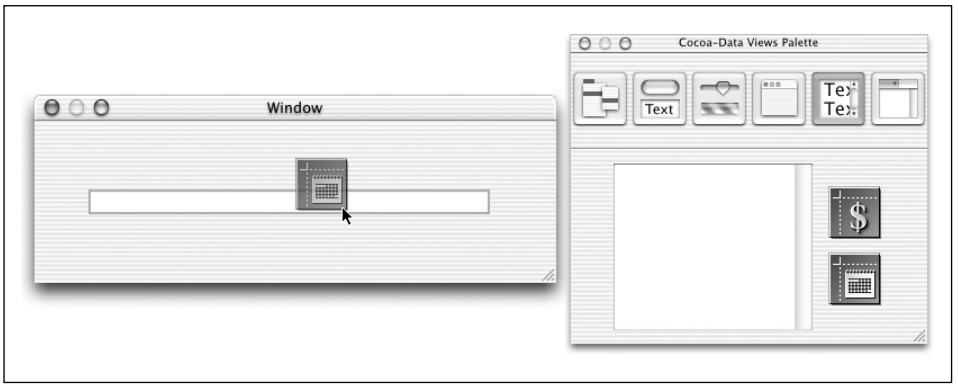
프로그램적으로 또는 Interface Builder를 사용하여 포매터 객체를 만들고, 설정하고, 변경할 수 있다. 자신만의 특수한 포매터 객체(즉, 전화 번호)를 만들어, 팔레트에 추가할 수 있다.

포매터 객체는 셀에 연결된 객체의 텍스트 표현방식을 처리하며, 셀에 입력된 내용을 기본 객체로 변환할 수 있다. 프로그램적으로 포매터를 셀에 연결하기 위해 Interface Builder에서 포매터 객체를 셀에 첨부하거나 NSCell의 `setFormatter:` 메소드를 사용할 수 있다.

## 포맷된 셀의 예제

이 예제에서는 텍스트 셀이 출력되기 전에 날짜를 포맷할 수 있도록 이전 섹션에서 만든 프로젝트(nib 파일)를 변경하는 방법을 제공한다. Interface Builder는 기본 팔레트에서 2개의 포매터 객체(날짜를 포맷하는 포매터와 숫자를 포맷하는 포매터)를 제공한다. 다음은 날짜를 포맷하는 포매터를 사용하는 방법을 설명한다.

1. Nib Files 프로젝트를 연다.
2. Inferface Builder에서 `MainMenu.nib` 파일을 열기 위해 더블 클릭한다.
3. 팔레트 윈도우에서 Data Views Palette를 선택한다.
4. <그림 6-20>과 같이 날짜 포매터 객체를 텍스트 필드에 드래그한다.




<그림 6-20> 데이터 포매터를 텍스트 필드에 추가하기

5. 텍스트 필드를 선택하는 한편, Info 윈도우가 보이지 않으면, Info 윈도우(Command-Shift-I)를 불러온다.
6. <그림 6-21>과 같이 Info 윈도우의 Formatter 패인에서 **%c** 포맷을 가진 가로행을 선택하여 날짜 포맷을 지정한다.
7. nib 파일을 저장한다.
8. 프로젝트를 구축하고 운용한다. <그림 6-22>와 같은 내용이 나타나야 한다.

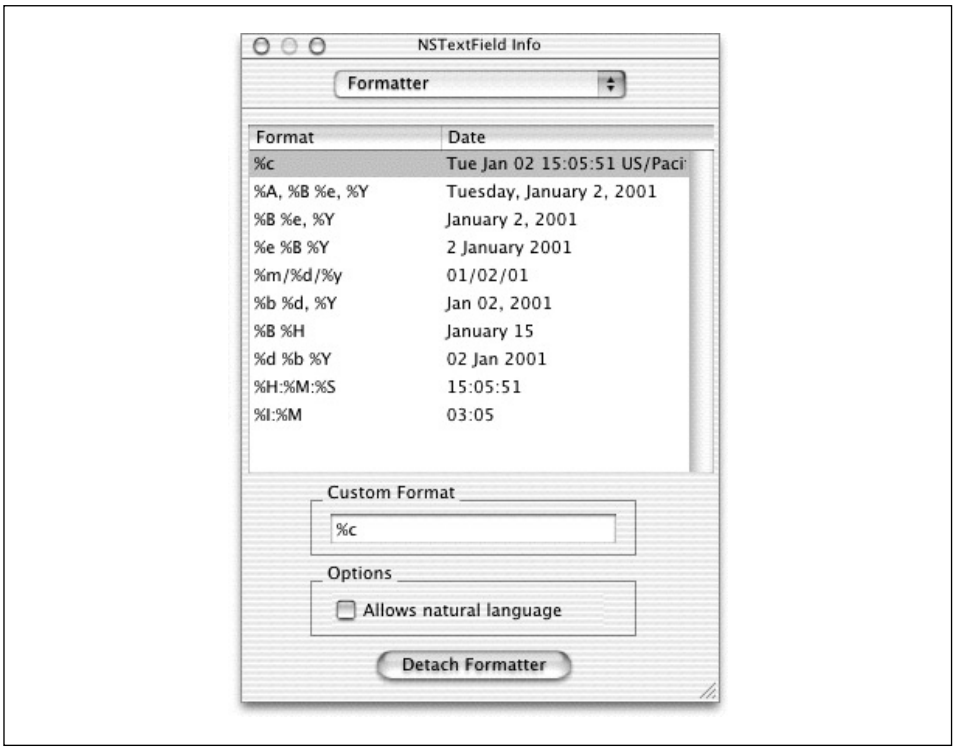
## 타켓/액션

타켓/액션 패턴은 응용 프로그램에 사용자의 의향이 전달될 수 있도록 사용자 인터페이스 컨트롤에 의해 사용자 액션에 응답하는 메카니즘의 일부이다. 타켓/액션 패턴은 컨트롤(구체적으로 컨트롤의 셀)과 컨트롤의 타겟간의 관계가 1대1임을 명시한다. 사용자가 사용자 인터페이스 컨트롤을 클릭하면, <그림 6-23>과 같이 컨트롤은 액션 메시지를 타겟 객체에 전송한다.

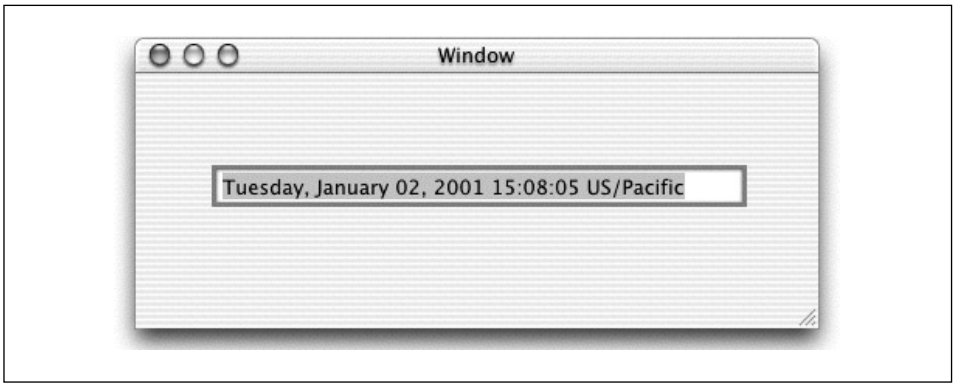
일반적으로, 타켓/액션 관계는 Interface Builder를 사용하여 정의된다. Interface Builder에서 타켓 객체로 전송될 특정 액션 메시지와 함께 컨트롤에 대한 타겟 객체를 선택한다. 또한, 타켓/액션 관계는 응용 프로그램이 운용되는 동안에 설정(또는 변경)될 수 있다.

### 타켓/액션 예제

다음 단계에서는 타켓/액션 패턴을 사용하는 간단한 응용 프로그램을 구축하는 과정을 설명한다. 이 예제에서 메인 윈도우에 있는 버튼을 클릭하면, 날짜 및 시간이 텍스트 필드에서 업데이트된다.



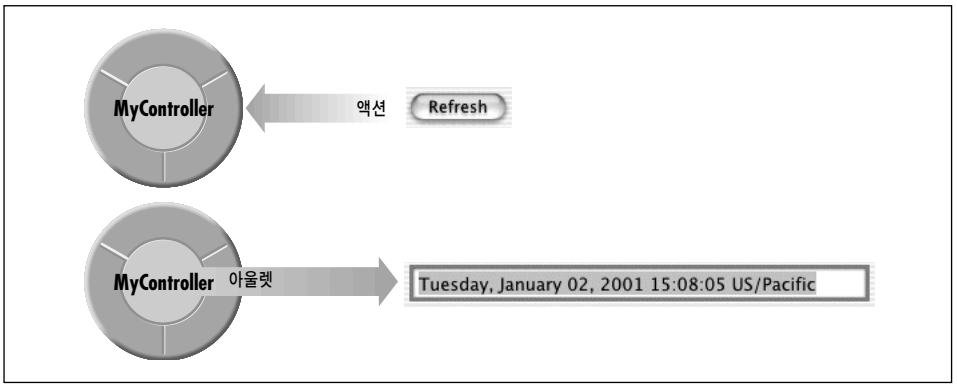

<그림 6-21> 날짜 포매터 구성하기




<그림 6-22> 날짜 포매터를 사용한 Nib 파일 응용 프로그램

## 프로젝트와 사용자 인터페이스 만들기

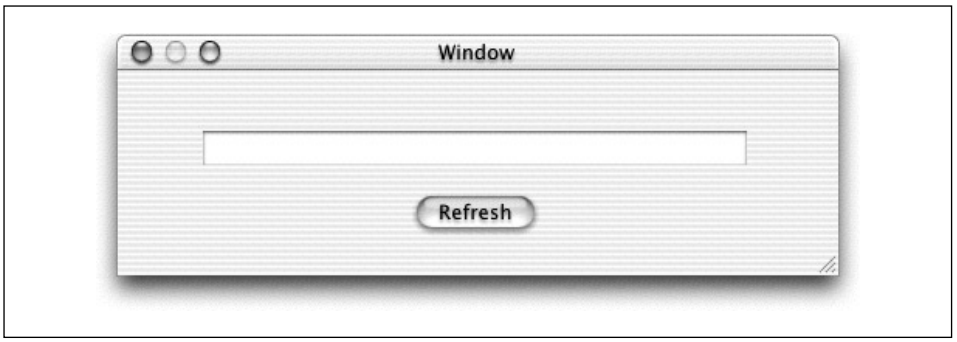
첫번째 예제에서는 Project Builder를 사용하여 프로젝트를 만드는 방법을 상세하게 설명했으므로, 이 예제에서 다시 반복하지 않는다. 그 단계를 수행하는 방법을 잊어버렸다면, "Cocoa에서 그래픽 사용자 인터페이스 만들기" 섹션의 Nib Files 프로젝트를 참조한다.




<그림 6-23> 타켓/액션

1. Project Builder가 실행되어 있지 않다면, 실행시킨다.
2. TargetAction이라는 새로운 Cocoa 응용 프로그램 프로젝트를 만든다.
3. main nib 파일을 연다.
4. 텍스트 필드를 메인 윈도우으로 드래그하고, 기본적인 폭을 약 4배로 넓힌다.
5. 버튼을 메인 윈도우로 드래그한다.
6. 버튼을 더블 클릭하고, **Refresh**를 입력하여 버튼의 이름을 지정한다.

작업이 완료되면 <그림 6-24>와 같은 화면이 나타나야 한다.




<그림 6-24> 타켓/액션 응용 프로그램 인터페이스

## 컨트롤러 클래스 만들기

응용 프로그램을 위한 컨트롤러 클랙스를 만들기 위해 NSObject의 서브클래스를 만든다.

1. `MainMenu.nib` 윈도우의 Classes 탭을 클릭한다.
2. 클래스 리스트에서 NSObject를 선택한다.
3. 새로운 서브클래스를 만들기 위해 Return을 누르고, MyObject 텍스트를 바꾸기 위해 **`MyController`**를 입력한다.

## 클래스의 아웃렛 정의하기

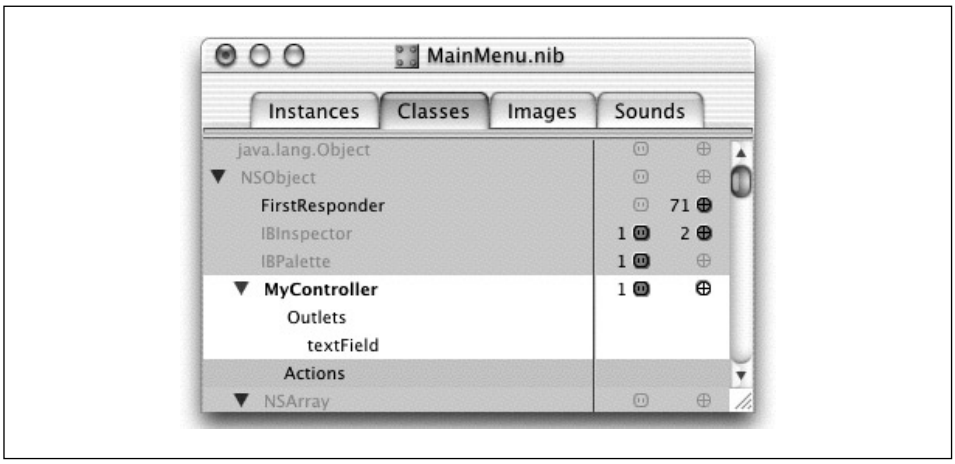
컨트롤러 클래스는 텍스트 필드용 아웃렛을 필요로 한다. 그래야만 메시지를 텍스트 필드에 전송할 수 있다.

1. Classes 패인에서 MyController를 선택한다.
2. 클래스 오른쪽에 있는 전자 아웃렛 아이콘을 클릭한다.
3. 새로운 아웃렛을 만드려면 Return을 누른다.
4. 이 아웃렛에 **`textField`**라는 이름을 지정하고, Return을 누른다.

## 클래스를 위한 액션 정의하기

**`refresh:`**라는 MyController의 액션 메소드 1개를 정의한다. 사용자가 Refresh 버튼을 클릭하면, **`refresh:`** 메시지는 타겟 객체인 MyController 인스턴스로 전송된다. 액션은 사용자가 컨트롤 객체를 조작할 때 객체로 전송된 메시지와 사용할 메소드로 전송된 메시지 모두를 참조한다.

1. <그림 6-25>와 같이 Classes 패인에서 MyConverter 아래에 있는 Actions을 클릭한다.




*<그림 6-25> 액션 정의하기*

2. 새로운 액션을 만들려면 Return을 누르고, **refresh**라는 메소드 이름을 입력한다. IB는 **:**를 추가한다.

### 클래스의 인스턴스 만들기

컨트롤러 클래스의 프록시 인스턴스를 만들어야 하기 때문에 컨트롤러 객체와 UI 객체를 연결한다. MyController를 선택하려면,

1. Classes 리스트에서 MyController를 클릭한다.
2. Classes 메뉴에서 Instantiate를 선택한다. 인스턴스는 Instances 패인에 나타난다.

### 인터페이스 컨트롤을 클래스 액션에 연결하기

MyController가 사용자 인터페이스의 버튼으로부터 액션 메시지를 수신할 수 있도록 하려면, 버튼을 MyController에 연결해야 한다. 버튼 객체는 아웃렛을 사용하여 버튼의 타겟에 대한 참조를 갖는다.(아웃렛의 이름을 `target`으로 지정한다)

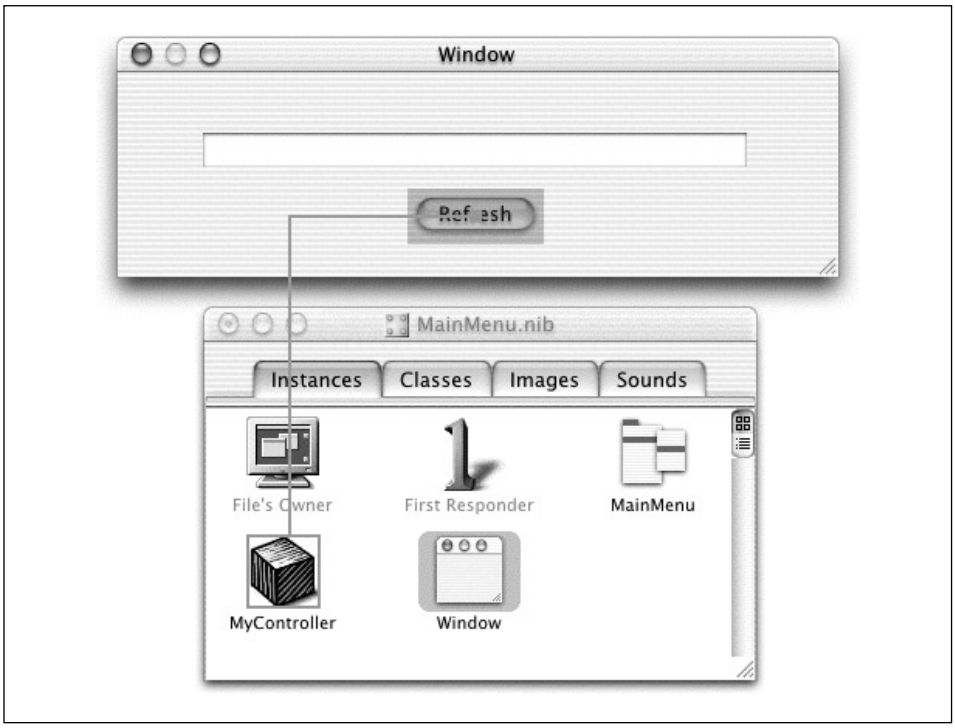
Interface Builder의 Connections Info 윈도우에서 타켓/액션 연결을 볼 수 있다.(또는 완성할 수 있다) Connections Info 윈도우의 상단 우측 세로열은 타겟 객체의 클래스에 의해 정의되고, Interface Builder에 의해 알려진 모든 액션 메소드를 나열한다.

1. Refresh 버튼으로부터 `MainMenu.nib` 윈도우의 MyController 인스턴스로 Control 키를 누른 채 연결 라인을 드래그한다. <그림 6-26>과 같이 인스턴스의 윤곽이 잡히면, 마우스 버튼을 해제한다.
2. Connections 패인에서 Outlets 세로열의 `target`이 선택되었는지 확인한다.
3. <그림 6-27>에서 보이는 바와 같이 오른쪽 세로열에서 `refresh:`를 선택한다.
4. Connect 버튼을 클릭한다.
5. 메인 nib 파일을 저장한다.

Cocoa를 처음 접한 개발자는 Interface Builder에서 액션과 아웃렛을 연결할 때 가끔 혼동을 하는 경우가 있다. 연결 라인을 긋는 방법을 배우려면 간단한 방식을 준수해야 한다. 메시지가 가는 방향으로 연결 라인을 그으면 된다.

- 액션을 연결하려면, 버튼이나 텍스트 필드와 같은 사용자 인터페이스의 컨트롤 객체로부터 액션 메시지를 수신해야 하는 커스텀 인스턴스로 연결 라인을 그린다.
- 아웃렛을 연결하려면, 커스텀 인스턴스에서 응용 프로그램의 다른 객체로 연결 라인을 그린다.

타켓/액션과 아웃렛 연결간의 차이점은 <그림 6-28>에 나타나 있다.




<그림 6-26> 버튼을 객체 인스턴스에 연결하기

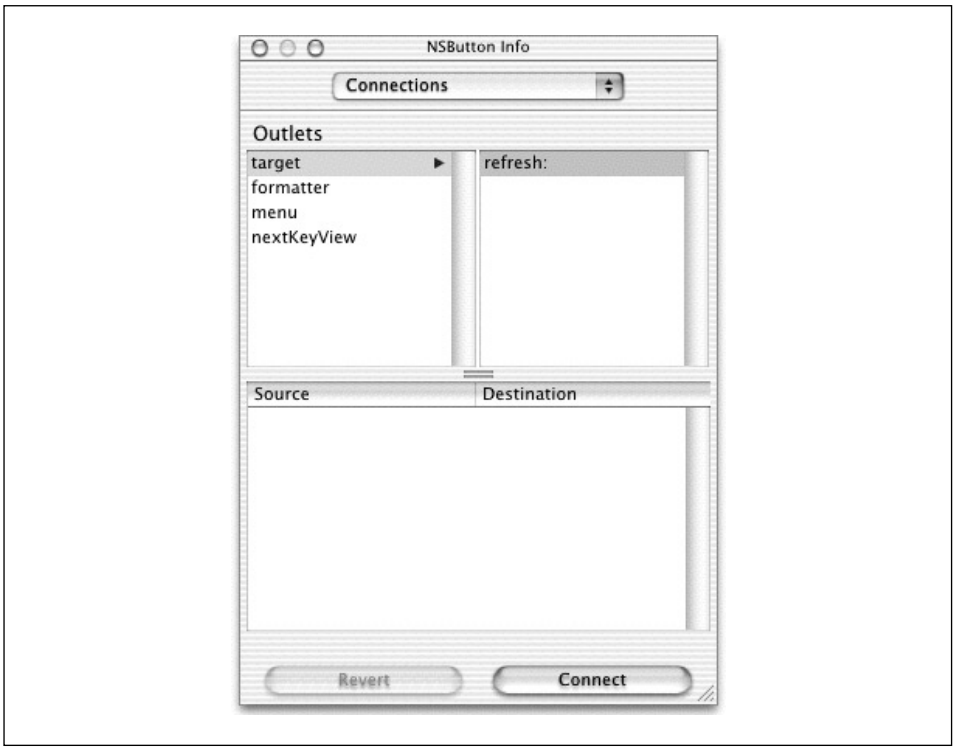
연결을 지정하는 또 다른 방식은 어디에서 어디로 연결해야 할 것인지를 고려하는 것이다. 커스텀 객체는 아웃렛을 사용하여 다른 객체를 찾아야 한다. 그러면, 커스텀 객체에서 다른 객체로 연결이 이루어진다. 컨트롤 객체는 액션을 사용하여 커스텀 객체를 찾아야 한다. 그러면, 컨트롤 객체로부터 연결이 이루어진다.

## 커스텀 클래스를 인터페이스로 연결하기

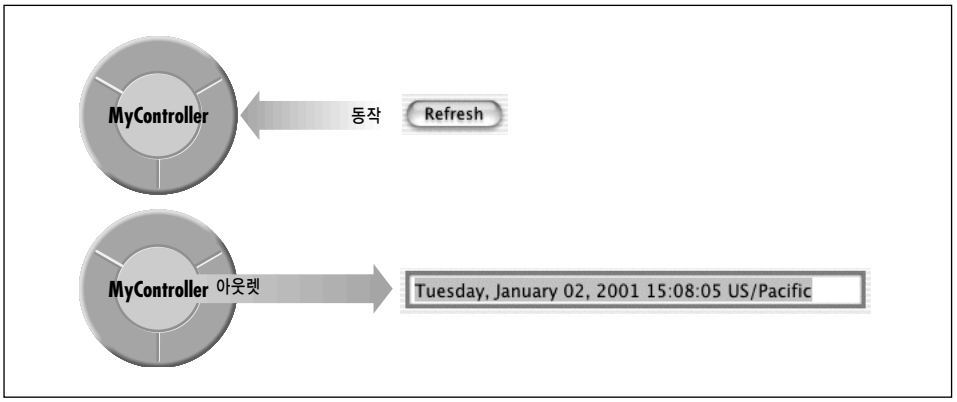
**refresh:** 액션 메소드가 발생하면 알 수 있도록 응용 프로그램은 일종의 피드백을 제공해야 한다. 이 예제를 위해, Refresh 버튼을 클릭하면, 현재 시간과 날짜가 출력된다.

"Cocoa에서 그래픽 사용자 인터페이스 만들기" 섹션에서 수행했던 대로 메인 윈도우의 텍스트 필드로 MyController를 연결한다. **refresh:** 메소드는 텍스트 필드에 텍스트를 전송하기 위해 이 아웃렛을 사용한다.

1. nib 파일 윈도우의 Instances 패인에서 MyController 인스턴스로부터 텍스트 필드로 Control 키를 누른채 연결 라인을 드래그한다. 인스턴스의 윤곽이 잡히면, 마우스 버튼을 해제한다.




<그림 6-27> 타겟을 액션에 연결하기




<그림 6-28> 타겟/액션 및 아웃렛 연결

2. Interface Builder는 Info 윈도우의 Connections 패인을 불러온다. `textField` 아웃렛을 선택한다.
3. Connect 버튼을 클릭한다.

### 소스 파일 만들기

MyController의 아웃렛, 액션, 연결라인을 선언하였으므로, 클래스를 위한 소스 파일을 만들어, Project Builder 프로젝트에 추가해야 한다.

1. nib 파일윈도우의 Classes 패인으로 간다.
2. MyController 클래스를 선택한다.
3. Classes 메뉴에서 Create Files을 선택한다.
4. `h`와 `.m` 파일 옆의 Create 세로열에서 체크박스를 선택하였는지 확인한다.
5. TargetAction 타겟 옆에 있는 체크박스를 선택하였는지 확인한다.
6. Choose 버튼을 클릭한다.
7. nib 파일을 저장한다.

이제, 응용 프로그램의 Interface Builder를 종료하고, Project Builder를 사용하여 응용 프로그램을 완성시켜 보도록 하겠다.

### *MyController*의 액션 메소드 구현하기

MyController의 `refresh:` 메소드 구현을 작성한다.

1. Project Builder에서 `MyController.h`와 `MyController.m`을 Classes 그룹으로 이동시킨다.
2. `MyController.m`을 선택하여, 아래와 같이 `refresh:` 코드를 삽입한다.

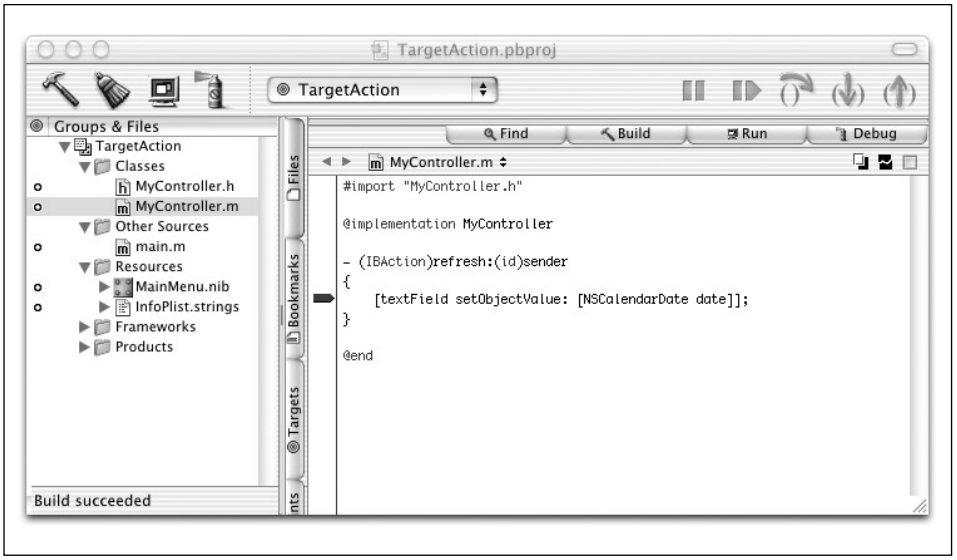
```
@implementation MyController
- (IBAction)refresh:(id)sender
{
    [textField setObjectValue:[NSCalendarDate date]];
}
@end
```

메소드는 단순히 현재 시간과 날짜를 텍스트 필드로 전송하므로, Refresh 버튼을 클릭할 때마다 업데이트된다.

### 응용 프로그램 구축 및 디버그

디버깅 작업을 원활하게 수행하려면, Project Builder는 GNU 디버거인 GDB 위에 그래픽 사용자 인터페이스를 배치한다. 디버거와 친숙해지려면, 디버거를 사용하여 `refresh:` 메소드에서 브레이크포인트를 설정해본다.

1. Project Builder의 Groups & Files 리스트에서 **MyController.m**을 클릭한다.
2. Text 패인에 **refresh:** 메소드를 배치한다.
3. 코드 리스트 좌측에 있는 세로열을 클릭하여 브레이크포인트를 설정한다. <그림 6-29>를 참조한다.




*<그림 6-29> 브레이크포인트 설정하기*

4. 프로젝트를 구축하려면 Build 버튼을 클릭한다.
5. Debug 버튼을 클릭하여 디버거를 실행시키거나 Command-R을 누른다.
6. 응용 프로그램이 구동되면, 윈도우에서 Refresh 버튼을 클릭한다.
7. Refresh 버튼을 클릭하면, 디버거가 활성화되어, 브레이크포인트에서 정지하고, 개발자는 콜 스택과 지역변수의 값을 확인할 수 있다.

복잡한 디버깅 작업을 수행하려면, GDB 콘솔을 사용하면 된다. Console을 보이게 하려면 윈도우의 우측 상단 코너에 있는 Console 탭을 클릭한다. Console 윈도우에서 클릭하여, Return 버튼을 누르면, **(gdb)** 프롬프트가 나타난다. 사용자 인터페이스에는 나타나지 않는 이 프롬프트에서 수 많은 GDB 커맨드들을 입력할 수 있다. 이 같은 커맨드에 관한 온라인 정보를 얻으려면 프롬프트에서 **help**를 입력하면 된다.

# 객체 소유권, 보유, 처리

객체 소유권과 처리의 문제는 객체 지향 프로그래밍에서는 자연스러운 관심사이다. 객체가 생성되어, 응용 프로그램의 다양한 "소비자" 객체로 전달될 때, 어떤 객체가 그 객체를 처리할 책임을 져야 하는가? 그리고 언제 객체를 처리해야 하는가? 객체가 더 이상 필요하지 않는데도 제거되지 않는다면, 메모리가 누설된다. 그러나, 객체가 너무나 빨리 제거되면, 그 객체를 떠맡고 있던 다른 객체에서 문제가 발생하고, 결국 응용 프로그램이 충돌을 일으킨다.

Foundation 프레임워크는 객체가 더 이상 필요하지 않을 때 제거될 수 있도록 하는 메카니즘과 방식을 도입한다.

방식은 상당히 간단하다. 개발자는 자신이 소유한 모든 객체를 처리할 책임이 있다. 개발자는 객체를 할당하거나 복사하는 방식으로 만들어진 객체를 소유한다. 또한, 보유하고 있는 객체들도 개발자의 소유이다. 반대로 생각해보면, 개발자는 자신이 보유하거나 생성하지 않은 객체는 결코 제거할 수 없다.

## 객체 초기화 및 제거

일반적으로, Cocoa에서는 객체(**alloc**)를 할당하고, 그 객체(**init** 또는 변형)를 초기화하여 객체를 만든다. 예를 들면,

```
NSArray *myArray = [[NSArray alloc] init];
```

어레이의 **init** 메소드를 사용하면, 메소드 구현은 메소드의 인스턴스 변수를 기본 값으로 초기화하고, 다른 구동 작업을 완성한다. 마찬가지로, 객체를 제거하면, 만들어진 객체를 해제하고, 할당된 메모리를 제거하기 위해 **dealloc** 메소드가 발생한다.

그러나, 또 다른 문제가 제기된다. 객체의 소유자가 프로그램 범위내에서 객체를 해제해야 한다면, 어떤 방식으로 다른 객체에 그 객체를 제공해야 하는가? 해답은 **autorelease** 메소드에 있다. 이 메소드는 나중에 해제될 것을 대비하여 수신자를 표시하고, 다른 객체들이 객체를 사용할 수 있도록 소유자 객체의 영역밖에서도 이 객체가 상주할 수 있도록 한다.

**autorelease** 메소드는 객체 제거에 대한 자동해제 메카니즘의 더 광범위한 개념으로 이해되어야 한다. 이 프로그램 메카니즘을 통해 개발자는 객체 소유권과 처리 방식을 구현한다.

## 참조 카운팅

Cocoa에서 각 객체는 각각의 참조 카운트를 가지고 있다. 객체를 할당하거나 복사할 때, 참조 카운트는 1에 설정된다. 참조 카운트를 줄이려면 객체에 **release**를 전송한다. 참조 카운트가 0에 도달하면, NSObject는 객체의 **dealloc** 메소드를 발생시켜,객체를 파괴한다. 그러나, 다음에 나오는 객체의 소비자는 **retain**를 전송하여 참조 카운트를 늘려 파괴를 지연시킬 수 있다. 작업을 완료할때까지 객체가 제거되지 않도록 객체를 보유한다.

## *Autorelease Pool*

각각의 응용 프로그램은 최소한 1개의 오토릴리즈 풀(이벤트 사이클용)을 가지고 있다. 오토릴리즈 풀은 이벤트 해제 표시가 된 객체를 추적하여 알맞은 시간에 이들을 해제한다. **autorelease** 메시지를 객체에 전송하여 풀에 객체를 배치한다. 응용 프로그램의 이벤트 사이클은 코드 실행이 종료되고, 컨트롤이 응용 프로그램 객체로 반환되면(대체적으로, 사이클이 끝날 무렵) 응용 프로그램 객체가 **release**를 오토릴리즈 풀에 전송하는 방식으로 이루어진다. 그러면, 풀은 들어있는 각 객체를 해제한다. 그 이후, 풀에 있는 객체의 참조 카운트가 0이면, 객체는 제거된다. <그림 6-30>은 과정을 보여준다.

1. **myObj**가 객체를 만든다.

```
anObj = [[MyClass alloc] init];
```

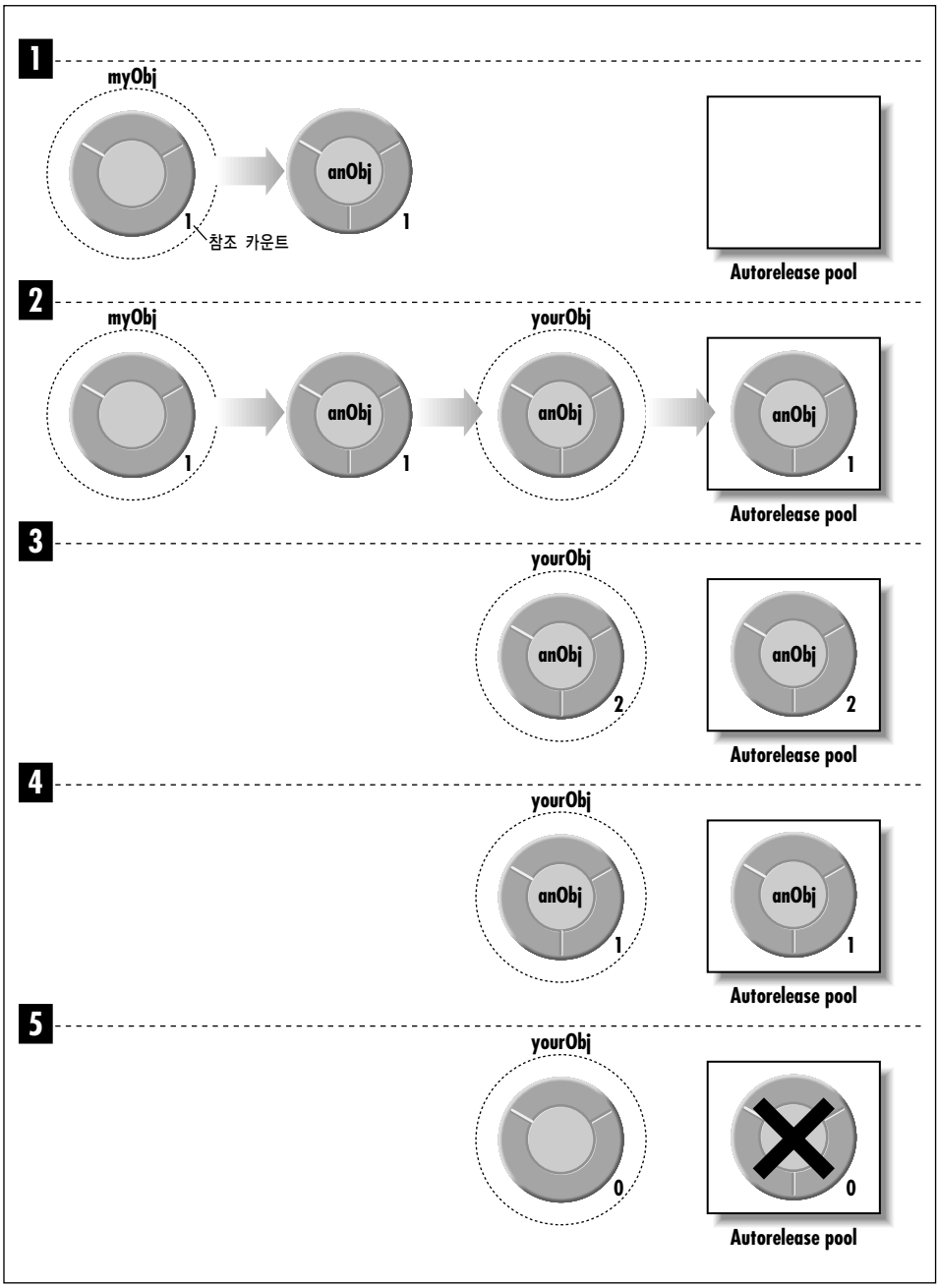
2. **myObj**가 자동해제된 **yourObj**로 객체를 반환한다. 이로써, 객체가 오토릴리즈 풀에 배치된다. 즉, 오토릴리즈 풀이 객체를 추적하기 시작한다.

```
return [anObj autorelease];
```

3. **yourObj**는 객체를 보유한다. **retain** 메시지는 참조 카운트를 2로 늘린다.(**myObj**와 **yourObj**는 **anObj**에 대한 참조를 가진다)

```
[anObj retain];
```

4. 이벤트 사이클이 끝날 무렵, 오토릴리즈 풀은 **release**를 모든 객체에 전송한다. 이로 인해 참조 카운트가 줄어든다. 참조 카운트가 0인 객체는 제거된다. **anObj**의 참조 카운트는 전에는 2였기 때문에 오토릴리즈 풀이 **release**메시지를 **anObj**에 전송해도, **anObj**의 참조 카운트는 1이 되므로, 제거되지 않는다.
5. **yourObj**가 더 이상 **anObj**를 필요로 하지 않으면, 오토릴리즈 풀에 객체를 배치하여 **autorelease**를 **anObj**에 전송한다. 사이클이 끝날 무렵, 오토릴리즈 풀은 다시 한번 **release**를 객체에 전송한다.




<그림 6-30> 오토릴리즈 풀에서 객체의 이벤트 사이클

그러면, 어떤 객체가 그 객체를 만든 메소드의 범위내에서만 사용되면, release를 전송하여 즉시 제거할 수 있다. 그러나, 객체를 만든 범위 밖에서 사용된 경우라면, 더 이상 필요로 하지 않기 때문에 다른 객체로 반환되거나 전달되는 모든 객체에 autorelease를 전송한다.

(release나 autorelease를 retain보다 먼저 수행하지 않는다면)다른 객체에서 수신한 객체를 해제해서는 안된다. 이들 객체를 소유할 수 없으며, 이들 객체의 진짜 소유자가 궁극적으로 이 객체들을 제거할 수 있다. 또한, 수신된 객체는 객체를 수신한 메소드 내에서만 유효하다고 할 수 있다. 그 메소드는 안전하게 객체를 발생자에게 반환할 수 있다.

release나 autorelease는 생성 객체가 허용한 횟수에 전송한 retain 메시지의 수 만큼 더하여 객체로 전송해야한다. free 또는 dealloc을 Cocoa 객체에 전송하면 안된다.(예외 사항: 클래스의 dealloc 메소드를 오버라이드하였을 경우)

## 보유된 객체의 관계

어떤 객체를 보유하고 있다면, 이 객체의 소유자와 이 객체를 보유하고 있는 다른 객체가 이 객체를 공유하는 경우가 있다. 이러한 상태를 개발자가 원할 수도 있지만, 이는 바람직하지 못한 결과를 초래할 수 있다. (접근자 메소드로부터 반환된)객체의 인스턴스 변수를 보유하고 있을 때, 객체의 소유자가 해제되어 버리면, 인스턴스가 참조한 객체는 당연히 사용할 수 없다. 소유자 객체의 인스턴스 변수를 보유하고 있을 때, 그 인스턴스 변수가 나중에 변경된 수정 가능 객체를 참조한다면, 코드는 의외의 객체를 참조할 수 있다.

## 복사와 보유 비교하기

객체를 복사할 것인지 보유할 것인지를 결정할 경우, 객체들을 *Value object*나 *Entity object*로 분류하는 것이 도움이 된다. Value 객체는 NSNumbers나 NSStrings와 같은 객체들로 개별적이고, 제한된 데이터를 캡슐화한다. Entity 객체는 NSViews와 NSWindows 같은 객체들로 보조 객체를 관리하고 조정하는 역할을 한다. Value 객체의 경우, 객체(객체는 NSCopying 프로토콜을 준수해야 한다)의 스냅샷이 필요하면 copy를 사용한다. 객체를 공유하고자 한다면 retain을 사용한다. 항상, Entity 객체를 보유한다.

Cocoa에서 객체를 복사하는 방법에 관한 자세한 내용은 NSCopying 프로토콜에 대한 참고 문서를 참조한다.

## 접근자의 참조 카운팅

접근자 메소드는 객체의 인스턴스 변수를 얻고, 설정하기 위해 사용된다. 값을 반환하는 접근자 메소드 선언은 괄호안의 반환값 뒤에 있는 인스턴스 변수의 이름이다. 인스턴스 변수 값을 설정하는 접근자 변수는 인스턴스 변수(첫글자는 대문자)의 이름 뒤에서 set으로 시작한다.

set 메소드의 인수는 인스턴스 변수 유형을 취하고, 메소드는 Void를 반환한다.

인스턴스 변수 값을 클래스가 아닌 어떤 객체에 의해서도 변경되지 않도록 하려면, 인스턴스 변수의 set 메소드를 제공하지 않는다. set 메소드를 제공하면, 인스턴스변수 값을 지정할 때 클래스 객체가 set 메소드를 사용하는지 확인해야 한다. 이는 클래스의 서브클래스에 중요한 관련이 있다.

값-객체 인스턴스의 변수를 설정하는 접근자 메소드에서는 대체로(항상 그런것은 아니다) 객체를 복사하여 사용하며,공유하지는 않는다.(객체를 공유하면 사전 예고없이 변경될 수 있다) <예제 6-2>에서 설명한 대로, 오래된 객체에는 **autorelease**를 전송하고, 새로운 객체에는 **copy**(**retain**이 아님)를 전송한다.

*<예제 6-2> setTitle: 메소드*

```
- (void)setTitle:(NSString *)newTitle
{
    [title autorelease];
    title = [newTitle copy];
}
```

접근자 메소드(특히, 세터 메소드)를 구현하는 방법을 결정하는 것은 까다로울 수 있다. 개발자가 염두해야 할 수 많은 문제들이 있다. 이 섹션의 나머지 부분에서는 이러한 문제들 중에 가장 중요하다고 여겨지는 사항을 설명한다.

## *해제 시점*

<예제 6-2>에서, **title**은 NSString 객체의 포인터이기 때문에 설정되기 전에 해제되어야 한다. 그래서 **title**이 먼저 자동 해제되지 않고, **newTitle**에 설정된다면, 원래의 NSString은 누설된다. **title**이 새로운 NSString의 포인터가 되었을 경우에는, **title**의 이전 값을 해제할 방법은 없다.

## *복사 시점*

NSStrings과 같은 리프 값 객체들은 보유하는 것보다 복사하는 것이 낫다. 다음 예제를 검토해본다.

```
NSMutableString *foo = [NSMutableString stringWithCString:"foo"];
[myWindow setTitle:foo];
[foo appendString:@"bar"];
```

NSWindow가 **setTitle:**에서 자신에게 전달된 스트링을 보유하고 있다면(복사하는 대신), 윈도우의 타이틀은 **foo**가 아닌 **foobar**로 나타난다. 이것은 의도된 결과가 아니다.

작업을 안전하게 수행할 수 있다면, (즉, 수신자가 실제로 수정 불가능 NSString이라면) copy는 실제로 객체의 참조 카운트를 늘린다.(암시적 보유)

### 해제 또는 자동 해제

인스턴스 변수를 변경하기 전에 해제할 것인지 자동 해제할 것인지 여부도 관건이다. <예제 6-2>의 `title`은 자동 해제 되지 않고, 해제되면, 바로 사용할 수 없게 된다. `newTitle`이 `title`과 동일한 객체의 포인터라면, 막 설정되려는 것을 해제 했기 때문에 `title`은 쓸모없게 된다.

`setTitle:` 메소드를 발생시키는 객체의 또 다른 문제점을 고찰해본다.

```
{
    /* ... */
    title = [myWindow title];
    [myWindow setTitle:newTitle];
    /*...*/
    // title is now garbage because it was released by setTitle:
}
```

`title`이 유효한 NSString일 것이라고 예상되면, `setTitle:` 다음에 나오는 코드는 장애를 일으킨다.

경험에 의하면 Foundation 객체(예를 들면, NSArray와 NSDictionary)들은 결코 자동 해제되지 않는다. 이는 성능을 최대화하기 위함이다. 그러나, Application Kit 레벨과 사용자 인터페이스 클래스에서 `autorelease`는 자동 해제된 반환 값의 일반적인 Application Kit API를 유지하기 위해 사용된다. 반환 값은 오토릴리즈 풀이 비워질때 까지 호출 문맥에서 유효하다.

클래스를 구현할 때, 클래스의 트래이드오프(Trade-off)와 위험성을 인식하고 있어야 한다. 클라이언트가 사용하기 곤란한 객체를 다룰 경우에 `release`를 사용하는 것이 더 나은 선택이 될 수 있다. 또한, `release`를 사용하면 코드에서 보유/해제 버그를 추적하는 작업이 더욱 수월해진다. 반면, `autorelease`는 클라이언트가 접근할 수 있는 인스턴수 변수를 위해 사용해야 한다.

### *Deallication 버그를 제거하기 위한 정보*

객체를 제거할 때 발생하는 문제는 Cocoa 프로그래머 입문과정에서 드문 현상은 아니다. 객체를 너무 많이 제거했거나 객체를 제거해야 하는데도 객체를 제거하지 않을 수 있다. 이 두가지 상황은 심각한 장애를 불러일으킨다. 첫번째 경우는 존재하지 않는 객체를 코드가 참조함으로써 런타임 오류를 발생시키고, 두번째 경우는 메모리 누설을 불러온다.

Cocoa 코드에서 Deallication 버그를 방지할 수 있는 몇 가지 방법을 제시한다.

- 각 release나 autorelease를 객체에 전송하기 위해 객체로 전송된 alloc, copy, mutableCopy 또는 retain 메시지가 있는지 확인한다.
- 객체를 콜렉션에 추가하면, 객체는 보유된다. 객체를 콜렉션에서 제거하면, 객체는 해제된다. 콜렉션 객체(NSArray 같은)를 해제하면, 그 안에 저장된 객체가 모두 해제된다.

# 7

# *Currency Converter* 튜토리얼

이 장은 제6장, *중요한 Cocoa 패러다임*에서 논의된 Cocoa 프로그래밍의 패러다임에 대한 이해를 돕기 위해 1개의 윈도우 응용 프로그램을 구축하는 방법을 자세하게 설명한다. 먼저, 이 응용 프로그램 개발에 대한 일반적인 작업 흐름을 살펴보면 다음과 같다.

1. 응용 프로그램 디자인
2. 프로젝트 생성(Project Builder)
3. 인터페이스 생성(Interface Builder)
4. 클래스 정의(Interface Builder)
5. 클래스 구현(Project Builder)
6. 프로젝트 구축(Project Builder)
7. 응용 프로그램 운용 및 시험

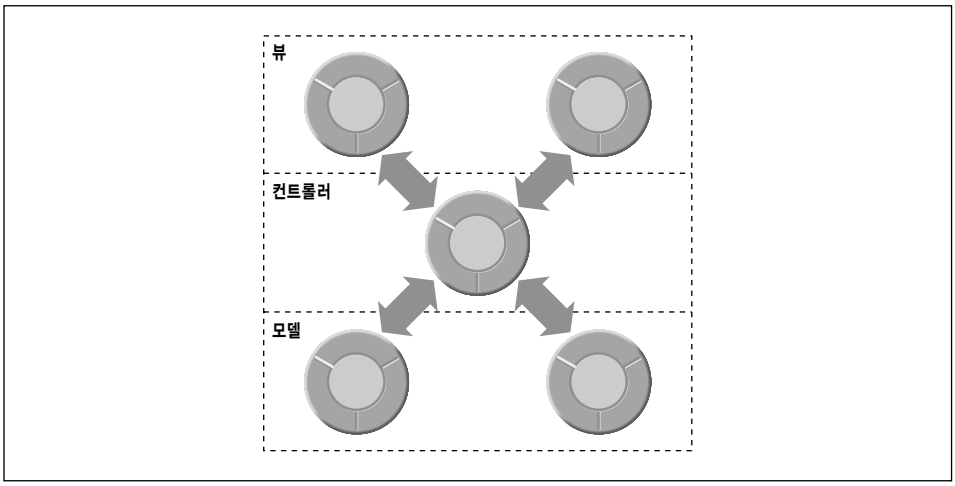
이 장에서 구축할 응용 프로그램은 Currency Converter(환율 변환)라는 프로그램이다. Currency Converter는 1달러를 다른 환율로 변환시키는 단순한 유틸리티이다. 그러나, 보기에는 단순한 응용프로그램이지만, 그 이면에는 디자인 구조가 뒷받침하고 있다. 이 응용 프로그램의 디자인은 객체 지향 프로그램의 디자인 모델인 Model-View-Controller(MVC) 패러다임을 기반으로 한다. MVC는 응용 프로그램을 역할과 책임에 따라 여러 객체 유형으로 분리한다. 이 디자인 패러다임은 유지 보수, 확장 및 재사용이 가능한 코드 개발을 목표로 구성되었다. Currency Converter의 구성 예제를 통해 이 같은 사실은 더욱 명확해진다.

# Currency Converter 응용 프로그램 디자인

객체 지향 응용 프로그램은 응용 프로그램의 객체를 식별하여, 역할과 책임을 명확하게 정의한 디자인을 기반으로 해야 한다. 일반적으로, 코드를 작성하기 전에 디자인 작업을 한다. 수 많은 응용 프로그램을 디자인하기 위해 반드시 훌륭한 툴을 사용할 필요는 없다. 연필과 종이만 있어도 충분하다.

## Model-View-Controller(MVC) 패러다임

MVC는 <그림 7-1>에서 보여준 모델, 뷰 및 컨트롤러를 의미한다. MVC 디자인에서 모델 객체는 데이터를 저장하고, 이 데이터를 조작하는 로직을 정의한다. 뷰 객체는 사용자 인터페이스(윈도우나 버튼)에서 볼 수 있는 객체를 의미한다. 컨트롤러 객체는 모델 객체와 뷰 객체사이에서 조정자로 동작한다.



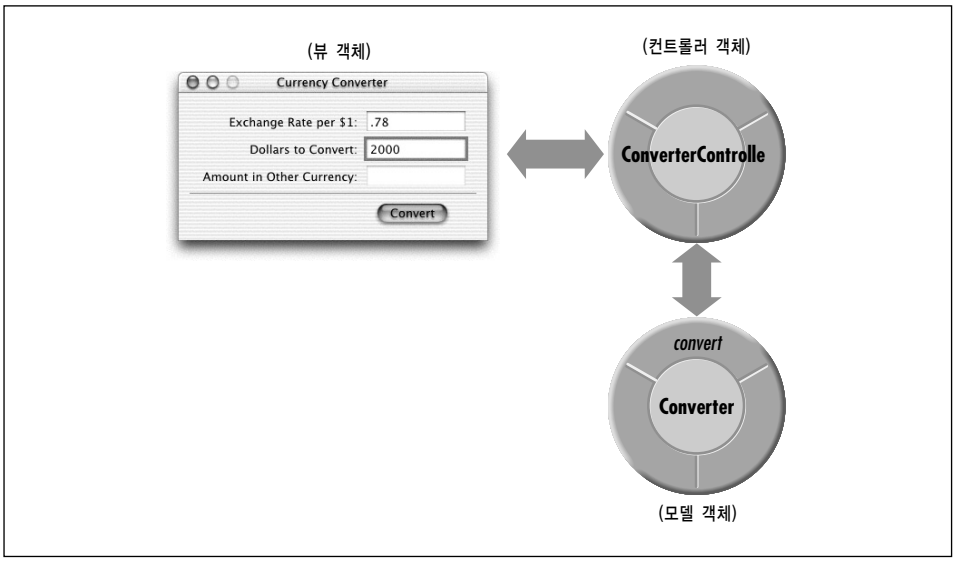

*<그림 7-1> MVC 디자인에서 모델, 뷰 및 컨트롤러의 분리*

MVC의 장점은 사용자 인터페이스의 상태와 이벤트를 인식하는 역할에서 모델 객체를 자유롭게 해주며, 모델 객체의 프로그램 인터페이스를 인식하는 역할에서 뷰 객체를 자유롭게 해주는 컨트롤러의 중심적인 조정 역할에 있다.

엄밀히 말하면, MVC는 모든 환경에 부합하는 디자인은 아니다. 때때로 MVC의 기능들을 통합해야 하는 상황도 발생한다. 예를 들어, 아케이드 게임같은 그래픽 집약 응용 프로그램에서 뷰와 모델 기능을 통합한 여러 개의 뷰 객체를 생성할 수 있다. 어떤 단순한 응용 프로그램에서 컨트롤러와 모델 기능을 통합할 수 있다. 이들 객체들은 컨트롤러의 후크를 통해 모델 객체의 특수 데이터 구조와 로직을 인터페이스에 연결한다.

## *Currency Converter 디자인에서의 MVC*

Currency Converter는 레디메이드 Application Kit객체의 콜렉션을 사용하여 구현된 사용자 인터페이스(뷰), 컨버터(모델) 및 ConverterController(컨트롤러)와 같은 커스텀 객체로 구성된다. 컨버터 객체는 환율 양을 산출하고, 그 값을 반환하는 역할을 수행한다. 사용자 인터페이스와 컨버터 객체 사이에는 컨트롤러 객체인 ConverterController 객체가 있다. ConverterController는 컨버터 객체와 UI 객체사이의 동작을 조정한다. <그림 7-2>는 이 같은 관계를 보여준다.




*<그림 7-2> Currency Converter 디자인에서 MVC*

ConverterController 클래스는 응용 프로그램에서 중추적인 역할을 담당한다. ConverterController 클래스는 모든 컨트롤러 객체들 처럼 인터페이스 및 모델 객체와 통신하고, 응용 프로그램 고유의 작업을 처리한다. 또한, 사용자가 필드에 입력한 값을 컨버터 객체에 건내 주고, 컨버터에서 결과를 수신하여 인터페이스 필드에 배치한다.

컨버터 클래스는 컨트롤러가 건내준 2개의 인수에서 값을 산출하고, 결과를 반환한다. 컨버터 클래스는 사용자 인터페이스의 구현 내용과 분리되기 때문에 다른 응용 프로그램에서 간편하게 재사용할 수 있다.

Currency Converter를 위한 이 디자인은 객체 지향 프로그래밍과 MVC만의 고유 기능을 제공하므로, 단순한 응용 프로그램을 디자인할 수 있다. 이 디자인에서는 응용프로그램의 컨트롤러 클래스인 ConverterController가 계산 기능을 수행하기 때문에 컨버터 클래스 없이도 작업을 행할 수 있다. 하지만 처음부터 훌륭한 디자인을 강조해서 나쁠 것은 없다고 본다.

# Currency Converter 프로젝트 만들기

Currency Converter를 위한 디자인의 요건이 갖춰졌으므로, 응용 프로그램을 구축해보기로 하겠다.

## 응용 프로그램 프로젝트 만들기

응용 프로그램을 완성하려면 먼저 Project Builder 프로젝트를 만든다.

1. Project Builder를 구동한다.
2. File 메뉴에서 New Project를 선택한다.
3. New Project 패널에서 Cocoa Application 프로젝트를 선택하고, Next 버튼을 클릭한다.
4. 응용 프로그램의 이름을 `Currency Converter`라 한다.
5. Set을 클릭하여 위치를 선택하고, 선택한 위치에 프로젝트를 저장한다. 기본적인 위치를 사용하려면, 6단계로 진행한다.
6. Finish를 클릭한다.

## Project 인덱싱

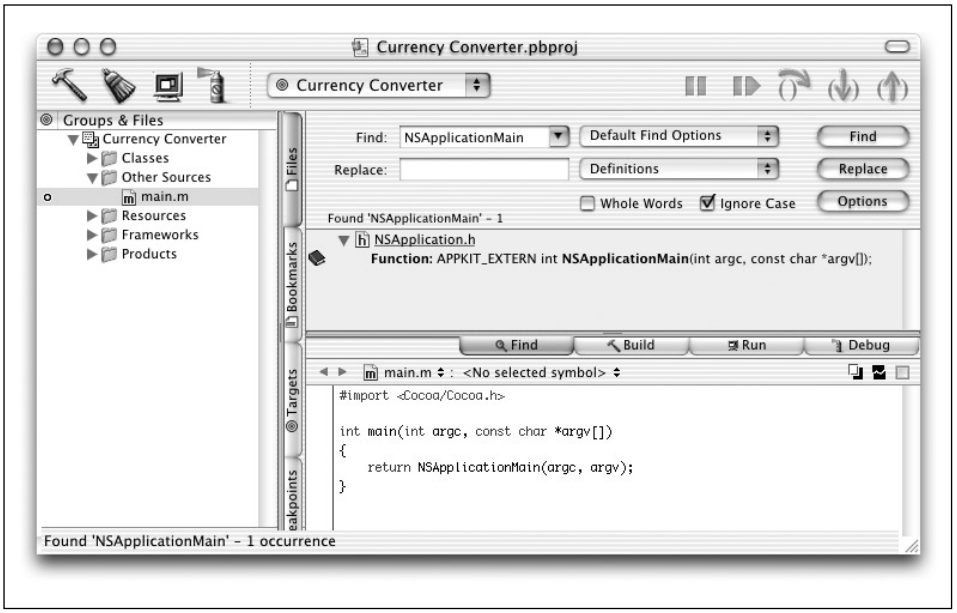
프로젝트를 만들고 난 후, Project Builder의 인덱스가 있으면 편리하다. 인덱스 작업을 하는 동안, Project Builder는 프로젝트의 모든 심볼(클래스, 메소드, 글로벌 등)을 디스크에 저장한다. 이로써, Project Builder는 프로젝트의 광범위한 정보에 신속하게 접근할 수 있다.

Mac OS X의 첫 번째 릴리즈에 포함된 Project Builder 버전은 프로젝트가 먼저 구축되지 않으면, 프로젝트 인덱스에 장애를 일으키는 버그가 발생한다. 이 버전으로 작업을 수행할 경우, 프로젝트를 구축하고 난 뒤, 프로젝트 인덱스를 생성해야 한다.

인덱스를 만들려면, Project 메뉴에서 Index Project를 선택한다(Command-Option-I). 인덱스가 만들어지면, Project Find를 사용하여 프로젝트 코드와 심볼의 시스템 헤더를 찾을 수 있다. 또한, Project Find에서 바로 참조 도큐멘트로 갈 수 있다. 처음으로 Cocoa 프로젝트의 인덱스를 만들 경우, Project Builder가 프로젝트의 헤더뿐만 아니라 모든 Cocoa 헤더에 대한 인덱스를 만들어야 하기 때문에 약간의 시간이 소요된다.

프로젝트의 인덱스가 다 만들어지면, Cocoa 클래스나 메소드의 도큐멘트에 쉽게 접근할 수 있다. 예를 들면, 프로젝트의 Other Sources 그룹에 있는 `main.m` 파일을 확인하면, `NSApplicationMain` 함수가 호출되었는지를 알 수 있다.

1. Project Builder의 메인 윈도우에서 Find 탭을 클릭한다. <그림 7-3>에서 보여진 바와 같이 Find 패인이 나타난다.
2. Find Type 팝업이 Definitions으로 설정되었는지 확인한다.
3. Find 필드에 **NSApplicationMain**을 입력하고, Find 버튼을 클릭한다.




*<그림 7-3> Project Builder의 Find 패널*

상기 그림을 보면, Project Builder는 **NSApplication.h**에서 1개의 정의를 찾아냈다. 헤더 파일명을 클릭하면, Project Builder는 도큐멘트 패인에 있는 정의를 제공한다. Find 왼쪽에 북 아이콘을 클릭하면, Project Builder는 Help Viewer를 구동하고, 이 함수의 참조 도큐멘트에 접근할 수 있도록 한다. Help Viewer는 Project Builder에서 참조 도큐멘트를 찾기 전에 Developer Center의 Cocoa 영역에서 먼저 찾아보도록 요구할 수 있다.

## *Currency Converter 인터페이스 만들기*

Currency Converter의 인터페이스는 상당히 단순하며, 단지 몇 개의 텍스트 필드와 버튼으로 구성되어 있다. 그러나, Currency Converter를 만들려면 Interface Builder에서 사용할 수 있는 객체의 레이아웃 툴에 대한 지식이 필요하다.

## Main Nib File 열기

먼저, Interface Builder에서 응용 프로그램의 사용자 인터페이스를 만든다.

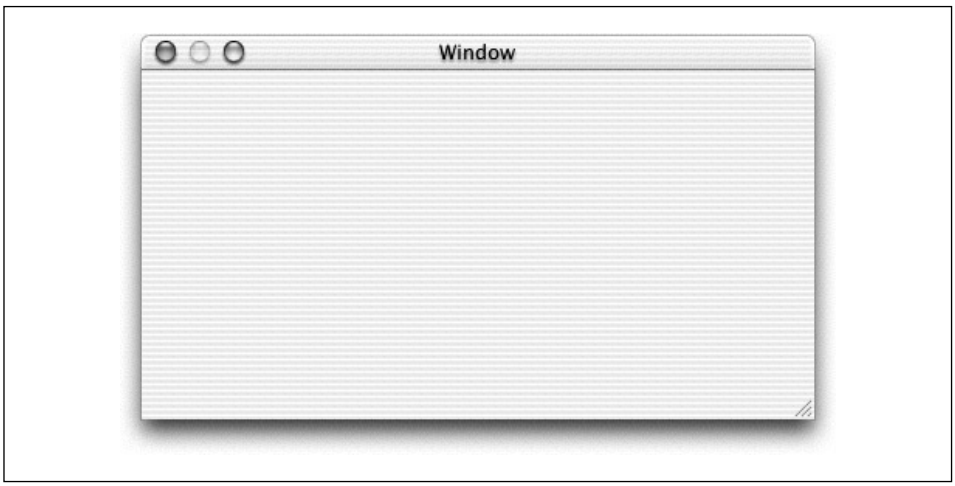
1. `MainMenu.nib`를 Project Builder의 Resources 그룹에 배치한다.
2. 더블 클릭하여 파일을 열면 Interface Builder를 실행되고, nib 파일이 열린다.

nib 파일이 열리면, 기본적인 메뉴 바와 Window라는 윈도우가 나타난다.

## Window 크기 조정하기

이 섹션은 추가될 UI 객체를 수용하기 위해 응용 프로그램의 메인 윈도우 크기를 조정하는 방법을 설명한다.

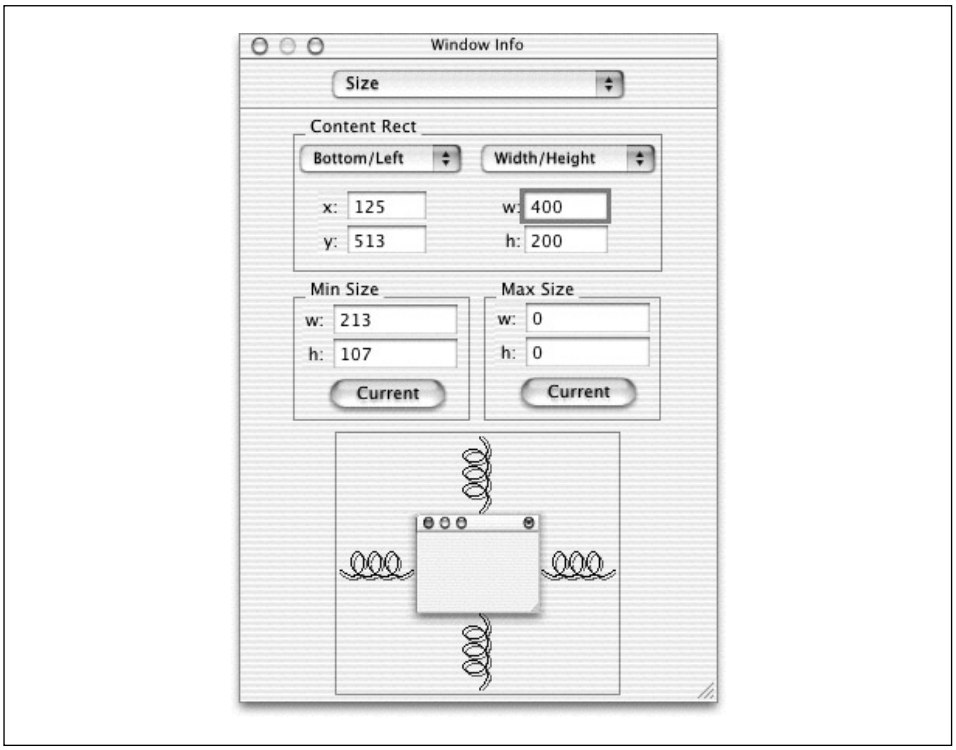
1. <그림 7-4>를 참조하여, 윈도우를 오른쪽 하단 끝으로 드래그한 뒤 작게 조정한다.




*<그림 7-4> Currency Converter의 메인 윈도우*

   Window Info 윈도우의 Size 메뉴를 사용하여 윈도우 크기를 더욱 정확하게 조정할 수 있다.

2. Tools 메뉴에서 Show Info를 선택한다.
3. 팝업 메뉴에서 Size를 선택한다.
4. Content Rect 영역의 우측 팝업 메뉴에서 Width/Height를 선택한다. 넓이(w) 필드에 400을 입력하고, 높이(h) 필드에 200을 입력한다.




<그림 7-5> *Window Info 윈도우*

## *Window 타이틀 및 속성 설정하기*

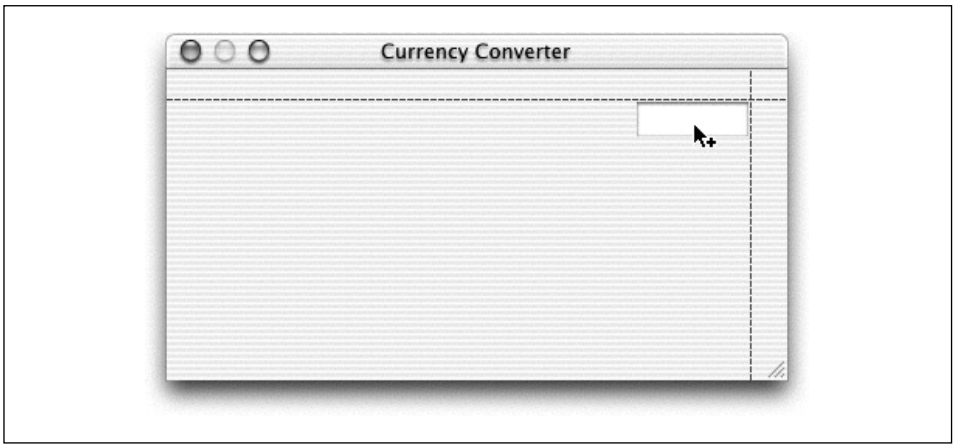
Info 윈도우가 열리면, 다른 속성들을 설정한다.

1. Info 윈도우의 팝업 메뉴에서 Attributes를 설정하고, 윈도우의 타이틀을 Currency Converter로 변경한다.
2. Launch Time 옵션에서 Visible을 선택하였는지 확인한다.
3. Controls 영역에 있는 Resize 체크박스를 선택해제한다.

## *Text Field 배치하기; 크기 조정 및 초기화하기*

Currency Converter에서는 텍스트 필드를 사용하여 사용자 입력을 받고, 변환된 값을 보여준다.

1. 텍스트 필드 객체를 <그림 7-6>에서와 같이 Currency Converter 윈도우로 드래그한다. 윈도우의 경계면이나 인접한 객체로부터 적당한 거리에 객체를 드래그하면, Interface Builder는 팝업 가이드를 보여주고, Aqua 인터페이스 가이드라인에 따라 객체를 배치할 수 있도록 도움을 준다.




<그림 7-6> 텍스트 필드 배치하기

2. 핸들을 잡고 윈도우를 크게 하고자 하는 방향으로 드래그하여 텍스트 필드의 크기를 조정한다. 이 경우, <그림 7-6>과 같이 핸들을 왼쪽으로 드래그하여 텍스트 필드를 확장한다.

Currency Converter에서는 첫번째 텍스트 필드와 동일한 크기를 가진 텍스트 필드가 있어야 한다. 이 작업을 수행하려면 팔레트에서 객체를 드래그하여 같은 크기로 만들거나 처음 객체를 복사하는 방법이 있다.

## 객체 복사하기

윈도우에 추가한 텍스트 필드를 탬플리트로 사용하려면, 텍스트 필드를 2번 복사하여 텍스트 필드 객체를 만든다.

1. 텍스트 필드를 선택하지 않았으면, 선택한다.
2. Edit 메뉴에서 Duplicate를 선택한다.(Command-D) 원래 필드에서 약간 떨어져서 새로운 텍스트 필드가 나타난다.
3. 첫번째 텍스트 필드 아래에 새로운 텍스트 필드를 놓는다. 가이드가 나타나, 두번째 텍스트 필드를 제 위치에 놓는 방법을 알려준다.
4. 세번째 텍스트 필드를 만들려면, Command-D를 다시 사용한다. IB는 이전 Duplicate 명령어를 기억하여, 새롭게 만들 텍스트 필드에 자동으로 적용한다.

## Text Field의 속성 바꾸기

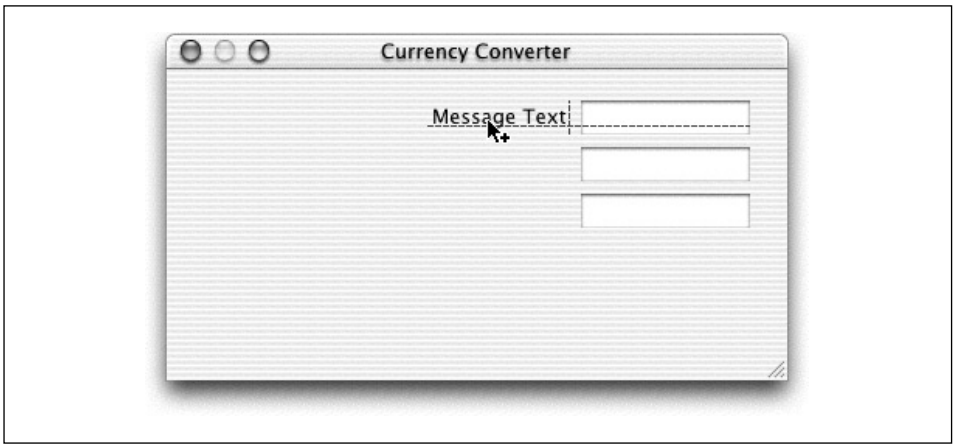
하단의 텍스트 필드는 계산 결과를 보여주기 때문에 다른 텍스트 필드와 속성이 다르다.

1. 세번째 텍스트 필드를 선택한다.
2. Info 윈도우를 불러와 팝업 메뉴에서 Attributes를 선택한다.
3. 사용자가 필드의 내용을 바꾸지 못하도록 Inspector의 Options 섹션에서 Editable 속성이 동작하지 못하도록 한다. 사용자가 다른 응용 프로그램의 내용을 베껴와, 붙일 수 있도록 Selectable 속성을 유지한다.

## Field에 Label할당하기

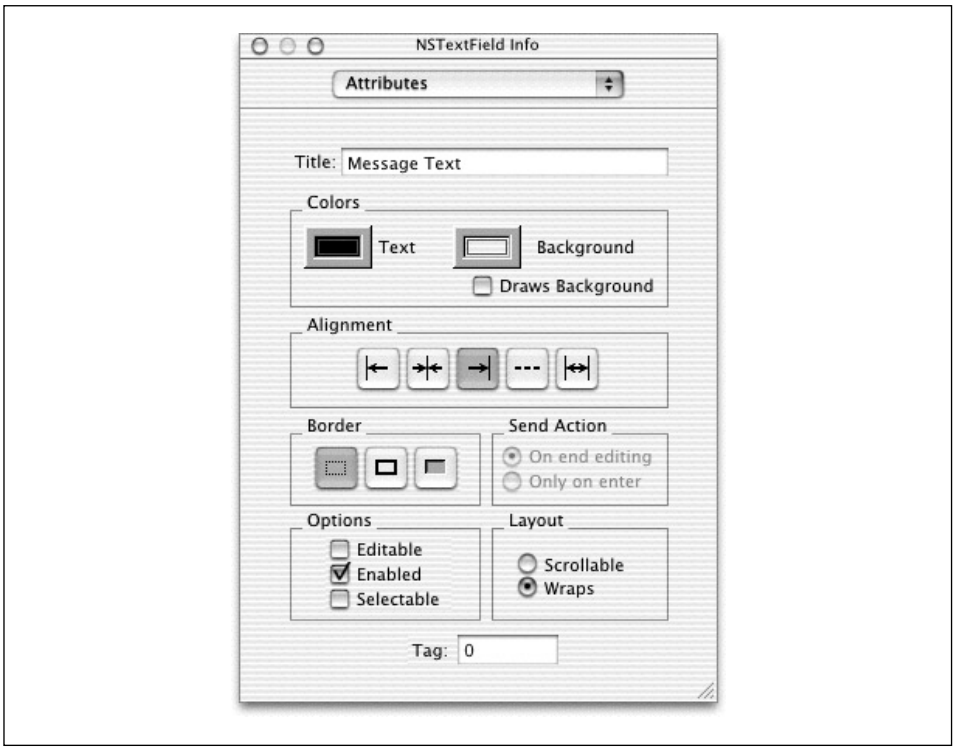
텍스트 필드에 라벨이 없으면 혼란을 줄 수 있다. 그래서, Views 팔레트에서 레디메이드 라벨 객체를 사용하여 라벨을 단다.

1. <그림 7-7>을 참조하여, Views 팔레트에서 윈도우로 Message Text 객체를 드래그한다.

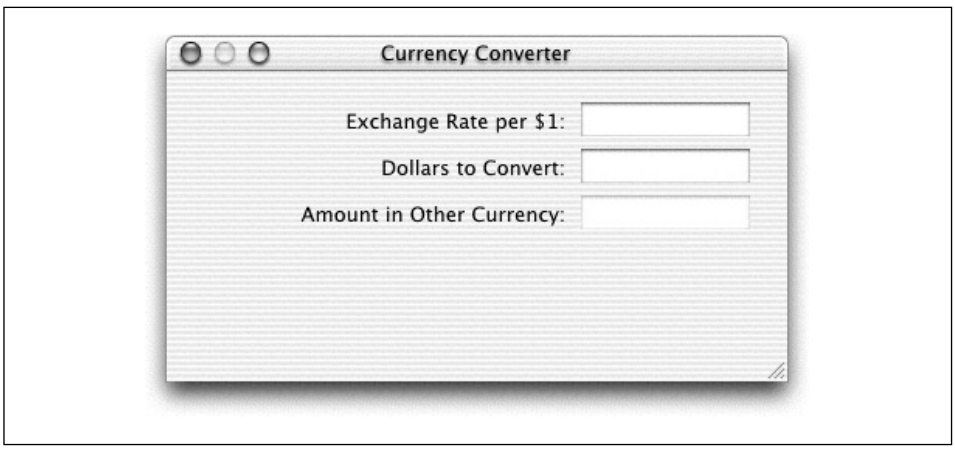



*<그림 7-7> 텍스트 필드와 라벨 배치하기*

2. 텍스트를 오른쪽에 정렬한다. <그림 7-8>과 같이 Message Text객체를 선택하여 Info 윈도우의 Alignment 좌측에서 세번째 버튼을 클릭한다.
3. <그림 7-9>와 같이 텍스트 라벨을 2번 복사하고, 텍스트를 입력한 뒤 정렬한다.




<그림 7-8> NSTextField Info 윈도우

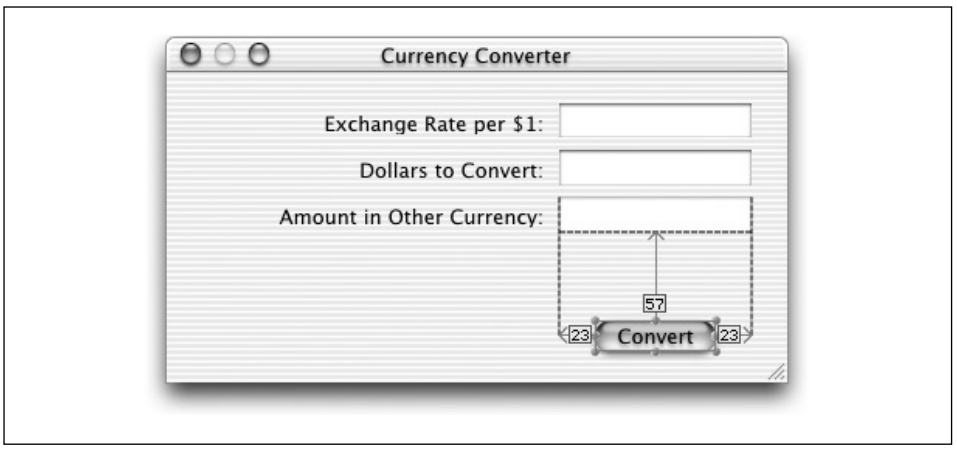



<그림 7-9> Currency Converter의 텍스트 필드 및 라벨 정렬하기

## 인터페이스에 버튼을 추가하고, 초기화시키기

환율변환은 버튼을 클릭하거나 Return 키를 누르면 수행될 수 있다.

1. Views 팔레트에서 버튼 객체를 드래그하여 윈도우의 오른쪽 하단에 놓는다.
2. 버튼 타이틀을 더블 클릭하여 텍스트 라벨을 선택하고, 타이틀을 Convert로 변경한다.
3. NSButton Info 윈도우에서 Attributes를 선택하고, Equiv:라는 라벨이 붙은 팝업메뉴에서 Return을 선택한다. 이로써, 마우스 클릭과 Return 키에 응답할 수 있는 용량을 버튼에 제공할 수 있다.
4. 텍스트 필드 아래에 버튼을 배치한다. 먼저, Aqua 가이드가 나타날 때 까지 버튼을 아래로 드래그한다. 버튼을 선택한 채로 Option 키를 눌러, 마우스 버튼을 해제한다. 커서가 움직이면, Interface Builder는 커서가 가리키고 있는 버튼에서 객체까지의 거리를 보여준다. <그림 7-10>에서와 같이 Option 키를 누르고, 하단 텍스트 필드 위에 커서를 놓은 채로 화살표 키를 사용하여 정가운데에 버튼을 배치한다.



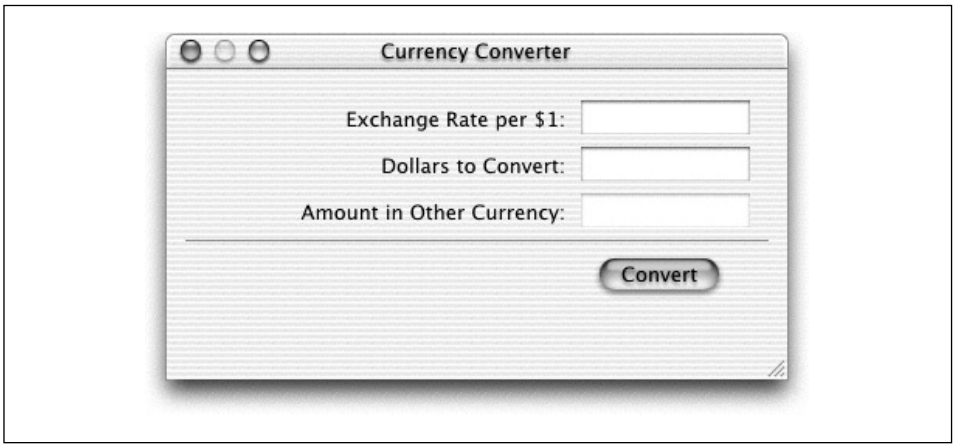

*<그림 7-10> 텍스트 필드와 Convert 버튼 배치하기*

## 수평 장식 라인 추가하기

Currency Converter의 인터페이스는 텍스트 필드와 버튼 사이에 라인을 그려 마무리한다. 수평라인을 그리려면,

1. Views 팔레트에서 인터페이스로 수평 구분선 객체를 드래그한다. 수평 구분선은 Views 팔레트 오른쪽 하단의 Box 객체 아래에 위치한다.

2. <그림 7-11>을 참조하여, 라인이 윈도우를 가로질러 뻗을 때까지 라인의 끝점을 드래그한다.




*<그림 7-11> 수평선을 인터페이스에 추가하기*

3. Aqua 가이드가 나타날 때까지 Convert 버튼을 누른다.

## *Aqua 레이아웃 및 객체 정렬*

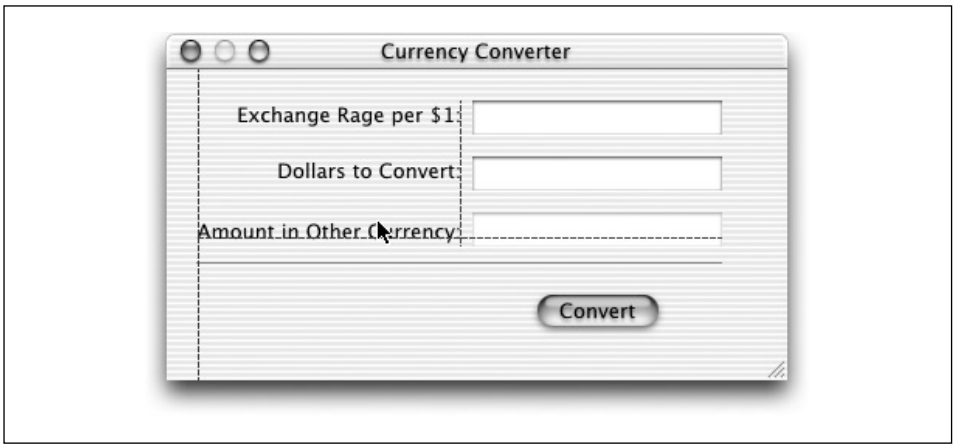
시각적으로 멋진 사용자 인터페이스를 만들려면, 인터페이스 객체를 행과 열로 정렬하면 된다. 눈 대중으로 어림잡아 객체를 정렬하는 작업은 어려운데다, x/y 좌표를 손으로 하나하나 입력하려면 지루하고, 시간 낭비라는 생각이 들 수 있다. 또한, Aqua 객체 위짓의 정렬 작업은 객체에 음영이 있고, UI 가이드라인 매트릭스가 음영을 고려하지 않기 때문에 더욱 어려울 수 있다. Interface Builder는 비쥬얼 가이드와 레이아웃 사각형을 사용하여 객체를 정렬할 수 있도록 도움을 준다.

Cocoa에서 모든 드로잉은 객체 프레임의 영역안에서 수행된다. Interface Builder에는 음영이 있기 때문에 프레임의 경계를 정렬하면,(Mac OS 9으로 작업했듯이) 객체들은 시각적으로 볼때 올바르게 정렬되지 않는다. 예를 들면, Aqua UI 가이드라인에서 Push 버튼은 20픽셀 높여야 한다고 설명하지만, 실제적으로 버튼과 버튼의 음영으로 인해 Push 버튼의 높이는 32픽셀 프레임이 되어야 한다. 여기서 정렬해야 하는 것은 레이아웃 사각형이다.

Layout 메뉴의 Show Layout Rectangles(Command-L)를 통해 IB에서 객체의 레이아웃 사각형을 볼 수 있다. 또한, IB Size Inspector는 팝업을 사용하여 프레임과 레이아웃 사각형 사이를 왔다 갔다 한다. 그래서, 수동으로 적절한 값을 설정할 수 있다.

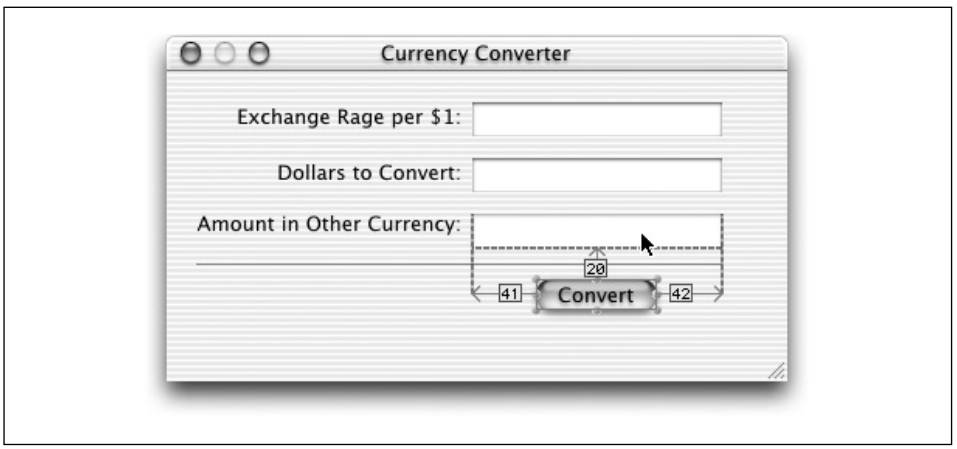
Interface Builder에서는 윈도우에서 객체를 정렬할 수 있는 몇 가지 방법을 제공한다.

- <그림 7-12>에서 보여진 바와 같이 Aqua 가이드에 따라 객체를 마우스로 드래그한다.



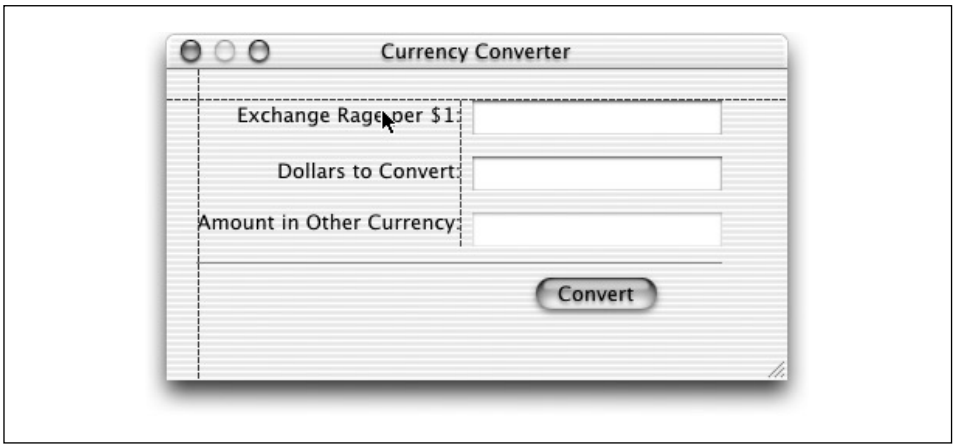

*<그림 7-12> Aqua 가이드를 사용한 객체 정렬*

- <그림 7-13>에서 보여진 바와 같이 화살표 키(그리드를 해제하고, 선택한 객체를 1픽셀 이동시킨다)를 누른다.

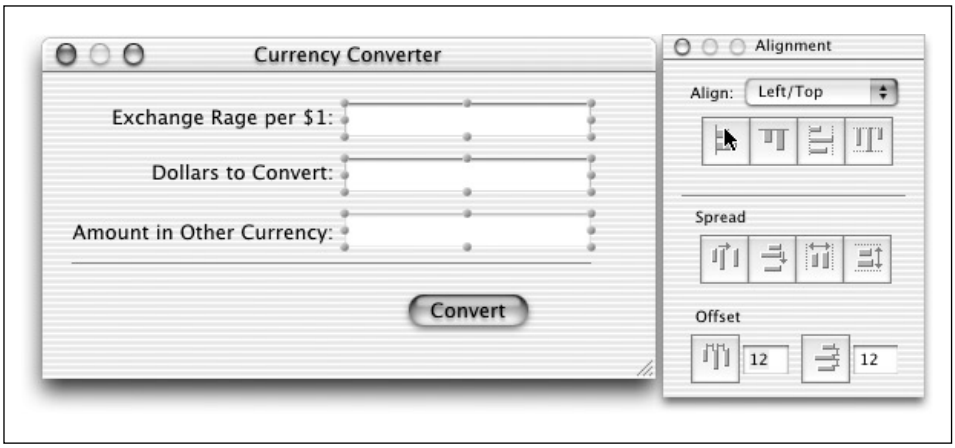



*<그림 7-13> 화살표 키를 사용하여 객체 정렬*

- <그림 7-14>에서 보여진 바와 같이 참조 객체를 사용하여 선택한 객체를 가로행과 세로열로 놓는다.
- <그림 7-15>에서 보여진 바와 같이 내장된 정렬 기능을 사용한다.



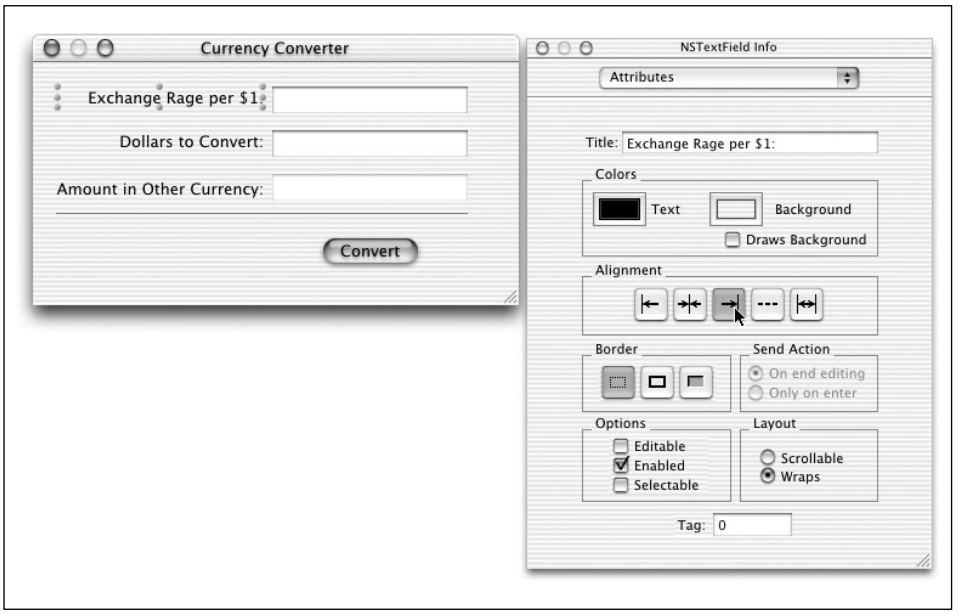

<그림 7-14> 참조를 사용한 객체 정렬



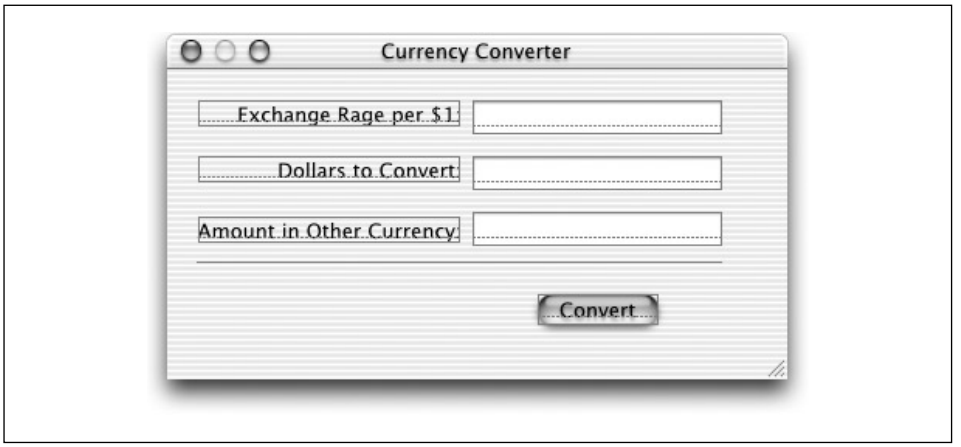

<그림 7-15> Alignment 팔레트를 사용한 객체 정렬

- <그림 7-16>에서 보여진 바와 같이 Info 윈도우의 Size 화면에 원점을 표시한다.
- <그림 7-17>에서 보여진 바와 같이 레이아웃 사각형을 사용한다.

다양한 정렬 커맨드와 툴이 있는 Layout 메뉴에서 Alignment와 Guides 서브메뉴를 살펴보기로 하겠다. 또한, 정렬 툴(Tools 메뉴에서 Alignment를 선택하여)을 사용할 수 있다. 정렬 툴은 플로팅 윈도우를 제공한다. 이 플로팅 윈도우에는 다양한 타입의 정렬기능을 수행하는 버튼이 있다




<그림 7-16> *Info 윈도우를 사용하여 객체 정렬*




<그림 7-17> *레이아웃 사각형을 사용하여 객체 정렬*

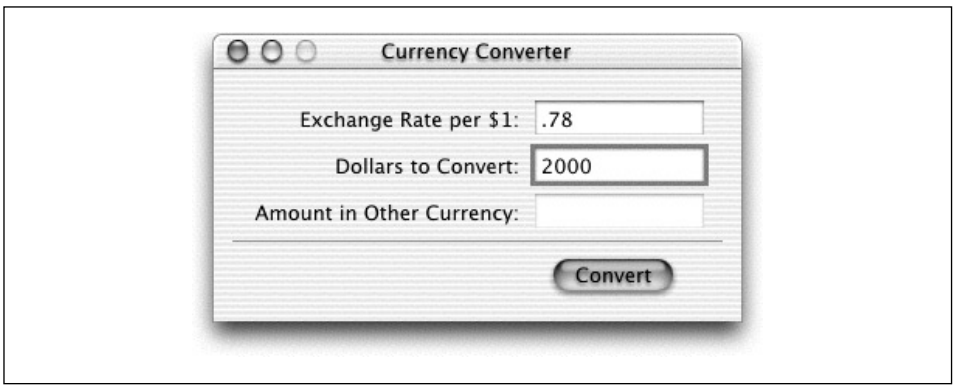
## *인터페이스 객체를 중앙에 놓고, 윈도우의 크기 조정하기*

Currency Converter의 인터페이스는 거의 완성되었다고 볼 수 있다. 마무리 작업은 모든 객체가 중앙에 위치하고, 각 경계면이 올바르게 정렬될 수 있도록 윈도우의 크기를 조정하는 것이다. 현재, 객체는 상단과 우측면만이 정렬되어 있다.

Currency Converter의 경우, 자동화된 Aqua 가이드와 몇 개의 Layout 커맨드를 사용하여 작업을 지속할 수 있다.

1. 세번째 텍스트 라벨(Amount in Other Currency)을 선택하면, 선택이 확대되어(Shift-클릭) 다른 2개의 텍스트 라벨이 포함된다.
2. Layout 메뉴에서 Size to Fit을 선택하여 모든 라벨의 넓이를 최소로 조정한다.
3. Layout 메뉴에서 Same Size를 선택하고, 텍스트 라벨을 동일한 크기로 만든다.
4. Aqua 가이드가 나타나면, 라벨을 윈도우의 왼쪽으로 드래그하여 해제한다.
5. 3개의 텍스트 필드를 선택하고, 올바른 위치에 배치할 수 있도록 가이드를 참조하여 다시 왼쪽으로 드래그한다.
6. 수평 구분선을 단축하고, 버튼을 텍스트 필드 아래로 이동시킨다.
7. 하단의 Convert 버튼과 우측의 텍스트 필드로부터 적당한 거리를 두기 위해 가이드를 참조해 윈도우의 크기를 조정한다.

이 때, 응용 프로그램의 윈도우는 <그림 7-18>과 동일해야 한다.




*<그림 7-18> Currency Converter의 최종 인터페이스*

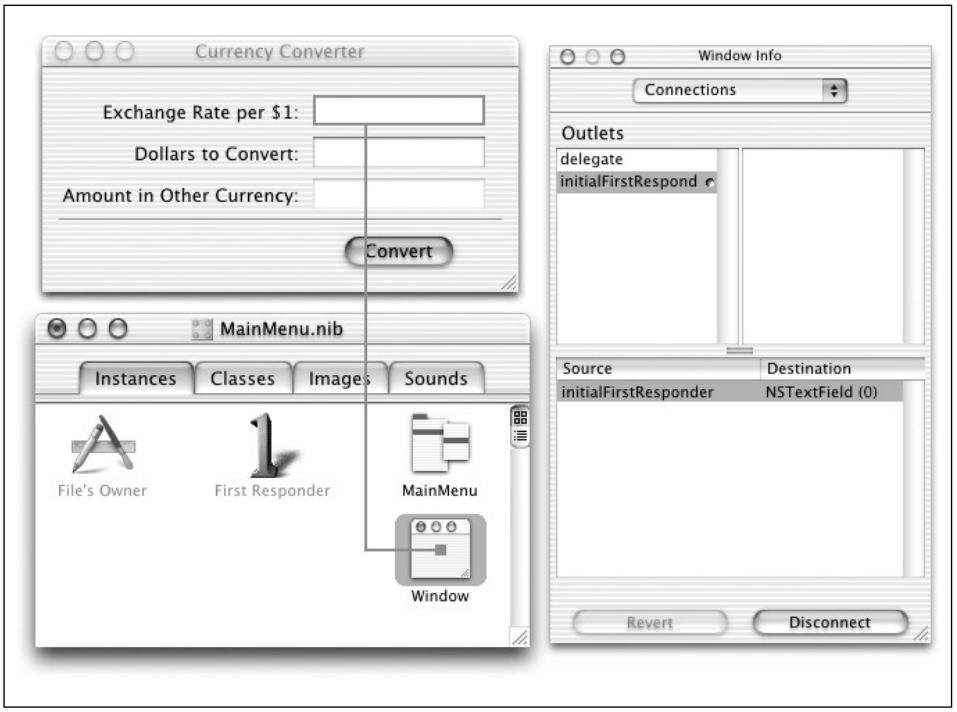
## 텍스트 필드 간의 탭 만들기

Currency Converter를 구성하는 마지막 단계에서는 시각적인 기능보다는 동작 기능에 치중해야 한다. 개발자는 첫 번째 편집 필드에서 두 번째로, 두 번째 편집 필드에서 첫 번째로 탭할 수 있는 기능을 사용자에게 제공해야 한다.

Interface Builder 팔레트의 여러 객체는 `nextKeyView`라는 인스턴스 변수를 가지고 있다. 이 변수는 사용자가 Tab 키(또는 Shift-Tab을 눌렀을 경우엔 이전 객체)를 누를 경우 키보드 이벤트를 수신하기 위해 다음에 나올 객체를 식별한다. 필드간의 탭 기능이 필요하면 `nextKeyView` 변수를 통해 필드를 연결해야 한다.

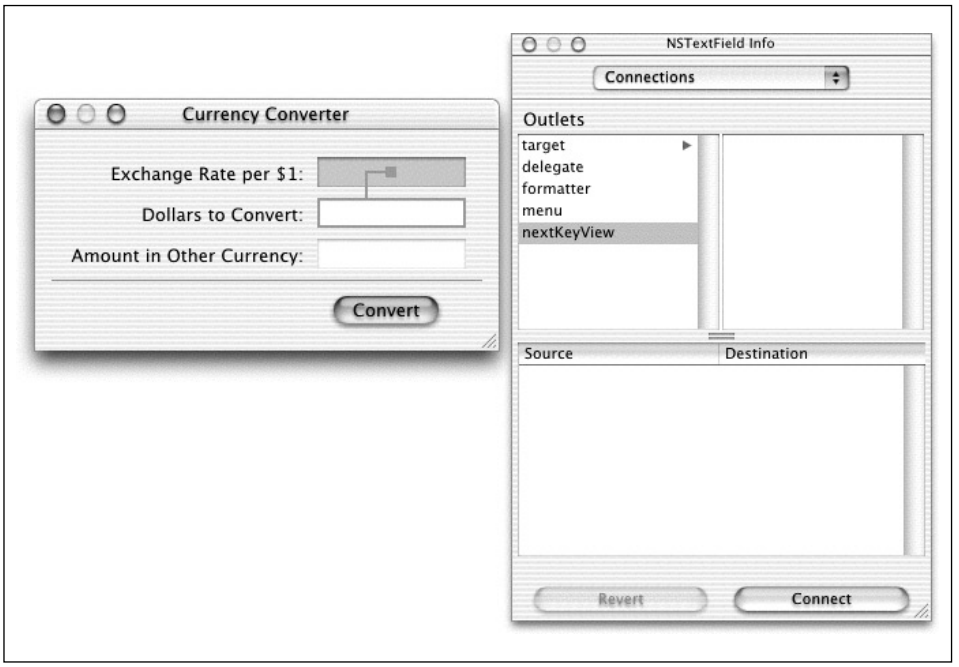
텍스트 필드에서 필드간 탭 기능을 사용하려면 텍스트 필드를 응용 프로그램 윈도우의 `initialFirstResponder`로 만들어야 한다. `initialFirstResponder`는 키보드에서 이벤트를 제일 먼저 수신하는 윈도우의 객체이다. 제8장, *이벤트 처리*에서 FirstResponder와 Responder 체인에 관하여 자세하게 설명한다.

1. `MainMenu.nib` 윈도우의 Instances 패인에서 Window 인스턴스를 클릭하고, Currency Converter의 첫 번째 텍스트 필드로 연결라인을 Control키를 누른채 드래그한다. <그림 7-19>에서와 같이 `initialFirstResponder` 아웃렛을 선택하고, Connect를 클릭한다.




<그림 7-19> *Currency Converter의 initialFirstResponder 연결하기*

2. <그림 7-20>에서와 같이 첫 번째 텍스트 필드를 선택하고, 두 번째 텍스트 필드로 연결라인을 Control 키를 누른채 드래그한다. `nextKeyView`를 선택하고, Connect를 클릭한다. `nextKeyView` 아웃렛은 Tab 키를 누르면 이벤트에 응답할 다음 번째 객체를 식별한다.




*<그림 7-20> 텍스트 필드에 필드 간 탭 연결*

3. 상기 단계를 반복한다. 그러나, 두 번째 필드를 첫 번째 필드로 연결한다.

## 인터페이스 시험하기

이제, Currency Converter 인터페이스가 완성되었다. Interface Builder에서는 1줄의 코드를 작성하지 않고서도 인터페이스를 시험할 수 있다.

1. File→Save All을 선택하고, 작업 내용을 저장한다.
2. File→Test 인터페이스를 선택한다.
3. 텍스트 필드간의 탭, 자르기, 붙이기등 다양한 기능을 시도한다.
4. 작업이 완료되면, Interface Builder Application 메뉴에서 Quit New Application을 선택하여 테스트 모드를 닫는다.

Interface Builder에서 Currency Converter 윈도우의 화면 위치는 응용 프로그램이 구동될 때 윈도우의 초기 위치로 사용된다.

화면의 좌측 상단에 윈도우를 배치한다. 그러면, 편리하게 초기 위치로 사용할 수 있다.

# Currency Converter의 클래스 정의하기

이 섹션에서는 Currency Converter의 컨트롤러와 모델 클래스를 정의한다.

## ConverterController 서브클래스 만들기

제6장, *중요한 Cocoa 패러다임*에서 nib 파일 윈도우의 Classes 디스플레이에서 클래스를 정의했다. ConverterController 클래스를 생성하려면,

1. Interface Builder에서 `MainMenu.nib` 윈도우의 Classes 화면을 선택한다.
2. ConverterController라는 NSObject의 서브 클래스를 만든다.

## ConverterController의 아웃렛 정의하기

ConverterController는 텍스트 필드 인터페이스로 액세스되어야 하기 때문에 아웃렛을 만들어야 한다. ConverterController는 Converter 클래스(정의될 예정)와 통신할 수 있어야 하며, 따라서 그 같은 용도로 사용할 4번째 아웃렛을 필요로 한다.

1. Classes 윈도우에서 ConverterController를 선택한다.
2. 클래스의 우측에 아웃렛 아이콘을 클릭한다.
3. Classes 메뉴에서 Add Outlet을 선택한다.
4. 이 아웃렛을 **`rateField`**라 명명하고, Return 키를 누른다.
5. `rateField` 아웃렛을 만들었기 때문에, 아웃렛을 더 만들려면 Return을 눌러야 한다. `dollarField` 아웃렛을 만들려면, 이 작업을 반복한다. `totalField` 아웃렛을 만들려면 역시 이 작업을 반복한다.
6. `converter` 아웃렛을 ConverterController에 추가한다.

## ConverterController 액션 정의하기

ConverterController는 **`convert:`**라는 1개의 액션 메소드를 보유한다. 사용자가 Convert 버튼을 클릭하면, **`convert:`** 메시지가 타겟 객체인 ConverterController의 인스턴스로 전송된다. 액션은 사용자가 버튼을 클릭하거나 다른 컨트롤 객체를 조작할 때 전송되는 메시지나 발생된 메소드를 참조한다.

1. Classes 메뉴에서 Add Action을 선택한다.
2. 메소드의 이름을 **`convert`**라고 입력한다. IB는 **`:`**를 추가한다.

## Converter Class 정의하기

Converter 클래스는 특수 기능을 상속할 필요가 없다. 특수 기능을 상속하지 않고서도, NSObject의 서브클래스를 생성할 수 있다. 이 모델의 인스턴스는 인터페이스와 직접 통신하지 않기 때문에 아웃렛이나 액션을 필요로 하지 않는다.

1. Classes 화면에서 Converter를 NSObject의 서브클래스로 만든다.
2. `MainMenu.nib`을 저장한다.

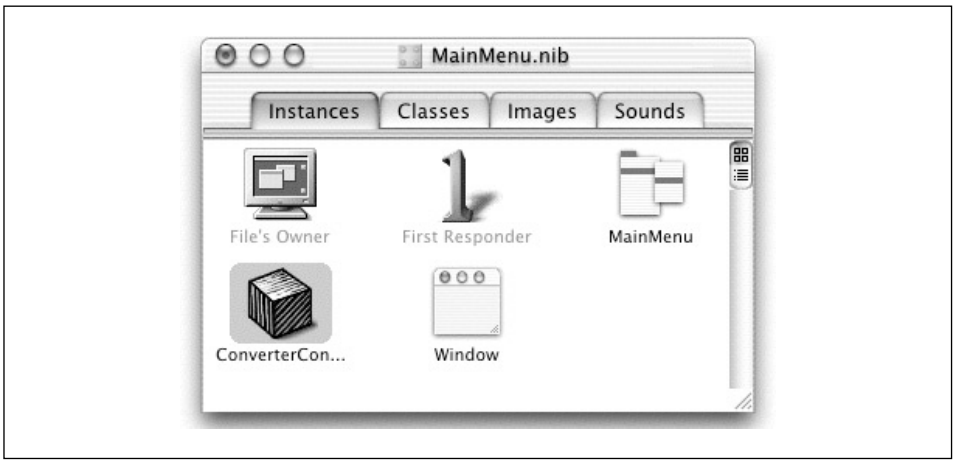
# ConverterController를 인터페이스에 연결하기

이 섹션에서는 ConverterController의 인스턴스를 만들고, Interface Builder를 사용하여 컨트롤러 객체의 아웃렛을 사용자 인터페이스의 객체에 연결하는 방법을 설명한다.

## 클래스의 인스턴스 만들기

Interface Builder의 클래스를 정의하는 최종 단계에서는 클래스의 인스턴스를 만들고, 아웃렛과 액션을 연결한다.

1. Classes 윈도우에서 ConverterController를 선택한다.
2. Classes 메뉴에서 Instantiate를 선택한다. <그림 7-21>에서 하이라이트되어있는 그림처럼, 인스턴스는 Instances 뷰에 나타난다.

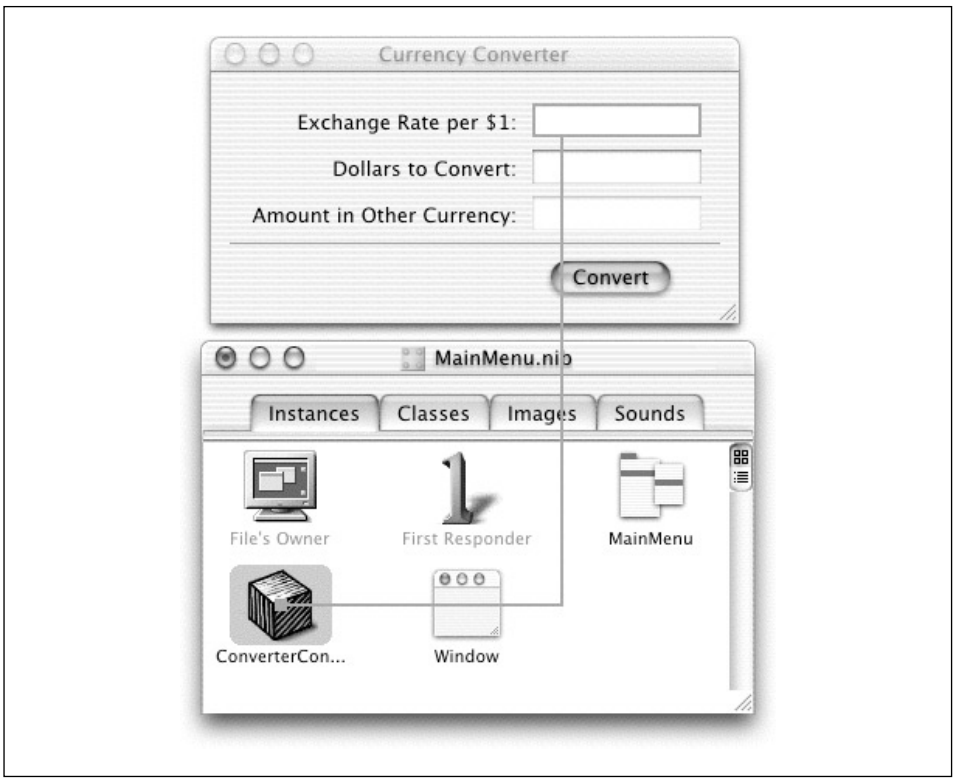



<그림 7-21> ConverterController 인스턴스

## 커스텀 클래스를 인터페이스에 연결하기

이제, ConverterController 객체를 사용자 인터페이스에 연결할 수 있다. 인터페이스의 특정 객체에 ConverterController 객체를 연결하여, ConverterController의 아웃렛을 초기화시킨다. ConverterController는 아웃렛을 사용하여 인터페이스에서 값을 얻어 설정한다.

1. <그림 7-22>에서와 같이 nib 파일 윈도우의 Instances 디스플레이에 있는 ConverterController 인스턴스에서 첫 번째 텍스트 필드로 연결 라인을 Control 키를 누른 채 드래그한다. 텍스트 필드의 윤곽이 잡히면, 마우스 버튼을 해제한다.




*<그림 7-22> ConverterController 인스턴스를 텍스트 필드에 연결하기*

2. Interface Builder는 Info 윈도우의 Connections 화면을 불러온다. 첫번째 필드(`rateField`)에 해당하는 아웃렛을 선택한다.
3. Connect 버튼을 클릭한다.
4. 동일한 단계를 수행하고, ConverterController의 `dollarField`와 `totalField` 아웃렛을 해당 텍스트 필드에 연결한다.

## 인터페이스 컨트롤을 클래스의 액션에 연결하기

버튼이 컨트롤러 객체에 메시지를 실시간으로 전송할 수 있도록 버튼을 Interface Builder에서 액션 메소드에 Convert 버튼을 연결해야 한다.

1. Convert 버튼에서 nib 파일 윈도우의 ConverterController 인스턴스로 연결라인을 Control키를 누른 채 드래그한다. 인스턴스의 윤곽이 잡히면, 마우스 버튼을 해제한다.
2. Connections 화면에서 아웃렛 열의 `target`을 선택하였는지 확인한다.
3. Actions 열에서 `convert:`를 선택한다.
4. Connect 버튼을 클릭한다.
5. `MainMenu.nib` 파일을 저장한다.

## ConverterController를 Converter Class에 연결하기

ConverterController의 아웃렛을 연결할 때 1개의 `converter` 아웃렛이 연결되지 않았음을 알 수 있다. 이 아웃렛은 Currency Converter 응용 프로그램에서 Converter 클래스의 인스턴스를 식별한다. 하지만, 이 인스턴스는 아직 존재하지 않는다.

1. Converter 클래스의 인스턴스를 만든다.
2. ConverterController와 Converter 간의 아웃렛을 연결한다. 힌트: ConverterController에서 Converter 인스턴스로 Control키를 누른 채 드래그한다.
3. `MainMenu.nib`을 저장한다.

# Currency Converter의 Classes 구현하기

Currency Converter 응용 프로그램을 구축하는 최종 단계에서는 예전 단계에서 정의한 클래스를 구현한다.

## 소스 파일 만들기

1. nib 파일 윈도우의 Classes 화면으로 넘어간다.
2. ConverterController 클래스를 선택한다.
3. Classes 메뉴에서 Create Files을 선택한다.
4. `.h` 및 `.m` 파일 옆의 Create 열에서 체크 박스를 선택하였는지 확인한다.

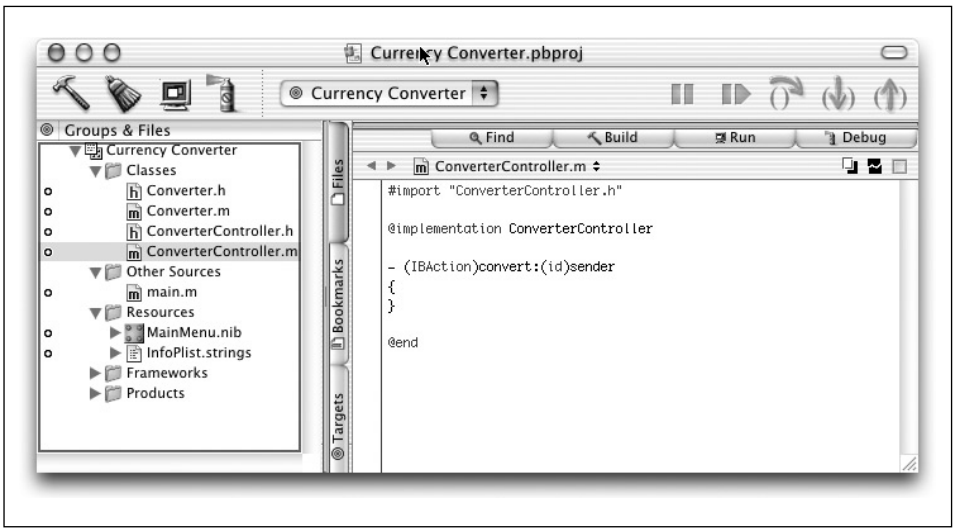
5. Currency Converter 옆의 체크박스를 선택하였는지 확인한다.
6. Choose 버튼을 클릭한다.
7. Converter 클래스를 위해 이 단계를 반복한다.
8. nib 파일을 저장한다.

이제, 이 응용 프로그램의 Interface Builder를 종료하고, Project Builder를 사용하여 응용 프로그램을 마무리한다.

## *Project Builder에서 인터페이스(헤더) 파일 검토하기*

Interface Builder에서 헤더 및 소스 파일을 Currency Converter 프로젝트에 추가하려면, 같은 디스크 폴더에 있는 다른 소스 파일로써 같은 그룹 폴더에 헤더 및 소스 파일을 배치한다. 새롭게 만든 파일이 클래스의 구현 파일이기 때문에 Interface Builder가 자동으로 작업을 수행하지 않을 경우, Classes 그룹으로 이동시킨다.

1. Project Builder의 메인 윈도우를 클릭하여 활성화시킨다.
2. Groups & Files 리스트에서 4개의 파일을 선택하여 <그림 7-23>에서와 같이 Classes 그룹으로 드래그한다.




*<그림 7-23> 소스 파일을 Classes 그룹에 추가하기*

## 메소드 선언 추가하기

인스턴스 변수나 메소드 선언을 Interface Builder가 만든 헤더 파일에 추가할 수 있다. 일반적으로 이 작업을 수행하긴 하지만, ConverterController의 경우에는 이 작업이 반드시 필요한 것은 아니다. 그러나, ConverterController 객체가 계산 결과를 받을 수 있도록 메소드를 Converter 클래스로 추가해야 한다. 메소드를 `Converter.h`에 선언한다.

1. 프로젝트 브라우저에서 `Coverter.h`를 선택한다.
2. `convertAmount:atRate:` 선언을 삽입한다.

```
#import <Cocoa/Cocoa.h>
@interface Converter:NSObject
{
}
- (float)convertAmount:(float)amt atRate:(float)rate;

@end
```

이 선언은 `convertAmount:atRate:`가 `float`의 인수 2개를 취하고, `float` 값을 돌려보낸다는 것을 의미한다. 메소드 이름에 `convertAmount:`와 `atRate:`와 같이 콜론이 있다면, 이는 인수를 제공하는 키워드를 나타낸다.

이제, 구현 파일을 업데이트해야 한다.

## Currency Converter의 클래스를 구현하기

Converter 클래스의 경우, `Converter.h`에 선언된 메소드를 구현한다. 메소드는 `@implementation <class name>`및 `@end` 사이에서 구현된다. 따라서, 여기에 Converter 코드를 추가한다.

1. Project Builder의 메인 윈도우에 있는 Classes 그룹에서 `Converter.m`을 선택한다.
2. `convertAmount:`용 코드를 삽입한다.

```
#import "Converter.h"
@implementation Converter
- (float)convertAmount:(float)amt atRate:(float)rate
{
    return (amt * rate);
}
@end
```

메소드는 단순히 2개의 인수를 다중화하고, 결과를 리턴한다.

3. 다음은, Interface Builder가 생성한 ConverterController.m에서 convert:의 "비어있는" 구현을 업데이트한다.

```
- (IBAction)convert:(id)sender
{
    float rate, amt, total;

    amt = [dollarField floatValue];
    rate = [rateField floatValue];

    total = [converter convertAmount:amt atRate:rate];

    [totalField setFloatValue:total];
    [rateField selectText:self];
}
```

4. 소스 파일의 상단에 다음의 코드를 추가하여 ConverterController.m가 Converter.h를 임포트하는지 확인한다.

```
#import "Converter.h"
```

convert: 메소드는 다음을 수행한다.

- rate와 dollar 필드에 입력된 부동 소수점 값을 얻는다.
- convertAmount:atRate: 메소드를 발생시키고, 반환 값을 얻는다.
- setFloatValue:를 사용하여 Amount in Other Currency 텍스트 필드(totalField)에 반환 값을 작성한다.
- selectText:를 rate 필드로 전송한다. 이는 필드에서 텍스트를 선택하거나 텍스트가 없을 경우, 사용자가 다른 계산을 시작할 수 있도록 커서를 삽입한다.

## *Currency Converter 구축 및 운용*

이 섹션에서는 응용 프로그램을 구축하고 운용하는 방법을 설명한다.

### *프로젝트 구축하기*

Project Builder 패널에서 빌드 프로세스를 시작한다.

1. 소스 코드 파일과 프로젝트 변경사항을 저장한다.
2. 메인 윈도우에서 Build 버튼을 클릭한다.

Build 버튼을 클릭하면, 빌드 프로세스가 시작된다. Project Builder가 완료되고, 오류가 발견되지 않으면, 프로젝트 윈도우의 좌측 하단에 Build Succeeded가 나타난다.

## Currency Converter 운용하기

동일한 Rates와 Dollar를 입력하고, Convert를 클릭한다. 그리고, 필드에 텍스트를 선택하고, Application 메뉴에서 Services를 선택한다. 이 메뉴는 선택된 텍스트를 처리할 수 있는 다른 응용 프로그램을 나열한다.

물론, 응용 프로그램이 복잡해질수록 더욱 철저하게 시험해야 한다. 시험을 통하여 전체 디자인에서, 인터페이스에서, 커스텀 클래스 정의에서, 또는 메소드와 함수의 구현에서 수정을 필요로하는 오류나 결함을 찾을 수 있다.

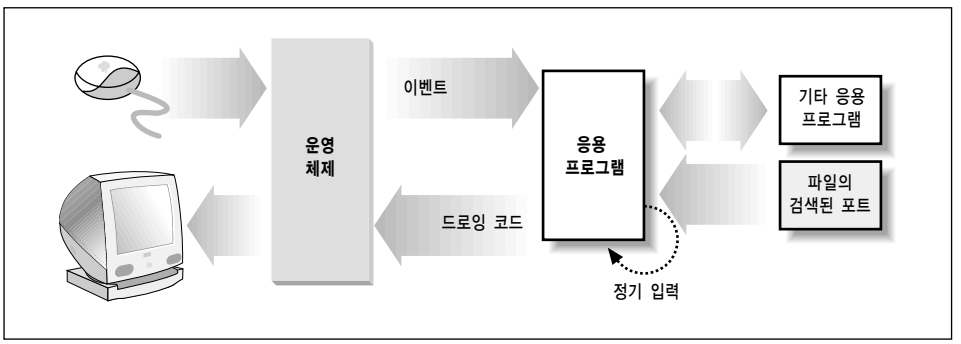
Currency Converter가 단순한 응용 프로그램이라 하더라도, 이 프로그램은 앞 장에서 제공한 모든 개념과 기법을 결합한다. 지금쯤, Cocoa 응용 프로그램을 개발하기 위해 필요한 능력을 좀 더 습득했을것이다. 다음 사항을 다시 한번 복습해 보기로 하겠다.

- Model-View-Controller 패러다임을 사용하여 응용 프로그램을 디자인한다.
- Interface Builder로 GUI를 구성한다.
- 인터페이스를 시험한다.
- 클래스의 아웃렛과 액션을 지정한다.
- 클래스 인스턴스를 아웃렛과 액션을 통해 인터페이스로 연결한다.
- Class 구현의 기본 원리
- 응용 프로그램을 구축하고, 오류를 해결한다.

# 8

# 이벤트 처리

그래픽 인터페이스는 마우스 클릭이나 키 스트로크 같은 사용자 이벤트로 구동된다. 그러나, 운용중인 응용 프로그램은 사용자 인터페이스에서 발생하지 않은 네트워크 인터페이스를 통해 도달한 패킷, 주기적인 타이머 발생, USB 포트에 연결된 입력 장치 또는 드라이브로 삽입된 CD등과 같은 이벤트도 수신할 수 있다. 궁극적으로 객체 지향 프로그램의 이벤트는 <그림 8-1>에서 보여진 바와 같이 응용 프로그램의 객체로 전송될 메시지를 발생시킨다. 이 장은 사용자와 프로그램이 발생시킨 이벤트를 중점적으로 설명하며, 프로그래머가 Cocoa에서 이벤트를 차단, 처리 및 조정하는 방법에 관해 설명한다.



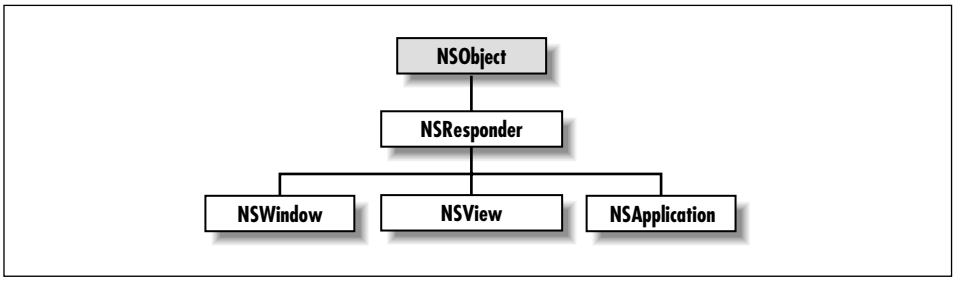

*<그림 8-1> 이벤트를 수신하는 Cocoa 응용 프로그램*

## 사용자 발생 이벤트에 대한 응답

이 섹션에서는 사용자 이벤트 처리 관점에서 Cocoa 응용 프로그램의 구조를 설명한다. 또한 코어 프로그램 프레임워크의 클래스 및 뷰 계층, 이벤트 사이클, 리스폰더 체인등에 대해 심층적으로 분석한다. 추후, 응용 프로그램의 윈도우에 컬러 도트를 드로잉하여 사용자의 마우스 클릭에 응답하는 단순한 응용 프로그램을 구축해 본다.

## 4개의 응용 프로그램

NSResponder, NSWindow, NSView 및 NSApplication(<그림 8-2> 참조) 클래스는 운용중인 응용 프로그램에서 핵심적인 역할을 수행한다. 각 클래스는 응용 프로그램의 사용자 인터페이스 드로잉이나 이벤트에 응답하기 같은 동작에서 중요한 역할을 담당한다. 각 클래스의 상호작용 구조를 "코어 프로그램 프레임워크"라고 한다.




<그림 8-2> 4개의 응용 프로그램 클래스 계층

### NSResponder

NSResponder는 추상화 클래스이다. 그러나 상속된 모든 클래스에서 이벤트를 처리할 수 있다. 또한, NSResponder는 다양한 마우스 및 키보드 이벤트가 발생했을때 생성되는 메시지와 응용 프로그램의 객체 사이에서 이벤트 처리의 메카니즘을 정의한다. 특히 이벤트가 처리될 때까지 리스폰더로 이벤트를 전달하는 리스폰더 체인을 정의한다. 리스폰더 체인은 "이벤트 사이클 및 첫 번째 리스폰더" 섹션에서 설명한다.

### NSWindow

NSWindow 객체는 화면상에서 각각의 물리적 윈도우를 관리한다. 또한, 윈도우의 프레임 영역을 드로잉하고, 윈도우 닫기, 이동하기, 크기 조정하기 및 기타 조작에 대한 사용자 액션에 응답한다.

NSWindow의 주요 역할은 컨텐트 영역에서 응용 프로그램의 사용자 인터페이스(즉 사용자 인터페이스의 일부)를 보여주는 것이다. 컨텐트 영역은 타이틀 바 하단과 윈도우 프레임의 내부에 있는 영역이다. 윈도우의 컨텐트가 둘러싸고 있는 뷰 계층이며, 뷰 계층의 맨 하단은 컨텐트 영역을 차지하는 컨텐트 뷰이다. NS윈도우는 사용자 이벤트 위치를 기반으로 첫 번째 리스폰더로 동작하기 위해 컨텐트 영역에서 NSView를 할당한다.

NSWindow는 커스텀 객체를 델리게이트로 할당하여, 다른 활동에 관여할 수 있도록 한다.

### *NSView*

윈도우의 컨텐트 영역에서 볼 수 있는 객체가 바로 NSView이다.(실제로 NSView는 추상화 클래스이기 때문에, 이들 객체는 NSView 서브클래스의 인스턴스이다) NSView 객체는 드로잉을 수행하며, 마우스와 키보드 이벤트에 응답한다. 각 NSView는 특정 윈도우와 연결된 사각형 영역을 제공한다. NSView는 이 영역내에서 이미지를 만들어, 사각형 내에서 발생하는 이벤트에 응답한다.

윈도우에서 NSViews는 계층 상단의 컨텐트 뷰와 함께 뷰 계층에 논리적으로 배열된다. NSView는 윈도우, 슈퍼뷰 및 서브뷰를 참조한다. 또한, 이벤트에 대한 첫 번째 리스폰더가 될 수 있으며, 이벤트 체인에서 다음 순서의 리스폰더가 될 수 있다. NSView의 프레임과 영역은 화면 위치, NSView의 크기 및 드로잉을 위한 좌표계를 정의한다.

### *NSApplication*

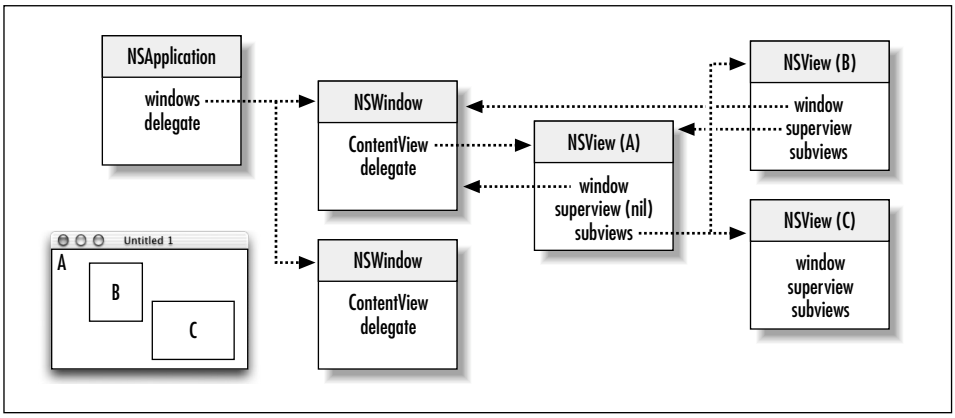
모든 응용 프로그램은 1개의 NSApplication 객체를 보유하여 응용 프로그램의 전반적인 동작을 감시 및 조정한다. 이 객체는 이벤트를 해당 NS윈도우(교대로, NSViews로 이벤트를 분배)로 디스패치한다. NSApplication 객체는 윈도우를 관리하고, 활성화 및 비활성화 상태뿐만 아니라 상태 변경을 감지 및 처리한다. 전역 변수 NSApp는 각 응용 프로그램에서 NSApplication 객체를 제공한다. 코드와 NSApp를 조정하려면, 커스텀 객체를 델리게이트로 할당해야 한다.

## *뷰 계층*

컨텐트 뷰는 삼면이 타이틀 바와 프레임으로 둘러싸인 윈도우의 컨텐트 영역 내에 위치한다. 또한, <그림 8-3>에서 보여진 바와 같이 윈도우의 뷰 계층에 있는 NSView이다. 개념적으로 보면, 1개 이상의 NSViews는 나뭇 가지처럼 컨텐트 뷰에서 분기될 수 있으며, 다른 NSViews가 보조 NSViews에서 분기될 수 있다. 컨텐트 뷰를 제외한 각 NSView은 상위로 단지 1개의 NSView를 가질 수 있다. NSView의 보조 뷰를 서브뷰로 정하고, 상위 뷰를 슈퍼뷰로 정한다.

화면상에서 포위 관계는 슈퍼뷰와 서브뷰의 관계를 결정한다. 슈퍼뷰는 서브뷰를 둘러싼다. 이 관계를 참조하여 화면에서 드로잉을 구현한다.

- 서브뷰를 조정하여 간단하게 슈퍼뷰를 구축한다.(예를 들면, NSBrowser는 혼합된 NSView의 인스턴스이다)




<그림 8-3> 뷰 계층

- 서브뷰는 슈퍼뷰의 좌표계에 위치한다. 그래서, NSView를 이동하거나 좌표계를 변경할 경우, 모든 서브뷰는 동시에 움직이고, 변경된다.
- NSView는 독립적인 좌표계를 갖추고 있기 때문에 좌표계나 슈퍼뷰의 위치가 변경되어도 따라서 변경되지 않는다.

코어 프로그램 프레임워크는 해당 객체에 액세스하기 위해 응용 프로그램에 대한 몇 가지 방안을 제공한다. 그래서, 계층에 있는 객체의 아웃렛이나 인스턴수 변수를 모두 정의할 필요가 없다.

- 전역 변수 NSApp는 NSApplication 객체를 식별한다. 해당 메시지를 NSApp에 전송함에 따라, 응용 프로그램의 NSWindow 객체(`windows`), 키 및 메인 윈도우(`keyWindow` 및 `mainWindow`), 현재 이벤트(`currentEvent`), 메인 메뉴(`mainMenu`), 응용 프로그램의 델리게이트(`delegate`)를 얻을 수 있다.
- NSWindow 객체를 확인하면, NSWindow의 컨텐트 뷰를 얻을 수 있고(`contentView`를 전송함으로써), 그것으로부터 윈도우의 모든 서브뷰를 얻을 수 있다. 메시지를 NSWindow 객체로 전송하면 새로운 이벤트(`currentEvent`), 새로운 첫 번째 리스폰더(`firstResponder`) 및 델리게이트(`delegate`)를 얻을 수 있다.
- NSView가 참조하는 대부분의 객체들은 NSView에서 얻을 수 있다. 여기서, NSView의 윈도우, 슈퍼뷰, 서브뷰를 찾아볼 수 있다.

## *NSView의 서브클래스 만들기*

NSView의 커스텀 서브클래스(또는 NSView에서 상속받은 클래스)를 만들어 커스텀 이벤트 처리 또는 드로잉을 수행하고자 할 경우, 기본 절차를 동일하게 적용한다.

1. Interface Builder에서 NSView의 서브클래스를 정의한다. 그리고, 헤더 및 구현 파일을 만든다.
2. Views 팔레트에서 윈도우로 CustomView 객체를 드래그하고, 크기를 조정한다. 그리고 난 뒤, CustomView 객체를 선택한 채로, Info 윈도우의 Custom Class 화면을 선택하고, 커스텀 클래스를 선택한다. 아웃렛과 액션을 연결한다.
3. 지정된 초기화기 `initWithFrame:`를 오버라이드하여 커스텀 초기화를 수행한다. 이 메소드의 인수는 Interface Builder에서 설정된 NSView의 프레임 사각형이다.
4. `drawRect:`를 구현하여 드로잉한다.

제13장, *To Do: 확장*에서 마우스 클릭에 응답하는 NSButtonCell(NSView에서 상속받지 않음)의 서브클래스를 만드는 방법에 관해 설명한다. 커스텀 NSViews가 이벤트를 처리하는 방식은 어려울 수 있다. 커스텀 NSViews를 사용자 액션에 응답하게 하려면, 몇 가지 작업을 수행해야 한다.

- NSView에서 선택을 처리하고자 할 경우, `acceptsFirstResponder`를 오버라이드하여 `YES`를 반환한다.(NSView 동작은 기본적으로 `NO`를 반환한다)
- 원하는 NSResponder 이벤트 메소드(`mouseDown:`, `mouseDragged:`, `keyDown:` 등)를 오버라이드한다.

```
- (void)mouseDown:(NSEvent *)event {
    if ([event modifierFlags] & NSControlKeyMask){
        [self doSomething];
}
```

윈도우, 눌러진 변경자 키, 문자 및 키 코드, 그리고 기타 정보를 통해 사용자 액션 위치를 파악하려면 NSEvent 인수를 조회한다.

`display`를 NSView로 전송하면, `drawRect:` 메소드 및 각 서브뷰의 `drawRect:`가 발생한다. 이 메소드는 NSView의 영역에서 구현된다. `drawRect:` 인수는 드로잉이 발생하는 사각형 영역에 있다. 이는 뷰 영역의 일부가 업데이트되어야 한다는것을 뷰에게 알리는 것이다. NSView을 드로잉하려면, 다음과 같은 작업을 수행한다.

- NSBezierPath, NSString, NSFont, 또는 NSColor 클래스를 사용한다.
- `NSRectFill` 및 `NSFrameRect`(`NSGraphics.h`)과 같은 Application Kit 기능을 호출한다.

- NSImage를 구성한다.
- PDF 연산에 해당하는 C 함수를 호출한다.

상태가 변경되고, 객체를 다시 드로잉해야 할 경우, `YES` 인수를 가진 `setNeedsDisplay:`를 발생시킨다.

NSView의 드로잉과 구성에 관한 자세한 내용은 부록A, *Cocoa에서 드로잉하기*를 참조한다.

## 이벤트 사이클 및 첫 번째 리스폰더

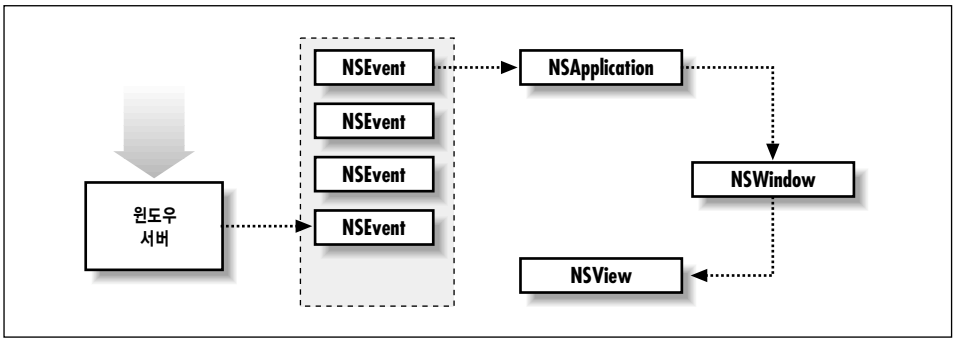
어떤 면에서 보면, NSApplication 객체는 응용 프로그램의 마스터 컨트롤러이다. NSApplication 객체의 주요 역할은 *이벤트 루프*를 수행하는 것이다. Cocoa 플랫폼에 대기하고 있는 이벤트를 1개씩 선택하여, 어떤 객체로 하여금 이벤트를 처리하게 할 것인지를 결정한다. 그리고, 특이사항을 포함한 NSEvent 객체를 건내주는 방식으로 메시지를 전송한다. 이벤트 메시지는 NSApplication에서 해당 윈도우, 윈도우의 뷰(일반적으로, 컨트롤), 타겟 객체순으로 전송된다.

버튼을 클릭하였을때 버튼이 이를 어떻게 인식하는 지 알아보기로 하자. 커스텀 객체는 타겟/액션, 델리게이션을 통해 직접적으로 버튼 클릭에 응답한다.("델리게이션" 섹션에서 보다 자세한 내용을 설명한다) 응용 프로그램 객체가 메시지에 응답을 완료하면, 컨트롤이 해제되고, NSApplication으로 반환된다. 그곳에서 루프를 다시 반복하여, 다음 순서로 대기하고 있는 이벤트를 처리한다.

일반적으로, 이 사이클(*이벤트 사이클*)은 응용 프로그램(링크된 모든 프레임워크를 포함)이 윈도우 서버로 Quartz 코드 스트림을 전송하여, 응용 프로그램 인터페이스를 드로잉할 때 구동한다. 이 때, 응용 프로그램은 메인 이벤트 루프를 시작하고, 사용자로부터 입력을 받기 시작한다. 사용자가 마우스를 클릭하거나 드래그하고, 또는 키보드에 입력할 때 윈도우 서버는 액션을 감지 및 처리하고, 응용 프로그램에 이벤트로 전송한다.

### 이벤트 큐 및 이벤트 디스패칭

응용 프로그램을 구동하면, NSApplication 객체 (NSApp)는 이벤트 루프를 시작하고, 윈도우 서버에서 이벤트를 수신하기 시작한다. NS이벤트가 도달함에 따라, 수신한 순서대로 이벤트 큐를 정한다. <그림 8-4>에서 보여진 바와 같이 NSApp는 상위 이벤트를 수신하여 분석한 뒤 해당 객체에 이벤트 메시지를 전송한다.(이벤트 메시지는 NSResponder에 의해 정의되며, 특정 이벤트에 해당한다) NSApp가 이벤트 처리를 종료하면 다음 이벤트를 수신하여 응용 프로그램이 종료될 때까지 처리과정을 반복한다.




<그림 8-4> 이벤트 큐

이벤트 라우팅은 이벤트 유형에 따라 결정된다. NSApp은 대부분의 이벤트 메시지를 사용자 액션이 발생한 NSWindow로 전송한다. NSWindow는 마우스 이벤트를 계층 뷰에 있는 1개의 객체, 즉 NSView로 전송한다. 키 이벤트는 첫 번째 리스폰더로 전송된다. NSView가 첫 번째 리스폰더 상태를 수신하여, 이벤트 메시지에 해당하는 NSResponder 메소드를 정의한 이벤트에 응답할 수 있다면, 이벤트는 처리된다. NSView가 이벤트를 처리할 수 없다면, 객체가 메시지를 처리할 때까지 리스폰더 체인의 다음 순서에 있는 리스폰더로 이벤트를 전송한다.(자세한 내용은 "첫번째 리스폰더 및 리스폰더 체인" 참조) 객체가 이벤트를 처리할 때까지 리스폰더 체인를 순회한다.

NSWindow는 윈도우 이동 이벤트, 윈도우 크기 조정 이벤트, 윈도우 노출 이벤트 등 일부 이벤트를 처리하나 NSView에 이들 이벤트를 전송하지는 않는다.(NSWindow가 자체적으로 이들 이벤트를 처리하기 때문에 NSResponder에서 이벤트를 정의할 필요는 없다) 또한, NSApp는 응용 프로그램 활성화 및 응용 프로그램 비활성화 이벤트를 처리한다.

## 이벤트 유형과 추적

윈도우 서버는 사용자 액션을 이벤트로 취급한다. 또한, 이벤트를 윈도우와 연관지어 윈도우를 생성한 응용 프로그램에 이들 이벤트를 보고한다. 이벤트는 사용자 액션에서 파생된 정보로 구성된 NSEvent 인스턴스 객체이다.

NSResponder가 정의한 모든 이벤트 메소드(`mouseDown:` 및 `keyDown:`)는 NSEvent를 인수로 취급한다. NSEvent를 조회하여 윈도우, 윈도우 내부의 이벤트 위치, 이벤트가 발생한 시간(시스템 구동과 관련)등을 파악할 수 있다. 또한 변경자 키(Command, Shift, Option 및 Control)를 눌렀을 때 문자 및 키, 다양한 유형의 정보를 포함하고 있는 코드를 확인할 수 있다.

또한, NSEvent는 자신이 제공하는 여러 이벤트 유형을 포함한다. 수 많은 이벤트 유형(즉, NSEvent-Type)이 존재하며, 이들을 4개의 카테고리로 나눌 수 있다.

- **키보드 이벤트.** 이 이벤트는 키를 누르거나, 누른 키를 해제할 때, 또는 변경자 키를 바뀔 때 발생한다. 물론, 키 누름 이벤트가 가장 유용하게 사용된다. 키 누름 이벤트를 처리할 때, NSEvent에 `characters` 메시지를 전송하여 문자나 이벤트에 연결된 문자를 결정한다.
- **마우스 이벤트.** 이 이벤트는 마우스 버튼(즉, 위 아래로 움직임) 상태를 변경하거나, 마우스를 드래그할 때 발생한다. 또한, 이 이벤트는 버튼을 누르지 않고 단순히 마우스를 움직일 때 발생한다.
- **Tracking-rectangle 이벤트.** 응용 프로그램이 윈도우에 Tracking Rectangle을 설정할 것을 요청할 경우, 윈도우 서버는 이 이벤트를 발생시킨다. 윈도우 서버는 커서가 사각형에 들어가거나 나올 때 마우스로 입력 및 닫기 이벤트를 만든다.
- **주기적 이벤트.** 타이머가 이 이벤트를 발생시킨다. 주기적인 이벤트는 특정 시간 간격을 경과할 경우 응용 프로그램에 통보한다. 응용 프로그램은 주기적인 이벤트를 특정 주파수의 이벤트 큐에 배치할 것을 요청할 수 있다. 일반적으로, 이 이벤트는 루프를 추적할 동안 사용된다.(이 이벤트는 NSWindow로 전송되지 않는다)

이벤트를 응용 프로그램에 전송하였을 때 Cocoa의 이벤트 추적 기능을 통해 이벤트를 확인한다. `NSTraceEvents` 플래그를 설정하여 응용 프로그램의 이벤트를 추적할 수 있다. 응용 프로그램의 이벤트를 추적하는 방법은 여러가지가 있다. 그러나 이들 방식은 Mac OS X 커맨드 라인 인터페이스를 사용해야 한다. 이 인터페이스는 `/Applications/Utilities`에 위치한 Terminal 응용 프로그램에서 사용할 수 있다. 이 응용 프로그램을 찾아 구동한다.

윈도우 서버에서 Project Builder 응용 프로그램을 전송한 이벤트를 확인하려면, 먼저 응용 프로그램의 구동 여부를 확인해야 한다. 그리고 난뒤, 다음 명령어를 Terminal 윈도우에 입력한다.

```
/Developer/Applications/Project\ Builder.app/Contents/MacOS/Project\ Builder
-NSTraceEvents YES
```

이 명령어는 Project Builder 응용 프로그램을 구동시켜, `NSTraceEvents` 플래그를 `YES` 값으로 설정한다. "Project" 및 "Builder"로 분리된 공간 앞에 Backslash를 표시한다. Backslash는 커맨드라인 인터프리터가 다음 영역이 경로명(커맨드의 다른 파라미터를 분리하는 공간과는 대조적임)의 일부라는 것을 인식할 수 있도록 하기 위해 사용한다.

마우스를 Project Builder의 사용자 인터페이스 위로 움직여 버튼을 클릭하면 Terminal 윈도우에서 나타나는 각각의 이벤트를 볼 수 있다. 추적 정보는 Project Builder 세션을 위해서만 사용할 수 있기 때문에 이벤트 추적을 가능케 하는 이 메소드를 "One shot"이라 한다. 다음번에 응용 프로그램을 구동할 때는 정상적으로 동작한다.

이벤트를 추적하는 또 다른 메소드는 응용 프로그램의 환경설정에서 `NSTraceEvent` 플래그를 설정하는 것이다. 이 작업을 수행하면, 응용 프로그램의 환경설정을 변경하여 운용을 중단할 때 까지 이벤트 추적을 할 수 있다. 이 메소드는 이벤트 추적 출력이 Terminal 윈도우 대신에 Mac OS X 콘솔로 전송되기 때문에 상기 언급한 메소드와는 약간의 차이가 있다. 콘솔은 `/Applications/ Utilities`에 위치한 Console 응용 프로그램에서 사용할 수 있다. 콘솔은 응용 프로그램(및 운영 체제)이 상태 및 디버깅 정보를 보여주는 곳이다. 이제, Console 응용 프로그램을 배치하여 구동한다.

커맨드 라인에서 Project Builder의 환경설정을 변경하려면, `defaults` 커맨드를 사용한다. 이 커맨드는 응용 프로그램의 그래픽 사용자 인터페이스에서 사용 불가한 응용 프로그램 환경 설정을 해독하고 작성할 수 있도록 한다. 다음을 Terminal 윈도우에 입력하여 Project Builder의 이벤트를 추적한다.

```
defaults write com.apple.ProjectBuilder NSTraceEvents YES
```

Project Builder가 운용중이라면 종료한다. 그리고 Finder에서 다시 구동한다. Project Builder와 상호 작용함에 따라 Console 윈도우에 나타난 이벤트 추적 정보를 확인해야 한다. 이벤트 추적을 멈추려면 Terminal 윈도우에 다음과 같이 입력한다.

```
defaults delete com.apple.ProjectBuilder NSTraceEvents
```

Cocoa의 이벤트 추적 장치가 제공하는 모든 이벤트 정보를 분류하려면 시간이 소요되지만 이 툴은 중요한 디버깅 툴이다.

## *첫번째 리스폰더 및 리스폰더 체인*

응용 프로그램에서 각 NSWindow는 첫 번째 리스폰더 상태를 제공하는 뷰 계층에서 객체를 추적한다. 그 객체는 다름아닌 윈도우의 키보드 이벤트를 수신하는 NSView이다. NSWindow는 기본적으로 첫 번째 리스폰더가 되지만, 윈도우 내에 있는 NSView는 사용자가 마우스로 클릭하면 첫 번째 리스폰더가 된다.

또한, NSWindow `makeFirstResponder:` 메소드를 사용하여 프로그램에서 첫 번째 리스폰더를 설정할 수 있다. 게다가 첫 번째 리스폰더 객체는 버튼이나 매트릭스 같은 NSControl이 전송한 액션 메시지의 타켓일 수 있다. `setTarget:`은 `nil` 인수를 가진 NSControl(또는 셀)로 전송하여 이 작업을 수행한다.

nib 파일 윈도우의 Instances에서 NSControl 및 첫번째 리스폰더 간의 타겟/액션을 연결하여 Interface Builder에서 동일한 작업을 수행할 수 있다.

NSWindows, 응용 프로그램 객체 및 응용 프로그램의 모든 NSViews는 NSResponder에서 상속되었다는 것을 기억해야 한다. NSResponder는 기본적인 메시지 처리 동작을 정의한다. 이벤트는 리스폰더 체인으로 전달된다. 수 많은 Application Kit 객체들이 이 동작을 오버라이드하므로 응답한 객체에 도달할 때 까지 체인을 거쳐간다.

리스폰더 체인에서 다음번 리스폰더는 응용 프로그램의 NSView, NSWindow 및 NSApplication 객체간의 상호 관계에 의해 결정된다. NSView의 경우, 다음번 리스폰더는 NSView의 슈퍼뷰가 된다. 컨텐트 뷰의 다음번 리스폰더는 NSWindow이다. 거기서, 이벤트는 NSApplication 객체로 전달된다.

첫 번째 리스폰더로 전송된 액션 메시지의 경우, 가능한 리스폰던트를 통해 지나간 흔적이 더욱 명확해진다. 먼저, 메시지는 리스폰더 체인을 통해 NSWindow, NSWindow의 델리게이트순으로 전달된다. 그리고, 이전 순서가 Key 윈도우에서 발생했다면, 메인 윈도우에 대해서도 동일한 경로를 따라간다. 이 때, NSApplication 객체가 응답에 실패하면, NSApplicaton의 델리게이트로 이동한다.

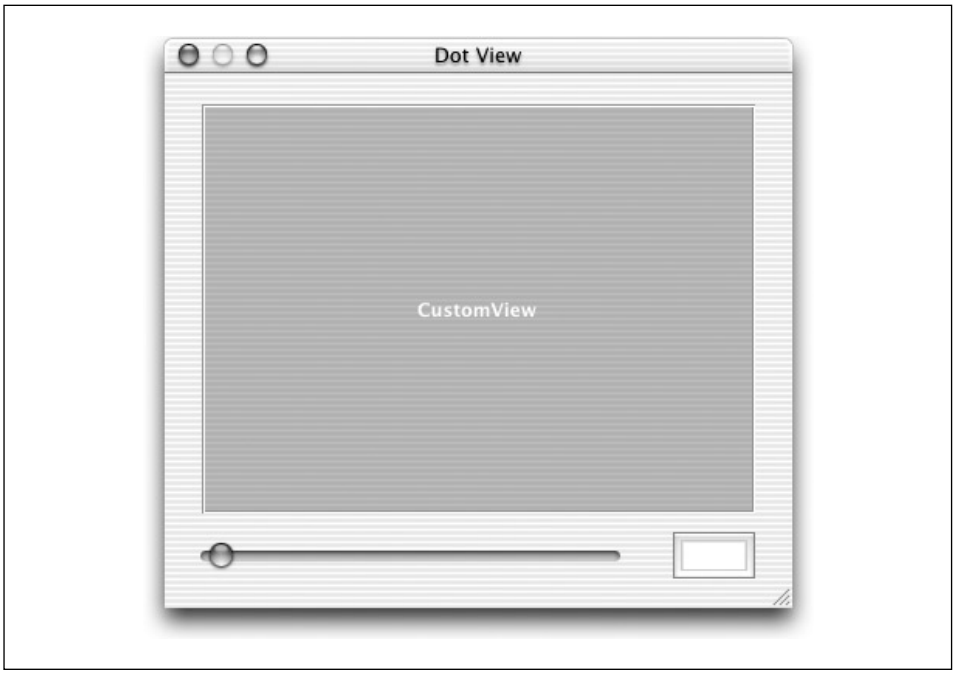
***Key 윈도우 및 첫 번째 리스폰더.*** 멀티 윈도우 데스크탑 환경은 화면에 수 많은 개방형 윈도우를 제공한다. 사용자는 마우스로 윈도우를 선택하여 활성화시킨다. 그러면, 윈도우는 키가 되고, 윈도우의 첫 번째 리스폰더는 사용자가 발생시킨 이벤트 타겟이 된다.

다른 윈도우를 선택하였다면, 그 윈도우가 키가 되고, 첫 번째 리스폰더는 새로운 객체가 된다. 객체를 선택하지 않거나, 윈도우에 컨트롤이 없다면, 윈도우가 자신의 첫 번째 리스폰더가 된다. 윈도우가 나타날 때, 키스트로크를 사용할 수 있는 첫 번째 로직 컨트롤이 첫 번째 리스폰더가 될 수 있도록 `initialFirstResponder`를 구성한다. `initialFirstResponder`가 `nil`일 경우 윈도우가 선택한 기본적인 객체는 상당히 합리적이라 할 수 있다.

## *Dot View 응용 프로그램 만들기*

이 섹션에서는 컬러 도트를 드로잉함으로써 마우스 클릭에 응답하는 커스텀 NSView 서브클래스를 사용하여 응용 프로그램 구축 방법을 설명한다. 이 예제를 통해 커스텀 이벤트를 처리하는 방법을 검토할 기회를 가질 수 있으며, 추가적으로 Cocoa의 메인 드로잉 클래스인 NSBezierPath를 사용하여 단순한 모양을 표현하는 방법을 배울 수 있다.

1. Dot View라는 새로운 Cocoa 응용 프로그램 프로젝트를 만든다.
2. 메인 nib 파일을 연다.
3. 메인 윈도우를 Dot View라 명명한다.
4. NSView라는 DotView의 서브클래스를 만든다.
5. DotView 파일을 만들어, 프로젝트에 추가한다.
6. <그림 8-5>에서 보여진 바와 같이 More Views 팔레트에서 윈도우로 커스텀 뷰를 드래그하여, 수평 슬라이더와 컬러 웰을 추가한다.




*<그림 8-5> 테이블 뷰 객체를 인터페이스에 추가하기*

7. Info 윈도우를 불러와서 CustomView을 선택하고, 뷰 클래스를 DotView로 변경한다.
8. <예제 8-1>에서 보여진 바와 같이 `DotView.h`를 열어 선언을 추가한다. 3개의 인스턴스 변수는 클래스가 드로잉하는 뷰의 위치, 컬러 및 크기와 같은 도트의 속성을 나타낸다. 이 메소드 선언에 관한 내용은 코드 리스트에 수록된 주석에 간략하게 나와 있다.

9. Project Builder에서 Interface Builder의 **MainMenu.nib** 윈도우로 **DotView.h**를 드래그한다. 이로써, Interface Builder가 파일을 분석하고, DotView 클래스에 추가된 액션과 아웃렛을 검색할 수 있다.
10. Interface Builder에서 수평 슬라이더를 Control키를 누른채 클릭하고, DotView로 드래그하여 연결한다. 타겟/액션을 Info 윈도우의 Connections 패인에 있는 **setRadius:** 메소드로 연결한다.
11. 컬러 웰에서 DotView로 드래그하여 연결하고, 타겟/액션을 **setColor:** 메소드에 연결한다.
12. DotView에서 슬라이더와 컬러 웰로 아웃렛을 연결한다.
13. nib 파일을 저장한다.

*<예제 8-1> DotView.h*

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

@interface DotView : NSView {
NSPoint center;
    NSColor *color;
    float radius;

    // Outlets for the color well and the slider
    IBOutlet NSColorWell *colorWell;
    IBOutlet NSSlider *sizeSlider;
}

// Standard view create/free methods
- (id)initWithFrame:(NSRect)frame;
- (void)dealloc;

// Drawing
- (void)drawRect:(NSRect)rect;
- (BOOL)isOpaque;

// Event handling
- (void)mouseUp:(NSEvent *)event;

// Custom methods for actions this view implements
- (IBAction)setRadius:(id)sender;
- (IBAction)setColor:(id)sender;

@end
```

인터페이스를 구축하고 DotView와 도트 드로잉 컨트롤 간의 연결을 정의하였으므로, DotView class 클래스를 구현한다.

1. 다음 코드에서 보여진 바와 같이 **DotView.m**을 열어, **initWithFrame:** 구현을 추가한다. 이 메소드는 DotView의 인스턴스 변수를 초기화한다.

```
- (id)initWithFrame:(NSRect)frame
{
    self = [super initWithFrame:frame];
    center.x = 50.0;
    center.y = 50.0;
    radius = 10.0;
    color = [[NSColor redColor] retain];
    return self;
}
```

2. 다음과 같이 **dealloc**를 구현한다. 예외 상황은 없다.

```
- (void)dealloc {
    [color release];
    [super dealloc];
}
```

3. **awakeFromNib** 메소드를 구현하여, 1단계에서 제공된 **initWithFrame:** 메소드에서 정의되 동일한 값으로 컬러 웰과 슬라이더를 초기화한다. 이같은 설정 작업을 수행하려면, 언아카이빙이 마무리되었을 때 nib에 있는 모든 객체로 **awakeFromNib** 메시지를 전송한다. 언아카이빙의 역할은 컬러웰을 생성하고, **colorWell** 아웃렛으로 연결을 구축하는 것이기 때문에 nib 파일이 언아카이빙될 때 까지 이 작업을 수행할 수 없다. nib 파일을 언아카이빙하기 전에는 아웃렛을 초기화할 수 없다. 따라서, 아웃렛을 통해 전송된 메시지는 아무런 쓸모가 없다. 이런 식으로 응용프로그램을 먼저 구동하면, 사용자 인터페이스 컨트롤은 도트를 드로잉하기 위해 사용될 초기값을 반영한다.

```
- (void)awakeFromNib {
    [colorWell setColor: color];
    [sizeSlider setFloatValue:radius];
}
```

4. **drawRect:** 메소드는 뷰에서 도트를 드로잉한다. 먼저, 흰색으로 드로잉하여 뷰를 확연히 구분할 수 있도록 한다. 그 다음, 도트의 영역을 산정한다. 그리고, 현재의 컬러를 뷰의 인스턴스 변수에 저장된 값으로 설정한다. 마지막으로, 뷰에서 사각형을 드로잉하려면 NSBezierPath를 사용한다.

```
- (void)drawRect:(NSRect)rect {
    NSRect dotRect;

    [[NSColor whiteColor] set];
    NSRectFill([self bounds]);

    dotRect.origin.x = center.x - radius;
    dotRect.origin.y = center.y - radius;
    dotRect.size.width  = 2 * radius;
    dotRect.size.height = 2 * radius;
```

```
        [color set];
        [[NSBezierPath bezierPathWithOvalInRect:dotRect] fill];
    }
```

5. **isOpaque**구현을 추가한다. **isOpaque** 뒤에 어떤 뷰도 필요로 하지 않고 전체 영역을 다시 드로잉하는 뷰는 **YES** 값을 리턴하기 위해 **isOpaque**를 오버라이드한다. 이로써, 화면 서브시스템의 성능이 최적화된다.

```
- (BOOL)isOpaque {
    return YES;
}
```

6. 뷰 이벤트를 처리하려면 NSView 서브클래스의 NSResponder(NSView의 서브클래스)메소드를 오버라이드하는 메소드를 권장할만하다. 그러한 메소드를 **mouseUp:**이라 한다. **mouseUp:**은 사용자가 마우스 버튼을 해제할 때 발생한다. 아래와 같이 **mouseUp:**을 추가한다. 모든 NSResponder 메소드는 인수로서 이벤트를 수신한다. 이벤트는 윈도우의 좌표계에서 마우스 위치를 나타낸다. 이 위치를 지역 뷰의 좌표계로 전환하기 위해 (뷰 인수로써 **nil**과 함께) **convertPoint: fromView:**를 사용할 수 있다. 새롭게 center를 작성하면, 다시 나타나야 할 뷰를 표시하기 위해 **setNeedsDisplay:YES**를 호출한다(Application Kit가 자동으로 수행함).

```
- (void)mouseUp:(NSEvent *)event
{
    NSPoint eventLocation = [event locationInWindow];
    center = [self convertPoint:eventLocation fromView:nil];
    [self setNeedsDisplay:YES];
}
```

7. **setRadius:** 구현을 추가한다. **setRadius:**는 사용자 인터페이스에서 슬라이더가 발생시킨 액션 메소드이다. 이 메소드를 사용하여 도트의 반경을 변경할 수 있다. 이 메소드는 전송자가 부동소수점을 반환할 수 있다고 가정한다. 따라서 값을 요구하고, DotView의 인스턴스 변수를 설정한 뒤, 다시 나타나야 할 뷰를 표시한다. 기존의 값과 새로운 값의 동일 여부를 확인함으로써 최적화가 이루어진다.

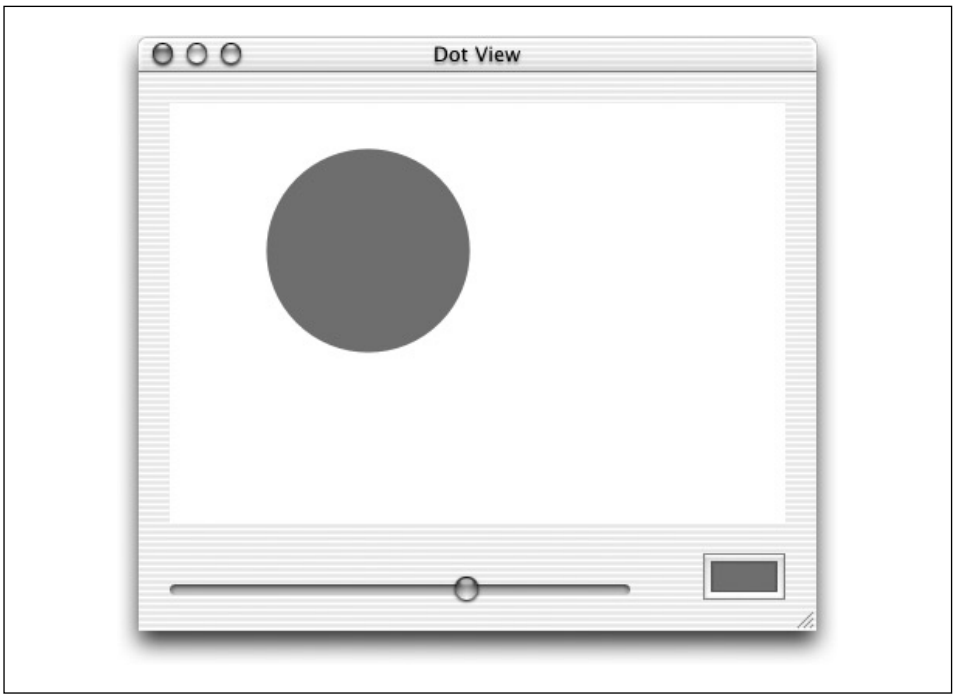
```
- (void)setRadius:(id)sender {
    radius = [sender floatValue];
    [self setNeedsDisplay:YES];
}
```

8. **setColor:** 구현을 추가한다. **setColor:**는 컬러 도트를 변경할 수 있는 컬러 웰이 발생시킨 액션 메소드이다. 이 메소드는 전송자가 컬러를 반환할 수 있다(NSColorWell은 이 작업을 수행할 수 있다)고 가정한다.

setColor:는 전송자로부터 값을 수신하여 이전의 컬러를 해제하고, 다시 나타나야 할 뷰를 표시한다. **setRadius:** 메소드의 경우, 기존의 값과 새로운 값의 동일 여부를 확인 점검함으로써 최적화가 이루어진다.

```
- (void)setColor:(id)sender {
    [color autorelease];
    color = [[sender color] retain];
    [self setNeedsDisplay:YES];
}
```

9. <그림 8-6>에서 보여진 바와 같이, 응용 프로그램을 구축 및 운용하고, 몇 개의 도트를 만들어본다.




*<그림 8-6> Dot View 응용 프로그램을 사용하여 도트 드로잉하기*

Dot View는 단순한 응용 프로그램이지만, Cocoa 프로그래밍을 상당히 흥미롭게 만든다. 간단하게 몇 단계를 수행하면 마우스 클릭에 응답하여 원을 드로잉할 수 있는 뷰를 생성할 수 있다. 원의 크기와 컬러는 슬라이더와 컬러 웰을 사용하여 동적으로 구성할 수 있다. 제공된 툴을 사용하여 몇 분동안 프로젝트를 구축하고, 시험해 볼 수 있다.

# 프로그램 발생 이벤트에 응답하기

객체 지향 응용 프로그램에서는 수 많은 상황을 통해 객체가 다른 객체와 시스템에서 어떻게 상호작용하는 지 여부를 파악할 수 있다. 이 섹션은 Cocoa에서 객체 대 객체 통신의 델리게이션과 통지에 관한 기법을 설명한다. 이 디자인 패턴은 관련 개념을 완벽하게 이해해야만 복잡한 응용 프로그램을 개발할 수 있는 Cocoa 프로그래밍에서 상당히 광범위하게 사용된다. 실제로, 델리게이션과 통지를 사용하여 제10장, *Travel Advisor 튜토리얼*, 제12장, *To Do: 기본 원리* 및 제13장에서 제공하는 고급 기능의 튜토리얼을 완성할 수 있다.

## 델리게이션

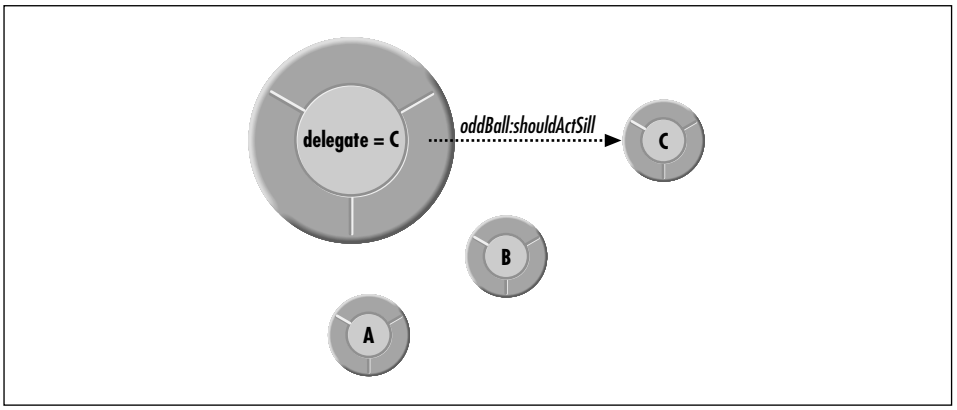
델리게이션은 커스텀 서브클래스를 생성하지 않고서도 객체 동작을 변경할 수 있는 방법으로 생각하면 된다. 델리게이트는 특정 이벤트가 발생했을 때 다른 객체에서 메시지를 수신하는 헬퍼 객체이다. 객체는 델리게이트가 동작과 결정과정에 영향을 줄 수 있도록 델리게이트로 리퀘스트를 전송한다.

역할을 위임한 객체는 "흥미로운" 이벤트가 발생했을 때 델리게이트로 전송될 델리게이트 메시지와 함께 *delegate* 아웃렛을 선언해야 한다. 델리게이트가 되려면, 객체는 1개 이상의 델리게이트 메소드를 구현해야 한다. 델리게이트의 예상 역할에 따라 델리게이션 메시지 유형이 결정된다.

- 어떤 메시지는 이벤트가 발생한 이후에 만들어진 단순히 정보만을 제공한다. 이 메시지는 델리게이트가 델리게이팅 객체에 맞춰 액션을 조정할 수 있도록 한다.
- 어떤 메시지는 이벤트가 발생하기 전에 전송된다. 델리게이트는 이들 메시지를 통해 액션을 거부하거나 허용할 수 있다.
- 기타 델리게이션 메시지는 셀로 브라우저를 메우는 것과 같은 특정한 태스크를 델리게이트에 할당한다.

<그림 8-7>은 `oddBall:shouldActSilly:` 메시지를 델리게이트로 전송하는 모습을 보여준다. 델리게이트 메시지에는 델리게이팅 객체로부터 수신한 리퀘스트가 포함되어 있다. 델리게이트 객체는 델리게이트 메소드 구현에서 `true` 또는 `false`를 델리게이트 메소드에서 리턴함으로써 "act silly" 요청을 승인 또는 거부한다.

Interface Builder에서 연결하여 커스텀 객체를 프레임워크 객체의 델리게이트로 설정할 수 있다. 또는, `setDelegate:` 메소드를 사용하여 설정 여부를 결정할 수 있다. 커스텀 클래스는 클라이언트 객체에 대한 델리게이트 변수와 델리게이션 프로토콜을 정의할 수 있다. 객체는 자신의 델리게이트를 "소유"하지 않기 때문에(아웃렛 뿐만 아니라) 델리게이트 또한 보유하지 않는다.




<그림 8-7> 델리게이션

## 간단한 델리게이트 예제

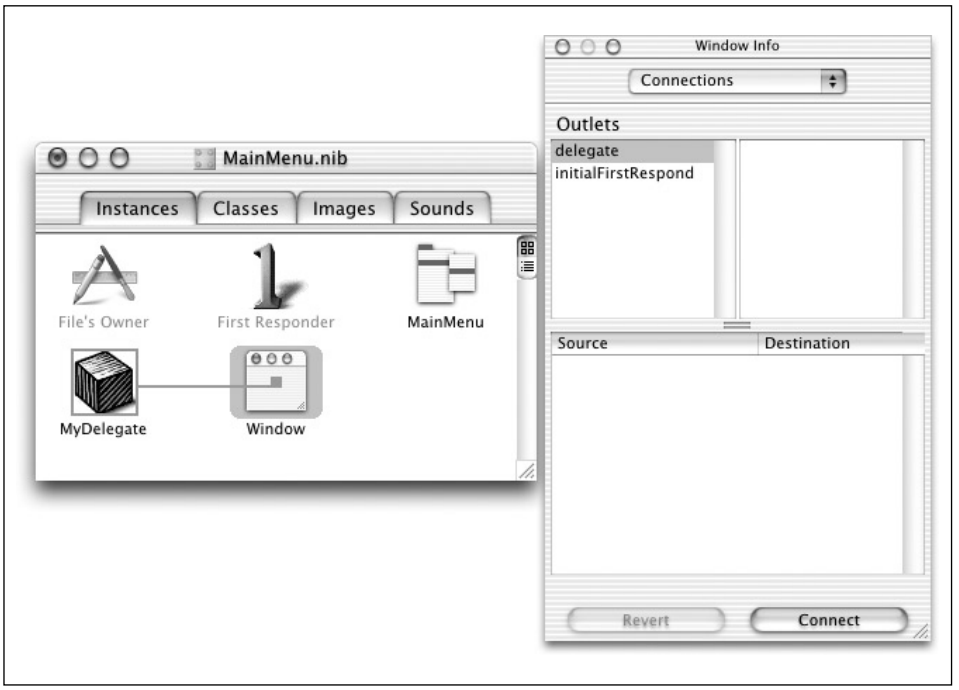
델리게이션은 윈도우의 동작을 변경하기 위해 Cocoa에서 일반적으로 사용된다. NSWindow은 델리게이트로 입력하기에 적합한 윈도우의 영역에서 크기조정, 숨기기, 이동하기, 닫기와 같은 각 유형의 이벤트를 위해 델리게이트 메소드를 선언한다. 모든 델리게이트가 윈도우의 영역에서 모든 가능 이벤트를 위해 입력을 제공해야 하는 것은 아니다. 그래서, Cocoa에서는 델리게이트와 관련이 있는 델리게이트 메소드만을 구현할 수 있다. 델리게이트가 구현한 델리게이트 메소드가 어떤 것인지 알아 보려면 델리게이팅 객체 메카니즘을 사용한다. 구현되지 않은 델리게이트 메시지는 전송되지 않는다.

예를 들어, `windowShouldClose:` 메소드는 윈도우를 닫기 위한 NSWindow의 델리게이트 메소드이다. 사용자가 윈도우의 close 버튼을 클릭하고 난뒤, 윈도우를 닫기 바로 직전에 윈도우 객체는 `windowShouldClose:` 메시지를 델리게이트에 전송한다. 일반적으로, 델리게이트는 사용자가 필요로 하는 코드를 구현하여 `windowShouldClose:` 메소드에서 윈도우의 변경 내용을 저장할 수 있다. 델리게이트 메소드에서 윈도우 객체로 반환된 값은 윈도우가 실제로 닫였는지 또는 열려있는지 여부에 영향을 준다.

다음 예제는 기본적인 Window Closing 다이얼로그를 보여줘서 `windowShouldClose:` 메소드에 응답할 수 있는 NSWindow의 델리게이트 클래스를 구현한다.

1. Delegate라는 새로운 Project Builder 프로젝트를 만들고, 메인 nib 파일을 연다.
2. MyDelegate라는 NSObject의 새로운 서브클래스를 생성한다.
3. MyDelegate의 인스턴스를 만든다.

4. <그림 8-8>에서 보여진 바와 같이 MainMenu.nib 윈도우의 Instances 패인에서 Window로부터 MyDelegate로 Control 키를 누른 채 연결라인을 드래그한다.




*<그림 8-8> Window에서 MyDelegate로 Control 키를 누른 채 드래그하기*

5. Window Info윈도우의 Connections 패인에서 델리게이트 아웃렛을 선택하고, Connect를 클릭하였는지 확인한다.
6. nib 파일을 저장한다.
7. MyDelegate 클래스 파일을 만들어, 프로젝트에 추가한다.

nib 파일에서 Window 및 MyDelegate간의 델리게이트 관계를 구축한다. 이제, MyDelegate용 델리게이트 메소드를 구현한다.

1. `MyDelegate.h`를 열어, 다음의 메소드 선언을 추가한다.

```
- (BOOL)windowShouldClose:(NSWindow *)sender;
```

2. `MyDelegate.m`을 열어, 다음 메소드 구현을 추가한다.

```
- (BOOL)windowShouldClose:(NSWindow *)sender
{
    int answer;

    answer = NSRunAlertPanel(@"Close", @"Are you certain?",
                    @"Close", @"Cancel", nil);
```

```
    switch (answer) {
                case NSAlertDefaultReturn:
                    return YES;
                default:
                    return NO;
            }
        }
```

3. 예제 응용 프로그램을 구축하여 운용한다.

응용 프로그램의 윈도우에서 close 버튼을 클릭하면, 대화 상자가 나타나 윈도우를 닫을 것인가를 묻는다. 윈도우에 편집한 데이터가 실제로 내장되어 있는 완전한 기능을 갖춘 응용 프로그램에서는 윈도우를 닫기 전에 windowShouldClose 델리게이트 메소드를 사용하여 윈도우에서 작업한 내용을 저장한다.

***시트 사용하기.*** 이 델리게이션 예제는 도큐멘트에 첨부된 새로운 유형의 대화 상자인 Aqua 시트를 구현할 때 나타난다. 시트는 도큐멘트 관계를 명확하게 구분하기 위해 윈도우 타이틀(<그림 8-9> 참조)에서 슬라이드로 분리된다. 시트는 시트를 첨부한 윈도우에만 있는 양식이다. 그래서 시트를 닫기 전까지는 작업을 진행할 수 있다.

경보시트인 **NSBeginAlertSheet**를 보여주는 함수가 비동기식이기 때문에 시트를 추가하는 작업은 기본 대화상자를 사용하는 것보다 더욱 복잡하다. 바꿔 말하면, **NSBeginAlertSheet**는 호출자에게 컨트롤을 리턴하기 전에는 시트를 닫기위해 대기하지 않는다. 대신, **NSBeginAlertSheet**는 시트를 제공한 후 즉시 리턴한다. 시트와 사용자의 상호작용을 확인하려면 델리게이트 객체로 **NSBeginAlertSheet** 레퍼런스를 전송해야 한다. 또한, 시트를 닫을 때 콜백으로 발생된 메소드 셀렉터도 전송한다. 콜백이 발생하면, 사용자가 클릭한 버튼을 가리키는 결과 코드를 전송한다. 이전의 윈도우 델리게이트와는 달리 시트는 임시적 또는 "형식적" 델리게이트를 사용한다. 델리게이트의 연결을 시트가 닫힐 때까지 지속된다.

**NSBeginAlertSheet** 파라미터 리스트를 처음 접하면, 어렵다고 생각할 수 있지만 실제로, 사용 방법은 생각보다 어렵지 않다. 다음은 파라미터에 관한 간략한 개요를 제공한다.

- **title.** 볼드체로 시트의 상단에 나타나는 시트의 타이틀.
- **defaultButton.** 시트의 기본적인 버튼 라벨; 일반적으로 OK를 의미.
- **alternateButton.** 시트의 대체 버튼 라벨;일반적으로, Cancel을 의미.
- **otherButton.** 세 번째 버튼 라벨. nil을 전송하면, 2개의 버튼이 시트에 나타난다.
- **docWindow.** 시트를 첨부한 NSWindow 레퍼런스.


<그림 8-9> Aqua 시트

- **modalDelegate.** 사용자가 시트를 닫을 때 응답할 객체 레퍼런스.
- **didEndSelector.** **modalDelegate**이 구현한 메소드 셀렉터. 이 메소드는 모달 세션이 종료되고, 시트가 닫히기 전에 발생한다.
- **didDismissSelector.** **modalDelegate**가 구현한 메소드 셀렉터. 이 메소드는 시트가 닫힌 후에 호출된다. 이 기능을 필요로 하지 않을 경우, **NULL**을 전송한다.
- **contextInfo.** **didEnd**와 **didDismiss** 메소드의 파라미터로써 모달 델리게이트로 전송을 정의할 수 있는 추가 데이터.
- **msg.** 시트 하단에 나타나는 옵션 메시지 스트링. 스트링은 **printf**-style 퍼센트 확장자 시퀀스를 포함할 수 있다.
- **optional params.** 이 **printf**-스타일 파라미터는 **msg** 스트링을 포맷하기 위해 사용할 수 있다.

파라미터가 어떤 용도로 사용되는지 설명했기 때문에 경보 시트로 가서 구현해 본다. 수행할 작업은 호출을 NSBeginAlertSheet에 추가하고, 시트와 사용자가 상호작용하는 결과를 처리할 델리게이트 메소드를 구현한다. 단계는 다음과 같다.

1. Project Builder에서 MyDelegate.h를 열어, 다음 메소드 선언을 추가한다. 메소드는 모달 세션이 종료되면(사용자가 버튼을 클릭하면 됨)발생한다.

```
- (void)didEndShouldCloseSheet:(NSWindow *)sheet
        returnCode:(int)returnCode
        contextInfo:(void *)contextInfo;
```

2. MyDelegate.m을 연다.
3. 다음과 같이 windowshouldclose 메소드를 변경한다.

   sender(델리게이팅 윈도우를 참조)가 docWindow 파라미터뿐만 아니라 contextInfo를 위해 전달되는지 확인한다. contextInfo 값은 didEnd 메소드의 파라미터로써 모달 델리게이트로 되돌려진다. 그래서, 어떤 윈도우를 닫을 것인지 결정하기 위해 사용될 수 있다.

```
- (BOOL)windowShouldClose:(NSWindow *)sender
{
    NSBeginAlertSheet(@"Close",         // sheet title
        @"OK",                          // default button label
        @"Cancel",                      // alternate button label
        nil,                            // other button label
        sender,                         // document window
        self,                           // modal delegate
        @selector(didEndShouldCloseSheet:returnCode:contextInfo:),
                                        // didEnd selector
        NULL,                           // didDismiss selector
        sender,                         // context Info
        @"Should this window close?",   // message
        nil);                           // params for msg string

    // Don't decide to close window until results of sheet
    // interaction are known.
    return NO;
}
```

4. didEndShouldCloseSheet 구현을 추가한다.

```
- (void)didEndShouldCloseSheet:(NSWindow *)sheet
        returnCode:(int)returnCode
        contextInfo:(void *)contextInfo
```

```
{
    if (returnCode == NSAlertDefaultReturn)
        [(NSWindow *)contextInfo  close];
}
```

5. 응용 프로그램을 구축 및 운용한다.

***델리게이트 메소드 찾기*** 클레스의 다른 델리게이트 메소드에 관심이 있다면, 클래스의 헤더 파일을 확인하면 된다. 또 다른 방법으로는 런타임 시, 호출된 메소드가 어떤 것인지 찾아보면 된다. 상기 언급했듯이, 델리게이팅 객체는 구현한 델리게이트 메소드가 어떤 것인지 알아보기 위해 델리게이트를 조회할 수 있다. 델리게이팅 객체는 **respondsToSelector:** 메시지를 델리게이트에 전송하여 조회한다. 이 메소드를 MyDelegate 클래스에 오버라이드하여, 객체가 수신한 모든 조회를 확인할 수 있다.

이 작업을 수행할 코드는 단순하다. 다음 메소드 구현을 **MyDelegate.m**에 추가한다.

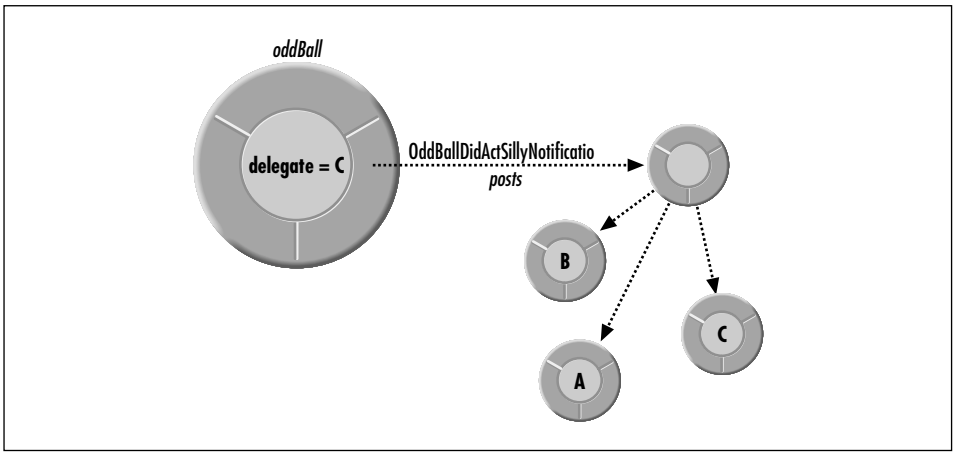
```
- (BOOL)respondsToSelector:(SEL)aSelector
{
    NSLog(@"respondsToSelector: %@",NSStringFromSelector(aSelector));
    return [super respondsToSelector:aSelector];
}
```

이 메소드는 전송자가 요청한 메소드명을 단순히 보여주기만 한다. 응용 프로그램을 컴파일하고, 운용하여 Project Builder 메인 윈도우의 Run 패인에서 출력을 확인한다.

## *통지*

Cocoa에서 객체간의 이벤트를 전달하는 일반적인 방식으로 *통지*를 사용한다. 통지는 이벤트를 수신하는 응용 프로그램의 모든 객체로 브로드캐스트된 메시지를 의미한다. 통지는 정보를 제공하는 델리게이션 메시지처럼 옵저버에게 이벤트가 발생했다는 정보를 알려준다.(<그림 8-10 참조)

또한, 통지는 이벤트 관련 데이터를 전송할 수 있다. 통지는 옵저버와 이벤트가 인터페이스하는 것을 허용하지 않는다는 점에서 델리게이션과 차이점을 보인다. 객체는 수 많은 옵저버를 보유하지만, 델리게이트는 단 1개만을 보유한다.




<그림 8-10> 통지

통지는 다음과 같이 진행된다.

- 응용 프로그램에서 발생하는 이벤트(데이터베이스에 레코드를 추가)에 관심있는 객체들은 이벤트 옵저버로써 통지 센터(NSNotificationCenter 인스턴스)에 등록한다. 등록하는 동안, 이벤트가 발생하면 옵저버는 메소드 중 1개가 통지 센터에 의해 발생되어야 하다는 것을 명시한다.
- 데이터베이스에 레코드(즉 일부 이벤트)를 추가하는 객체는 통지(NSNotification 인스턴스)를 통지 센터에 게재한다. 통지 객체는 통지를 식별할 태그, 통지를 게재한 객체 ID, 부가 데이터 딕셔네리로 구성된다.
- 통지 센터는 이전에 명시한 메소드를 발생시켜 통지에 포함한 메시지를 각 등록된 옵저버에게 전송한다.

통지는 객체가 아이덴티티를 밝히지 않아도 응용 프로그램의 다양한 객체들과 객체의 동작과 상태를 동기화할 수 있도록 한다. 또한, 통지 큐를 통해 비동기식으로 통지를 전송하고, 유사한 통지 그룹을 통합할 수 있다.

## *통지를 수신하는 레지스터*

통지를 게재한 클래스는 정적인 NSString 객체로써 헤더 파일에서 통지 이름을 정의한다. 예를 들어, NSTableView(제9장, *데이터 함수*에서 상세하게 설명)는 데이터를 행과 열로 나타낸다.

객체가 테이블 뷰로 변경된 내용을 감시할 수 있도록 NSTableView.h에서 4개의 통지를 정의한다.

```
APPKIT_EXTERN NSString *NSTableViewSelectionDidChangeNotification;
APPKIT_EXTERN NSString *NSTableViewColumnDidMoveNotification;
APPKIT_EXTERN NSString *NSTableViewColumnDidResizeNotification;
APPKIT_EXTERN NSString *NSTableViewSelectionIsChangingNotification;
```

통지 중 1개를 수신하려는 객체는 통지 센터에 등록할 때 NSTableView.h를 임포트한 뒤, 통지명을 사용해야 한다. 예를 들면,

```
[[NSNotificationCenter defaultCenter] addObserver:self
       selector:@selector(tableViewSelectionChanged:)
       name:NSTableViewSelectionDidChangeNotification
       object:someTableView];
```

selector 파라미터는 옵저빙 객체가 구현한 메소드 명이다. 옵저빙 객체는 someTableView에서 명시된 통지를 수신할 때 통지 센터에 의해 발생된다.

통지를 게재한 객체의 델리게이트는 자동으로 통지의 옵저버로 등록된다. Cocoa에서는 통지명과 같이 나오는 비정형 프로토콜로 선언된 셀렉터를 사용하여 델리게이트를 등록한다. 예를 들어, NSTableView는 다음의 비정형 프로토콜을 선언한다.

```
@interface NSObject(NSTableViewNotifications)
- (void)tableViewSelectionDidChange:(NSNotification *)notification;
- (void)tableViewColumnDidMove:(NSNotification *)notification;
- (void)tableViewColumnDidResize:(NSNotification *)notification;
- (void)tableViewSelectionIsChanging:(NSNotification *)notification;
@end
```

객체가 테이블 뷰의 델리게이트를 생성할 경우, 객체는 비정형 프로토콜에서 셀렉터를 사용하여 모든 통지를 수신하기 위해 자동으로 등록된다. 델리게이트 객체가 통지 중 1개에 응답하려면, 비정형 프로토콜에서 선언된 관련 메소드를 구현해야 한다.

## 간단한 통지 사례

이 섹션에서는 텍스트 필드를 사용하여 간단하게 통지를 보여줄 응용 프로그램 생성 방법에 관해 설명한다. 모든 텍스트 필드는 텍스트가 변경된 경우 NSControlTextDidChangeNotification 통지를 게재한다. 이 응용 프로그램에는 NSControlTextDidChangeNotification 메시지가 텍스트 필드에서 게재되었을 때 통보될 수 있도록 텍스트 필드 객체의 옵저버로 등록된 간단한 컨트롤러 클래스가 있다.

***새로운 프로젝트 만들기*** Notification 응용 프로그램의 Project Builder 프로젝트를 만들어, 메인 nib 파일을 연다.

1. Notification 이라는 새로운 Cocoa 응용 프로그램 프로젝트를 만든다.
2. 메인 nib 파일을 연다.

***텍스트 필드 및 컨트롤러 클래스 추가하기*** 텍스트 필드 객체를 응용 프로그램의 메인 윈도우에 추가한다. 텍스트 필드에서 텍스트를 추가, 변경 또는 삭제하는 작업은 응용 프로그램의 컨트롤러를 발생시켜 통보를 게재할 수 있도록 한다.

1. 텍스트 필드 객체를 윈도우로 드래그한다.
2. NSObject의 서브 클래스를 만들고, 새로운 클래스 MyController를 호출한다.
3. 아웃렛을 `textField`라는 MyController에 추가한다.
4. MyController의 인스턴스를 만든다.
5. MyController에서 텍스트 필드 객체로 연결을 드래그한다.
6. `textField` 아웃렛을 연결한다.
7. MyController 파일을 만들고, 프로젝트에 추가한다.

***MyController를 옵저버로 만들기.*** 통지 센터는 게시판과 같은 역할을 한다. 객체들은 특정 객체나 일반 객체에서 제공된 통지에 관심이 있으면 등록한다. 일반적으로, 응용 프로그램에는 단 1개의 통지 센터가 있다. 이는 NSNotificationCenter의 인스턴스이며, `defaultCenter` 클래스 메소드를 사용하여 액세스할 수 있다.

일반적으로, 객체는 초기화 메소드에서 옵저버로써 등록한다. nib 파일에 저장된 객체는 언아카이빙되고, `awakeFromNib` 메시지를 전송할 때 까지 대기해야 한다. MyController의 `awakeFromNib` 메소드를 오버라이드하여 클래스를 텍스트 필드의 `NSControlTextDidChangeNotification` 옵저버로 만든다. 다음 메소드 구현을 추가한다.

```
- (id) awakeFromNib
{
    [[NSNotificationCenter defaultCenter] addObserver:self
            selector:@selector(textDidChange:)
            name:NSControlTextDidChangeNotification
            object:textField];
}
```

textField가 `NSControlTextDidChangeNotification`을 게재할 때 마다 MyController는 `textDidChange:` 메시지가 MyController로 전송되어야 한다고 요구하면서 `addObserver:selector:name:object:` 메시지를 기본적인 통지 센터에 전송한다. 통지 센터에 게재된 모든 `NSControlTextDidChangeNotification`을 알고자 한다면, `object:` 파라미터의 `nil`을 전송한다.

이제, `textDidChange:` 메소드 구현을 MyController에 추가한다.

```
- (void)textDidChange:(NSNotification *)notification

{
    NSLog(@"Notification received - %@\n", [notification name]);
}
```

`textDidChange:` 메소드는 단순히 통지명을 나타낸다.

통지 센터는 옵저버 객체를 갖고 있지 않다. 그래서 옵저버 객체를 해제하기 전에 옵저버를 삭제하려면 주의해야 한다. 이로써, 통지 센터가 더 이상 존재하지 않은 객체로 메시지를 전송하는 작업을 미연에 방지할 수 있다. MyController는 init 메소드의 옵저버로써 등록되어 있기 때문에 객체는 `dealloc` 메소드의 통지 센터에서 해제되어야 한다. 다음 `dealloc` 구현을 `MyController.m`에 추가한다.

```
- (void)dealloc
{
     [[NSNotificationCenter defaultCenter] removeObserver:self];

     [super dealloc];
}
```

객체가 다른 객체를 통지 센터에 등록하고, 통지 센터에서 객체를 제거한 후에 해제하지 않았다면, 응용 프로그램은 메모리를 누설한다. 다음 규정을 준수한다. A객체가 B객체를 통지 센터에 등록했다면, A객체는 B객체를 해제하고, 통지 센터에서 B객체를 제거할 임무를 수행해야 한다.

***응용 프로그램 구축 및 운용하기.*** 텍스트 필드 객체의 텍스트를 추가하거나 변경할 경우, Project Builder Run 패인에서 통지 로그가 나타나는지 여부를 확인해야 한다.

# 9

# 데이터의 기능성

이 장은 여러 유형의 응용 프로그램에서 사용되는 Cocoa의 두가지 데이터 처리 기능을 설명하기 위해 Expense라는 간단한 튜토리얼 응용 프로그램을 사용하는 방법을 설명한다. 제11장 *Cocoa의 멀티플 도큐멘트 아키텍처*, 제12장 *To Do: 기본 원리* 및 제13장 *To Do: 확장*에서 설명하는 고급 튜토리얼을 완성하려면 이들 기능과 친숙해져야 한다.

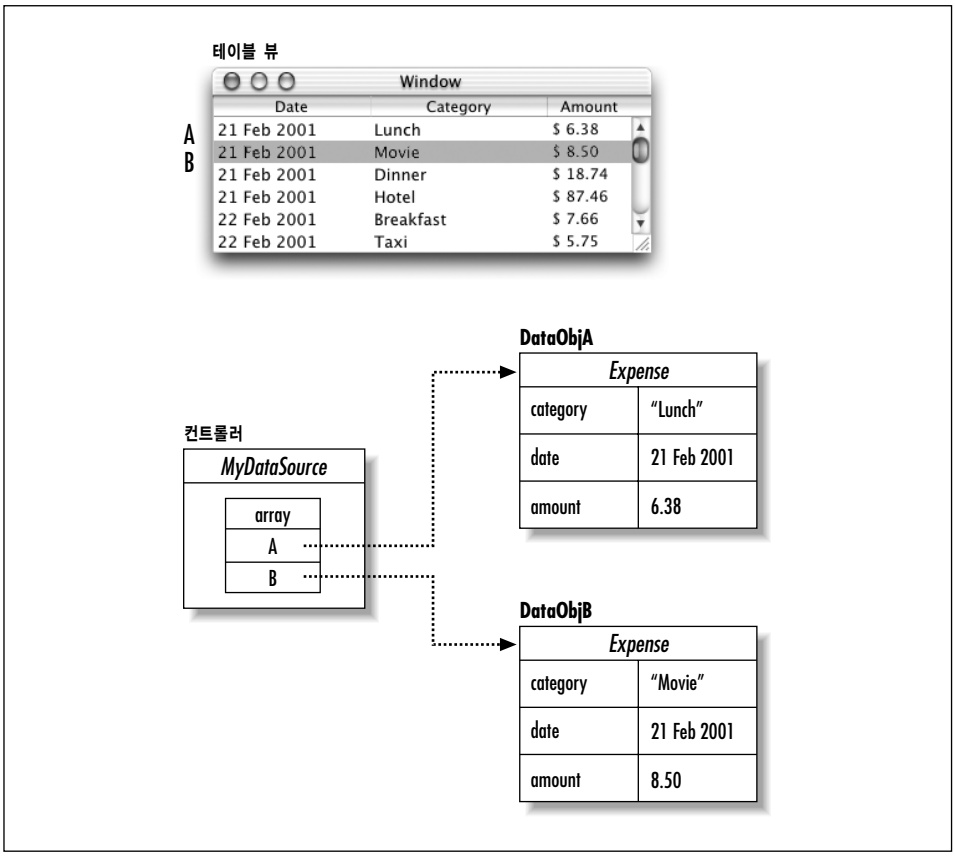
이 장의 첫번째 섹션에서는 열과 행으로 이루어진 데이터를 표현하는 사용자 인터페이스 객체인 테이블 뷰에 관해 설명한다. 테이블 뷰를 구현하면 컨트롤, 셀, 이들을 둘러싸고 있는 객체 간의 상호 작용에 대해 심층적으로 파악할 수 있다. 또한, 테이블 뷰를 처리하다 보면 데이터를 표현하기 위해 테이블 뷰에 제공하는 헬퍼 객체인 데이터 소스의 개념을 이해할 수 있다.

이 장의 두번째 섹션에서는 Cocoa의 아카이브/언아카이브 메카니즘을 사용하여 데이터 객체 그룹을 "평면화(flatten)"하는 방법(객체 지향 프로그래밍에서는 *serialization*라 한다)을 설명한다. 그래서, 데이터 객체 그룹은 저장 장치에 영구히 저장되어, 추후 사용하기 위해 찾아볼 수 있다.

## 테이블 뷰와 데이터 소스

테이블 뷰는 데이터를 행과 열로 표현하는 객체이다. 테이블 뷰에서 가로행은 데이터 모델의 한개 객체와 맵핑한다. 반면에 세로열은 가로행 객체 속성과 맵핑한다. 어떤 세로열은 파생되었거나 산출된 값을 제공한다. 객체 속성의 서브셋은 자주 테이블 뷰에서 표현된다. <그림 9-1>을 참조한다.

실제로, NSTableView은 스크롤 뷰에 묶여있는 여러 개의 객체들이다. 스크롤 뷰의 내부에는 데이터를 나타내고, 편집하는 NSTableView 인스턴스가 있다. 테이블 뷰의 상단에는 NSTableHeaderView객체가 있다.




*<그림 9-1> 데이터 모델을 테이블 뷰의 가로행과 세로열로 맵핑하기*

헤더 뷰 아래에는 1개 이상의 세로열(NSTableColumn의 인스턴스)이 구성되어 있다. 테이블 뷰의 세로열은 여러 구성 가능한 속성들을 제공한다.

- **헤더 셀.** 이 셀은 세로열의 타이틀을 나타낸다.
- **식별자.** 식별자는 세로열을 데이터 모델 객체의 속성과 맵핑하기 위해 사용하는 스트링 객체 값이다. 식별자는 속성 이름일 수도 있고, 태그로 동작하는 숫자일 수도 있다.
- **데이터 셀.** 이 셀은 테이블 뷰 내부에 있는 단 한개의 객체이다. 응용 프로그램이 요청할 경우, 커스텀 셀을 사용하기 위해 1개의 세로열을 구성할 수 있다. NSTextFieldCell은 기본적인 `dataCell` 타입이다.
- **포매터.** 테이블 뷰의 세로열은 텍스트 필드처럼 포매터를 사용할 수 있다. 포매터는 세로열에 있는 모든 셀에 적용된다.

NSTableView는 테이블의 세로열을 정의하는 한개 이상의 NSTableColumns을 소유한다. 테이블 뷰는 스크롤 뷰에서 테이블 뷰 위에 있는 NSTableHeaderView를 소유할 수 있다. 테이블 뷰가 어떻게 배치되는지 알려면 테이블 뷰 내부에 있는 NSTableColumns을 사용하고, 각 가로행을 드로잉하거나 편집하려면 각 세로열로부터 `dataCell`을 사용한다.(편집이 끝나면, `dataCell`을 복사한다) NSTableHeaderView가 어떻게 배치될 것인지를 알려면 NSTableView로부터 NSTableColumns을 사용한다. 또한, 세로열의 제목을 작성하기 위해 각 세로열로부터 `headerCell`을 사용한다.

NSTableView는 대부분의 사용자 인터페이스처럼 NSControl의 서브클래스이다. 이 경우, 타겟/액션 연결을 지원한다. 또한, 사용자가 편집 셀 이외에 다른 것을 더블 클릭할 때 `doubleAction`을 설정할 수 있다. 이 경우는 Interface Builder에서 설정할 수 없고, NSTableView의 `setDoubleAction:` 메소드를 사용한다.

NSTableView는 심플러 NSControls와는 달리 보여줄 데이터를 저장하지 않는다. 대신, 테이블 뷰는 데이터 소스를 제공하는 커스텀 "헬퍼" 객체로부터 보여줄 데이터를 얻을 수 있다. 데이터 소스는 NSTableDataSource의 비정형(Informal) 프로토콜을 구현하여 테이블 뷰로 데이터를 공급한다. 테이블 뷰의 데이터 소스 프로토콜은 2개의 메소드로 구성된다.

```
-(int)numberOfRowsInTableView:(NSTableView *)tableView;
```

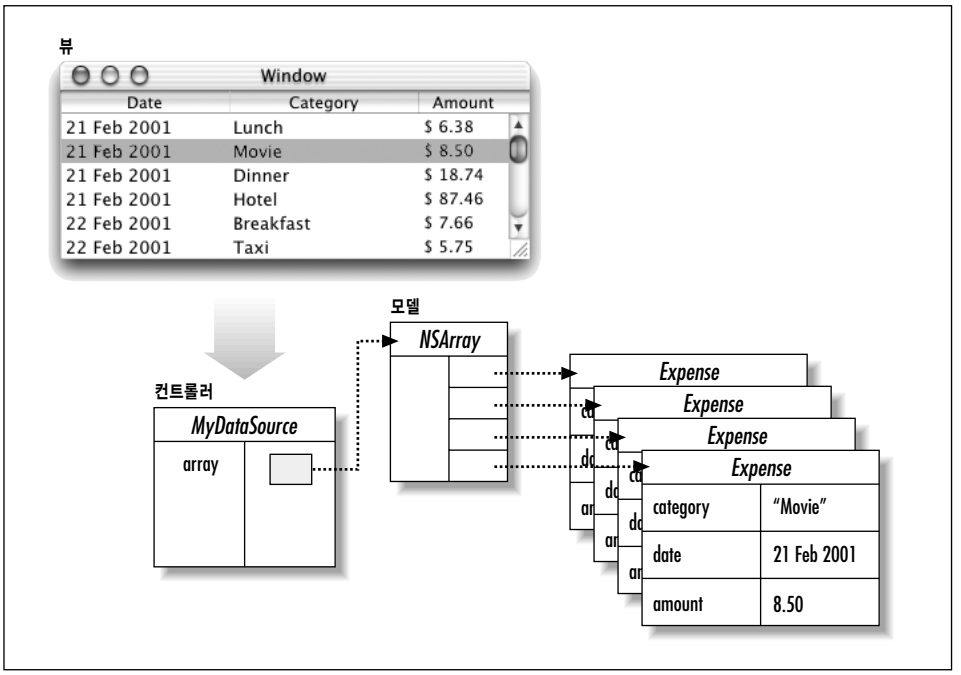
및

```
-(id)tableView:(NSTableView *)tableView
    objectValueForTableColumn:(NSTableColumn *)tableColumn row:(int)row;
```

첫 번째 메소드는 데이터 모델에 가로행 데이터가 몇 개가 있는지 데이터 소스에게 요청하는 방법을 제공한다. 두 번째 메소드는 가로행과 세로열 번호를 좌표로 사용하여 데이터 모델에서 객체 값을 검색하기 위해 테이블 뷰에 의해 사용된다. 사용자가 데이터를 테이블에 입력하거나 변경하려면, 데이터 소스로 돌려보낸 변경사항을 저장하기 위해 세 번째 메소드를 구현해야 한다.

MVC 측면에서 보면, 데이터 소스는 모델 객체(일반적으로 어레이)와 뷰 객체(일반적으로 테이블 뷰 또는 아웃라인 뷰)와 통신하는 컨트롤러 객체이다. <그림 9-2>는 이들 관계를 보여준다.

여러 Application Kit의 클래스(NSOutlineView 및 NSComboBox)는 다른 객체가 데이터 소스로 동작할 수 있도록 유사한 비정형 프로토콜을 선언한다.




<그림 9-2> MVC 컨트롤러 객체인 데이터 소스

## 간단한 테이블

이 섹션에서는 테이블 뷰에 나타날 데이터를 공급하는 데이터 소스를 통해 아주 간단한 응용 프로그램을 구현하는 방법을 설명한다. 이 장의 뒷 부분에서는 테이블 뷰가 일상 경비를 추적하는 유틸리티로 사용될 수 있도록 응용 프로그램을 확장하는 방법을 설명한다.

### 프로젝트 만들기

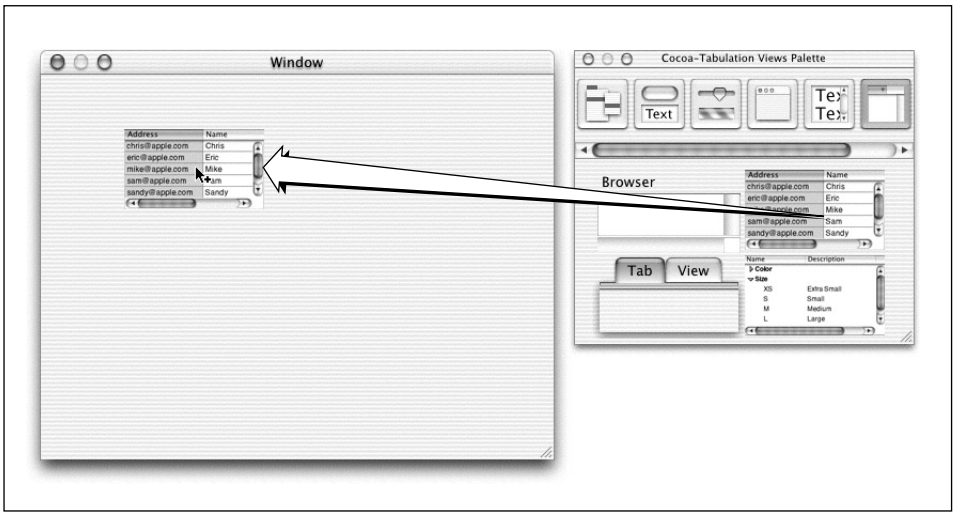
응용 프로그램을 위해 새로운 Project Builder 프로젝트를 만든다.

1. Expense라는 새로운 Cocoa 응용 프로그램을 만든다.
2. 메인 nib 파일을 연다.

### 테이블 뷰 추가하기

Expenses 응용 프로그램에 필요한 유일한 UI 객체는 테이블 뷰이다. 테이블 뷰를 윈도우에 추가한다.

1. <그림 9-3>에서 보이는 바와 같이 테이블 뷰를 Tabulation Views 팔레트에서 드래그한다.
2. 윈도우에 나타내기 위해 테이블 뷰의 크기를 조정한다.




<그림 9-3> 테이블 뷰를 인터페이스에 추가하기

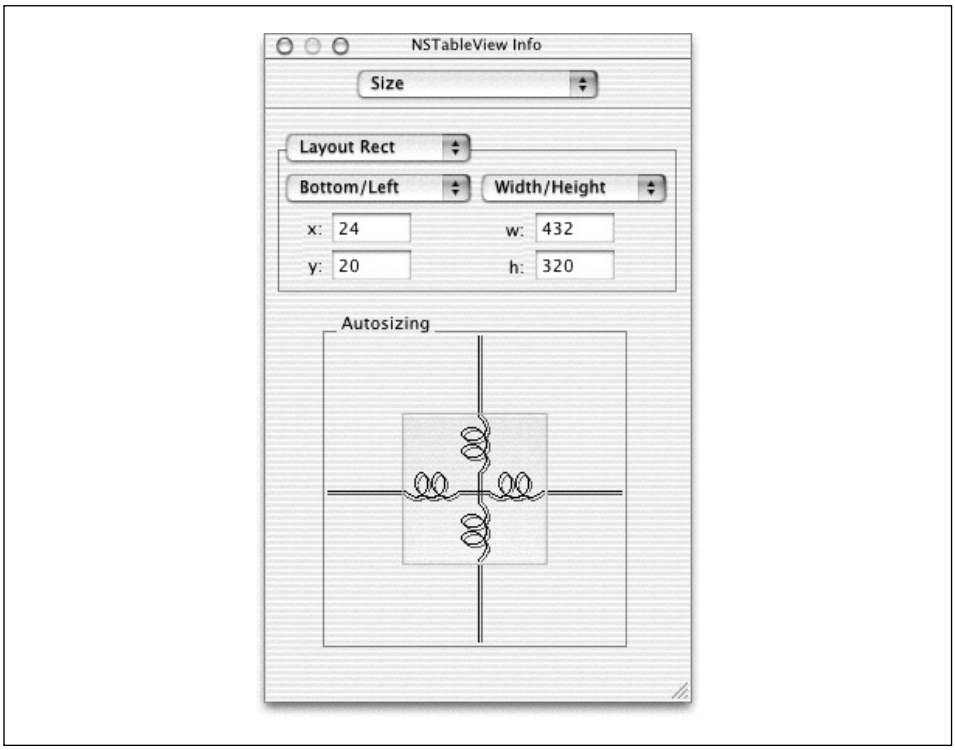
이제, 인터페이스를 시험해보면 윈도우 크기를 조정하더라도 테이블 뷰에는 어떤 영향도 주지 않는다는 것을 확인할 수 있다. 테이블 뷰의 크기를 윈도우와 함께 조정하려면, 테이블 뷰를 둘러싸고 있는 NSScrollView의 Size 패인에서 Autosizing 설정값을 변경해야 한다.

1. NSTableView Info 윈도우를 불러오기위해 테이블 뷰를 클릭한다.
2. <그림 9-4>와 같이 Size 패인을 선택하고, 라인을 클릭하여 이들이 매칭될 수 있도록 한다.

Autosizing 상자는 현재 선택된 객체(이 경우, 테이블 뷰를 둘러싸고 있는 윈도우를 의미)를 둘러싸고 있는 뷰를 나타낸다. Autosizing 상자내에 있는 사각형 영역은 선택된 객체를 의미하며, 스프링과 라인은 자유자재로 이동할 수 있다는 것을 보여준다. 그림에서 제공하는 설정값은 테이블 뷰의 경계와 테이블 뷰를 둘러싸고 있는 윈도우 사이의 간격이 고정적이라는 것(자동 크기조정 상자내에 있는 사각형에서 자동 크기조정 상자로 연결된 직선 라인)을 나타낸다. 그러나, 테이블 뷰 크기는 변경할 수 있다.(사각형 안에 있는 스프링) 라인을 클릭하면, 스프링으로 변경된다. 반대로 스프링을 클릭하면, 라인으로 변경된다. 어떤 결과가 발생하는 지 확인하려면, 다른 설정값을 입력해보거나, 매 시간 인터페이스를 시험해본다.

테이블 뷰를 완전하게 구성하려면, NSTableColumn 객체를 비롯하여 NSTableView 객체 속성을 설정해야 한다.

1. NSTableView를 선택한다.
2. <그림 9-5>에서 보여진 바와 같이 NSTableView Info 윈도우의 Attributes 패인에서 속성을 설정한다.



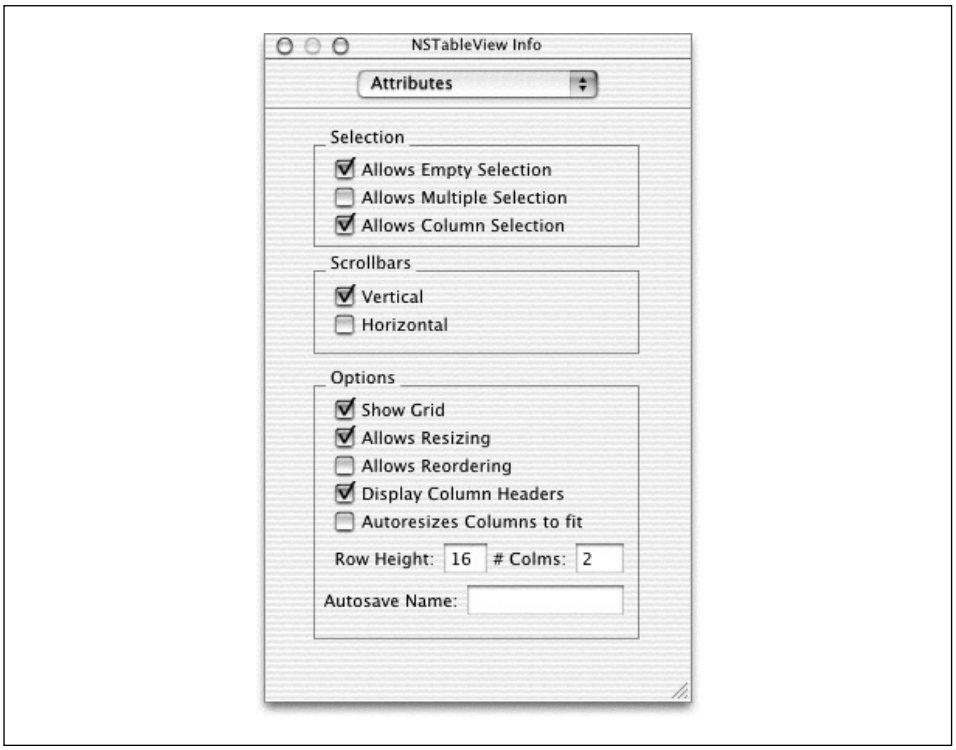

<그림 9-4> NSTableView의 크기 속성 변경하기

## 테이블에 세로열 만들기

이 섹션에서는 테이블의 세로열 넓이를 똑같이 만들고, 제목을 표현하는 방법을 설명한다.

1. 세로열 넓이를 똑같이 만든다. 가장 왼쪽에 있는 세로열을 선택하여(더블 클릭해야 함) 커서를 세로열의 우측 경계에 놓은 뒤, 커서가 화살표로 변경될 때를 기다린다. 세로열 끝을 드래그하여 세로열 뷰를 2개로 똑 같이 나눈다. 작업을 완료하면, 선택한 세로열을 클릭한다.
2. 세로열을 더블 클릭한다. **Column 1**을 입력하고, Return을 누른다.
3. 두 번째 세로열도 이 같은 작업을 반복하고, **Column 2**라고 이름을 지정한다.
4. 한 번에 한 개씩, 세로열을 선택하고, Table Column Info 윈도우를 불러온다. Attributes 패인을 선택하고, 제목이 중앙에 위치하도록 한다.

작업이 완료되면, 윈도우는 <그림 9-6>과 똑같아야 한다.




<그림 9-5> *NSTableView 속성*

## *데이터 소스 클래스 선언하기*

데이터 소스는 NSTableView에 데이터를 공급하는 응용 프로그램의 어느 객체나 될 수 있다. 이 장의 뒷 부분에서 테이블 뷰로 복잡한 데이터 모델을 구축하는 방법을 설명할 예정이다. 그러나, 복잡한 데이터 모델을 구현하려면, 테이블 뷰에 시험 데이터를 공급할 수 있는 전용 데이터 소스 객체를 만들어야 한다.

1. NSObject의 서브클래스를 만들고, 이름을 **`MyDataSource`**라 지정한다.
2. MyDataSource 클래스의 인스턴스를 만든다.
3. 테이블 뷰 객체에서 Instances 윈도우의 데이터 소스 객체로 연결라인을 드로잉한다. 연결 라인을 드로잉하기 전에 스크롤 뷰를 둘러싸고 있지 않은 테이블 뷰를 선택하였는지 확인한다. 테이블 뷰를 선택하면, 회색 음영이 만들어 진다.
4. Info 윈도우의 Connections 패인에서 `dataSource` 아웃렛을 선택하고, Connect를 클릭한다.
5. MyDataSource 파일을 만들고, Project Builder 프로젝트에 추가한다.


<그림 9-6> 2개의 세로열로 구성된 테이블 뷰

## 데이터 소스 메소드 구현하기

MyDataSource 클래스에서 테이블에 데이터를 제공하려면 2개의 메소드가 있어야 한다. 첫 번째 메소드는 데이터의 가로행이 몇 줄인지 테이블 뷰에 알려주고, 두 번째 메소드는 테이블에서 셀 값을 리턴한다. 테이블 뷰와 호환하려면, 제로 베이스 인덱싱을 통해 데이터 요소(일반적으로 소트 레코드)를 식별할 수 있어야 한다.

1. Project Builder에서 `MyDataSource.h`를 열고, 다음 선언을 추가한다.

```
- (int)numberOfRowsInTableView:(NSTableView *)tableView;
- (id)tableView:(NSTableView *)tableView
        objectValueForTableColumn:(NSTableColumn *)tableColumn
        row:(int)row;
```

2. `MyDataSource.m`을 열고, 다음 메소드 구현을 추가한다.

```
- (int)numberOfRowsInTableView:(NSTableView *)tableView
{
    return 10;
}
- (id)tableView:(NSTableView *)tableView
        objectValueForTableColumn:(NSTableColumn *)tableColumn
        row:(int)row
```

```
{
    return [NSCalendarDate date];
}
```

**numberOfRowsInTableView:** 구현은 이 같은 단순한 예제에서 항상 10을 반환한다. 이 메소드는 응용 프로그램에서 어레이 같은 컨테이너 객체를 조회하고, 어레이에서 수 많은 아이템을 리턴한다.

**tableView:objectValueForTableColumn:row:** 구현은 단순히 테이블에서 각 셀의 현재 시간과 날짜를 반환한다. 마찬가지로, 이 메소드는 기능을 구현할 때 가로행과 세로열에 기반한 컨테이너 객체에서 데이터를 검색하고, 테이블 뷰에 데이터를 반환한다.

### 프로젝트 구축 및 운용하기

간단한 테이블을 만들려면, 프로젝트를 구축하여 응용 프로그램을 운용하면 된다. 보시다시피, 윈도우의 크기를 조정하면 테이블 뷰 크기도 따라서 조정된다. 세로열 사이의 분할된 선을 드래그하여 세로열 넓이를 변경할 수 있다. 뷰에 있는 아이템을 클릭하면, 데이트 소스를 업데이트할지 여부를 묻는다.

## 좀 더 기능이 향상된 *Table View*의 예제

테이블 뷰를 구축한다 하더라도, 실제로는 거의 구현되지 않는다. 데이터 소스 클래스는 진정한 데이터 모델이 아니며, 사용자가 테이블에서 데이터를 변경할 방법은 없다고 할 수 있다. 이 섹션에서는 이와 같은 결점을 해결하기 위해 Expense 응용 프로그램을 변경하는 방법을 설명한다. 테이블 뷰의 새로운 예제 버전은 아주 단순한 Expense 응용 프로그램을 제공한다.

### *Expense* 클래스 만들기

먼저, 테이블 뷰 응용프로그램에 데이터 모델 클래스를 추가한다. Expense 추적 응용 프로그램을 만들었기 때문에 Expense를 표현할 클래스가 필요하다. Expense, Category 및 Amount 데이터를 저장한 클래스를 만든다.

1. Project Builder에서, Expense 예제 프로젝트를 연다.(실행되지 않았을 경우)
2. File 메뉴에서 New File를 선택한다.
3. 클래스 Objective-C Assistant 대화 상자가 나타난다. 리스트에서 Objective-C Class를 선택한다.
4. 새로운 파일을 **Expense.m**이라 정하고, Finish를 클릭한다. Project Builder로 헤더 및 구현 파일을 만들고, 기본적인 템플리트를 삽입하여, 프로젝트에 이들을 추가한다.

5. Expense.h를 열고, 다음의 인스턴스 변수 선언을 추가한다. { } 괄호안에 이 선언을 삽입해야 한다.

```
NSCalendarDate *date;
NSString *category;
NSDecimalNumber *amount;
```

6. 다음의 접근자 메소드 선언을 추가한다. { } 괄호와 @end 지시문 사이에 이 선언을 삽입해야 한다.

```
- (NSCalendarDate *)date;
- (void)setDate:(NSCalendarDate *)value;
- (NSString *)category;
- (void)setCategory:(NSString *)value;
- (NSDecimalNumber *)amount;
- (void)setAmount:(NSDecimalNumber *)value;
```

7. Expense.m에 #import와 @implementation 지시문 사이에 다음 라인을 추가한다.

```
static NSString *defaultCategory = @"Food";
```

8. 구현 섹션에서 init 메소드를 추가한다.

```
- (id)init {
    self = [super init];
    [self setDate: [NSCalendarDate date]];
    [self setCategory: defaultCategory];
    [self setAmount: [NSDecimalNumber zero]];
    return self;
}
```

9. 다음 dealloc 메소드를 추가한다.

```
- (void)dealloc {
    [date release];
    [category release];
    [amount release];
    [super dealloc];
}
```

10. 마지막으로, 접근자 메소드 구현을 추가한다.

```
- (NSCalendarDate *)date {
    return date;
}

- (void)setDate:(NSCalendarDate *)value {
    [date autorelease];
    date = [value copy];
}

- (NSString *)category {
    return category;
}
```

```
- (void)setCategory:(NSString *)value {
        [category autorelease];
        category = [value copy];
    }

    - (NSDecimalNumber *)amount {
        return amount;
    }

    - (void)setAmount:(NSDecimalNumber *)value {
        [amount autorelease];
        amount = [value copy];
    }
```

이제, 기능이 완전히 갖춰진 Expense 클래스가 완성되었다. Expense 응용 프로그램에 대한 수정이 완료되면, 데이터 소스 객체는 Expense 객체의 어레이를 포함한다. 테이블 뷰의 각 가로행은 어레이에서 해당 Expense 객체를 제공한다. 테이블 뷰의 세로열은 가로행의 Expense 객체에 있는 인스턴스 변수에 해당한다.

### 테이블 뷰에 세로열 추가하기

테이블 뷰에 세 번째 세로열을 만든다. 3개의 세로열은 각각 Date, Category 및 Amount에 해당한다.

1. 메인 nib 파일을 연다.
2. NSTableView 객체를 선택하고, Info 윈도우의 Attributes 패인에서 세로열을 3으로 변경한다.
3. 3개의 세로열이 똑같은 넓이가 되도록 크기를 조정한다. 새로 만들어진 세 번째 세로열은 두 번째 세로열의 우측에 나타난다. 두 번째 세로열을 선택하여, 우측 경계선을 좌측으로 움직여 세 번째 세로열을 만든다.

### 테이블에 세로열 만들기

테이블의 세로열 제목을 만들고, 포매터를 추가하여 응용 프로그램의 UI를 더욱 매력적으로 만든다.

1. 3개의 세로열에 각각 **Date**, **Category** 및 **Amount**라는 제목을 만든다.
2. Info윈도우를 사용하여, 세 번째 세로열의 라벨을 중앙에 놓는다.
3. Date 세로열을 선택하여, 날짜 포매터를 Date 줄의 제목칸으로 드래그한다. 커서로 날짜 포매터를 입력하면, 제목칸의 윤곽이 강조된다. Date 세로열의 Info 윈도우에 있는 Formatter 패인에서 맘에 드는 날짜 포맷을 선택한다. 또한, Allow Natural Language 상자를 선택한다.

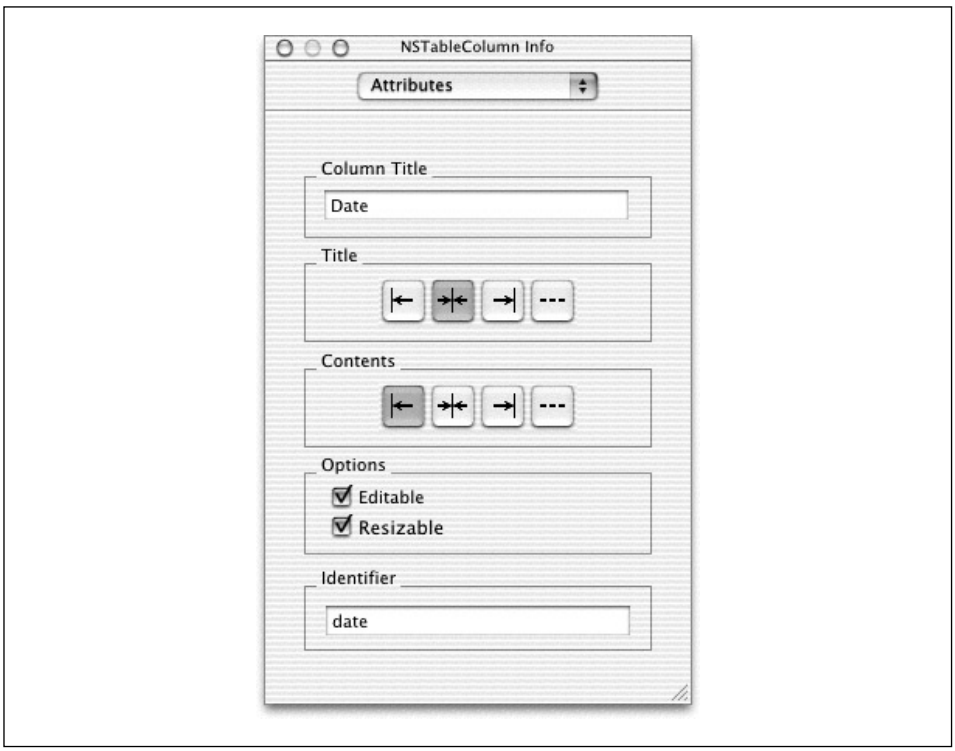
포매터의 장점은 날짜를 입력하기 위해 "today", "yesterday", "tomorrow"와 같이 스트링을 사용할 수 있도록 한다.

4. Amount 세로열을 만들기 위해 날짜 포매터 대신에 번호 포매터를 사용하여 3단계를 반복한다.

### 테이블 세로열 식별자 추가하기

세로열 식별자는 데이터 모델 객체에서 세로열을 속성과 맵핑시키기 위해 사용하는 객체 값이다. 테이블 뷰가 데이터 소스에 데이터를 요청하는 메시지를 전송하면, 데이터 소스가 어떤 세로열에 데이터를 삽입할 것인지, 검색할 데이터 모델의 속성은 어떤 것인지 결정할 수 있도록 세로열 식별자를 제공한다.

세로열 식별자를 설정하기 위해 Interface Builder에서 테이블 뷰의 각 세로열을 선택하여, Info 윈도우의 Attributes 패인(<그림 9-7 참조>)을 사용한다.




<그림 9-7> 테이블 뷰 세로열에 세로열 식별자 추가하기

Date 세로열에는 "Date", Category 세로열에는 "Category", Amount 세로열에는 "Amount"를 사용한다. 식별자로 스트링 값을 사용할 수 있다. 잠시 후면 이 같은 스트링을 사용해야할 이유가 발생한다.

## 데이터 소스 클래스 업데이트하기

사용자 인터페이스로 작업을 완료하였기 때문에, 앞서 만든 Expense 클래스를 사용할 수 있도록 데이터 소스 클래스를 일부 변경해야 한다. 이 장의 앞 부분에서 설명한 데이터 소스의 새로운 구현은 Expense 객체의 어레이를 데이터 모델로 그대로 유지한다.

1. Project Builder에서 **MyDataSource.h**를 연다.
2. Expense 객체의 인스턴스 변수를 추가한다.

```
NSMutableArray *expenses;
```

3. 접근자 메소드의 메소드 선언을 추가한다.

```
- (NSMutableArray *)expenses;
- (void)setExpenses:(NSMutableArray *)newExpenses;
```

4. **MyDataSource.m**을 열어, **Expense.h**를 임포트한다.

```
#import "Expense.h"
```

5. 접근자 메소드 구현을 추가한다.

```
- (NSArray *)expenses {
    return expenses;
}

- (void)setExpenses:(NSMutableArray *)newExpenses {
    [expenses autorelease];
    expenses = [newExpenses retain];
}
```

6. **init**와 **dealloc** 메소드 구현을 추가한다.

```
- (id)init {
    self = [super init];
    [self setExpenses: [NSMutableArray array]];
    return self;
}

- (void)dealloc {
    [expenses release];
    [super dealloc];
}
```

7. **numberOfRowsInTableView:** 구현을 변경하여, Expense 객체 어레이의 아이템 번호를 리턴한다.

```
- (int)numberOfRowsInTableView:(NSTableView *)tableView
{
    return [expenses count];
}
```

8. **tableView:objectValueForTableColumn:row:** 구현을 변경하여 Expense 객체 어레이에서 값을 검색한다.

이 메소드는 Expense 객체에서 데이터를 검색하기 위해 세로열 식별자를 사용한다. 이 메소드는 메시지를 전송하는 방식으로 역할을 수행한다. 먼저, 세로열 식별자는 메시지 파라미터로써 전송된 테이블의 세로열 객체로부터 검색된다. 그 다음, Expense 객체는 메시지 파라미터로써 전송된 가로행 번호를 사용하여 Expense 어레이에서 찾아볼 수 있다.

이제부터, 좀 까다로워진다. Interface Builder에서 할당된 테이블의 세로열 식별자 스트링이 Expense 객체의 인스턴스 변수 이름과 동일하기 때문에 Expense 객체에서 해당 속성 값을 찾기 위해 *키-값 코딩*을 사용한다. 키-값 코딩을 통해 인스턴스 변수에 간단하게 액세스할 수 있다. 객체가 인스턴스 변수를 갖고 있으면, 변수 이름을 **valueForKey:**라는 메시지의 파라미터로 사용할 수 있으며, 객체에서 그 변수 값을 얻을 수 있다.

```
- (id)tableView:(NSTableView *)tableView
    objectValueForTableColumn: (NSTableColumn *)tableColumn
    row: (int)row
{
    NSString *identifier = [tableColumn identifier];
    Expense *expense = [expenses objectAtIndex: row];

    return [expense valueForKey: identifier];
}
```

키-값 코딩은 실질적으로 데이터 소스 메소드 구현을 단순화한다. 이 같은 편리함이 없었다면, Expense 객체의 어떤 접근자 메소드를 호출할 것인지 결정하기 위해 복잡한 **if...then** 지시문으로 세로열 식별자의 스트링 값을 시험했어야 했다. 따라서, 한 줄을 삽입하는 대신에 다음과 같은 긴 라인의 코딩 작업을 수행했을 것이다.

```
id value = nil;

    if ([identifier isEqual: @"date"])
        value = [expense date];
    else if ([identifier isEqual: @"category"])
        value = [expense category];
```

```
else
    value = [expense amount];

return value;
```

## 시험 데이터에 Expense 어레이 정렬하기

이제, 이 응용 프로그램을 구축 및 운용할 수 있다. 그러나, 데이터를 Expense의 비어 있는 어레이에 추가하는 방법을 구현하지 않았기 때문에 아직은 어떤 내용도 확인할 수 없다. 이 섹션에서는 **awakeFromNib** 메소드를 사용하여 Expense 어레이를 데이터로 채우고, 데이터 소스 객체를 설정하여 시험 데이터를 사용하는 방법을 설명한다.

1. Expense 객체 어레이에 시험 데이터를 채운후 **MyDataSource.m**에 메소드를 추가하고, **MyDataSource.h**에서 메소드를 선언한다. 각각의 객체는 루프 카운터와 동일한 현재 날짜, 기본적인 Expense 카테고리와 dollor amount를 사용하여 만들어진다.

```
- (NSMutableArray *)generateTestData
{

    NSMutableArray *array = [NSMutableArray array];
    int index;

    for (index = 0; index < 15; index++) {
        Expense *exp = [[Expense alloc] init];
        [exp setAmount:
                (NSDecimalNumber *)[NSDecimalNumber numberWithInt:index]];
        [array addObject: exp];
        [exp release];
    }

    return array;
}
```

2. **MyDataSource.m**의 **awakeFromNib**를 구현한다.

```
- (void)awakeFromNib
{
   [self setExpenses: [self generateTestData]];
}
```

## 응용 프로그램 구축 및 시험하기

응용 프로그램을 구축하고, 시험한다. <그림 9-8>과 동일한 화면이 나타나야 한다.


<그림 9-8> 시험 데이터가 있는 테이블 뷰

## 테이블 뷰에서 값 변경하기

사용자가 테이블 값을 변경하고, Expense 객체의 데이터 소스 어레이에서 관련 속성에 새로운 값을 복사하려면, 한번 더 **tableView:setObjectValue:forTableColumn:row:** 메소드를 구현해야 한다. 키-값 코딩을 통해 다음과 같이 구현한다.

```
- (void)tableView:(NSTableView *)tableView
    setObjectValue:(id)object
    forTableColumn:(NSTableColumn *)tableColumn
    row:(int)row
{
    NSString *identifier = [tableColumn identifier];
    Expense *expense = [expenses objectAtIndex: row];

    [expense takeValue: object forKey: identifier];
}
```

## 테이블 뷰에 데이터 입력하기

응용 프로그램을 다시 구축 및 운용한다. 새로운 값을 입력하려면 테이블의 셀을 더블 클릭한다. Return을 눌러 테이블의 새로운 값을 확인한다. 날짜 셀 중 하나인 "tomorrow"를 입력해본다.

## 심층 검토

몇몇 일반적으로 사용된 NSTableView 메소드는 다음과 같다.

- `reloadData:` 이 메소드는 테이블 뷰를 데이터 소스에서 다시 로딩시킨다.
- `noteNumberOfRowsChanged:` 이 메소드는 데이터 소스 크기가 변경되었다고 테이블 뷰에게 알려준다.
- `selectedRow:` 이 메소드는 현재 선택된 가로 인덱스를 리턴한다.
- `editColumn:row:withEvent:select:` 이 메소드는 NSTextField의 `selectText` 메소드와 함께 사용되며, 단일 셀을 편집 모드로 전환시킨다.

NSTableView는 `tableView:willDisplayCell:forTableColumn:row:` 및 `tableView:shouldEditTableColumn:row:`를 포함한 델리게이트 메소드를 정의한다. 데이터 소스를 테이블 뷰의 델리게이트로 만들 수 있으며, 테이블 뷰와 사용자가 더욱 밀접하게 상호작용할 수 있도록 이 같은 메소드를 구현하고, 동작을 변경한다.

NSTableView는 테이블 뷰에서 데이터 소스 추적 변경을 지원할 수 있도록 `NSTableViewSelectionDidChangeNotification`와 `NSTableViewColumnDidMoveNotification` 같은 통지를 제공한다.

NSTableView를 검토하고, NSTableView가 데이터 소스와 어떻게 상호작용하는지에 관한 방법을 알아보려면 예제 응용 프로그램을 변경해 본다. 그에 따라, Expense를 추가하거나 삭제할 수 있다.

`/Developer/Examples/AppKit`의 예제 응용 프로그램인 OutlineView을 검토하고 운용할 수 있다. NSOutlineView는 NSTableView보다 약간 복잡할 수 있지만, 지금까지 배운 내용을 활용하면 운용하는 방법을 이해하는 데에는 아무런 문제가 없다.

## 객체 네트워크 플래튼하기: 코딩과 아카이빙

실제로, 모든 응용 프로그램은 자신들 객체를 영구적으로 만들어야 한다. 예를 들면, 상기 섹션에서 생성한 Expense 응용 프로그램은 데이터 모델 상태를 저장하지 않는다. 그래서, 응용 프로그램을 닫으면 Expense 응용 프로그램은 전부 손실된다. Cocoa 응용 프로그램은 코딩 및 아카이빙을 사용하여 차후 검색을 위해 도큐멘트 내용과 다른 중요한 응용 프로그램 데이터를 디스크에 저장한다. 일부 응용 프로그램은 코딩 및 아카이빙을 사용하여 다른 응용 프로그램으로 객체를 전송한다.(예를 들면, 친구에게 Expense를 전송하고자 할 경우)

이 섹션에서는 코딩과 아카이빙을 사용하기 위해 상기 섹션에서 설명한 단순한 경비 추적 응용 프로그램을 변경하여 Expense 어레이를 저장하고, 다시 로딩하는 방법을 설명한다.

## NSCoder 및 NSArchiver

NSCoder가 구현한 코딩은 응용 프로그램의 객체들처럼(*object graph*) 객체 그룹을 연결하고, 객체의 상태, 구조, 관계 및 클래스 멤버를 캡쳐하여 데이터를 직렬화한다. NSCoder의 서브클래스인 NSArchiver는 파일의 직렬화된 데이터를 저장하여 동작을 확대한다.

객체 그래프의 루트 객체를 아카이빙하면, 그 객체뿐만 아니라 루트 객체가 참조한 모든 객체들, 즉, 2단계 객체가 참조한 모든 객체등이 아카이빙된다. 아카이빙을 하려면, 객체는 NSCoding 프로토콜(`encodeWithCoder:` 및 `initWithCoder:` 메소드로 구성)을 준수해야 한다.

## 코딩 및 아카이빙을 Expenses 응용 프로그램에 추가하기

1. Interface Builder의 메인 nib 파일을 연다.
2. `MainMenu.nib` 윈도우의 Instances 빌더내 Window Instances에서 MyDataSource Instances로 Control 키를 누른채 드래그하여 연결한다. Info 윈도우의 Connections 패인에서 MyDataSource를 Window의 델리게이트로 만든다.
3. Project Builder에서 TableView 예제 프로젝트를 연다.
4. `Expense.h`를 열고, 클래스 선언을 다음과 같이 변경한다. `<NSCoding>`를 추가하면 Expense 클래스가 코딩 프로토콜을 준수한다는 것을 선언한 것이다.

```
@interface Expense : NSObject <NSCoding>
{
```

5. `Expense.m`을 열어, NSCoding 메소드를 추가한다. 가장 자주 사용하는 NSCoder 메소드인 `encodeObject:`는 단일 객체를 인코딩(직렬화)한다. 비객체 유형의 경우, `encodeValueOfObjCType:at:`를 사용할 수 있다. 디코딩 순서는 인코딩 순서와 동일해야 한다. 날짜를 먼저 인코딩하기 때문에 디코딩할 경우에도 날짜가 먼저 디코딩되어야 한다. NSCoder는 encode 메소드에 해당하는 `decode` 메소드를 정의한다. `init` 메소드에서와 같이 초기화된 인스턴스인 `self`를 리턴하여 마무리 짓는다.

```
- (id)initWithCoder:(NSCoder *)coder {
    [self setDate: [coder decodeObject]];
    [self setCategory: [coder decodeObject]];
    [self setAmount: [coder decodeObject]];
    return self;
}
```

```
- (void)encodeWithCoder:(NSCoder *)coder {
    [coder encodeObject: [self date]];
    [coder encodeObject: [self category]];
    [coder encodeObject: [self amount]];
}
```

6. 다음의 **windowShouldClose:** 델리게이트 메소드를 MyDataSource에 추가한다. 이 메소드는 사용자가 윈도우에서 작업한 내용을 저장하고자 할 경우, 먼저 표준 경보 요청 사용자를 보여준다. 사용자가 Save를 클릭하면, Save 대화 상자(NSSavePanel)가 나타나 사용자가 파일명을 작성하고, 저장할 디렉토리를 선택할 수 있도록 유도한다. **setRequiredFileType:** 메소드는 파일이 인식가능한 유형이 될 수 있도록 Save 패널을 **.expenses**에 추가시킨다. 사용자가 Save를 클릭하면, NSArchiver는 Expense 객체의 전체 어레이를 인코딩하여 디스크에 저장한다. 사용자가 Cancel을 클릭하면, 윈도우는 닫히고, 어떤 내용도 저장되지 않는다.

```
- (BOOL)windowShouldClose:(NSWindow *)sender
{
    NSSavePanel *sp;
    int answer;

    answer = NSRunAlertPanel(@"Save Expenses",
        @"Do you want to save?",
        @"Save",
        @"Don't Save",
        nil);

    if (answer == NSAlertDefaultReturn) {
        sp = [NSSavePanel savePanel];
        [sp setRequiredFileType:@"expenses"];
        answer = [sp runModal];
        if (answer == NSOKButton ) {
            [NSArchiver archiveRootObject:[self expenses]
                    toFile:[sp filename]];
        }
    }

    return YES;
}
```

7. 마지막으로, 사용자가 파일을 로딩할 수 있도록 **awakeFromNib** 구현을 변경한다. 이 메소드는 표준 개방형 대화 상자를 제공하기 때문에 사용자가 로딩할 저장 데이터 파일을 선택할 수 있다. 타입 어레이는 사용자가 유효한 Expense 파일(즉 **.expenses** 파일 확장자를 가진 파일)을 선택할 수 없다는 것을 의미한다. 파일을 선택하면, NSUnarchiver는 저장장치에서 Expense 객체 어레이를 복구시킨다. 사용자가 Cancel를 클릭하면, 시험 데이터는 이전과 같이 작성된다.

```
- (void)awakeFromNib
{
    NSOpenPanel *op;
```

```
    int answer;

    op = [NSOpenPanel openPanel];
    answer = [op runModalForTypes: [NSArray arrayWithObject:@"expenses"]];

    if (answer == NSOKButton) {
        [self setExpenses: [NSUnarchiver unarchiveObjectWithFile:
                [op filename]]];
    } else {
        [self setExpenses:[self generateTestData]];
    }
}
```

# 10

# *Travel Advisor* 튜토리얼

이 장을 통해 개발자는 Cocoa 프로그래밍 기법을 확장할 수 있다.

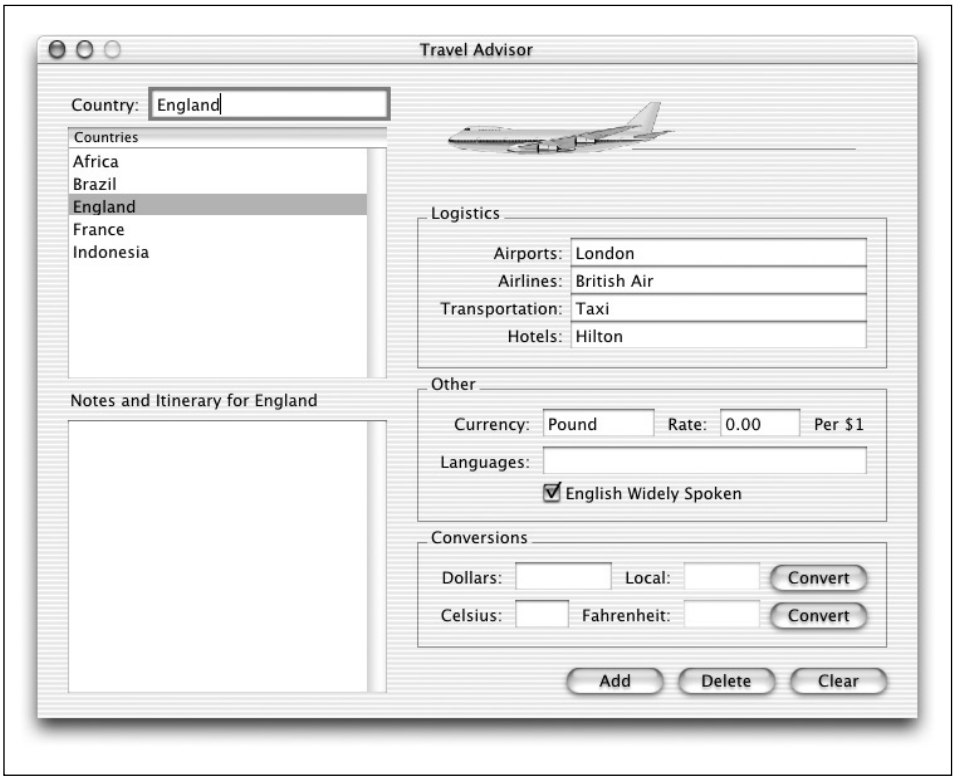
- 양식과 테이블 뷰(제9장, *데이터 기능성*에서 간단하게 소개한 것보다는 심층적으로 다룬다)를 사용한다.
- Interface Builder에서 객체를 그룹화한다.
- 이미지를 응용프로그램에 추가한다
- 필드를 포맷하여 사용한다
- 간단한 출력을 하기 위해 연결을 한다
- 콜렉션 객체 및 스트링 객체를 사용한다
- 다른 응용 프로그램에서 커스텀 객체를 재사용한다

상기 기능을 수행하는 매개체는 Travel Advisor라는 응용 프로그램이다. Travel Advisor는 사용자가 여행하는 국가들에 대한 기록을 입력하여 보거나 삭제할 수 있는 양식 기반 응용 프로그램이다. 이 응용 프로그램은 Model-View-Controller(MVC) 디자인을 활용하고, 앞장에서 소개한 여러 Cocoa 프로그래밍 기법을 심층적으로 활용할 기회를 제공한다.

## *Travel Advisor 디자인*

Travel Advisor는 기본적으로 디자인 측면에서 Currency Converter와 상당히 유사하다. 그러나, 프로그램 측면에서 훨씬 더 복잡하다. <그림 10-1>은 Travel Advisor의 사용자 인터페이스를 보여준다. Travel Advisor를 사용하려면, 특정 국가와 관련된 여행 관련 정보와 함께 그 국가명을 입력한다. Add를 클릭하면, 국가명이 국가 기록 테이블에 나타난다. 몇 개 국가들에 관한 데이터를 입력한 후, 양식에 나타난 국가들에 관한 정보와 테이블에서 특정 국가를 선택할 수 있다.

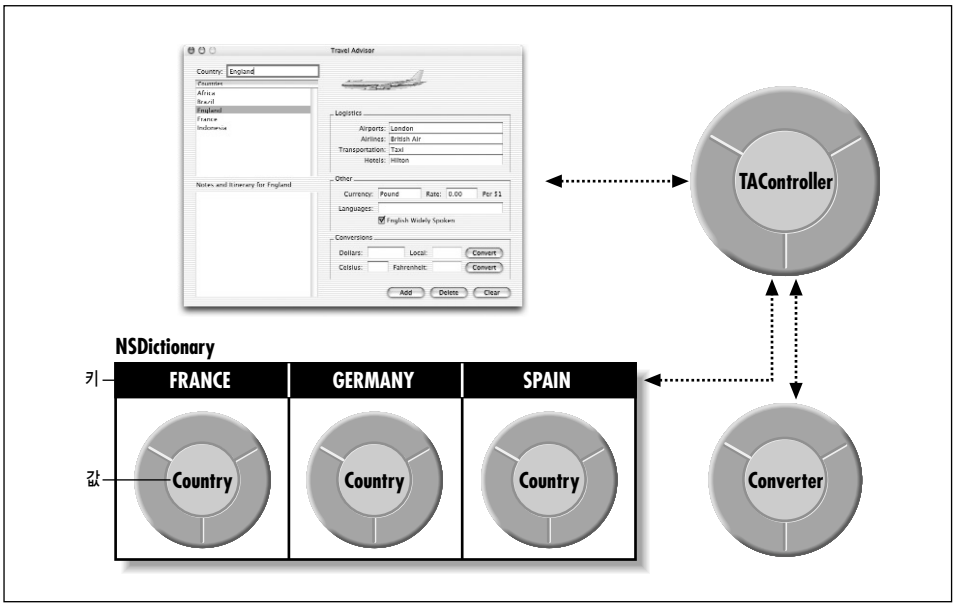
응용 프로그램을 닫기 전에 모든 여행 정보를 디스크에 저장할 수 있다. 그래서, 세션 사이에 정보가 손실되지 않도록 한다. 또한, 이 프로그램을 통해 환율과 온도를 참조할 수 있다.




*<그림 10-1> Travel Advisor의 사용자 인터페이스*

Travel Advisor는 Model-View-Controller 패러다임을 기반으로 한다. Controller 객체(TAController)는 Application Kit 객체로 구성된 사용자 인터페이스를 관리하고, 컨트롤러는 메시지를 Model 객체에 전송하여 다양한 처리 결과를 얻는다. <그림 10-2>는 상위 단계 디자인을 보여준다.

Travel Advisor에서는 다양한 통화율의 변화를 알아야 하기 때문에 Currency Converter 프로젝트에서 Converter 객체를 변경하지 않고, 재사용할 수 있다.




<그림 10-2> *Travel Advisor 디자인 개요*

## *Model 객체*

Travel Advisor는 사용자가 선택한 국가에 따라 고유 데이터를 보여줘야 한다. 이 작업을 실행하려면, 각 국가 데이터가 Country 객체에 저장되어야 한다. 이 객체는 국가에 관한 데이터를 캡슐화한다.(어떤 면에서 보면, 관계형 데이터베이스의 레코드와 유사한 면이 있다) 이 응용 프로그램은 "하드와이어(hardwired)" 방식에 의존하지 않고, 객체를 추적하는 방식으로 수 백개의 객체를 관리할 수 있다.

이 응용 프로그램의 또 다른 Model 객체는 Converter 클래스의 인스턴스이다. 이 인스턴스는 데이터를 저장하지 않지만, 특화된 동작을 제공한다.

## *View 객체*

Travel Advisor의 뷰 객체는 텍스트 필드, 체크박스, 버튼 등으로 구성된 Application Kit이다.

## *Controller 객체*

응용 프로그램의 Controller 객체는 TAController이다. TAController는 Controller 객체처럼 사용자 인터페이스(패러다임의 View)와 Country 객체를 캡슐화하는 Model 객체간의 데이터 흐름을 조정한다.

TAController는 인터페이스에서 사용자 선택을 기반으로 하여, 요청된 Country 객체를 검색하여, 보여줄 수 있다. 또한, 사용자가 변경한 사항을 해당 Country 객체에 저장할 수 있다. 이 작업을 수행하는 객체는 NSDictionary 객체(여기서는 딕셔네리라 지칭)이다. 제6장, *중요한 Cocoa 패러다임*에서 설명한 대로 딕셔네리는 객체를 저장하고, 키값으로 검색을 허용한다. 키는 관련 객체를 "검색"하기 위해 사용할 수 있는 고유 식별자이다. 객체를 얻으려면, 키를 인수(`objectForKey:`)로 사용하여 딕셔네리에 메시지를 전송한다. 예를 들면,

```
NSColor *aColor = [aDictionary objectForKey: @"BackgroundColor"];
```

이 예제에서 `aDictionary`는 스트링 키와 연결된 1개 이상의 NSColor를 보유한다. Dictionary 객체가 objectForKey: 메시지를 수신하면, `BackgroundColor` 인수와 연결된 객체를 검색한다.

Travel Advisor(이 장의 뒷 부분에서 설명함)에서 Country 객체는 인스턴스 변수로 국가명을 제공한다. 이 국가명은 딕셔네리 키로써 동작한다. 딕셔네리에 Country 객체를 저장할 경우, 객체 키로써 국가명(NSString 형태)을 저장할 수 있다. 추후, 딕셔네리에 국가명을 가진 `objectForKey:` 메시지를 전송하여 객체를 찾는다.

### *TAController가 데이터를 관리하는 방법*

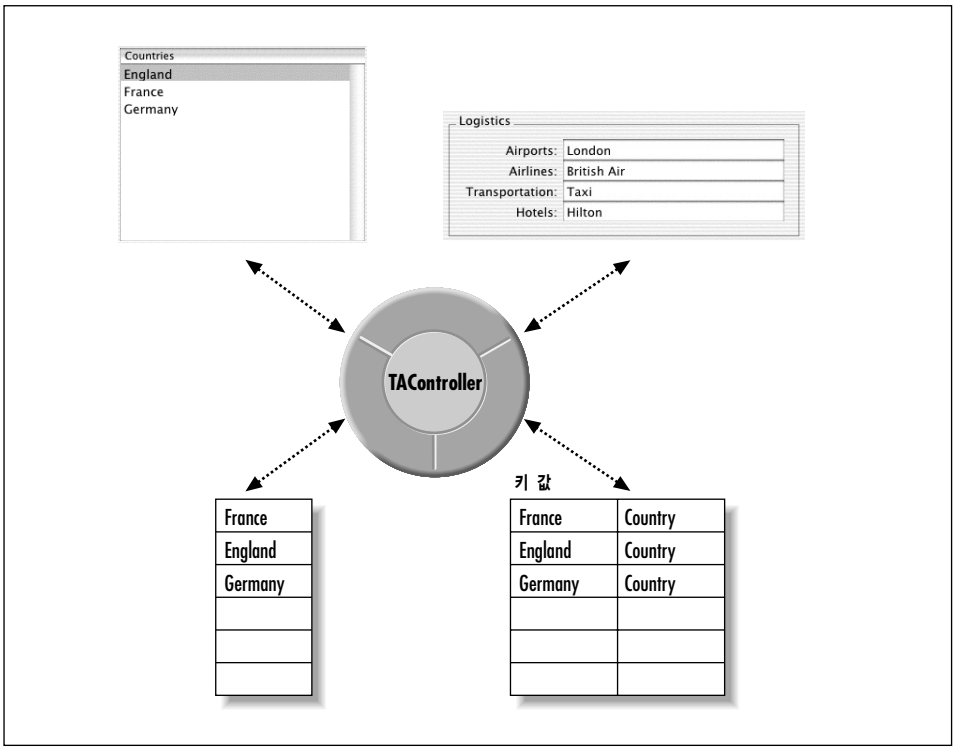
TAController 클래스는 <그림 10-3>에서 보여진 바와 같이 Travel Advisor 응용 프로그램에서 중심적인 역할을 한다. 응용 프로그램의 Controller 객체로써 Model 객체(Country 인스턴스)에서 인터페이스 필드로 데이터를 전송하고, 사용자가 데이터를 입력하거나 변경할 때 정상적인 Country 객체를 리턴한다. TAController는 현재 객체들과 테이블 뷰에 나타난 데이터를 조정해야 하고, 사용자가 테이블 뷰의 아이템을 선택하거나 Add 또는 Delete 버튼을 클릭하면 정상적인 작업을 수행해야 한다. 사용자 인터페이스 고유의 모든 커스텀 코드는 TAController에 상주한다. 이 동작 메카니즘은 어레이를 필요로 한다.

### *데이터 소스 정보 저장하기*

TAController는 이용 가능한 국가 리스트를 보여주기 위해 사용된 테이블 뷰의 데이터 소스를 관리한다. 또한, 알파벳별로 분류된 어레이 객체(NSArray)에서 모든 Country 객체를 포함하고 있는 딕셔네리에 키를 저장한다. 테이블 뷰에서 데이터를 요청할 경우, TAController는 어레이에서 분류된 객체 리스트를 "공급"한다.

### *Country 객체 생성*

디자인에서 중요한 요소는 Country 객체를 만드는 방식이다. Interface Builder가 Country 객체를 만드는 대신에, TAController 객체는 Add 버튼을 클릭한 사용자에 응답하는 방식으로 Country 객체를 만든다.




<그림 10-3> TAController로 데이터 처리하기

### 델리게이션과 통지

<그림 10-3>에는 나타나있지 않지만, 디자인의 필수적인 요소로 델리게이션과 통지가 있다. 응용 프로그램을 구축할 때 이 디자인 패턴이 어떻게 활용되는지 살펴보기로 하자.

## Travel Advisor 인터페이스 만들기

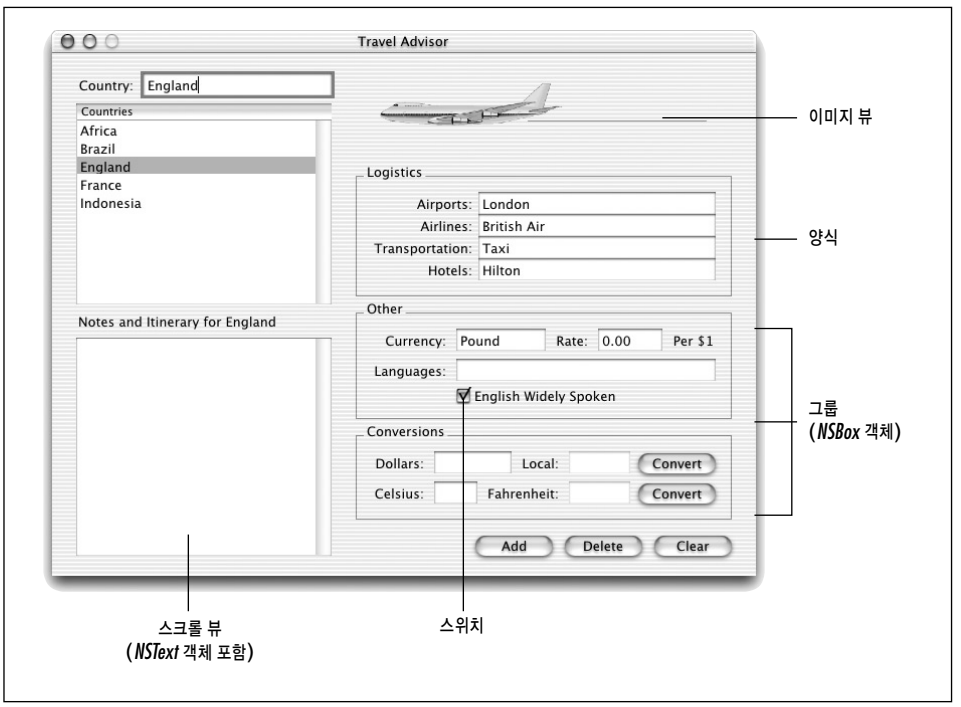
Travel Advisor 인터페이스를 만들 때, Currency Converter로 작업했을 때보다 훨씬 더 Interface Builder의 기능을 시험해볼 수 있는 기회를 가질 수 있다. 이 섹션에서는 Travel Advisor 인터페이스를 완성하기 위해 수행해야 할 작업내용을 설명한다.

1. 응용 프로그램 프로젝트를 만든다.
2. 응용 프로그램의 nib 파일을 연다.
3. 응용 프로그램 윈도우를 사용자화한다.
4. 텍스트 필드, 라벨과 버튼을 윈도우에 추가한다.

5. 양식 객체를 윈도우에 추가한다.
6. 사용자 인터페이스 객체를 그룹화한다.
7. 텍스트 뷰를 추가한다.
8. 테이블 뷰를 추가 및 구성한다.
9. 이미지를 인터페이스에 추가한다.
10. 메뉴 및 메뉴 아이템을 추가한다.
11. 포맷터를 추가한다.
12. 필드간 탭 기능을 작동하고, 출력하기 위해 연결을 한다.
13. 인터페이스를 시험한다.

## 시작하기

Travel Advisor 인터페이스의 여러 객체들은 제7장, *Currency Converter 튜토리얼*에서 이미 수행했었기 때문에 친숙해져 있어야 한다. <그림 10-4>는 이 튜토리얼에서 처음으로 접한 객체들을 보여준다.




<그림 10-4> 새로운 인터페이스 요소

## 응용 프로그램 프로젝트 생성하기

응용 프로그램의 새로운 프로젝트를 생성한다.

1. Project Builder를 구동한다.
2. File 메뉴에서 New Project를 선택한다.
3. New Project 패널에서, Cocoa Application 프로젝트 유형을 선택하고, Next 버튼을 클릭한다.
4. 응용 프로그램의 이름을 Travel Advisor라 지정한다.
5. 응용 프로그램을 특정 위치에 프로젝트를 저장하려면, Set을 클릭하여 위치를 선택한다. 기본적인 위치를 사용하려면, 6단계로 간다.
6. Finish를 클릭한다.

## 응용 프로그램 윈도우를 사용자화하기

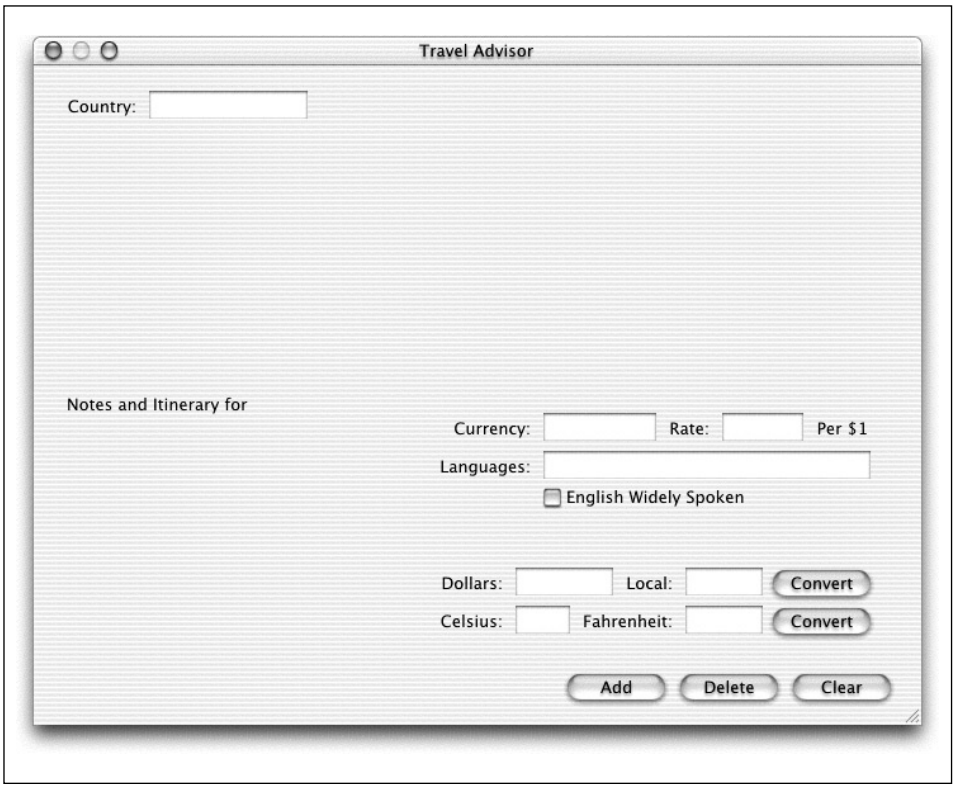
응용 프로그램의 메인 윈도우 이름을 다시 작성하고, 크기를 조정한다.

1. 프로젝트 브라우저에서 Resources 그룹을 연다.
2. `MainMenu.nib`를 더블클릭한다.(이로써, Interface Builder가 구동된다)
3. Interface Builder의 Info 윈도우가 실행되지 않았다면, 불러온다.(Command-Shift-I)
4. 윈도우 타이틀을 **`Travel Advisor`**로 변경한다.
5. Controls 영역에서, Resize 속성을 끈다.
6. Window Info의 팝업 메뉴를 클릭하고, Size를 선택한다.
7. Info 윈도우의 Content Rect 영역에서, 좌측 팝업 메뉴에 Top/Left가 나타나는지 확인한다.
8. x필드를 30, y필드를 80으로 설정한다. 이것은 응용 프로그램이 처음으로 구동될 때 Travel Advisor 윈도우의 시작 위치를 설정하는 것이다. Interface Builder에서 Travel Advisor 윈도우를 움직이면 값이 변경된다.
9. 또한, Info 윈도우의 Content Rect 영역에서, 우측 팝업 메뉴에 Width/Height가 나타나는지 확인한다.
10. Width를 700에 Height를 520에 설정한다.

## 텍스트 필드, 라벨 및 버튼 윈도우에 추가하기

응용 프로그램 사용자 인터페이스의 첫 단계를 완성한다.

1. <그림 10-5>에서 보여진 바와 같이 객체를 배치하고, 크기 조정하여 초기화한다. 힌트: Duplicate 기능(Command-D)을 자유자재로 사용할 수 있다면 이 단계를 훨씬 더 빨리 진행할 수 있다.

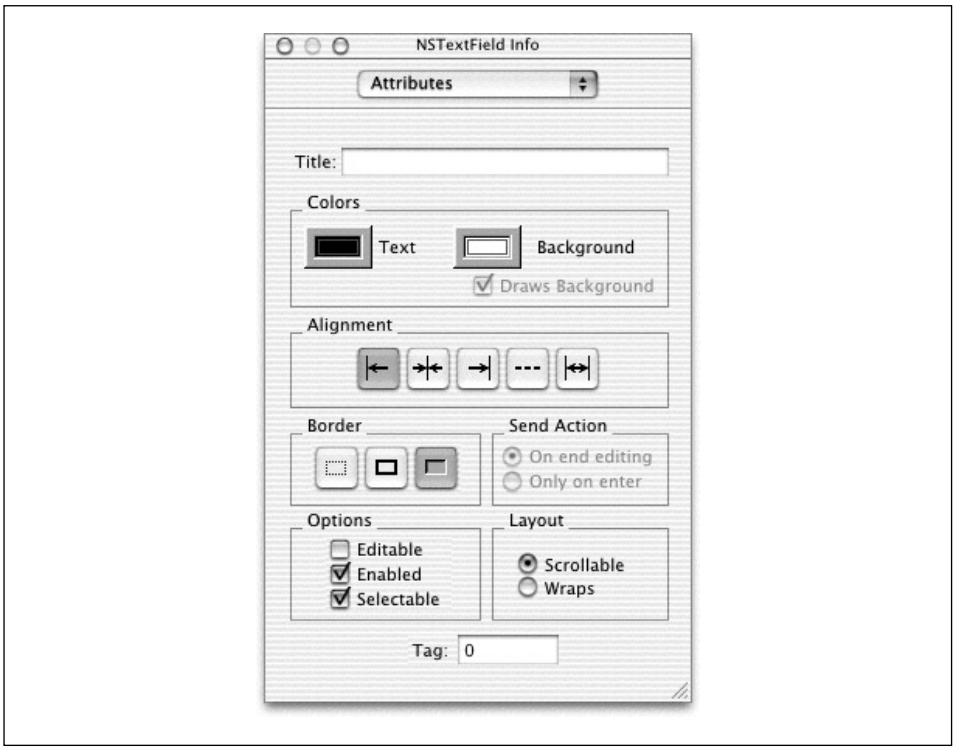



*<그림 10-5> Travel Advisor 인터페이스의 초기 레이아웃*

2. <그림 10-6>에서 보여진 바와 같이 Info 윈도우(Command-Shift-I)를 사용하여 Local 및 Fahrenheit을 편집할 수 없도록 텍스트 필드 속성을 변경한다.이 필드는 계산 결과를 나타내기 위해 사용한다. 이로써, 사용자는 편집할 필요가 없다.

### 버튼에 관한 상세한 정보

Interface Builder에서 English Widely Spoken 스위치를 선택하여 Attributes Info 윈도우를 불러오면, 스위치가 단순히 버튼의 유형이라는 것을 확인할 수 있다.(<그림 10-7> 참조)



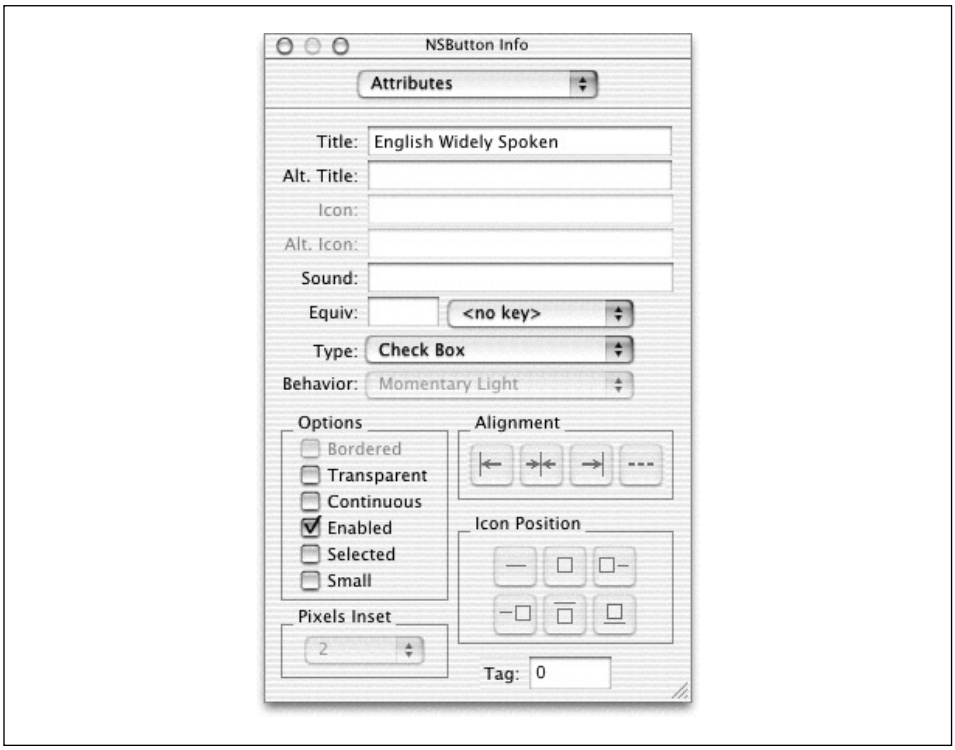

*<그림 10-6> Info 윈도우의 체크박스 속성*

버튼은 2가지의 상태를 가진 컨트롤 객체이다. 2가지의 상태란 OFF 또는 ON 상태이다. 이 상태는 사용자가 설정하거나 `setState:`를 사용하여 설정할 수 있다. 특정 유형의 버튼(특히, Currency Converter의 Convert 버튼같은 표준 버튼)의 경우, 액션 메시지를 타겟 객체로 전송한다. 스위치 및 무선 버튼같은 토글 유형 버튼은 상태를 반영한다. 응용 프로그램은 `state` 메시지를 통해 상태를 파악한다. 아이콘과 타이틀을 버튼의 OFF 및 ON 상태와 연결하고, 각각 연결된 타이틀과 아이콘을 배치하여 독자적인 버튼을 만들 수 있다.

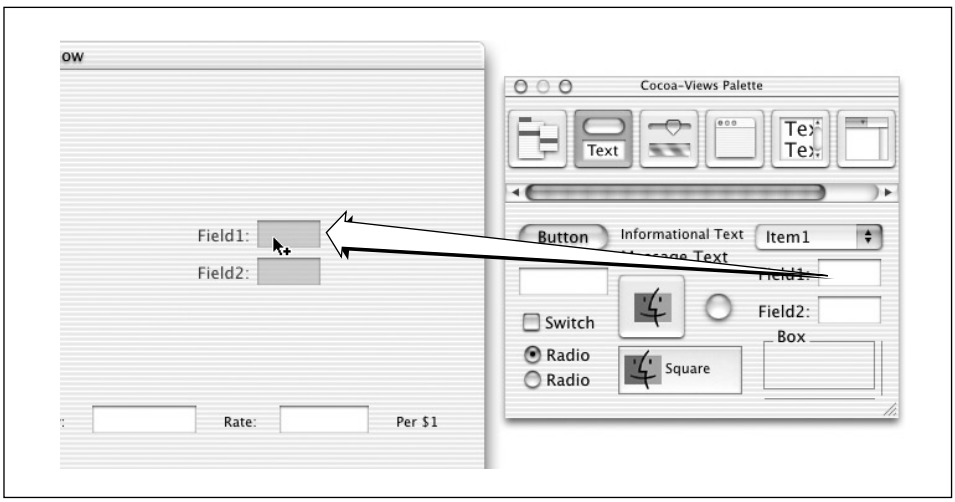
## 양식 객체를 인터페이스에 추가하기

국가에 관한 논리적 정보를 제공하는 양식 객체를 추가한다.

1. <그림 10-8>에서 보여진 바와 같이 Interface Builder의 Views 팔레트에서 양식 객체를 드래그한다.
2. 크기조정 핸들을 옆으로 드래그하여 양식 필드 크기를 크게 한다. Travel Advisor UI의 Other 섹션에서 Languages 필드와 같은 넓이의 필드를 만든다.

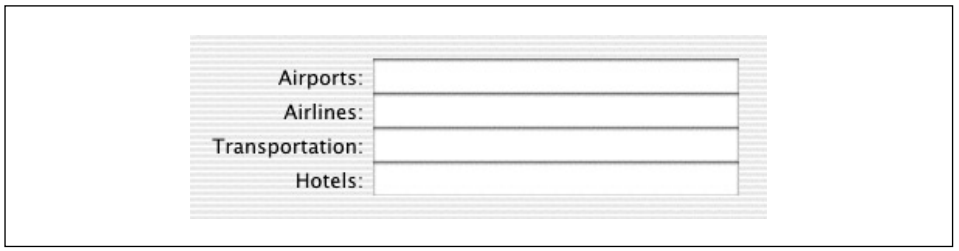



<그림 10-7> Info 윈도우의 체크박스 속성




<그림 10-8> 양식 객체를 추가하기

3. 중간 하단의 크기조정 핸들을 아래쪽으로 Option 키를 누른채 드래그하여 2개 이상의 양식 필드를 만든다.
4. 중간 하단의 크기조정 핸들을 위쪽으로 Command 키를 누른채 드래그하여 텍스트 필드사이의 공간을 제거한다.
5. 다음과 같이 필드 라벨의 이름을 작성한다.(팁: Tab 키를 사용하여 필드사이에서 움직여본다)
6. <그림 10-9>와 똑같이 라벨을 우측 정렬한다.

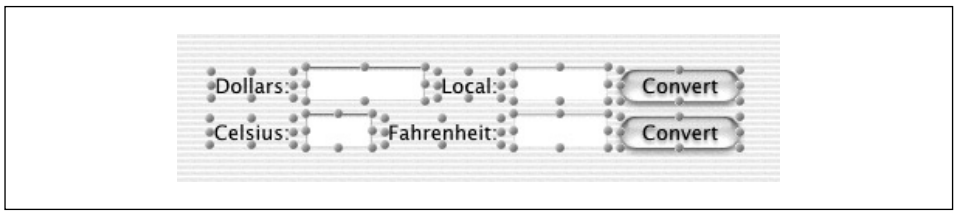



*<그림 10-9> 양식 객체 마무리하기*

## 인터페이스에서 객체를 그룹화하기

선택된 객체 그룹인 Travel Advisor 인터페이스에서 필드, 양식, 버튼의 타이틀 섹션을 만들려면, 객체를 그룹화하여 상자에 넣는다.

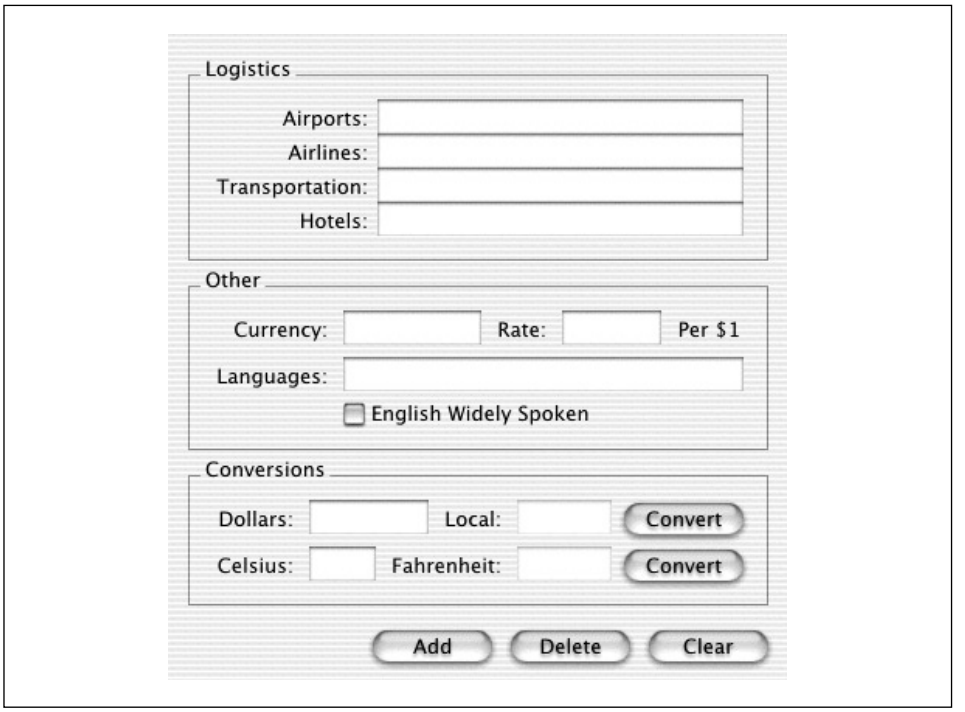
1. 2개의 Convert 버튼을 Dollars, Local, Celsius, Fahrenheit 같은 라벨과 텍스트 필드와 함께 선택한다. 그룹으로 객체를 선택하려면, 객체 주위의 선택 사각형을 드래그하거나 각 객체를 시프트키를 누른 채 클릭한다.(선택 사각형을 만드려면, 윈도우의 빈 지점으로부터 드래그한다) 모든 객체를 선택하면, <그림 10-10>처럼 나타난다.




*<그림 10-10> 그룹화를 위한 객체 선택하기*

2. Layout → Group In → Box (Command-G)를 선택한다. 타이틀 상자가 객체를 둘러싼다.
3. 타이틀을 선택하려면 더블 클릭한다.

4. 타이틀의 이름을 **Conversions**라 지정한다.
5. <그림 10-11>에서와 같이 Logistics 및 Other 그룹을 위해 상기 단계를 반복한다.




*<그림 10-11> 그룹화된 객체*

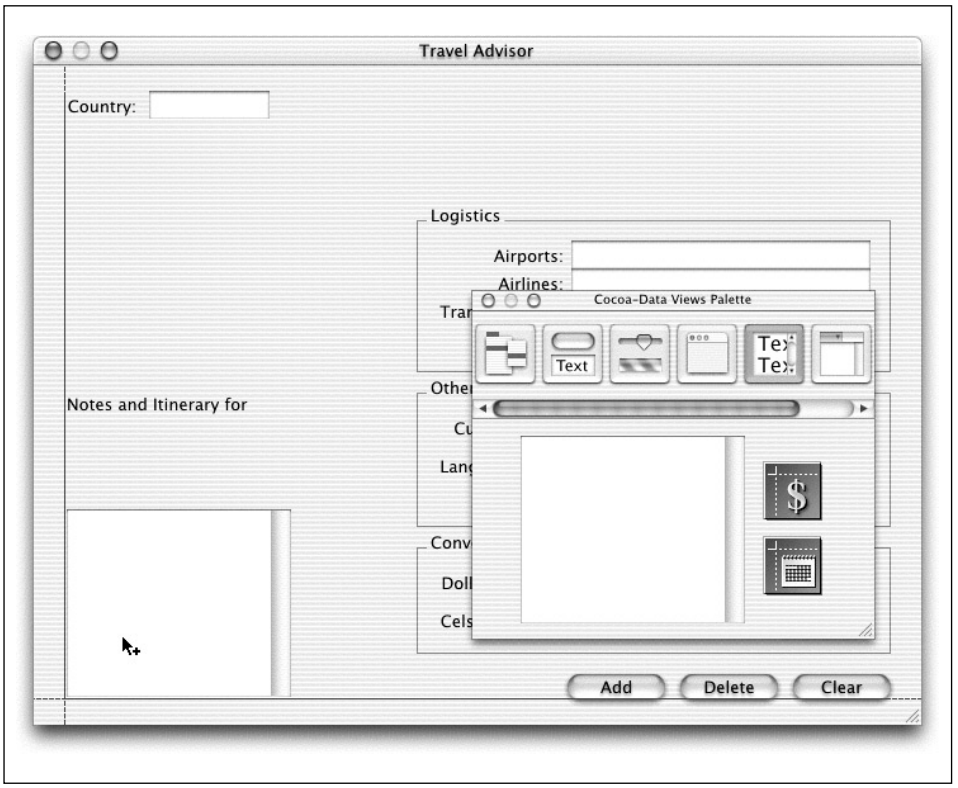
상자는 인터페이스 이름 섹션을 구성할 수 있는 유용한 방안이다. Interface Builder에서 상자를 사용하여 이동, 베끼기, 붙이기 및 작업을 수행할 수 있다. Travel Advisor의 경우에는 기본적인 상자 속성을 변경할 필요가 없지만, 원한다면 다른 상자를 선택할 수 있다.

NSBox 인스턴스는 모든 그룹화된 객체의 서브뷰이다(뷰는 윈도우에서 볼 수 있는 객체이다). 제8장, *이벤트 처리*에서 설명했듯이, 슈퍼뷰는 서브뷰를 둘러싸고 있으며, 서브뷰가 사용자 액션을 처리하지 못할 경우, 이에 응답한다.

## *Text View 추가하기*

DataViews 팔레트의 텍스트 뷰는 스크롤 뷰(NSScrollView 인스턴스)에 둘러싸인 텍스트 객체(NSTextView 인스턴스)로 구성된다. 사용자는 이 객체를 통해 프로그램에 거의 개입하지 않고, 임의 길이 텍스트를 통해 입력, 편집, 포맷 및 스크롤 할 수 있다.

1. <그림 10-12>에서 보여진 바와 같이 DataViews 팔레트에서 텍스트 뷰를 드래그하고, 윈도우의 좌측 하단에 드롭한다.




*<그림 10-12> 텍스트 뷰 객체 추가하기*

2. 텍스트 뷰가 Travel Advisor 윈도우의 좌측 하단을 점유할 수 있도록 크기 조정한다.
3. Notes and Itinerary For 라벨의 우측을 텍스트 뷰와 정렬시키기 위해 라벨 크기를 조정한다. 이로써, 국가명을 삽입하기 위해 여분의 공간을 남길 수 있다.

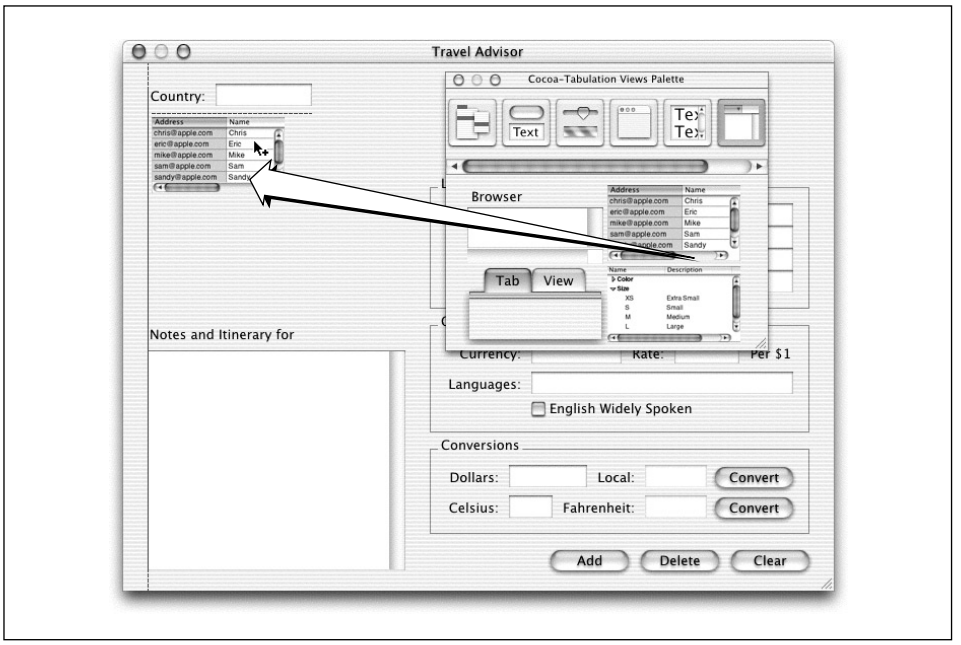
텍스트 뷰의 기본적인 속성을 굳이 바꿀 필요는 없다.(그러나 원한다면 설정할 수 있는 속성을 참조할 수 있다)

## *테이블 뷰를 추가 및 구성하기*

제9장에서 설명했듯이, 테이블 뷰는 테이블 데이터를 표현하고, 편집하기 위해 사용되는 객체이다. 데이터는 관련된 레코드로 구성되며, 독립적인 레코드는 가로행에, 레코드의 일반적인 필드(속성)는 세로열에 배치한다. 테이블 뷰는 데이터베이스 컴포넌트가 있는 응용 프로그램에 이상적이다.

이 섹션에서는 이용 가능한 국가 리스트를 보여주기 위해 테이블 뷰를 구성하는 방법을 설명한다.

1. Tabulation Views 팔레트에서 테이블 뷰 객체를 드래그한다.




*<그림 10-13> 테이블 뷰 객체 추가하기*

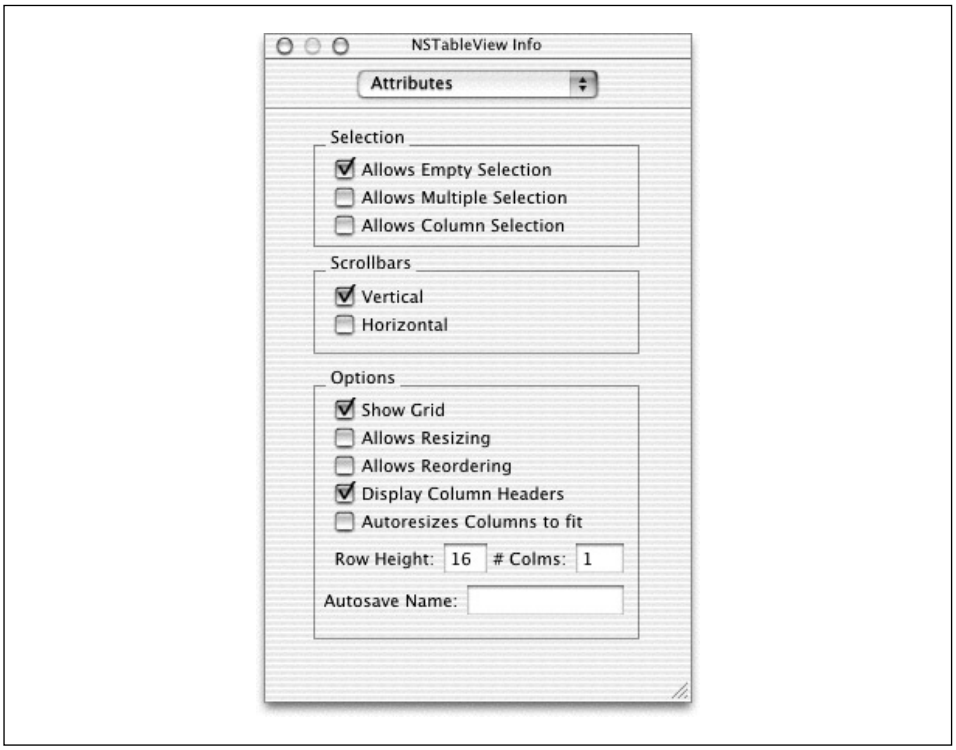
2. 테이블 뷰의 크기를 조정하여 Country 필드와 Notes 라벨간의 영역을 채운다.
3. 첫 번째 세로열 타이틀은 Countries로 설정한다. 세로열 헤더를 더블 클릭하고, 커서를 삽입한다. **`Countries`**를 입력하고, 세로열 바깥의 아무 곳이나 클릭하거나 Return을 누른다.
4. 테이블에 단 1개의 세로열을 만든다. 먼저, 세로열을 선택(세로열 헤더를 클릭)하여, Delete를 누른 뒤 불필요한 세로열을 삭제한다. 그 다음, Countries 세로열의 우측 경계면에 커서를 놓고, 커서가 화살표로 변경되기를 기다린다. 그리고, 뷰의 우측 경계면과 수평이 될 수 있도록 세로열 경계면을 클릭하여 드래그한다. 우측으로 너무 이동하면, 스크롤 바가 뷰 하단에 나타난다.

이 같은 일이 발생하면, 세로열 경계면을 좌측으로 옮긴 다음 다시 시도한다. 이 작업을 완료하면, 테이블 뷰의 바깥면을 클릭하여,세로열에서 빠져나온다.

5. Country 텍스트 필드의 우측 경계면이 테이블과 정렬할 수 있도록 크기를 조정한다.

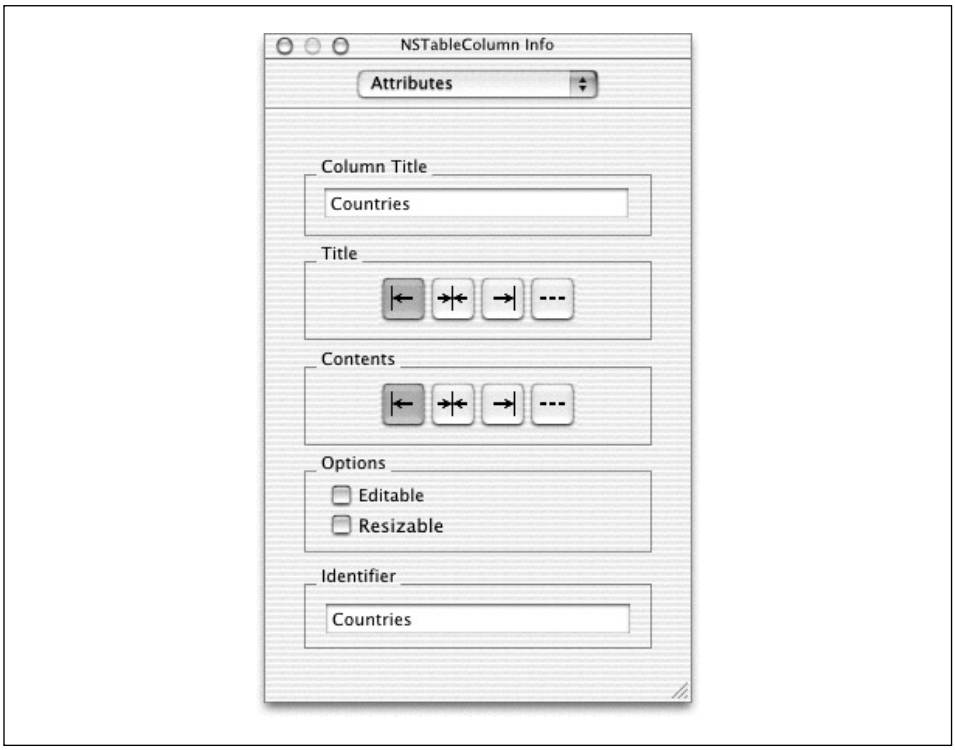
테이블 뷰를 구성하려면, NSTableView 및 NSTableColumn와 같은 2개의 컴포넌트 객체 속성을 설정해야 한다.

1. 테이블 뷰를 선택한다.
2. <그림 10-14>에서 보여진 바와 같이 NSTableView 속성을 설정한다. 이 테이블은 1개의 세로열 뷰로 구성되어 있고, 국가 명의 길이는 제한되어 있기 때문에 한번에 더 많은 국가들을 볼려면, 수평 스크롤바를 이용한다. Show Grid를 선택할지 여부는 개인적인 환경설정의 문제이지만, Allow Resizing과 Allow Reordering은 선택하지 않는다. 사용자가 테이블 내용에 직접 손대면 안된다.




*<그림 10-14> Info 윈도우의 테이블 뷰 속성*

3. 테이블 내부를 더블 클릭하고, Countries 라벨을 선택한다.
4. <그림 10-15>에서와 같이 NSTableColumn 속성을 설정한다.




<그림 10-15> Info 윈도우의 테이블 세로열 속성

5. Identifier 필드에서 세로열을 식별하기 위해 이름을 입력한다. Travel Advisor의 이름은 세로열 제목과 동일하게 한다.

## 이미지를 인터페이스에 추가하기

이미지를 사용하여 상당히 훌륭한 비쥬얼 효과를 인터페이스에 추가할 수 있다. 버튼은 가끔 이미지를 제공하기 위해 선호되는 객체이다.(다양한 버튼 상태를 나타내기 위해 각기 다른 이미지를 원할 경우) 하지만, 버튼이 제 기능을 하지 못하면 버튼이 보여주는 이미지는 흐릿해 보인다. 장식적인 이미지를 나타내려면, 버튼보다는 이미지 뷰(NSImageView)를 사용한다.

Cocoa에서 선호되는 이미지 파일 포맷은 Tag Image File Format(TIFF)이다. NSImage 클래스가 지원하는 어떤 포맷(다음을 포함한)도 사용할 수 있다.

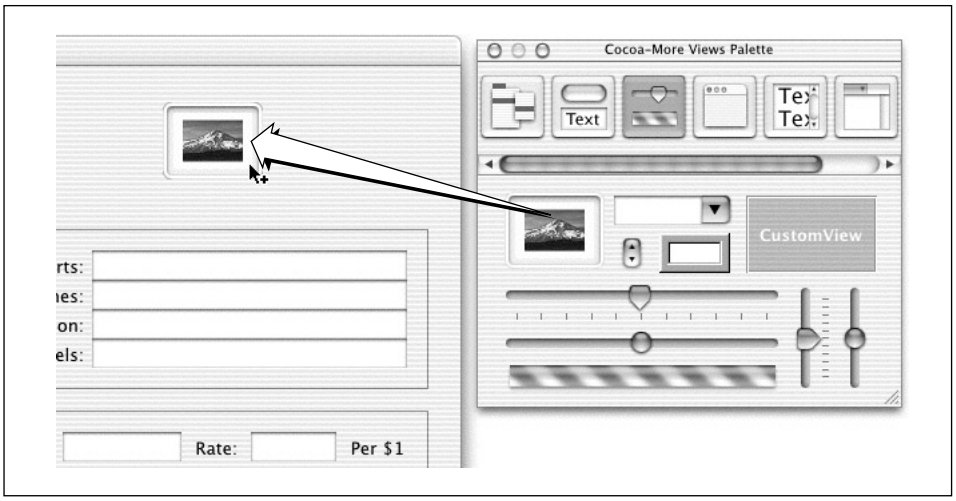
- Portable Document Format(PDF)
- Apple의 PICT 포맷

- 윈도우 Bitmap 포맷(BMP)의 비트맵 데이터
- 태그가 붙지 않은(raw) 비트맵 데이터
- NSImage 클래스에 등록된 NSImageRep 서브클래스가 지원하는 이미지 데이터
- 사용자 설치 필터 서비스가 지원하는 필터링될 수 있는 데이터

추가적인 내용을 보려면 NSImage와 NSImageRep의 도큐멘트를 참조한다. Cocoa는 QuickTime이 지원하는 이미지 포맷을 수용하지만, QuickTime을 사용하면 추가 메모리와 성능 오버헤드를 야기한다. 따라서, 응용 프로그램의 사용자 인터페이스에 있는 정적 이미지를 위해 권장할만 하지 않다.

Interface Builder는 이미지 파일을 간편하게 관리하기 위해 Project Builder 프로젝트에 포함된 이미지를 참조하고, 사용할 수 있다.

1. Project Builder의 Project 메뉴에서 New Group을 선택한다.
2. 그룹을 **`Images`**라 정하고, Resources 그룹에 배치한다.
3. Project 메뉴에서 Add File을 선택하고, 샘플 파일에 포함된 `Airplane.tiff`를 추가한다.
4. <그림 10-16>에서 보여진 바와 같이 Interface Builder의 More Views 팔레트에서 윈도우로 이미지 뷰를 드래그한다.



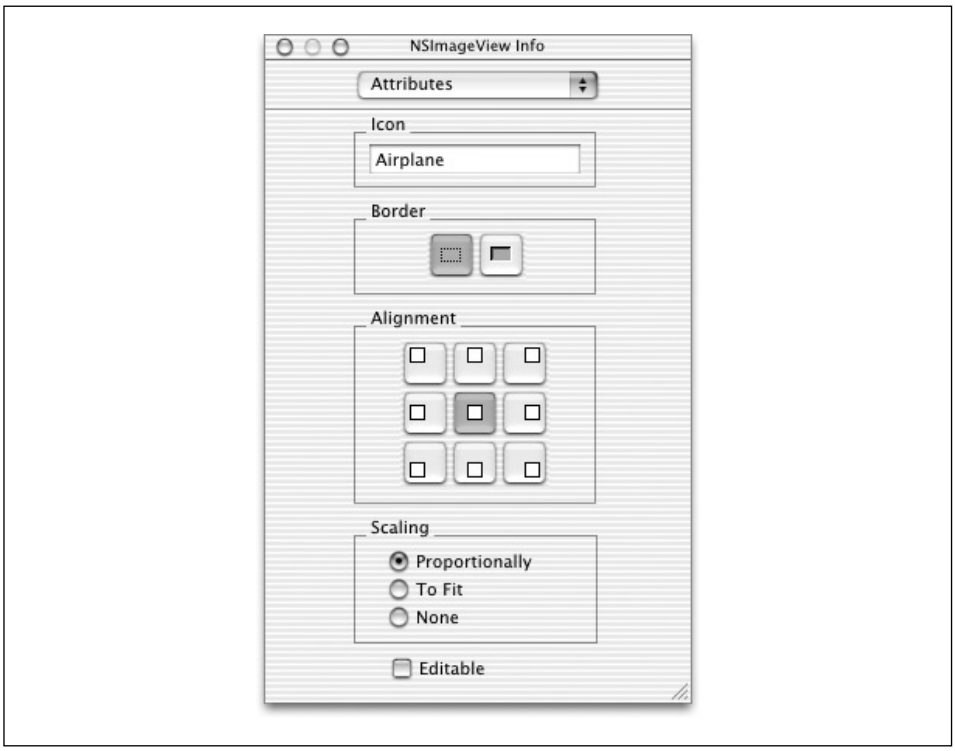

*<그림 10-16> 이미지 뷰 객체를 추가하기*

5. `MainMenu.nib` 윈도우에서 Images 탭을 클릭한다.
6. 이미지 프록시를 이미지 뷰로 드래그한다.

Interface Builder에서 직접 이미지를 추가할 수 있다. 버튼이나 이미지 뷰로 이미지를 드롭하면, Interface Builder는 이미지 파일을 프로젝트(현재, 프로젝트가 Project Builder에서 실행된다면)에 추가하고, nib 파일에서 이를 참조하기 위해 저장한다.

이제, 이미지 속성을 구성한다.

1. <그림 10-17>에서 보여진 바와 같이, Interface Builder에서 이미지 뷰의 Attributes Info 윈도우를 불러와, 속성을 설정한다.



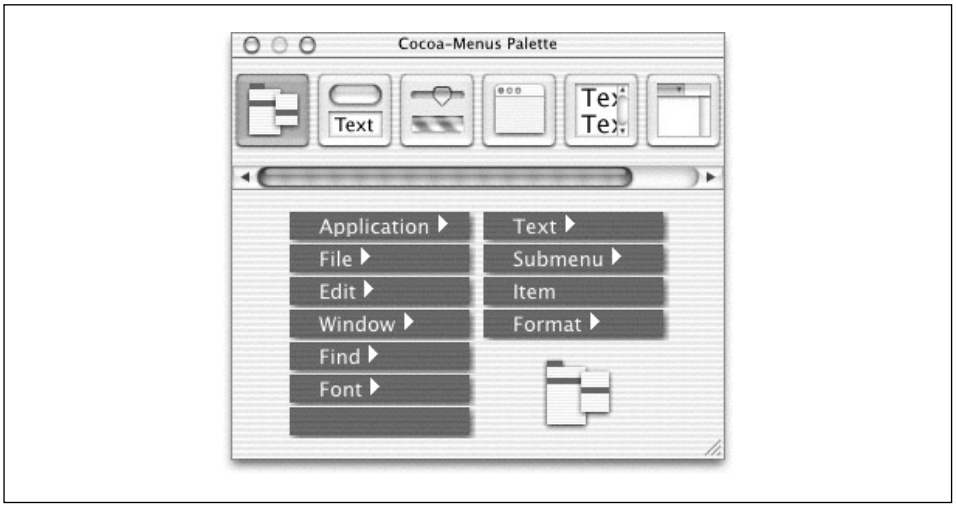

*<그림 10-17> Info 윈도우의 이미지 뷰 속성*

2. 이미지 뷰(및 둘러싸인 이미지)를 타이틀바와 Logistics 그룹 사이에 충분히 들어갈 만큼 작게 만든다.
3. Airplane 뒤에 "velocity" 라인을 추가한다.(Tip: 수평 구분선을 사용한다)

## *메뉴 및 메뉴 아이템 추가하기*

Travel Advisor 메뉴는 기본적으로 서브 메뉴와 커맨드로 구성된다. 기본적인 설정에 포함되지 않으며, Menus 팔레트에서 찾을 수 없는 서브메뉴와 메뉴 커맨드가 필요할 때가 있다. Submenu 및 Item 셀을 사용하여, 사용자화된 메뉴와 메뉴 아이템을 만든다.

1. Interface Builder에서 Menus 팔레트를 선택한다. 팔레트 윈도우는 <그림 10-18>과 동일해야 한다.

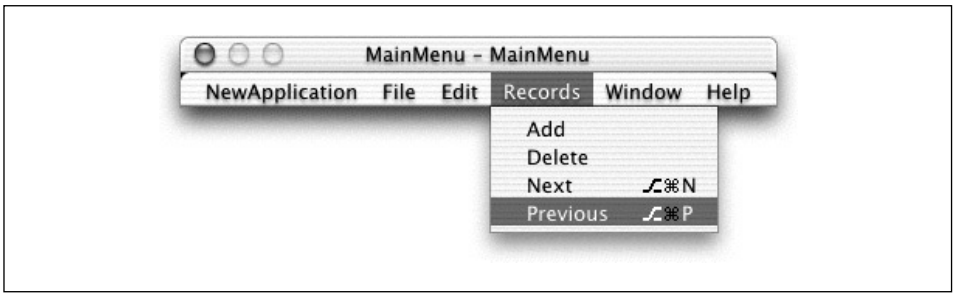



*<그림 10-18> Menus 팔레트*

2. 일반 Submenu 아이템을 드래그하여 Edit 및 Window 서브메뉴 사이에 드롭한다.
3. Submenu를 더블 클릭하여 메뉴 타이틀을 선택한다. 이름을 **Records**로 변경한다.
4. 새로운 Records 메뉴를 클릭하여 Item 커맨드를 노출시킨다.
5. Item을 클릭하고, Command-D를 사용하여 이를 3번 복사한다.(전부 4개를 만든다)
6. 커맨드 이름을 **Add**, **Delete**, **Next** 및 **Previous**로 변경한다.(Tip: Tab 키를 사용하여 메뉴의 엔트리 사이를 이동한다)
7. Next와 Previous 커맨드 우측에 Command-key를 추가한다. 키를 할당하려면, 메뉴 커맨드의 우측 영역을 더블 클릭하고(작은 사각형이 출력된다), 할당하고자 하는 키를 입력한다. Records 메뉴는 <그림 10-19>와 동일해야 한다.
8. File 메뉴에서 Print...를 Print Notes....로 변경한다.

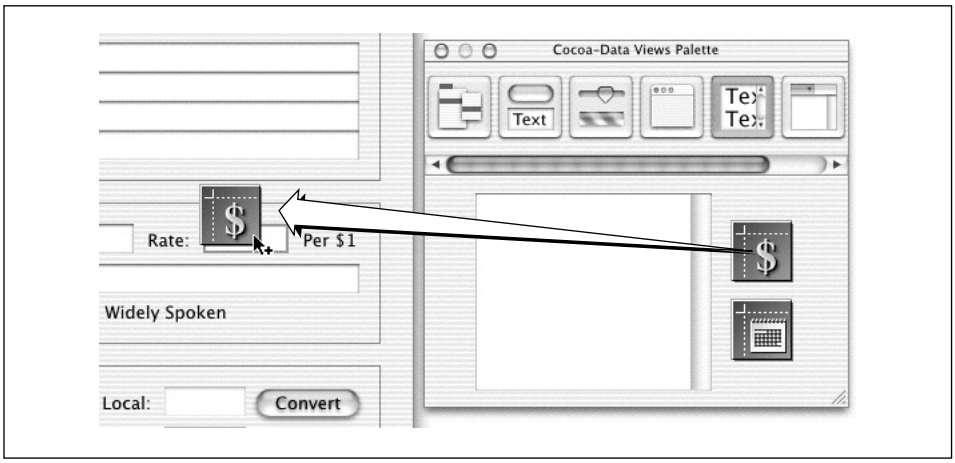
## *포맷터 추가하기*

제6장에서 설명한대로, 포맷터는 특정 객체 값을 특정 온스크린으로 변환시키는 객체이다. 이 섹션에서는 포맷터를 사용자 인터페이스의 텍스트 필드에 추가하여, 현재 값을 정상적으로 나타내는 방법을 설명한다.



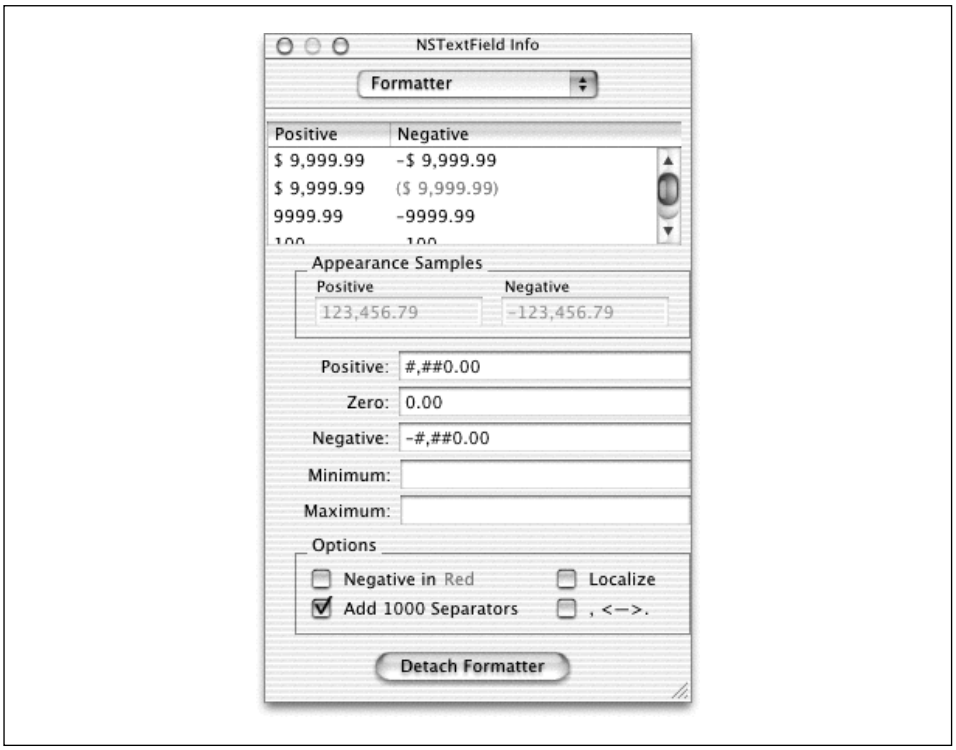

<그림 10-19> 메뉴 아이템을 기록하기

1. Palette 윈도우에서 DataViews 팔레트를 선택한다.
2. <그림 10-20>에서 보여진 바와 같이, 숫자 포맷터 객체를 드래그하여, Rate 필드 위에 드롭한다.




<그림 10-20> 숫자 포맷터 추가하기

3. Rate 필드를 클릭하여, 선택한 뒤 Info 윈도우(Command-Shift-I)를 불러온다.
4. <그림 10-21>에서 보여진 바와 같이, Info 윈도우의 Formatter 화면에서 테이블 뷰의 가로행을 9999.99 포맷으로 선택하여 등급 포맷을 명시한다.
5. Dollars 및 Local 필드를 위해 이 방식을 반복하지만, 적합한 포맷에 적용한다.




*<그림 10-21> Info 윈도우의 텍스트 필드 포맷터*

## 필드간 탭 기능과 출력을 위해 연결하기

Application Kit가 정의한 아웃렛과 액션을 통해 Travel Advisor의 수 많은 객체를 연결할 수 있다. 상기 튜토리얼에서 기억할 점은 소스 객체를 Control 키를 누른채 클릭하고, 목적지 객체로 라인을 드래그하여 연결한 뒤 객체를 Interface Builder에서 연결해야 한다는 것이다.

Cocoa에서 윈도우는 이벤트의 초기 포커스가 되어야 하는 윈도우에서 객체를 위한 `initialFirstResponder` 아웃렛을 제공한다. 텍스트 필드는 사용자가 필드간 탭기능을 사용할 수 있도록 연결된 `nextKeyView` 아웃렛을 지원한다. 양식은 탭 기능을 위한 `nextKeyView` 아웃렛을 제공한다.(필드는 양식내에서 이미 상호 연결되어 있기 때문에 연결할 필요는 없다)

1. nib 파일의 윈도우 아이콘에서 Country 필드로 연결한다.
2. Info 윈도우의 Connections 디스플레이에서 `initialFirstResponder`를 선택하고, Connect를 클릭한다.

3. `nextKeyView` 아웃렛을 통해 상단에서 하단으로 필드와 양식을 연결한다. Country 필드를 Logistics 양식으로 연결하여 구동한다.
4. Languages 필드에서 루프를 생성하면서 Country 필드와 연결한다.
5. Conversions 섹션에서 편집 가능 텍스트 필드를 유사한 루프를 통해 연결한다.

Application Kit는 응용 프로그램에 연결할 수 있는 프리셋 액션을 제공한다. 스크롤 뷰의 NSTextView 객체는 NSView에서 상속한 모든 객체들처럼 내용을 출력할 수 있다. 이 기능을 활용하려면, 메뉴 커맨드를 NSTextView 액션 메소드에 "연결"한다.

1. Print Notes 메뉴 커맨드를 선택하려면 클릭한다. NSMenuItem Info 윈도우의 Connection 영역은 `FirstResponder.print`로 이미 연결이 되어 있다는 것을 보여준다. FirstResponder로 연결을 해제한다.
2. Print Notes 메뉴 커맨드를 스크롤 뷰의 텍스트 객체에 연결한다.
3. Info 윈도우의 Connections 화면에서 `print:` 액션 메소드를 선택한다.
4. Connect 버튼을 클릭한다.

### 인터페이스 시험하기

Travel Advisor 인터페이스가 완성되었다. Interface Builder의 File 메뉴에서 Test Interface(Command-R)를 선택하여, 저장(Command-S)한 뒤 시험한다. 다음 사항을 시험한다.

- Tab 키를 반복적으로 누른다. 커서가 양식 필드사이를 어떻게 점프하는지, Languages 필드에서 Country 필드로 어떻게 루프되는지 확인한다. Shift-Tab을 눌러 커서가 반대 방향으로 가도록 한다.
- 텍스트 일부를 텍스트 뷰에 입력한다. 그리고, Print Notes 메뉴 아이템을 클릭한다. 그러면, Print 대화 상자가 나타난다. 텍스트 객체의 내용을 출력한다.
- 텍스트 뷰에서, 스크롤박스가 스크롤바에 나타날 때까지 반복해서 Return 키를 누른다.

## Travel Advisor 클래스 정의하기

Travel Advisor는 Country, Converter 및 TAController를 제공한다. TAController만이 아웃렛과 액션을 지원한다. Converter 클래스 경우, 새롭게 정의하기보다는 Currency Converter 프로젝트로부터 프로젝트에 추가하여 재사용한다.

## Country 및 TAController 클래스 지정하기

Country 및 TAController 클래스를 생성하려면, NSObject를 서브클래스로 만든다.

1. Interface Builder에서 nib 파일 윈도우의 클래스 화면을 불러온다.
2. NSObject를 슈퍼클래스로 선택한다.
3. 클래스 메뉴에서 Subclass를 선택한다.
4. MyObject 공간에 **Country**를 입력한다.
5. TAController의 경우도 이를 반복한다.

## TAController 아웃렛 및 액션 지정하기

인터페이스를 배치하였으므로, TAController 이 UI 객체와 통신할 수 있도록 아웃렛을 정의할 수 있다. 여기서, Currency Converter 프로젝트로부터 재사용할 아웃렛을 Converter 객체에 추가할 수 있다.

1. TAController를 선택하고, 클래스 메뉴에서 Add Outlet을 선택한다.
2. 다음 아웃렛을 추가한다.

```
celsiusField
commentsField
commentsLabel
converter
countryField
countryTableView
currencyDollarsField
currencyLocalField
currencyNameField
currencyRateField
englishSpokenSwitch
fahrenheitField
languagesField
logisticsForm
```

UI 객체가 TAController 메시지를 전송할 수 있도록 아웃렛뿐만 아니라 액션도 지정해야 한다.

1. TAController를 선택하고, 클래스 메뉴에서 Add Action을 선택한다.
2. 다음 액션 메소드를 정의한다.

```
addRecord:
blankFields:
convertCurrency:
convertTemp:
deleteRecord:
handleTVClick:
```

```
nextRecord:
prevRecord:
switchClicked:
```

## Converter Class 재사용하기

Cocoa에서 객체를 재사용하는 방법은 여러가지가 있다. 예를 들어, 약간 다른 동작을 얻기 위해 기존 클래스를 서브클래스화하는 작업은 슈퍼클래스를 재사용하는 한 방법이며, 또 다른 방법으로는 Converter 클래스 같은 기존 클래스를 프로젝트에 통합하는 것이다.

1. Project Builder의 프로젝트 브라우저에서 클래스 그룹을 선택한다.
2. Project 메뉴에서 Add Files을 선택하여, Currency Converter 프로젝트 디렉토리를 검색한다. `Converter.m` 및 `Converter.h`를 선택한다. Open을 클릭한다.
3. Travel Advisor에 파일을 추가하려면, Add를 클릭하기 전에 Copy 체크박스를 점검하고, Travel Advisor를 선택하였는지 확인한다.
4. Travel Advisor의 `MainMenu.nib` 파일을 연다.
5. Project Builder에서 Interface Builder의 `MainMenu.nib` 윈도우로 `Converter.h` 파일을 드래그한다. Interface Builder는 슈퍼클래스 및 선언된 모든 아웃렛과 액션을 찾아서 헤더 파일을 파싱한다. nib 파일의 Classes 패널은 Converter를 NSObject의 서브클래스로 리스트화한다.

이 과정을 완료하면, Converter 클래스는 Travel Advisor 프로젝트와 Travel Advisor의 메인 nib 파일에 복사된다.

## TAController 및 Converter Instance 생성하기

Country 클래스는 아웃렛이나 액션이 연결되지 않기 때문에 nib 파일에서 Country 클래스의 인스턴스를 만들 필요는 없다. 그러나, 다른 객체에 연결할 경우에는 TAController 인스턴스를 반드시 만들어야 한다. TAController는 사용자가 응용 프로그램의 인터페이스를 조작하고, Country 객체에서 들어오고 나가는 데이터를 조정할 때 사용자와 상호작용을 한다. 따라서, TAController는 인터페이스 객체에 액세스해야 하고, 액션 메시지의 타겟을 만들어야 한다. 또한, Converter 객체에 연결되어야 한다. 그렇게 해야만 Converter의 인스턴스가 만들어진다.

## TAController Instance 연결하기

TAController 아웃렛은 인터페이스 객체 및 Converter 객체와 통신할 수 있도록 이들과 연결되어야 한다. 여기서, 기억할 점은 메시지가 흐르는 방향대로 연결(Control 키를 누른채 드래그)이 이루어져야 한다는 것이다. 예를 들어, TAController는 텍스트를 드로잉하기 위해 메시지를 Celsius 텍스트 필드로 전송한다. 따라서, 이들 객체를 연결하려면 TAController 인스턴스에서 Control 키를 누른채 Celsius 텍스트 필드로 드래그하여 연결한다.

1. TAController를 아래 표에 수록된 아웃렛으로 연결한다. 몇 개의 아웃렛을 연결하고 나면, 이후 새로운 아웃렛을 연결하려고 할 때, Interface Builder가 개발자의 마음을 읽고 있는 듯 Connections Info 윈도우에서 해당 아웃렛을 미리 선택한다. 이는 테이블 리스트가 알파벳 순서대로 진행되기 때문에 발생하는 일일 뿐이며, 이로 인해 튜토리얼이 훨씬 더 신속하게 운용될 수 있다. 순서를 다르게 하여 TAController 인스턴스를 인터페이스 객체에 연결하면, 연결할 해당 아웃렛을 선택할 때 Info 윈도우의 리스트를 일일히 검색해야 하는 번거로움이 있다.

| 아웃렛 | 연결 대상 |
|---|---|
| celsiusField | Celsius 라벨이 붙은 텍스트 필드 |
| commentsField | 스크롤 뷰 내 텍스트 객체 |
| commentsLabel | Notes 및 Itinerary for 라벨 |
| converter | Converter 클래스의 인스턴스 (Instances 디스플레이의 큐브) |
| countryField | Country 라벨이 붙은 텍스트 필드 |
| countryTableView | Countries 세로열 아래 영역 |
| currencyDollarsField | Dollars 라벨이 붙은 텍스트 필드 |
| currencyLocalField | Local 라벨이 붙은 텍스트 필드 |
| currencyNameField | Currency 라벨이 붙은 텍스트 필드 |
| currencyRateField | Rate 라벨이 붙은 텍스트 필드 |
| englishSpokenSwitch | English Widely Spoken 라벨이 붙은 스위치(버튼) |
| fahrenheitField | Fahrenheit 라벨이 붙은 텍스트 필드 |
| languagesField | Languages 라벨이 붙은 텍스트 필드 |
| logisticsForm | Logistics 라벨이 붙은 Group (box); 4개의 필드가 회색 라인으로 경계선을 이루고 있으면 양식을 선택한다. |

2. 아래 표와 같이 액션을 통해 TAController 인스턴스를 인터페이스의 컨트롤 객체로 연결한다. 여기서 중요한 점은 인터페이스 객체에서 TAController 인스턴스로 드래그해야 한다는 것이다.

| 액션(Action) | 연결 대상 |
|---|---|
| addRecord: | 버튼 추가(그리고, Records 메뉴에서 Add 아이템을 추가) |
| blankFields: | 버튼 제거 |
| convertCurrency: | 버튼을 Local 필드 우측으로 전환 |
| convertTemp: | 버튼을 Fahrenheit 필드 우측으로 전환 |
| deleteRecord: | 버튼 메뉴(및 Records에서 Delete 아이템) 삭제 |
| handleTVClick: | 테이블 뷰(Countries 세로열 아래 영역을 더블 클릭하여 선택함) |
| nextRecord: | Records 메뉴의 Next Record 아이템 |
| prevRecord: | Records 메뉴의 Previous Record 아이템 |
| switchClicked: | English Widely Spoken 스위치 |

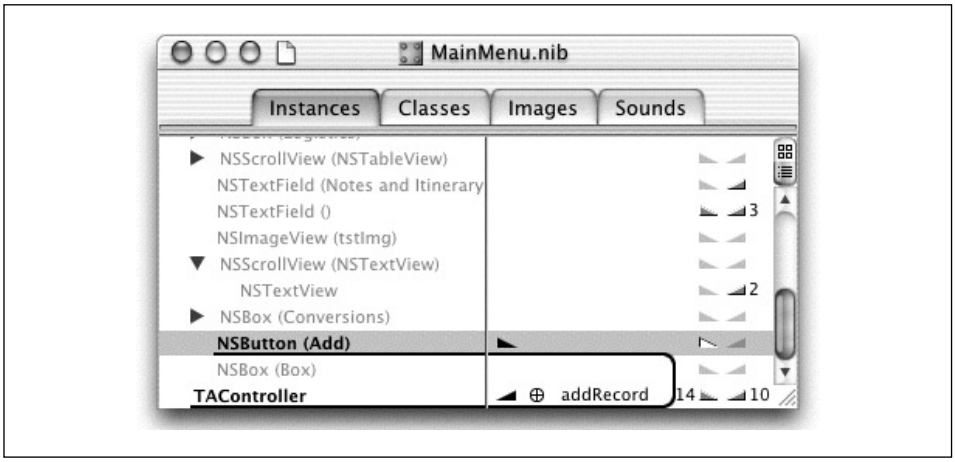
## Outline Mode에서 View 연결

Interface Builder의 nib 파일 윈도우는 nib 파일에서 객체를 볼 수 있는 모드와 이들 객체를 연결할 수 있는 모드를 제공한다. 지금까지는 윈도우와 커스텀 객체를 제공하는 Instances 화면의 아이콘 모드에서 작업을 수행했다. Instances 화면은 윈도우와 커스텀 객체들을 그림처럼 생생하게 표현한다.

*Outline* 모드는 계층적 리스트에서 Outline 객체를 나타낸다. Outline 모드의 장점은 객체간의 연결을 그래픽으로 나타낼 뿐만 아니라 모든 객체를 표현할 수 있다는 점이다.

`MainMenu.nib` 윈도우의 Instances 뷰에서 Outline 모드를 입력할 수 있다. 뷰의 수직 스크롤바 위에 있는 Outline 뷰 아이콘을 클릭하면 된다.

<그림 10-22>에서 보여진 바와 같이, Outline 뷰는 메인 Travel Advisor 윈도우와 TAController 인스턴스의 Add 버튼의 연결을 보여준다. 연결은 아이콘(액션은 크로스헤어, 아웃렛은 전기 아웃렛)과 텍스트 라벨을 비롯해 두 객체를 링크한 검은 라인으로 식별한다. Outline 뷰에서 우측 삼각형은 객체로부터의 연결을 나타내고, 좌측 삼각형은 객체에 연결을 나타낸다. 아이콘 모드에서와 같이 연결 라인을 Control 키를 누른채 Outline 모드에서 아웃렛과 액션을 통해 객체를 연결할 수 있다.




*<그림 10-22> Nib File Outline 뷰*

## File's Owner에 관하여

모든 nib 파일은 nib 파일 윈도우에서 File's Owner 아이콘이 제공하는 1개의 소유자를 갖는다. 소유자는 nib 파일에서 저장되지 않은 객체와 응용 프로그램의 다른 객체 사이에서 메시지를 연결하는 nib 파일 외부에 있는 객체이다.

NSBundle의 `loadNibNamed:owner:`라는 두 번째 인수에서 파일의 소유자를 명시할 수 있다. Interface Builder에서 File's Owner 아이콘은 그 소유자의 "프록시" 객체이다. Owner를 Interface Builder의 객체에 할당했다하더라도 파일의 진짜 소유자에 관한 정보를 반드시 보장하는 것은 아니다.

메인 nib 파일에서 File's Owner는 전역 NSApplication 상수인 NSApp를 나타낸다. 메인 nib 파일은 응용 프로그램 프로젝트를 생성하면 자동으로 만들어진다. 그리고, 응용 프로그램을 구동하면, 로딩된다.

메인 nib 파일이외의 보조 nib 파일은 응용 프로그램이 필요로 할 때 로딩할 수 있는 객체 및 자원(예를 들면, Info 패널)을 제공한다. 보조 nib 파일의 소유자를 명시해야 한다.

nib 파일의 File's Owner 아이콘을 클릭하거나 Custom Class Info 윈도우를 열어 윈도우의 Interface Builder에서 현재 nib 파일의 소유자 클래스를 결정 또는 변경할 수 있다. 제11장, *Cocoa의 멀티플 도큐멘트 아키텍처*에서 멀티도큐멘트 응용 프로그램 생성 방법을 실행하려면 이 기법을 사용해야 한다.

## 델리게이트 아웃렛 연결하기

File's Owner 프록시를 사용하여 TAController를 NSApp 객체의 델리게이트로 만드는 작업을 진행한다. TAController는 NSApp(NSApplication 객체)의 델리게이트로서 특정 이벤트가 발생했을 때 메시지를 수신한다.

여러 메시지 중에서, NSApp은 응용 프로그램이 종료될려는 시점을 통보하기 위해 메시지를 델리게이트로 전송한다. 이 메시지를 수신하면, Country 객체가 포함된 Dictionary를 아카이빙(저장)할 수 있도록 TAController를 구현한다.

1. File's Owner에서 TAController 객체로 드래그하여 라인을 연결한다. 방향은 File's Owner(응용 프로그램 객체)에서 TAController 객체로 연결해야 한다.
2. Info 윈도우의 Connections 화면에서 `delegate`를 선택하고, Connect를 클릭한다.

## Source Code File 만들기

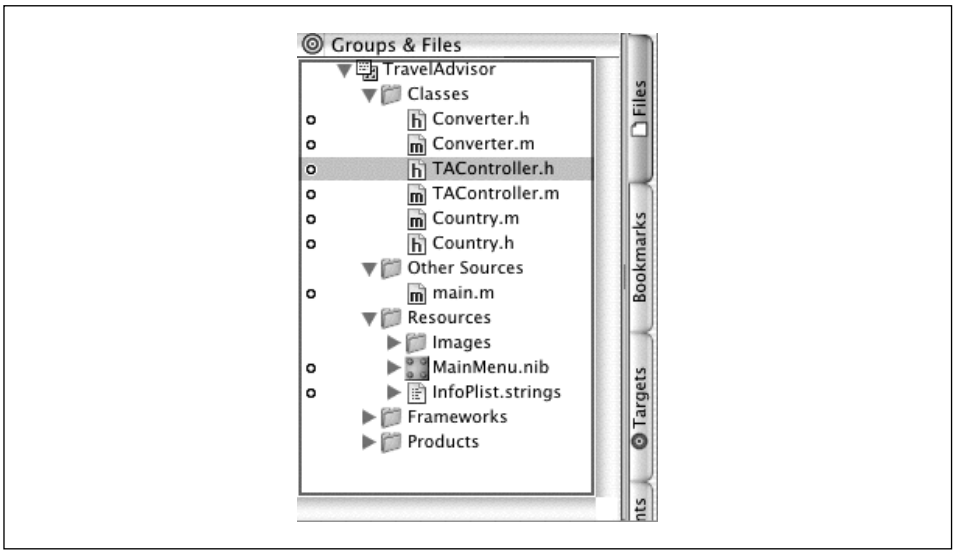
Travel Advisor 클래스의 헤더 및 구현 파일을 만들려면, Interface Builder로 작업을 완료한다.

1. `MainMenu.nib`을 저장한다.
2. nib 파일 윈도우의 Classes화면에서 TAController 클래스를 선택한다.

3. Classes 메뉴에서 Create Files을 선택한다.
4. Travel Advisor 프로젝트 디렉토리를 선택한다.
5. Country 클래스의 경우도 이를 반복한다.

Currency Converter 프로젝트에서 파일을 이미 임포트했기 때문에 Converter 클래스를 위해서는 이 작업을 수행할 필요가 없다.

파일을 만들고 나면, Project Builder로 전환하고, 새롭게 추가된 파일이 <그림 10-23>에서 보이는 바와 같이 클래스 그룹에 들어 있는지 확인한다. 이 작업은 Project Builder가 파일을 처리하는 방식에 영향을 주지 않는다. 그러나, 프로젝트가 일관적으로 정리될 수 있도록 지원한다.




*<그림 10-23> Project Builder Groups & Files*

# *Travel Advisor 클래스 구현하기*

Travel Advisor의 사용자 인터페이스를 만들고, Interface Builder에서 객체들을 연결하였기 때문에 응용 프로그램 클래스를 구현할 수 있다.

## *Currency Converter 재사용하기*

Interface Builder를 사용하여, Convert 버튼을 클릭하면, TAController의 `convertCurrency:` 메소드가 발생할 수 있도록 액션을 설정한다.

이제, 수행할 작업은 코드를 통해 사용자로부터 계산 값을 확보한 뒤, 이들 값을 사용하여 Converter 객체를 발생시키고, 결과를 사용자 인터페이스로 전송한다.

1. Project Builder의 Groups & Files 뷰에서 `TAController.m`를 클릭하여 연다.
2. 파일의 상단에서 다음 라인을 추가한다. 이로써, TAController가 Converter 클래스 메소드에 액세스할 수 있다.

```
#import "Converter.h"
```

3. 이제, `convertCurrency:` 메소드의 비어있는 선언을 변경한다. 이 작업이 Objective-C에서 사용할 수 있는 가장 컴팩트한 방식은 아니지만, 문제 해결 단계를 분명하게 정의한다. 먼저, 사용자가 입력한 값을 사용자 인터페이스에서 검색한다. 다음, converter 객체의 `convertAmount:atRate:` 메소드를 발생시켜, 전환 작업을 수행할 수 있도록 한다. 마지막으로, `currencyLocalField` 값이 Converter 객체가 돌려보낸 결과를 반영하여 설정된다.

```
- (IBAction)convertCurrency:(id)sender
{
    float rate, dollars, result;

    dollars = [currencyDollarsField floatValue];
    rate = [currencyRateField floatValue];
    result = [converter convertAmount:dollars atRate:rate];

    [currencyLocalField setFloatValue:result];
}
```

## 응용 프로그램 구축 및 시험하기

진행에 앞서, 환율이 예상대로 전환되는지 확인하기 위해 Travel Advisor를 구축한다.

1. Project Builder에서 Build 버튼을 클릭하거나 Command-B를 입력한다.
2. Run 버튼을 클릭하여 Travel Advisor를 구동한다.
3. Rate 및 Dollars 필드에 합당한 값을 입력한다. Dollars 필드 값이 달러 심볼로 자동적으로 포맷되는지 확인한다.
4. Local 필드 옆의 Convert 버튼을 클릭한다.

정상적인 값을 돌려받았는가? 그렇다면, 축하드립니다. 여러분은 유능한 Cocoa 프로그래머가 되었다. 값을 돌려 받지 못하면, Interface Builder의 `MainMenu.nib`을 열어, Convert 버튼을 TAController 인스턴스에 제대로 연결한다. 비정상적인 값을 돌려받으면, TAController 아웃렛이 해당 텍스트 필드로 올바르게 연결되었는지 확인한다. 오류가 발견되면, nib 파일을 저장하고, 응용 프로그램을 재구축한 뒤 다시 한번 시도한다.

## 온도 변환 구현하기

TAController의 convertTemp: 메소드를 구현한다. 필요한 아웃렛(Celsius, Fahrenheit)과 액션(convertTemp:)을 지정하여 연결한다. 그러면, 남는 것은 구현 파일이다. 공식은 다음과 같다.

```
F = (9/5)C + 32
```

단순한 작업을 수행할 때는 새로운 Converter 클래스를 구현할 필요가 없다. 코드를 TAController의 convertTemp: 메소드에 배치하면 된다.

```
- (IBAction)convertTemp:(id)sender
{
    [fahrenheitField setFloatValue:
            (((9.0/5.0) * [celsiusField floatValue]) + 32.0)];
}
```

온도 변환시, 장애가 발생하면 Interface Builder의 아웃렛과 액션을 점검한다.

## Country Class 구현하기

Country 클래스는 데이터를 저장하고, 액세스 메소드를 지원하며, 제공된 국가에 관한 정보를 저장하는 Travel Advisor 객체이다.

### 인스턴스 변수 선언하기

아웃렛이 제공되지 않을 경우, Country 클래스는 Travel Advisor의 사용자 인터페이스 필드에 해당하는 수 많은 인스턴스 변수를 정의한다.

1. Project Builder에서 Country.h를 선택한다.
2. <예제 10-1>에서 선언을 추가한다. 이 인스턴스 변수는 국가를 설명하는 속성 정보를 갖는다. 인스턴스 변수를 선언한 뒤 Country 클래스가 NSCoding 프로토콜을 채택하였다고 선언한다.

*<예제 10-1> Country 인스턴스 변수*

```
@interface Country : NSObject <NSCoding>
{
    NSString *name;
    NSString *airports;
    NSString *airlines;
    NSString *transportation;
    NSString *hotels;
    NSString *languages;
    BOOL     englishSpoken;
    NSString *currencyName;
    float    currencyRate;
    NSString *comments;
}
```

## *Country* 클래스 메소드

이제, Country 클래스가 지원하는 메소드를 선언해야 한다. 이 메소드는 다음과 같은 3개의 카테고리로 나눠진다.

- 객체 초기화 및 해제
- 객체 아카이빙과 언아카이빙
- 접근자 메소드

***메소드 선언하기.*** 인스턴스 변수 섹션과 @end를 마무리하는 괄호사이에 클래스의 메소드 선언을 추가한다. 다음 코드에서 선언을 인스턴스 변수 뒤에 추가한다.

```
/* Initialization and De-allocation
*/
- (id)init;
- (void)dealloc;

/* Archiving and Unarchiving */
- (void)encodeWithCoder:(NSCoder *)coder;
- (id)initWithCoder:(NSCoder *)coder;

/* Accessor Methods */
- (NSString *)name;
- (void)setName:(NSString *)str;

- (NSString *)airports;
- (void)setAirports:(NSString *)str;

- (NSString *)airlines;
- (void)setAirlines:(NSString *)str;

- (NSString *)transportation;
- (void)setTransportation:(NSString *)str;

- (NSString *)hotels;
- (void)setHotels:(NSString *)str;

- (NSString *)languages;
- (void)setLanguages:(NSString *)str;

- (BOOL)englishSpoken;
- (void)setEnglishSpoken:(BOOL)flag;

- (NSString *)currencyName;
- (void)setCurrencyName:(NSString *)str;

- (float)currencyRate;
- (void)setCurrencyRate:(float)val;

- (NSString *)comments;
- (void)setComments:(NSString *)str;
```

## Country 객체의 init 메소드 구현하기

먼저, init 메소드는 상속된 인스턴스 변수를 초기화할 수 있도록 슈퍼(슈퍼클래스)의 init 메소드를 발생시킨다. init 메소드에서는 이 작업을 항상 먼저 수행해야 한다. init 메소드는 NSString 인스턴스 변수를 비어 있는 스트링으로 초기화한다. @""는 Quote로 표시된 텍스트에서 수정 불가능 상수 NSString 객체를 생성하는 컴파일러 지원 구조이다. 상수인 이들 객체는 해제될 수 없다. 그래서, 일반적인 레퍼런스 카운팅 기법을 따르지 않는다. 그 같은 기법을 준수하지 않는다 하더라도 이들 객체는 스트링이 정상적으로 할당된 NSStrings처럼 취급될 수 있도록 Retain 및 Release 메시지를 수신한다.

1. Country.m을 연다.
2. <예제 10-2>에서 init 메소드 구현을 작성한다.

*<예제 10-2> Country의 init 메소드 구현*

```
- (id)init
{
    [super init];

    [self setName:@""];
    [self setAirports:@""];
    [self setAirlines:@""];
    [self setTransportation:@""];
    [self setHotels:@""];
    [self setLanguages:@""];
    [self setCurrencyName:@""];
    [self setComments:@""];

    return self;
}
```

비객체 인스턴스 변수를 널(Null) 값(nil, zero, NULL 등)으로 초기화할 필요는 없다. 왜냐하면, 런타임 시스템이 이 작업을 수행하기 때문이다. 그러나, 다른 시작 값을 취한 인스턴스 변수를 초기화해야 한다. 또한, 객체가 비어있는 것으로 예상될 때 nil로 대체하지 않는다. 반대의 경우도 마찬가지이다. Objective-C 키워드 nil은 zero ID(값)을 가진 널 "객체"를 나타낸다. 비어있는 객체(@""같은)는 진짜 객체로 "실질적인" 내용을 제공하지 않는다. self를 리턴함으로써 객체의 진짜 인스턴스를 리턴한다. 이 지점까지는 인스턴스가 정의되지 않은 것으로 간주한다.

## dealloc 메소드 구현하기

이 메소드에서 만들었거나 복사했거나 보유한 객체를 해제한다.(긴급하게 자동 해제되지 않은 객체들도 해제한다) Country 클래스 경우, 인스턴스 변수로써 묶인 모든 객체를 해제한다.

다른 보유 객체가 있다면, 이들을 해제한다. 동적으로 할당된 객체가 있다면, 이를 제거한다. 이 메소드를 완성하면, Country 객체를 해제한다. **dealloc** 메소드는 소유하고 있는 모든 객체를 해제하기 전에 Country 객체가 슈퍼클래스에 의해 해제되지 않도록 마지막으로 **dealloc**를 **super**로 전송해야 한다.

<예제 10-3>에서 **dealloc** 메소드 구현을 추가한다.

*<예제 10-3> Country dealloc 메소드 구현*

```
- (void)dealloc
{
    [name release];
    [airports release];
    [airlines release];
    [transportation release];
    [hotels release];
    [languages release];
    [currencyName release];
    [comments release];

    [super dealloc];
}
```

## 접근자 메소드 구현하기

"Get" 접근자 메소드(최소한, Travel Advisor 같은 인스턴스 변수가 수정 불가능 객체를 보유할 경우)는 단순히 인스턴스 변수를 리턴한다. 객체 값을 설정한 접근자 메소드의 경우, 먼저 **autorelease**를 현재 인스턴스 변수로 전송하여, 수신된 값을 변수로 복사(또는 보유)한다. 여기서 기억할 점은 **autorelease** 메시지는 유효 객체를 참조할 수 있도록 하며 새로운 이벤트 루프가 끝날 무렵에는 이전에 할당된 객체를 해제해야 한다.

인스턴스 변수가 비객체 값(정수 또는 Float)을 갖는다면, 오토릴리즈나 복사할 필요는 없고, 단지 새로운 값을 할당하면 된다.

객체를 생성하기 위해 객체를 복사하는 대신 보유 메시지를 전송해야 하는 경우가 많다. 그러나, NSStrings 및 Country 객체같은 Value-type 객체의 경우, 복사하는게 낫다.

1. 프로젝트 브라우저에서 **Country.m**을 선택한다.
2. 표준 포맷을 사용하여 클래스의 인스턴스 변수를 확보 및 설정하여 코드를 작성한다.

```
- (NSString *)name
{
    return name;
}

- (void)setName:(NSString *)str
{
```

```
        [name autorelease];
        name = [str copy];
    }
```

## TAController Outlet을 정적으로 유형화하기

Interface Builder는 id로 유형화된 **TAController.h** 파일에서 아웃렛을 선언한다. 동적으로 유형화할 필요가 없다면, 여분의 시간이 소요되긴 하지만 객체를 정적으로 유형화하는 것이 프로그램 실행에 도움이 된다.

1. **TAController.h**를 열고, Converter 클래스를 미리 선언한다. @interface 앞의 파일 상단에 **@class**를 추가한다.

   ```
   @class Converter;
   ```

   컴파일러는 클래스 헤더 파일의 모든 선언을 포함하지 않고서도 단순히 @class 지시문을 통해 Converter 클래스에 관해 알려준다. TAController 구현 파일은 완전한 클래스 정의에 액세스해야 한다. 그래서, 이 장의 앞 부분에서 추가했던 클래스 헤더 파일을 임포트한다.

2. <예제 10-4>에서 보여진 바와 같이, 인스턴스 변수 선언을 변경한다.

*<예제 10-4> TAController 인스턴스 변수*

```
@interface TAController : NSObject
{
    IBOutlet Converter *converter;
    IBOutlet NSTextField *countryField;
    IBOutlet NSTableView *countryTableView;

    IBOutlet NSTextField *commentsLabel;
    IBOutlet NSTextView *commentsField;

    IBOutlet NSTextField *celsiusField;
    IBOutlet NSTextField *fahrenheitField;

    IBOutlet NSTextField *currencyNameField;
    IBOutlet NSTextField *currencyDollarsField;
    IBOutlet NSTextField *currencyLocalField;
    IBOutlet NSTextField *currencyRateField;

    IBOutlet NSTextField *languagesField;
    IBOutlet NSButton *englishSpokenSwitch;

    IBOutlet NSForm *logisticsForm;
}
```

## 새로운 인스턴스 변수를 TAController.h에 추가하기

1. 인스턴스 변수 선언을 추가한다. countryDict 및 countryKeys 변수는 Country 객체를 추적하기 위해 사용된 딕셔네리와 어레이를 식별한다. recordNeedsSaving 플래그는 사용자가 사용자 인터페이스의 필드에서 정보를 변경하는 지 여부를 나타낸다.

```
NSMutableDictionary *countryDict;
NSMutableArray *countryKeys;
BOOL recordNeedsSaving;
```

2. @class 지시문과 @interface 지시문 사이에 enum 선언을 추가한다. 이 선언은 중요하지는 않지만, enum 상수는 Logistics 양식에서 셀을 식별하는 명확하고, 편리한 방법을 제공한다. cellAtIndex:와 같은 메소드는 제로 베이스 인덱싱을 통해 양식에서 편집 가능 셀을 식별한다. 이 선언은 Logistics 양식에서 인간 가독형 목적지를 각 셀에 제공한다.

```
enum LogisticsFormIndices {
    LGAirports=0,
    LGAirlines,
    LGTransportation,
    LGHotels
};
```

## blankFields: 메소드 구현하기

blankFields: 메소드는 비어있는 스트링 객체와 제로를 삽입하여 Travel Advisor 필드에서 나타난 내용은 무엇이든지 제거한다. <예제 10-5>에서 TAController.m로 구현을 추가한다.

*<예제 10-5> blankFields: 메소드 구현*

```
- (void)blankFields:(id)sender
{
    [countryField setStringValue:@""];

    [[logisticsForm cellAtIndex:LGAirports] setStringValue:@""];
    [[logisticsForm cellAtIndex:LGAirlines] setStringValue:@""];
    [[logisticsForm cellAtIndex:LGTransportation] setStringValue:@""];
    [[logisticsForm cellAtIndex:LGHotels] setStringValue:@""];

    [languagesField setStringValue:@""];
    [englishSpokenSwitch setState:NSOffState];

    [currencyNameField setStringValue:@""];
    [currencyRateField setFloatValue:0.0];
    [currencyDollarsField setFloatValue:0.0];
    [currencyLocalField setFloatValue:0.0];

    [celsiusField setFloatValue:0.0];
```

<예제 10-5> blankFields: 메소드 구현 (계속)

```
    [fahrenheitField setFloatValue:0.0];

    [commentsLabel setStringValue:@"Notes and Itinerary for"];
    [commentsField setString:@""];

    [countryField selectText:self];
}
```

countryField는 blankFields:에서 비어있는 스트링에 먼저 설정된다. 그 다음에는 Logistics 양식의 4개 셀이 제거된다. 양식에서 각 셀을 처리하기 위해 enum 상수를 사용하여 cellAtIndex: 메시지가 어떻게 양식 객체로 전송되는지 확인한다.

setState: 메시지는 스위치 버튼의 On과 OFF의 토글 컨트롤에 영향을 준다. YES 인수를 사용하면, 체크마크가 나타난다. NO 인수를 사용하면, 체크마크가 제거된다. setString: 메시지는 NSText 객체의 텍스트를 설정한다.

blankFields:는 사용자 인터페이스의 Clear 버튼에 연결된 액션이기 때문에 메소드를 지금 시험할 수 있다. Travel Advisor를 구축하여, 모든 필드에 값을 입력한 뒤 Clear 버튼을 클릭한다. 그러면, blankFields: 메소드가 정상적으로 운용되는지 확인할 수 있다. 필요하다면, 디버깅을 수행한다.

Country.h에 선언된 메소드를 아직 구현하지 않았기 때문에 프로젝트를 구축할 때 컴파일러 경고를 수신할 수 있다. 시험할 경우에는 이 경고를 무시해도 된다. 이 장의 뒷 부분에서 분실한 메소드를 구현하는 방법에 관해 설명한다.

## *TAController 및 데이터 조정*

TAController는 데이터 소스 및 데이터 디스플레이를 상호교환하는 데이터 조정자로 동작한다. 데이터 조정은 데이터를 어딘가에 저장하고, 추후 데이터를 필드에 배치하면서 사용자 인터페이스 필드에서 데이터를 처리하는 방식이다. TAController는 데이터 조정과 관련된 2개의 Private 메소드를 제공한다. populateFields:는 Travel Advisor 사용자 인터페이스 필드로 Country 인스턴스 데이터를 배치하고, extractFields:는 필드 정보를 사용하여 Country 객체를 업데이트한다.

### *extractFields: 메소드 구현하기*

컨트롤러의 extractFields: 메소드는 응용 프로그램의 사용자 인터페이스 객체에서 값을 찾아 새로운 Country 객체의 인스턴스 변수에 저장한다.

1. 다음 메소드 선언을 TAController.h에 추가한다.

```
- (void)extractFields:(Country *)aRec;
```

2. Country 객체를 참조해야 하기 때문에 Country 클래스를 미리 선언해야 한다.

```
@class Country;
```

3. TAController.m 를 열어, Country.h를 임포트한다.

```
#import "Country.h"
```

인터페이스 파일이 Country 객체를 선언한다 하더라도 구현 파일은 이 메소드를 인식해야 한다.

4. TAController.m에 <예제 10-6>의 extractFields: 메소드를 위한 코드를 입력한다.

*<예제 10-6> extractFields: 메소드 구현*

```
- (void)extractFields:(Country *)aRec
{
    [aRec setName:[countryField stringValue]];

    [aRec setAirports:[[logisticsForm
            cellAtIndex:LGAirports] stringValue]];
    [aRec setAirlines:[[logisticsForm
            cellAtIndex:LGAirlines] stringValue]];
    [aRec setTransportation:[[logisticsForm
            cellAtIndex:LGTransportation] stringValue]];
    [aRec setHotels:[[logisticsForm
            cellAtIndex:LGHotels] stringValue]];

    [aRec setCurrencyName:[currencyNameField stringValue]];
    [aRec setCurrencyRate:[currencyRateField floatValue]];
    [aRec setLanguages:[languagesField stringValue]];
    [aRec setEnglishSpoken:[englishSpokenSwitch state]];

    [aRec setComments:[commentsField string]];
}
```

extractFields:를 구현하였으므로, 이를 시험해본다. TAController의 addRecord: 메소드는 사용자 인터페이스의 Add 버튼에 연결된다. addRecord:는 UI로부터 데이터를 얻는 작업을 수행해야 한다. 그래서, addRecord: 구현에서 extractFields:로 호출을 추가하고, extractFields:가 발생할 때 브레이크포인트를 설정한다.

```
- (IBAction)addRecord:(id)sender
{
    Country *aCountry = [[Country alloc] init];

    [self extractFields:aCountry];
}
```

응용 프로그램을 구축하고, 디버깅한 다음, 코드를 연결하여 UI에 입력한 데이터가 정상적으로 Country 객체로 생성되는지 확인한다. gdb Console 패인 객체에 관한 정보를 출력하기 위해 gdb 커맨드 po를 시험해본다.

## *populateFields:* 메소드 구현

컨트롤러의 **populateFields:** 메소드는 **extractFields:**의 역함수라 할 수 있다. 새로운 Country 객체의 인스턴스 변수에서 값을 취하고, 그 값을 사용자 인터페이스에서 나타낸다.

1. 다음 메소드 선언을 TAController.h에 추가한다.

```
- (void)populateFields:(Country *)aRec;
```

2. TAController.m 파일을 실행시키고, <예제 10-7>의 **populateFields:** 메소드 코드를 입력한다.

*<예제 10-7> populateFields: 메소드 구현*

```
- (void)populateFields:(Country *)aRec
{
    [countryField setStringValue:[aRec name]];

    [[logisticsForm cellAtIndex:LGAirports] setStringValue:
            [aRec airports]];
    [[logisticsForm cellAtIndex:LGAirlines] setStringValue:
            [aRec airlines]];
    [[logisticsForm cellAtIndex:LGTransportation] setStringValue:
            [aRec transportation]];
    [[logisticsForm cellAtIndex:LGHotels] setStringValue:
            [aRec hotels]];

    [currencyNameField setStringValue:[aRec currencyName]];
    [currencyRateField setFloatValue:[aRec currencyRate]];
    [languagesField setStringValue:[aRec languages]];
    [englishSpokenSwitch setState:[aRec englishSpoken]];

    [commentsLabel setStringValue:[NSString stringWithFormat:
            @"Notes and Itinerary for %@", [aRec name]]];
    [commentsField setString:[aRec comments]];

    [countryField selectText:self];
}
```

**populateFields:**는 먼저 Country 필드에서 새로운 국가명을 나타내야 한다. **populateFields:**로 전송된 Country 레코드(**aRec**)의 **name** 인스턴스 변수에서 값을 검색한다. **[aRec name]**가 리턴한 객체는 수신자의 텍스트 컨텐트(이 경우, **countryField** 객체)를 설정하는 **setStringValue:** 메소드의 인수로써 사용된다. 그러면, 사용자 인터페이스 요소의 나머지는 업데이트된다.

마지막으로, 텍스트를 선택하였거나, 선택할 텍스트가 없다면 커서를 필드에 삽입할 수 있도록 `selectText:` 메시지를 Country 필드로 전송된다.

## 테이블 뷰 작업하기

Travel Advisor의 테이블 뷰는 단 1개의 세로열을 제공하며, 여행 정보가 들어있는 응용 프로그램의 국가 리스트를 보여주기 위해 사용된다. 9장에서 테이블 뷰와 데이터 소스를 이미 검토했기 때문에 이 섹션을 이해하기가 좀 더 수월할 것으로 보인다.

### 테이블 뷰의 데이터 소스 동작 구현하기

TAController의 `awakeFromNib` 메소드를 구현한다. `self`를 데이터 소스로 지정한다.

```
- (void)awakeFromNib
{
    [countryTableView setDataSource:self];
    [countryTableView sizeLastColumnToFit];
}
```

`[countryTableView setDataSource:self]` 메시지는 TAController 객체를 테이블 뷰의 데이터 소스로 인식한다. 테이블 뷰는 NSTableDataSource 메시지를 TAController로 전송하기 시작한다. (Interface Builder에서 NSTableView의 `dataSource` 아웃렛을 설정하여 동일한 작업을 수행할 수 있다).

### *Informal 프로토콜인 NSTableDataSource의 2개 메소드 구현하기*

데이터 소스로서의 역할을 수행하려면, TAController는 <예제 10-8>에서 보여진 바와 같이 Informal 프로토콜인 NSTableDataSource의 2개 메소드(`numberOfRowsInTableView:` 및 `tableView:objectValueForTableColumn:row:`)를 구현해야 한다.

*<예제 10-8> NSTableDataSource 프로토콜 구현*

```
- (int)numberOfRowsInTableView:(NSTableView *)theTableView {
    return [countryKeys count];
}

- (id)tableView:(NSTableView *)theTableView
    objectValueForTableColumn:(NSTableColumn *)theColumn
    row:(int)rowIndex
{
    if ([[theColumn identifier] isEqualToString:@"Countries"])
        return [countryKeys objectAtIndex:rowIndex];
    else
        return nil;
}
```

첫 번째 메소드는 `countryKeys` 어레이에서 국가명이 몇 개인지 리턴한다. 테이블 뷰는 몇 개의 가로 행을 만들어야 할지 결정하기 위해 이 정보를 사용한다.

두 번째 메소드는 세로열(항상, Countries여야 한다)이 정상적인지 판단하기 위해 세로열 식별자를 평가한다. 정상적이라면, 메소드는 `rowIndex`와 연결된 `countryKeys` 어레이에서 국가명을 리턴한다. 이 국가명은 세로열의 `rowIndex`에서 나타난다.(세로열의 어레이와 셀은 인덱스로 동기화된다는 것을 기억한다)

Informal 프로토콜인 NSTableDataSource은 이 튜토리얼에서 구현되지 않는 `tableView:setObjectValue:forTableColumn:row:`라는 메소드를 제공한다. 이 메소드는 데이터 소스로 하여금 사용자가 입력한 데이터를 테이블 뷰 셀로 추출할 수 있도록 한다. Travel Advisor 테이블 뷰는 읽기전용이기 때문에 구현할 필요는 없다.

멀티플 테이블 뷰를 내장한 응용 프로그램을 운용한다면, 각 테이블 뷰는 NSTableView 델리게이션 메소드를 발생시킨다. `theTableView` 인수를 판단하여, 테이블 뷰가 발생시킨 메소드를 구별할 수 있다.

## *Country-selection 메소드 구현하기*

마지막으로, 새로운 레코드를 요청하기 위해 마우스를 클릭하면 테이블 뷰는 응답을 해야 한다. 앞서 언급했듯이, Interface Builder에서 `handleTVClick:` 액션을 정의했다. 이 메소드는 수 많은 작업을 수행한다.

- 현재의 Country 객체를 저장하거나 새로운 객체를 생성한다.
- 새로운 레코드가 있다면, 데이터를 테이블 뷰에 제공하여 어레이를 재분류한다.
- 선택한 레코드를 보여준다.
- <예제 10-9>의 `handleTVClick:` 메소드를 구현한다. 이 메소드는 테이블 뷰에서 사용자 선택에 응답한다.

*<예제 10-9> handleTVClick: 메소드 구현*

```
- (IBAction)handleTVClick:(id)sender
{
    Country *aRec;
    NSString *countryName;
    int index = [sender selectedRow];

    if (index == -1) return;
    countryName = [countryKeys objectAtIndex:index];

    if (recordNeedsSaving) {
        [self addRecord:self];
        index = [countryKeys indexOfObject:countryName];
        [countryTableView selectRow:index byExtendingSelection:NO];
    }
```

*<예제 10-9> handleTVClick: 메소드 구현 (계속)*

```
    aRec = [countryDict objectForKey:countryName];
    [self populateFields:aRec];
}
```

handleTVClick:는 먼저 사용자가 선택한 가로행(즉, 국가)을 식별해야 한다. 가로행을 선택하지 않으면, NSTableView의 selectedRow 메소드는 -1과 handleTVClick:가 존재한다고 리턴한다. 그렇지 않으면, 가로행 인덱스는 국가명을 나타내기 위해 사용된다.

Travel Advisor는가 Country 객체 데이터를 추가하거나 변경할 때, recordNeedsSaving 플래그는 YES로 설정된다.(뒷부분에서 이 작업을 수행하는 방법을 설명한다) recordNeedsSaving가 YES라면, 코드는 변경된 레코드를 업데이트하거나 새로운 레코드를 삽입하는 addRecord: 메소드를 호출한다. 새로운 레코드를 삽입하면 테이블 내용이 바뀌기 때문에 정상적으로 나타내려면 선택한 가로행을 업데이트해야 한다. 이 작업을 수행하려면, indexOfObject:를 사용하여, countryKeys에서 국가명이 올바른 위치에 올 수 있도록 해야 한다. 테이블 뷰 내에서 해당 가로행을 선택한다.

마지막으로, 국가명은 딕셔네리에서 해당 Country 인스턴스를 얻기 위한 키로 사용된다. 따라서, populateFields:를 호출하여, Country의 인스턴스 변수 값으로 윈도우를 업데이트한다.

## 옵션 실행

사용자는 테이블 뷰를 클릭하는 것처럼 마우스 액션에서 알트(alternative) 키를 사용할 수 있다. 알트키를 사용하려면, Interface Builder에서 메뉴 커맨드를 추가하고, 커맨드 속성으로 키를 지정하여, 커맨드를 발생시키는 액션 메소드를 정의한 뒤 메소드를 구현한다.

nextRecord:와 prevRecord: 메소드는 사용자가 Records 메뉴에서 Next 또는 Previous를 선택하거나, Command-Option-N 및 Command-Option-P 키를 입력하면 발생한다. TAController.m에서 다음 내용을 염두에 두고 메소드를 구현한다.

1. 선택한 가로행의 인덱스(selectedRow)를 얻는다.
2. 어떤 키를 누르냐(또는 어떤 커맨드를 클릭하는지)에 따라 인덱스는 증가하거나 감소한다.
3. 테이블 뷰의 처음이나 끝부분이 마주치면, 선택을 "랩(wrap)"하게 된다.(힌트: countryKeys 어레이카운트를 사용한다)
4. 인덱스를 사용하여 새로운 가로행을 선택한다. 그러나, 더 이상은 선택하지 않는다.
5. handleTVClick:를 self로 전송하여 새로운 가로행에 마우스 클릭을 시험해한다.

## 프로젝트 구축하기

이제, Travel Advisor를 구축해야 한다. 코드나 nib 파일에 오류가 발생했는지 확인한다.

## 레코드를 추가 및 삭제하기

사용자가 Add Record를 클릭하여, Country 코드를 입력하면, **addRecord:** 메소드가 발생한다. 이 메소드는 Country 객체를 응용 프로그램의 Dictionary에 추가하는 작업외에 몇 가지 작업을 더 수행한다.

- 국가 이름을 입력하였는지 확인한다.
- 테이블 뷰가 새로운 레코드를 반영할 수 있도록 한다.
- 레코드가 이미 존재한다면, 이를 업데이트한다.(그러나, 변경될 경우에 한한다)
- <예제 10-10>의 **addRecord:** 메소드를 구현한다.

*<예제 10-10> addRecord: 메소드 구현*

```
- (IBAction)addRecord:(id)sender
{
    Country *aCountry;
    NSString *countryName = [countryField stringValue];

    //  Is there country data to be saved?
        if (recordNeedsSaving && ![countryName isEqualToString:@""])
{
        aCountry = [countryDict objectForKey:countryName];

        //  Is current object already in dictionary?
        //  ...  aCountry will be nil if new
            if (!aCountry) {
            //  Create Country obj, add to dict, add name to keys array
            aCountry = [[[Country alloc] init] autorelease];
            [countryDict setObject:aCountry forKey:countryName];
            [countryKeys addObject:countryName];

            //  Sort array and update table view
            [countryKeys sortUsingSelector:@selector(compare:)];
            [countryTableView reloadData];
            [countryTableView selectRow:
                    [countryKeys indexOfObject:countryName]
                    byExtendingSelection:NO];
        }

        //  Update the country object
        [self extractFields:aCountry];
        recordNeedsSaving = NO;

        [commentsLabel setStringValue:[NSString stringWithFormat:
                @"Notes and Itinerary for %@", countryName]];
```

*<예제 10-10> addRecord: 메소드 구현 (계속)*

```
            [countryField selectText:self];
        }
    }
```

이 메소드는 Country 객체를 NSDictionary "데이터베이스"에 추가한다. 이 코드는 국가 이름이 입력되고, 필드가 변경한대로 신호를 보냈는지 확인한다. 이 때, Country 객체가 Dictionary에서 제공된 이름을 사용하였는지 확인한다. 키 객체가 없다면, `objectForKey:`는 `nil`을 돌려보낸다. 이 경우, 코드는 새로운 Country 객체를 생성하고, Dictionary에 이를 추가하며, 이름을 키 어레이에 추가한다. 이 때, 어레이는 분류되며, 테이블 뷰는 업데이트된다. `reloadData` 메시지를 통해 테이블 뷰의 내용을 업데이트한다. `selectRow:byExtendingSelection:` 메시지는 테이블 뷰에 새로운 레코드를 제공한다.

맨 왼쪽의 if 다음에 나오는 `aCountry` 변수는 countryDict에 저장된 객체를 참조한다. Country 객체는 응용 프로그램 필드(`extractFields:`) 정보를 통해 업데이트되며, `recordNeedsSaving` 플래그를 리셋한다. 마지막으로, 텍스트 뷰의 라벨은 방금 추가한 국가를 반영하기 위해 업데이트되며, 국가명 필드가 하이라이트된다.

<예제 10-10>에서, `if(!aCountry)`를 살펴보기로 하자. `if(!aCountry)`는 `if(aCountry == nil)`를 간단하게 표현한 것이다. 같은 맥락으로, `if(aCountry)`는 `if(aCountry !=nil)`와 같다. 새롭게 생성된 Country 객체는 `autorelease` 메시지를 전송한다. Country 객체는 객체 값을 보유하고 있는 딕셔네리에 저장되어 있기 때문에 `addRecord:`는 새롭게 생성된 Country 객체의 소유권을 필요로 하지 않는다. Dictionary는 Country 객체를 자동해제하여 이벤트 루프가 끝날 무렵에 객체의 단독 소유자가 된다.

- `deleteRecord:` 메소드를 구현한다. 이 메소드는 구조적인 측면에서 `addRecord:`와 유사하지만, Country 레코드를 변경할 것인지를 신경 쓰지 않아도 되기 때문에 더 단순하다고 볼 수 있다. 레코드가 삭제되면, 테이블 뷰는 업데이트되고, 응용 프로그램의 필드는 제거되어야 한다.

## 필드 유효성

NSControl 클래스는 셀의 내용을 유효하게 하기 위해 API를 제공한다. 유효성은 셀 값이 한계에 도달하거나 어떤 기준을 충족하는 지 여부를 확인한다. Travel Advisor에서 사용자가 Rate 필드에 음수 값을 입력하지 않았다는 것을 확인해야 한다.

유효성 요청은 컨트롤이 델리게이트로 전송하는 `control:isValidObject:` 메시지이다. 이 경우, 컨트롤은 Rate 필드이다.

1. awakeFromNib에서 TAController를 필드의 델리게이트로 유효하게 만든다.

```
[currencyRateField setDelegate:self] ;
```

2. control:isValidObject: 메소드를 구현하여, 필드 값을 유효하게 한다.

```
- (BOOL)control:(NSControl *)control isValidObject:(id)obj
{
    if (control == currencyRateField) {
        if ([obj floatValue] <= 0.0) {
            NSRunAlertPanel(@"Travel Advisor",
                @"Rate cannot be zero or negative.", nil, nil, nil);
            return NO;
        }
    }
    return YES;
}
```

1개 이상의 유효한 필드 값을 가지고 있기 때문에 먼저, 어떤 필드가 메시지를 전송해야 할 지를 결정한다. 그리고, 필드값(두 번째 객체에서 전송됨)을 점검한다. 필드값이 음수이면, 메시지 상자가 나타나고, 값 입력을 차단하기 위해 `NO`를 리턴한다. 반대 경우에, `YES`를 리턴하고, 필드는 값을 허용한다.

상기 예제에서 값을 허용할 수 없는 이유를 사용자에게 통보하기 위해 `NSRunAlertPanel`를 호출한다. Travel Advisor는 값을 평가할 수 없지만, 함수는 메시지 상자에서 사용자가 클릭한 버튼이 어떤 것인지 나타내는 상수를 리턴한다. 코드의 논리는 사용자 입력에 따라 분기될 수 있다. 게다가, 함수는 (`printf` 양식 변환 지정자를 사용하여) 변수 정보를 메시지의 body에 삽입할 수 있도록 한다.

## *응용 프로그램 관리*

Travel Advisor의 대규모 코딩 작업이 완료되었다. 앞으로 6개 정도의 메소드를 더 구현하면 된다. 이중 일부 메소드는 모든 응용 프로그램이 수행해야 하는 태스크를 수행한다. 나머지 메소드는 Travel Advisor에서 필요한 기능성을 제공한다. 이 섹션에서는 다음사항을 설명한다.

- TAController 객체를 아카이빙 및 언아카이빙한다.
- TAController의 `init` 및 `dealloc` 메소드를 구현한다.
- 응용 프로그램이 종료되면, 데이터를 저장한다.
- 사용자가 레코드를 변경하면, 변경한 레코드를 표시한다.
- 변경된 환율값을 얻어 제공한다.

사용자가 Travel Advisor에 입력한 데이터는 파일 시스템에 저장되어야 한다. Travel Advisor의 아카이빙을 초기화하기에 가장 적합한 시간은 응용 프로그램이 종료될 때이다. TAController를 응용 프로그램 객체(NSApp)의 델리게이트로 만들 수 있다. 이제, 응용 프로그램이 종료되기 직전에 전송된 **applicationShouldTerminate:** 델리게이트 메시지에 응답한다.

<예제 10-11>에서와 같이 델리게이트 메소드 **applicationShouldTerminate:**를 구현한다.

*<예제 10-11> applicationShouldTerminate: 메소드 구현*

```
- (NSApplicationTerminateReply)applicationShouldTerminate:(id)sender
{
   NSString *storePath = [NSHomeDirectory()
            stringByAppendingPathComponent:@"Documents/TravelData.travela"];

   // save current record if it is new or changed
   [self addRecord:self];
   if (countryDict)
      [NSArchiver archiveRootObject:countryDict toFile:storePath];

   return NSTerminateNow;
}
```

이 함수는 아카이브 파일인 **TravelData**의 경로명을 구축한다. 이 파일은 사용자 홈 디렉토리의 Documents 폴더에 저장된다.

countryDict Dictionary가 있다면, TAController는 NSArchiver 클래스 메소드인 **archiveRootObject:toFile:**를 사용하여 저장한다. 딕셔네리가 아카이빙의 루트 객체로써 저장되기 때문에 딕셔네리가 참조하는 모든 객체(즉, 딕셔네리가 포함하고 있는 Country 객체) 또한 저장된다.

## *초기화 및 해제를 위해 TAController 메소드 구현하기*

<예제 10-12>의 **init**와 **dealloc** 메소드를 구현한다.

*<예제 10-12> init 및 dealloc 메소드 구현*

```
- (id)init
{
    NSString *storePath = [NSHomeDirectory()
            stringByAppendingPathComponent:@"Documents/TravelData.travela"];
    [super init];

    countryDict = [NSUnarchiver unarchiveObjectWithFile:storePath];

    if (!countryDict) {
        countryDict = [[NSMutableDictionary alloc] init];
        countryKeys = [[NSMutableArray alloc] initWithCapacity:10];
    } else {
```

*<예제 10-12> init 및 dealloc 메소드 구현 (계속)*

```
        countryDict = [[NSMutableDictionary alloc]
                initWithDictionary:countryDict];
        countryKeys = [[NSMutableArray alloc]
                initWithArray:[[countryDict allKeys]
                sortedArrayUsingSelector:
                @selector(caseInsensitiveCompare:)]];
    }

    recordNeedsSaving = NO;
    return self;
}

- (void)dealloc
{
    [countryDict release];
    [countryKeys release];
    [super dealloc];
}
```

**init** 메소드는 아카이브 파일인 **TravelData**를 사용자의 Documents 디렉토리에 배치하고, 경로를 리턴한다.

**unarchiveObjectWithFile:** 메시지는 지정된 파일에서 속성을 인코딩한 객체를 언아카이빙(즉, 복구)한다. 복구되어 리턴된 객체는 Country 객체의 NSDictionary(**countryDict**)이다.

NSDictionary가 복구되지 않으면, **countryDict** 인스턴스 변수는 **nil**로 남아 있는다. 이 같은 경우에, TAController는 비어있는 **countryDict** Dictionary 및 비어있는 **countryKeys** 어레이를 생성한다. 그렇지 않으면, 인스턴스 변수를 보유하여, Country 키 어레이를 구축한다.

**[countryDict allKeys]** 메시지는 Country 객체를 값으로 포함하고 있는 언어카이빙된 디렉토리인 **countryDict**에서 키 어레이(국가명)을 리턴한다. **sortedArrayUsingSelector:** 메시지는 NSString(다형성 및 동적 바인딩 사례)의 경우 어레이에서 객체 클래스가 정의한 **caseInsensitiveCompare:** 메소드를 사용하여 "로(Raw)" 어레이에서 아이템을 분류한다. 분류된 이름은 리턴된 값이기 때문에 임시(자동해제된) NSArray에 저장된다. 이 임시 어레이는 **countryKeys**로 할당된 수정 가능 어레이를 생성하기 위해 사용된다. 사용자가 국가를 추가 또는 삭제할 수 있기 때문에 수정 가능 어레이는 필요하다.

**dealloc** 메소드는 **init** 메소드가 생성한 객체를 해제한다. 그리고, **dealloc**의 슈퍼클래스 구현을 호출하여 객체가 제거될 때 까지 해제한다.

### 통지를 구현하여 변경된 기록을 추적하기

사용자가 Travel Advisor의 필드 데이터를 변경하면, 변경된 새로운 레코드를 표시해야 한다. 그래야만, 추후 저장할 때 알 수 있다. Application Kit는 응용 프로그램의 텍스트를 변경할 때 마다 통지를 브로드캐스트한다. 이 통지를 수신하려면, TAController를 통지 관찰자 리스트에 추가한다.

1. `awakeFromNib` 메소드에서, TAController를 `NSControlTextDidChangeNotification`가 배치된 모든 객체의 옵저버로 만든다.

```
[[NSNotificationCenter defaultCenter] addObserver:self
     selector:@selector(textDidChange:)
     name:NSControlTextDidChangeNotification object:nil];
```

2. Note 필드를 관찰해야 한다. Note 필드는 컨트롤 텍스트 객체가 아니다.

```
[[NSNotificationCenter defaultCenter] addObserver:self
     selector:@selector(textDidChange:)
     name:NSTextDidChangeNotification object:commentsField];
```

3. `textDidChange:`를 구현하여 `recordNeedsSaving` 플래그를 설정한다. Travel Advisor의 2개의 편집 가능 필드는 Conversion에서 사용되는 임시 값을 보유하지만, 저장하지는 않는다. if는 이들 필드가 통지를 발생시키는지 여부를 점검한다. 통지를 발생시킨다면, 플래그를 설정하지 않고, 리턴한다.(객체 메시지는 통지와 관련된 객체를 얻는다)

```
- (void)textDidChange:(NSNotification *)notification
{
    if (([notification object] == currencyDollarsField) ||
        ([notification object] == celsiusField)) return;

    recordNeedsSaving = YES;
}
```

4. `switchClicked:` 액션 메소드를 구현하여 English Widely Spoken 스위치로 변경된 사실을 통보받는다.

```
- (IBAction)switchClicked:(id)sender
{
    recordNeedsSaving = YES;
}
```

## *Archiving 및 Unarchiving 메소드 구현하기*

이 섹션에서는 Country 클래스의 아카이빙과 언아카이빙의 구현 방법을 설명한다. 이 단계가 완성되면, 국가의 전체 딕셔너리를 디스크에 저장할 수 있다. 그래서, 응용 프로그램이 종료되어도 여행 정보는 손실되지 않는다.

1. 다음과 같이 `Country.m`에서 `encodeWithCoder:` 메소드를 구현한다.

```
- (void)encodeWithCoder:(NSCoder *)coder
{
    [coder encodeObject:[self name]]];
    [coder encodeObject:[self airports]];
    [coder encodeObject:[self airlines]];
    [coder encodeObject:[self transportation]];
    [coder encodeObject:[self hotels]];
    [coder encodeObject:[self languages]];
    [coder encodeValueOfObjCType:"s" at:&englishSpoken];
    [coder encodeObject:[self currencyName]];
    [coder encodeValueOfObjCType:"f" at:&currencyRate];
    [coder encodeObject:[self comments]];
}
```

2. 다음과 같이 `initWithCoder:` 메소드를 구현한다.

```
- (id)initWithCoder:(NSCoder *)coder
{
    [self setName:[coder decodeObject]];
    [self setAirports:[coder decodeObject]];
    [self setAirlines:[coder decodeObject]];
    [self setTransportation:[coder decodeObject]];
    [self setHotels:[coder decodeObject]];
    [self setLanguages:[coder decodeObject]];
    [coder decodeValueOfObjCType:"s" at:&englishSpoken];
    [self setCurrencyName:[coder decodeObject]];
    [coder decodeValueOfObjCType:"f" at:&currencyRate];
    [self setComments:[coder decodeObject]];

    return self;
}
```

## *응용 프로그램 구축 및 운용하기*

Travel Advisor의 구축을 완료하면, Finder에서 아이콘을 더블 클릭하여 구동한다. 다음 시험을 통해 응용 프로그램을 실행한다.

- 몇 개의 레코드를 입력하고, 지리적 정보를 구성한다. 이 응용 프로그램에 대한 향후 여행정보는 아직은 신뢰할 만하지 않다.
- 테이블 뷰의 아이템을 클릭하고, 선택한 레코드가 나타났는지 확인한다. Command-Option-N 및 Command-Option-P를 누르고, 어떤 일이 발생하는 지 관찰한다.
- 값을 Conversion 필드에 입력하고, 어떻게 포맷되는지 확인한다. Rate 필드에 음수 값을 입력해 본다.
- 응용 프로그램을 닫고, 다시 구동한다. 응용 프로그램이 입력한 동일한 레코드를 출력하는 지 확인한다.

# III

# 멀티 윈도우 응용 프로그램

# 11

# *Cocoa의 멀티플 도큐멘트 아키텍처*

도큐멘트 기반 응용 프로그램은 현재 널리 사용되고 있는 일반적인 유형의 응용 프로그램이다. 이들 응용 프로그램은 동일한 프레임워크를 지원하지만, 파일에 저장될 수 있도록 고유하게 구성된 데이터를 제공한다. 워드 프로세서와 스프레드시트 응용 프로그램은 일반적으로 잘 알려진 도큐멘트 기반 응용 프로그램이다. 도큐멘트 기반 프로그램이 어떻게 체계화되어 있는지 알아보기 전에 응용 프로그램이 어떤 작업을 수행하는지 알아보기로 하자.

- 새 도큐멘트를 만든다.
- 파일에 저장된 기존의 도큐멘트를 불러온다.
- 사용자가 지정한 이름과 위치에 도큐멘트를 저장한다
- 저장된 도큐멘트를 불러온다
- 도큐멘트를 닫는다.(일반적으로 사용자가 변경사항을 저장한 이후)
- 도큐멘트를 출력하고, 페이지 레이아웃을 변경한다.
- 내부적으로 다양한 유형의 데이터를 제공한다.
- 도큐멘트의 편집 상태를 감시 및 설정하고, 메뉴 아이템을 유효하게 한다.
- 윈도우 타이틀을 정하고, 도큐멘트 윈도우를 관리한다.
- 응용 프로그램과 윈도우 델리게이션 메소드를 처리한다.(응용 프로그램이 종료된 경우)

Cocoa의 멀티 도큐멘트 아키텍쳐는 NSDocument, NSDocumentController 및 NSWindowController 클래스로 구성된다. 이들 클래스는 전체 "무상"으로 상기 기능을 지원한다. 작업 개발자는 멀티플 도큐멘트 아키텍쳐를 동적으로 사용하여 멀티 도큐멘트를 구현해야 한다.

아키텍쳐가 운용되는 방법을 이해하면, 멀티 도큐멘트 응용 프로그램을 몇 분내에 운용할 수 있다.

이 장은 Cocoa의 멀티플 도큐멘트 아키텍쳐에 관한 개요를 제공하고, NSDocument, NSDocumentController 및 NSWindowController간의 상호작용을 설명한다. 마지막 섹션에서는 간단한 멀티도큐멘트 RTF 텍스트 편집 응용프로그램을 구현하는 방법을 설명한다.

# 아키텍쳐 개요

사용자 관점에서 보면, 도큐멘트는 윈도우에 포함된 정보의 고유한 몸체라 할 수 있다. 사용자는 무제한으로 도큐멘트를 만들고, 만든 도큐멘트를 파일에 저장할 수 있다.

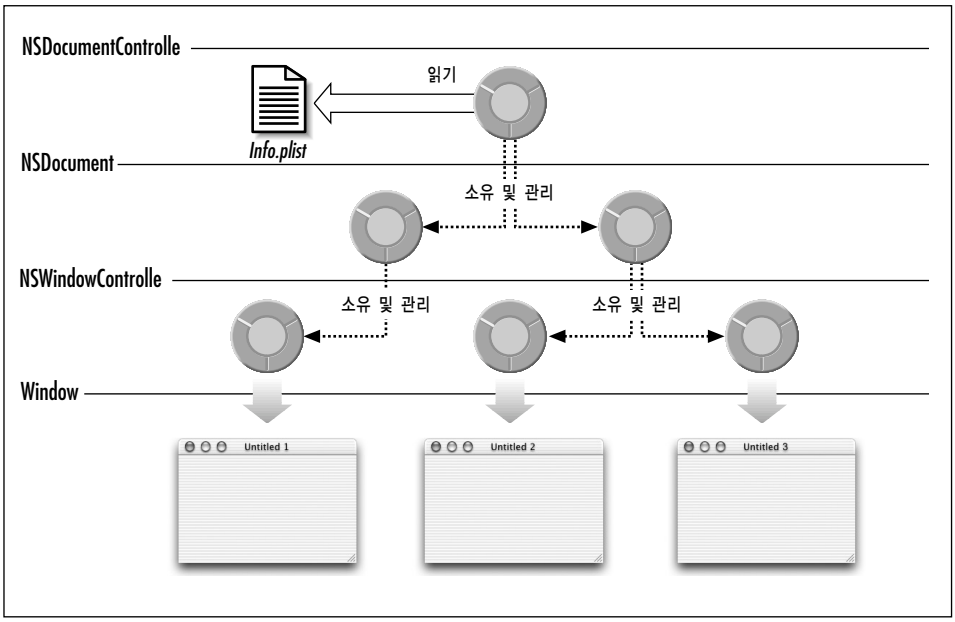
프로그래밍 관점에서 보면, 도큐멘트는 보조 nib 파일에서 복구된 객체와 자원, 그리고 이들을 관리하고 로딩하는 컨트롤러 객체로 구성된다. 이 도큐멘트 컨트롤러는 도큐멘트 인터페이스와 관련 자원을 제공하는 보조 nib 파일의 소유자라 할 수 있다. 도큐멘트 컨트롤러는 도큐멘트를 관리하기 위해 윈도우의 델리게이트 및 "컨텐트" 객체를 생성한다. 또한, 편집 상태를 추적하고 윈도우 닫기 이벤트를 처리하며 다른 상태에 응답한다.

사용자가 New(이와 동등한) 커맨드를 선택하면, 메소드가 응용 프로그램의 컨트롤러 객체에서 발생한다. 이 메소드에서 응용 프로그램 컨트롤러는 초기화 과정에서 도큐멘트 nib 파일을 로딩시키는 도큐멘트 컨트롤러 객체를 생성한다. 따라서, 도큐멘트는 응용 프로그램의 "코어" 객체와 독립적으로 운용된다. 응용 프로그램이 도큐멘트 상태 정보를 필요로 할 경우, 도큐멘트 컨트롤러를 조회할 수 있다.

사용자가 Save 커맨드를 선택하면, 응용 프로그램은 Save 패널을 제공하고, 사용자가 파일 시스템의 도큐멘트를 저장할 수 있도록 한다. 사용자가 Open 커맨드를 선택하면, 응용 프로그램은 사용자가 도큐멘트 파일을 선택하여 실행시킬 수 있도록 Open 패널을 제공한다.

3개의 Application Kit 클래스(NSDocumentController, NSDocument 및 NSWindowController)는 도큐멘트 기반 응용 프로그램의 아키텍쳐를 제공한다. 이 클래스 객체들은 응용 프로그램 도큐멘트를 생성, 저장, 실행 및 관리하는 작업을 분담하고 조정한다. 이들 객체는 <그림 11-1>에서 보이는 바와 같이 일 대 다층관계로 계층화되어 있다.

멀티 도큐멘트 응용 프로그램은 여러 NSDocument 객체들(각 New나 Open 기능을 위해 한개씩)을 만들고 관리하는 1개의 NSDocumentController를 제공한다. NSDocument 객체는 도큐멘트를 제공하는 윈도우를 위해 1개 이상의 NSWindowController 객체를 만들어 관리한다.




<그림 11-1> NSDocument, NSDocumentController 및 NSWindowController 관계

일부 객체는 NSApplication 및 NSWindow 델리게이트와 유사한 임무를 수행한다. NSDocument, NSDocumentController, NSWindowController는 디폴트가 아닌 응용 프로그램이나 윈도우의 델리게이트일 수 있다.

멀티플 도큐멘트 아키텍처는 3개의 Application Kit 클래스 뿐만 아니라 응용 프로그램과 운용될 수 있는 데이터가 어떤 유형인지 결정하기 위해 응용 프로그램의 Info 속성 리스트의 정보를 사용한다. 정보는 도큐멘트 유형의 어레이로 속성 리스트에 저장된다. 어레이에서 각 도큐멘트 유형은 다음 정보(딕셔네리로 패키지됨)를 포함한다.

- 도큐멘트 유형의 이름
- 도큐멘트의 데이터 유형에 해당하는 `rtf` 및 `txt`와 같은 파일명 확장자 어레이
- 도큐멘트의 데이터 유형에 해당하는 TEXT 및 PICT와 같은 Mac OS 스타일 유형 식별자의 어레이
- 데이터를 상호작용할 때 응용 프로그램의 역할을 정하는 스트링. 응용 프로그램은 제공된 데이터 유형에 대한 Editor나 Viewer일 수 있다.
- 데이터 유형을 처리하는 응용 프로그램에서 NSDocument 서브 클래스의 이름

응용 프로그램의 NSDocumentController는 Info 속성에서 정보를 사용하여 다음과 같은 작업을 수행한다.

- 개방형 대화 상자를 제공할 때 비정상적인 파일 유형을 자동으로 걸러낸다.
- 데이터 유형에 적합한 NSDocument 서브클래스를 제공한다.
- 사용자가 특정 유형의 도큐멘트를 저장하는 것을 방지한다.

Project Builder는 응용 프로그램의 도큐멘트 유형 어레이에서 엔트리를 생성하고, 편집하기 위해 간단한 사용자 인터페이스를 제공한다. 그래서, 속성 리스트를 직접 변경할 필요가 없다.

## 도큐멘트 실행

응용 프로그램에서 NSDocumentController 객체는 주로 도큐멘트를 생성하고 실행시켜 이들 도큐멘트를 추적하고 관리한다. NSDocumentController는 도큐멘트 객체의 내부 리스트를 유지하여, 현재 도큐멘트(현재, 윈도우가 키인 도큐멘트)를 추적한다. NSDocumentController는 응용 프로그램이 구동할 때, 응용 프로그램이 종료될 때, 시스템 전원이 꺼질 때, 도큐멘트가 실행되었거나 Finder에서 나타날 때 특정 응용 프로그램 이벤트에 적절하게 응답하여 하드와이어 방식으로 연결된다. 예를 들어, 사용자가 File 메뉴에서 New를 선택하면, NSDocumentController는 다음을 수행한다.

1. 응용 프로그램의 도큐멘트 유형 어레이에서 첫 번째 엔트리로 명시된 NSDocument 서브클래스의 인스턴스를 할당한다.
2. NSDocument 서브클래스의 init 메소드를 발생시켜 인스턴스를 초기화한다.

사용자가 File 메뉴에서 Open을 선택하면, NSDocumentController는 다음을 수행한다.

1. 응용 프로그램의 Info 속성 리스트(`Info.plist`)에서 데이터 유형을 사용하여 파일 리스트를 필터링하면서 Open 패널을 제공하고, 사용자 선택을 얻는다.
2. 파일 및 데이터에서 유형 정보를 사용하여 해당 NSDocument 서브클래스의 인스턴스로 할당한다.
3. 파일 내용이 도큐멘트 인스턴스로 로딩될 수 있도록 `initWithContentsOfFile:ofType:`을 발생시켜 객체를 초기화한다.

원한다면, 커스텀 도큐멘트 컨트롤러를 생성하여 같은 이벤트의 결과로 발생된 델리게이트 메소드를 구현한다. 그러나, default NSDocumentController 객체는 대부분 상황에 적합한 응용 프로그램 컨트롤러이며, 이를 서브클래스화할 필요가 없어야 한다. About 패널을 제공하고 응용 프로그램의 환경설정을 처리하는 것과 같은 추가 동작이 요구되면, NSDocumentController의 서브클래스보다는 커스텀 컨트롤러 객체가 이 작업을 수행하는 것이 적합하다.

## NSDocument의 역할

NSDocument 객체의 주요 역할은 도큐멘트와 연결된 영구적 데이터를 제공, 조작, 저장 및 로딩하는 것이다. (Info 속성 리스트의 DocumentTypes 어레이에 지정된 대로) 객체가 인식해야 하는 도큐멘트 유형에 기반한 도큐멘트 객체는 다음과 같은 기능을 수행해야 한다.

- 데이터를 윈도우에 제공하는 응용 프로그램에서 다른 객체를 제공한다. 도큐멘트 객체는 지원된 포맷으로 데이터를 제공할 수 있어야 한다.
- 데이터를 내부 데이터 구조로 로딩하고, 윈도우에 나타낸다. 도큐멘트 객체는 지원된 포맷으로 데이터를 수신할 수 있어야 한다.
- 파일 시스템의 특정 위치 파일에 도큐멘트 데이터를 저장한다.
- 파일에 저장된 도큐멘트 데이터를 해독한다.

윈도우 컨트롤러 지원으로 NSDocument는 자신의 윈도우에서 데이터의 디스플레이와 캡쳐를 관리한다. Application Kit의 특수 하드와이어 방식으로 키 윈도우에 연결된 NSDocument는 사용자가 도큐멘트를 저장, 출력, 복구 및 닫을 때 첫 번째 리스폰더 액션 메시지를 수신한다. 또한, Save 패널 및 Page Layout 패널을 운용하고 관리하는 방법을 인식한다.

완전히 구현된 NSDocument는 편집된 상태를 추적하고, 도큐멘트 데이터를 제공하며, 취소 및 복귀 기능을 수행한다. 제12장, *To Do: 기본 원리* 및 제13장, *To Do: 확장*에서 튜토리얼을 검토해 보면 알다시피, 기본적으로 이들 동작이 완전하게 제공되지는 않는다. 그러나, NSDocument는 각각을 구현할 때 훨씬 더 많은 동작을 지원한다.

편집 상태 추적 기능의 경우, NSDocument는 도큐멘트 변경 카운터를 업데이트하기 위해 API를 제공한다. 취소 및 복귀 기능의 경우, NSDocument는 NSUndoManager를 생성하고, Undo and Redo 메뉴 커맨드에 적합한 응답을 하며, 취소 및 복귀 기능을 수행할 때 변경 카운터를 업데이트한다.

NSDocument는 Page Layout 패널의 디스플레이와 출력시 사용되는 NSPrintInfo 객체의 후속 변경을 용이하게 한다.

## NSWindowController의 역할

NSWindowController는 일반적으로 nib 파일에 저장된 도큐멘트와 연결된 1개의 윈도우를 관리한다. 도큐멘트가 멀티플 윈도우를 제공할 경우, 각 윈도우는 윈도우 고유의 컨트롤러를 지원한다. 예를 들면, 도큐멘트는 메인 데이터 입력 윈도우와 윈도우 리스트 레코드를 제공한다. 각 윈도우는 자신의 NSWindowController를 제공한다.

자신의 NSDocument가 요청하면, NSWindowController는 윈도우가 포함된 nib 파일을 로딩하고, 이를 제공한다. 또한, 정상적으로 윈도우를 닫는 역할(확실히 저장한 이후)도 수행한다.

NSWindowController의 서브클래스는 옵션 사항이다. 응용 프로그램에서 기본적인 인스턴스를 사용할 수 있다. 서브클래스는 NSWindowControllers를 늘려, 다양한 nib-로딩 및 설정 작업을 수행하며, 윈도우의 타이틀을 사용자화할 수 있다.

어떤 응용 프로그램은 NSDocument 클래스에서 사용자-인터페이스-고유 로직을 이동시키기 위해 NSWindowController를 서브클래스화할 수 있다. 그래서, 모델의 고유한 기능에 집중할 수 있다. `/Developer/Examples/AppKit`의 Sketch 응용 프로그램은 이 기법을 사용한다.

# 도큐멘트 기반 응용 프로그램 구현하기

코드를 많이 작성하지 않고서도 도큐멘트 기반 응용 프로그램을 조합할 수 있다. 요구 사항을 최소화하려면 Application Kit은 default NSWindowController 인스턴스와 기본적인 NSDocumentController 인스턴스를 제공해야 한다. 개발자는 멀티 도큐멘트 프로젝트를 생성하고, 휴먼 인터페이스를 구성하여, NSDocument의 서브클래스를 구현한 뒤 응용 프로그램이 요구하는 커스텀 클래스나 동작을 추가해야 한다.

다음 절차는 상당히 간단한 Rich Text(RTF) 편집기의 생성과정을 설명한다. 구축하고 디버깅하려면 몇 일 또는 몇 주 걸리는 응용 프로그램을 Cocoa의 Help없이도 몇 분내에 몇 줄의 코드만으로 만들 수 있다.

## 도큐멘트 기반 응용 프로그램 패키지

Project Builder는 도큐멘트 기반 응용 프로그램 프로젝트 템플리트를 제공하여 응용 프로그램을 신속하게 개발할 수 있도록 한다. 이 프로젝트는 다음을 제공한다.

- **응용 프로그램의 메인 nib 파일** 이 nib 파일은 표준 Cocoa 응용 프로그램 메뉴 바를 제공한다. File과 Edit 메뉴의 메뉴 아이템들은 첫 번째 해당 리스폰더 액션 메소드에 이미 연결되어 있다.
- **응용 프로그램의 도큐멘트를 위한 nib 파일** 이 nib 파일은 UI 아이템을 추가할 수 있는 1개의 윈도우를 제공한다. MyDocument라는 NSDocument의 서브클래스는 nib 파일의 File's Owner를 생성한다. 또한, NSDocument(MyDocument 서브클래스)는 자신의 윈도우에 아웃렛을 가지고 있다.

- **골격만 갖춘 NSDocument 서브클래스 구현** 이 프로젝트는 도큐멘트 nib 파일의 NSDocument 서브클래스에서 파생된 `MyDocument.h`와 `MyDocument.m`를 제공한다. MyDocument.m 파일은 비중있는 메소드의 "Stub"를 정의한다.
- **info 속성 리스트의 도큐멘트 유형 엔트리** Targets 화면의 Application Settings 패인은 `Info.plist` 파일을 변경하기 위한 간단한 사용자 인터페이스이다. 파일은 도큐멘트 유형 어레이를 비롯해 전체 응용 프로그램 키의 Placeholder 값을 제공한다.

이 장의 나머지 부분은 이 탬플리트에서 구동되는 도큐멘트 기반 응용 프로그램을 만들기 위해 개발자가 수행해야 하는 작업이 무엇인지에 관해 설명한다.

## *프로젝트 만들기*

1. Project Builder를 구동하고, File 메뉴에서 New Project를 선택한다.
2. Simple RTF Edit라는 Cocoa 도큐멘트 기반 응용 프로그램 프로젝트를 만든다.
3. Resources 그룹을 연다. Interface Builder에서 이 파일을 열려면, `MyDocument.nib`를 더블클릭한다.

nib 파일은 상당히 단순한 파일로, 기본적으로 텍스트 스트링을 가진 단 1개의 윈도우를 제공한다. File's Owner 인스턴스를 선택하여 Inspector를 불러오면, 해당 MyDocument에 설정된 Attributes 패인에서 File's Owner를 확인할 수 있다. 또한, Connections 패인에서 윈도우와 연결된 아웃렛을 확인할 수 있다. 원한다면, `MainMenu.nib` 파일을 연다. 앞서 언급했듯이, 여러 응용 프로그램의 메뉴 아이템은 첫 번째 해당 리스폰더 액션 메소드에 연결되어 있다. 응용 프로그램의 NSDocumentController가 이 메소드를 구현하며, <표 11-1>에 나타나 있다.

*<표 11-1> 초기 멀티도큐멘트 응용 프로그램의 타겟/액션 구성*

| 파일 메뉴 명령어 | 첫 번째 리스폰더 액션 |
|---|---|
| New | `newDocument:` |
| Open | `openDocument:` |
| Save | `saveDocument:` |
| Save As | `saveDocumentAs:` |
| Save To | `saveDocumentTo:` |
| Save All | `saveAllDocuments:` |
| Close | `closeDocument:` |
| Revert | `revertDocumentToSaved:` |
| Print | `printDocument:` |
| Page Layout | `runPageLayout:` |

이제, MyDocument.m을 열어, NSDocument 서브클래스 구현을 검토한다. 4개 메소드를 제공한다.

- `- (NSString *)windowNibName.` 이 메소드는 클래스의 nib 파일명을 반환한다. 응용 프로그램의 NSDocumentController는 이 메소드를 사용하여 도큐멘트 객체를 위해 nib 파일을 배치하고 로딩한다.
- `- (void)windowControllerDidLoadNib:(NSWindowController *)aController.` 이 메소드는 도큐멘트 객체 nib 파일이 로딩되면 초기화를 제공한다.
- `- (NSData *)dataRepresentationOfType:(NSString *)aType.` NSDocumentController가 도큐멘트 데이터를 디스크에 저장할 경우, 지정된 포맷(데이터 유형)으로 도큐멘트 데이터를 저장한 NSData 객체를 반환하기 위해 이 메소드를 구현해야 한다.
- `- (BOOL)loadDataRepresentation:(NSData *)data ofType:(NSString *)aType.` 마찬가지로, 도큐멘트 컨트롤러가 도큐멘트 데이터를 전송할 때 도큐멘트 데이터를 로딩(제공)하는 이 메소드를 구현해야 한다.

프로젝트를 구축하고, 이를 시험한다. 알다시피, 전체적인 기반은 이미 구현되었다. 이제, 수행해야 할 작업은 텍스트 편집기를 만들기 위해 몇 개의 빈칸을 채우는 일이다.

## 인터페이스 구성하기

이 섹션은 응용 프로그램의 룩앤드필을 정의한다. Project Builder의 템플리트로 생성한 기본 nib 파일을 변경하여, 사용자가 RTF 텍스트를 보고, 편집할 수 있도록 텍스트 뷰를 추가한다.

1. MyDocument.nib를 연다.
2. 도큐멘트 윈도우를 작게 만들고, Your Document Contents라고 나타나는 기본적인 텍스트 객체를 삭제한다.
3. 팔레트의 DataViews 패인에서 윈도우로 NSTextView를 드래그한다.
4. 스크롤 바가 윈도우 전체를 점유할 수 있도록 크기를 조정한다. Info 윈도우의 Attributes 패인을 사용하여 스크롤 뷰의 경계를 클릭한다.
5. 스크롤 뷰를 선택하여, Inspector에서 Size 패인을 불러온다. 크기조정 옵션을 변경하여, 윈도우 크기가 변경되면, 스크롤 뷰도 따라서 변경될 수 있도록 한다.

6. Project Builder에서 `MyDocument.h`를 열어, 텍스트 뷰의 아웃렛 선언을 추가한다.

```
IBOutlet NSTextView *textView;
```

7. Project Builder Groups 및 Files 리스팅에서 Interface Builder의 `MyDocument.nib window`로 `MyDocument.h`를 드래그한다. 이로써, Interface Builder는 아웃렛을 연결할 수 있도록 파싱할 수 있다.
8. Instances 패인에서 File's Owner 인스턴스(MyDocument 인스턴스의 프록시임)로부터 텍스트 뷰로 라인을 드래그하여 연결하고, `textView` 아웃렛을 연결한다. 경고: 라인을 연결하기 위해 MyDocument의 인스턴스를 만들지 않는다. 실행시, MyDocument의 진짜 인스턴스인 File's Owner를 연결을 원할 수도 있다..
9. nib 파일을 저장한다.

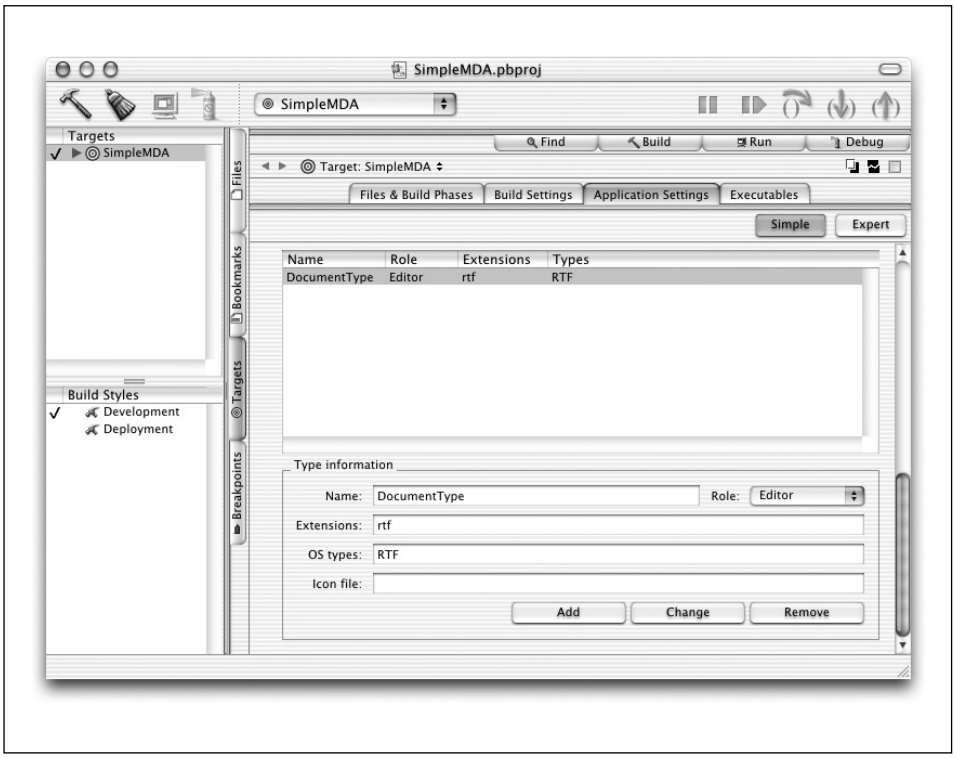
이제, 샘플 텍스트 편집기의 사용자 인터페이스가 완성되었다.

## *Info 속성 리스트 변경하기*

Simple RTF Edit 응용 프로그램은 단 한 종류의 데이터, RTF를 처리한다. 응용 프로그램의 Info 속성 리스트를 변경하여 도큐멘트 유형을 지원하는 것은 실로 간단하다.

1. Project Builder에서 메인 윈도우의 Targets을 선택한다.
2. 기본적인 타겟인 Simple RTF Edit를 선택한다.
3. Target의 Application Settings을 선택한다.
4. <그림 11-2>에서 보여진 바와 같이 도큐멘트 유형 편집기를 스크롤 다운한다.
5. 그림에서 보여진 바와 같이, 기본적인 도큐멘트 유형 엔트리를 변경한다. 엔트리를 클릭하여 값을 Type Information 영역에 입력하고, Change를 누른다.

Target 윈도우의 Application Settings 패인은 다양한 Application-wide 속성을 생성하여 변경할 수 있다. 실행 파일명, 메인 Cocoa 클래스명 같은 중요한 값은 기본적으로 제공한다. 많은 속성들은 완성된 응용 프로그램에서 중요한 역할을 한다. 그러나, 간단한 응용 프로그램에서는 이 같은 속성을 설정하지 않는다. 다음 장에서는 최종 튜토리얼 응용 프로그램을 구축할 때 이들 속성에 관해 자세하게 설명한다. 따라서, 이 장에서는 이들 속성에 대한 설명은 하지 않는다.




<그림 11-2> Project Builder의 도큐멘트 유형 편집기

## NSDocument 서브클래스

도큐멘트 기반 응용 프로그램의 Application Kit 아키텍처를 사용하는 프로그램은 NSDocument의 서브클래스를 최소한 1개는 생성해야 한다. 아키텍처는 NSDocument 메소드를 오버라이드하여, 오버라이드한 메소드를 특정 상황에서 사용할 수 있도록 한다. 이 섹션은 가장 일반적인 경우에 대해 설명한다.

### 데이터 기반형

`dataRepresentationOfType:` 메소드는 일반적으로 파일에 데이터를 작성한 준비가 되어 있는 유형의 도큐멘트 데이터(NSData 객체로 패키지됨)를 생성하고 반환하기 위해 구현되어야 한다. `loadDataRepresentation:ofType:` 메소드는 특정 유형의 도큐멘트 데이터를 포함하고 있는 NSData 객체를 도큐멘트의 내부 데이터 구조로 전환하고, 도큐멘트 윈도우에서 데이터를 제공하기 위해 구현되어야 한다. NSData 객체는 도큐멘트 파일을 읽는 도큐멘트에서 파생된다. 서브클래스는 이 메소드를 오버라이드해야 한다.

### 위치 기반형

writeToFile:ofType: 메소드는 dataRepresentationOfType:에서 데이터를 얻은 fileWrapperRepresentationOfType:로부터 데이터를 획득하여 기본적으로 데이터를 파일에 작성한다. readFromFile:ofType: 메소드는 파일에서 데이터를 읽고, NSFileWrapper 객체를 생성하여, loadFileWrapperRepresentation:ofType:에 제공한다. 이 객체를 단순히 파일에서 사용할 거라면, loadDataRepresentation:ofType: 메소드로 건내진다. loadFileWrapperRepresentation:ofType:를 오버라이드하여 디렉토리 객체를 처리해야 한다.

NSDocument에서 도큐멘트 데이터가 충분하게 읽히거나 작성되지 않는다면, NSDocument의 서브클래스가 데이터 기반형을 대신하여 이들 메소드를 오버라이드 할 수 있다. 그러나, 오버라이드를 구현하려면 데이터 기반형의 로딩 작업을 수행해야 한다.

### 윈도우 컨트롤러 생성

NSDocument의 서브클래스는 윈도우 컨트롤러를 생성해야 한다. 이 작업은 직접 또는 간접적으로 수행할 수 있다. 도큐멘트가 단 1개의 nib 파일(파일속에 1개의 윈도우가 있음)을 가지고 있다면, 서브클래스는 윈도우의 nib 파일명을 반환하기 위해 windowNibName을 오버라이드할 수 있다. 그 결과로 기본적인 NSWindowController 인스턴스가 도큐멘트에 생성된다. 도큐멘트가 멀티플 윈도우를 제공할 경우나 커스텀 NSWindowController 서브클래스의 인스턴스가 사용되어야 한다면, NSDocument 서브클래스는 이들 객체를 만들기 위해 makeWindowControllers를 오버라이드한다.

### 출력과 페이지 레이아웃

일반적으로, 도큐멘트 기반 응용 프로그램은 도큐멘트 데이터가 어떻게 출력되었는지 정의하기 위해 (NSPrintInfo 객체) 사용한 정보를 변경할 수 있다. 서브클래스는 shouldChangePrintInfo:를 오버라이드하여 이 같은 변경을 거부할 수 있다. 응용 프로그램이 도큐멘트 데이터를 출력해야 한다면, NSDocument의 서브클래스는 printShowingPrintPanel:를 오버라이드해야 한다.

### 백업 파일

도큐멘트를 저장할 때, NSDocument는 새로운 파일에 데이터를 작성하기 전에 기존 파일을 백업한다.(백업 파일은 새로운 파일과 동일한 이름을 갖지만, 확장자앞에 틸드를 사용한다) 일반적으로, 쓰기 작업을 성공적으로 구현하였다면, 백업 파일을 삭제한다. 서브클래스는 YES를 반환하고, 가장 최신 백업 파일을 보유하기 위해 keepBackupFile를 오버라이드한다.

### 메뉴 아이템

NSDocument는 Revert 및 Save As 메뉴 아이템의 Enabled 상태를 관리하기 위해 `validateMenuItem:`을 구현한다. 다른 메뉴 아이템을 유효화하면, 이 메소드를 오버라이드할 수 있다. 그러나, `super`를 반드시 구현해야 한다. 메뉴 아이템의 유효성에 관한 자세한 내용은 NSMenuValidation의 Informal 프로토콜의 설명서를 참조한다.

### 초기화기(Initializers)

NSDocument의 초기화기는 서브클래서의 중요한 사항이다. `init` 메소드는 지정된 초기화기로 `initWithContentsOfFile:ofType:`에 의해 발생된다. `init` 메소드는 새로운 도큐멘트를 만들 때 발생된다. `initWithContentsOfFile:ofType:` 메소드는 도큐멘트를 실행할 때 발생된다. 따라서, 실행된 도큐멘트에만 적용되는 초기화 메소드라면, `initWithContentsOfFile:ofType:`을 오버라이드해야 한다. 일반적인 초기화 메소드라면, 물론 `init`를 오버라이드해야 한다. 이 두 경우, `super`를 반드시 구현해야 한다.

## NSDocument 서브클래스 구현하기

이제, NSDocument의 메소드를 오버라이드하여 RTF 데이터를 읽고, 쓰는 기능을 지원한다.

1. `MyDocument.h`를 열어, 다음 인스턴스 변수를 추가한다. 이 변수는 파일에서 로딩된 로(Raw)RTF 데이터 참조를 보유한다.

```
NSData *dataFromFile;
```

2. 서브클래스 구현 파일인 `MyDocument.h`를 연다.
3. `dealloc` 메소드를 오버라이드하여 `dataFromFile:`을 마무리한다.

```
- (void)dealloc {
    [dataFromFile release];
    [super dealloc];
}
```

4. 객체가 내용을 저장할 수 있도록 `dataRepresentationOfType:` 메소드를 구현한다. 이 메소드는 텍스트 뷰의 모든 텍스트를 선택하여 RTF 데이터로써 반환된다.

```
- (NSData *)dataRepresentationOfType:(NSString *)aType {
    return [textView RTFFromRange:
            NSMakeRange(0, [[textView string] length])];
}
```

5. 파일에서 인터페이스의 텍스트 뷰 객체로 데이터를 로딩하는 메소드를 추가한다. 이 메소드는 디스크에서 파일을 열어, 이전에 저장한 상태로 되돌리기 위해 텍스트 뷰의 모든 텍스트를 선택하여, 반입된 데이터로 대체한다.

```
- (void)loadtextViewWithData:(NSData *)data {
    [textView replaceCharactersInRange:
            NSMakeRange(0, [[textView string] length])
            withRTF:data];
}
```

6. **loadDataRepresentation** 메소드를 구현한다. 이 메소드는 2가지 경우의 로딩 데이터를 처리해야 한다. 첫 번째 경우는 객체가 이전에 저장된 상태로 복귀한 때이다. 이 때, 텍스트 뷰가 이미 존재하여 반입된 데이터에 직접 로딩될 수 있다. 그러나, 파일이 열리면, **loadDataRepresentation**은 도큐멘트의 nib 파일을 로딩하기 전에 호출된다. 이 경우를 처리하려면, RTF 데이터를 보유하고, RTF 데이터 참조를 인스턴스 변수에 저장해야 한다. 도큐멘트 컨트롤러가 도큐멘트의 nib 파일을 로딩하면, 데이터가 텍스트 뷰로 로딩되어 해제될 때 도큐멘트 객체의 **windowControllerDidLoadNib** 메소드가 발생한다.

```
- (BOOL)loadDataRepresentation:(NSData *)data ofType:(NSString *)aType {
    if (textView) {
        [self loadtextViewWithData:data];
    } else {
        dataFromFile = [data retain];
    }

    return YES;
}
```

7. 마지막으로, **windowControllerDidLoadNib** 메소드를 구현한다. 이 메소드는 파일로 로딩되기를 기다리는 데이터가 존재하는지 여부를 점검한다. 데이터가 존재한다면, 데이터를 텍스트 뷰 객체로 로딩하고, 참조를 해제한다. 이 메소드는 텍스트 뷰 객체에서 취소 기능을 지원한다. NSTextView는 취소 기능을 구현하기 때문에 다중 취소/복귀 기능을 지원할 코드를 작성할 필요가 없다. 취소 기능의 구현 결과로 도큐멘트는 자동으로 변경 사항을 추적한다. 그래서, 윈도우가 닫히면 사용자가 도큐멘트를 저장해야 할 것인지를 인식한다.

```
- (void)windowControllerDidLoadNib:(NSWindowController *)aController {
    [super windowControllerDidLoadNib:aController];

    if (dataFromFile) {
        [self loadtextViewWithData:dataFromFile];
        [dataFromFile release];
        dataFromFile = nil;
    }
```

```
        [textView setAllowsUndo:YES];
    }
```

작업이 완료되었다. 응용 프로그램을 구축하여, 시험해본다. 새로운 도큐멘트를 만들어, 일부 텍스트에 입력한 뒤, 저장한다. 그리고, 취소/복귀 기능을 시험한다. 또 다른 응용 프로그램이 만든 RTF 도큐멘트를 열어보고, 출력해본다. 코드를 20줄도 작성하지 않고 이 모든 작업이 완성된다는 것은 놀라운 일이다.

# 12

# *To Do: 기본 원리*

제11장, *Cocoa의 멀티플 도큐멘트 아키텍처*에서 간단한 멀티도큐멘트 응용 프로그램을 생성하는 방법을 배웠다. 이 장은 Cocoa 멀티도큐멘트 아키텍처를 사용하여 To Do라는 도큐멘트 기반 응용 프로그램을 생성하는 방법을 설명한다. To Do는 간단한 개인 정보 관리자 프로그램이다.

이 장은 To Do 응용 프로그램의 기본 프레임워크를 생성하는 방법을 설명한다. 데이터 모델을 정의하여, 커스텀 캘린더 뷰가 갖춰진 도큐멘트 인터페이스를 만든다. 제13장, *To Do: 확장*에서 To Do 최종 버전을 완성하면, 사용자는 응용 프로그램을 통해 캘린더의 특정 날짜에 접근하고, 특정 날짜에 약속이나 작업 리스트를 입력할 수 있다. To Do 양식은 <그림 12-1>과 같다.

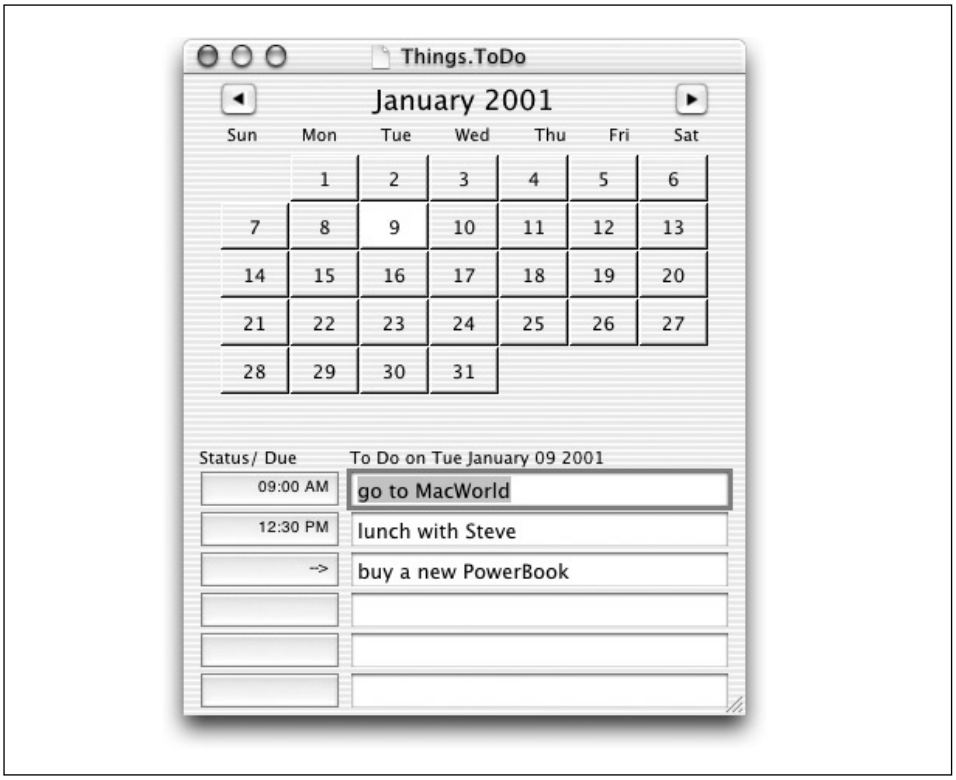
각각의 To Do 도큐멘트는 특정 용도의 데일리 "must-do" 아이템을 캡처한다. 예를 들어, 가정과 직장용 To Do 리스트를 만들 수 있다.

## *To Do 디자인*

To Do 응용 프로그램은 제10장, *Travel Advisor 튜토리얼*의 Travel Advisor 응용 프로그램보다 훨씬 복잡하다. To Do응용 프로그램은 3개의 nib 파일과 7개의 커스텀 클래스를 제공한다. <그림 12-2>는 클래스 및 로딩한 nib 파일의 인스턴스간의 상호관계를 보여준다.

그림에 나타난 객체는 Model-View-Controller 패러다임에 해당하기 때문에 이미 친숙해져 있을 것이다. ToDoItem 클래스는 응용 프로그램의 Model 객체를 제공한다. 이 클래스의 인스턴스는 도큐멘트에 나타난 아이템과 연결된 데이터를 캡슐화한다. 뷰 객체는 `MainMenu.nib`, `ToDoDocument.nib`과 `ToDoInfoWindow.nib` 파일에 포함되어있다.

그리고 컨트롤러 객체(실제로 1개 이상의 컨트롤러 객체)를 제공한다. Cocoa의 멀티플 도큐멘트 아키텍쳐가 제공하는 기본적인 컨트롤러를 비롯하여 도큐멘트 UI와 데이터 모델을 조정하는 ToDoDocument 객체 및 Info 윈도우가 있으며, To Do 도큐멘트 객체와 통신하는 Info 윈도우 컨트롤러가 있다.



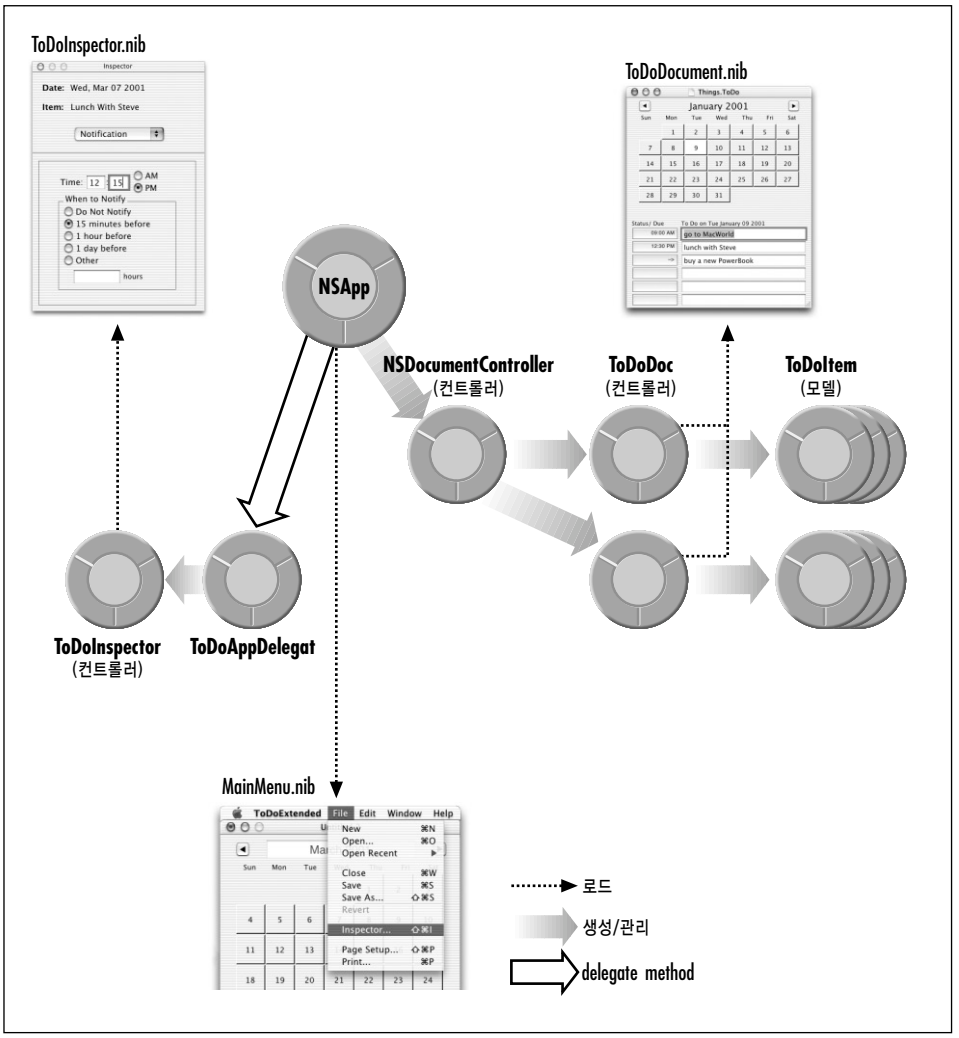

*<그림 12-1> To Do 응용 프로그램*

## *To Do 멀티 도큐멘트 디자인*

2가지 유형의 Controller 객체는 멀티 도큐멘트 응용 프로그램 디자인의 핵심적인 객체이다. 이들 객체는 응용 프로그램에서 다양한 영역의 역할을 수행한다. To Do는 기본적인 응용 프로그램 컨트롤러를 사용하여 응용 프로그램에 영향을 주는 이벤트를 관리한다. 각각의 ToDoDocument 객체는 도큐멘트 컨트롤러이며, 도큐멘트에 속한 모든 ToDoItems을 포함하여 1개의 도큐멘트를 관리한다.

Interface Builder가 메뉴 바에서 기본적으로 포함하고 있는 File 메뉴는 멀티 도큐멘트 응용 프로그램이 일반적으로 필요로 하는 명령어들을 제공한다. 사용자가 File 메뉴에서 New를 선택하면, 응용 프로그램 컨트롤러는 ToDoDocument 클래스의 인스턴스를 할당하고, 초기화한다. ToDoDocument 인스턴스를 초기화하면, 윈도우 컨트롤러는 `ToDoDocument.nib` 파일을 로딩한다.

사용자가 아이템을 도큐멘트에 입력하고, File 메뉴에서 Save를 선택하면, Save 시트가 나타나고, 사용자는 지정된 이름 아래의 파일 시스템에 도큐멘트를 저장한다. 추후, 사용자는 Open 대화 상자를 나타내는 Open 메뉴의 명령어를 사용하여 도큐멘트를 열 수 있다.



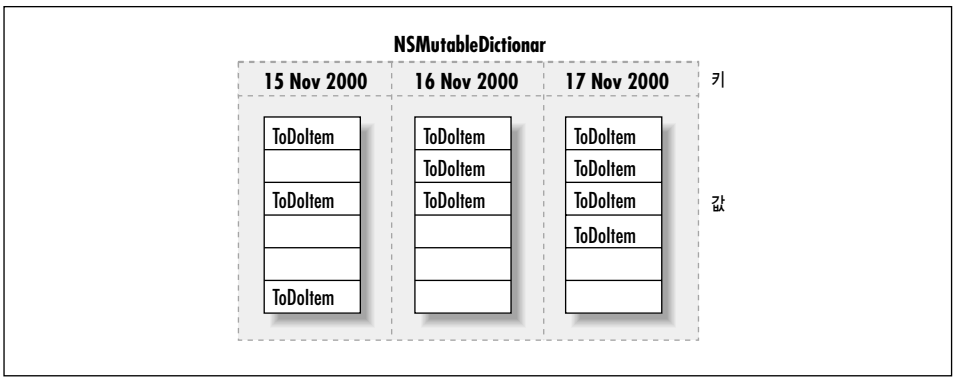

*<그림 12-2> To Do 디자인 개요*

To Do의 Controller 객체는 주로 윈도우와 응용 프로그램 객체에서 다양한 델리게이션 메시지에 응답하여, 객체 상태를 유지한다. 이 같은 이벤트의 일례는 사용자가 도큐멘트 윈도우를 닫을 경우나 데이터가 도큐멘트에 입력될 경우이다. 이벤트가 발생하면, 컨트롤러는 메시지나 통지를 다른 컨트롤러에 전송하여 상태를 알린다.

## 데이터를 저장하고, 액세스하는 방법

To Do 도큐멘트의 데이터 요소는 ToDoItems이다. 사용자가 아이템을 도큐멘트의 동작 리스트에 입력하면, ToDoDocument는 ToDoItem를 생성하고, 객체를 수정 가능 어레이(NSmutableArray)에 배치한다. ToDoItem는 리스트 매트릭스의 텍스트 필드 아이템과 동일한 어레이의 위치를 점유한다. 어레이의 객체와 매트릭스 아이템의 위치적 동일성은 디자인의 필수적인 요소이다. 예를 들면, 사용자가 도큐멘트 리스트의 첫번째 엔트리를 삭제하면, 도큐멘트는 어레이에서 해당 ToDoItem(index 0에서)을 삭제한다.

ToDoItems의 어레이는 특정 날짜와 연관이 있다. 따라서, 도큐멘트의 날짜는 키에 해당하는 날짜와 값에 해당하는 ToDoItems의 Arrary를 제공하는 딕셔네리(수정 가능한)로 구성된다. 사용자가 캘린더에서 날짜를 선택하면, 응용 프로그램은 딕셔네리에서 ToDoItems 어레이를 배치하기 위해 키로 사용할 날짜를 산출한다. <그림 12-3>는 ToDoItems, 아이템 어레이, 딕셔네리간의 관계를 보여준다.




*<그림 12-3> ToDoItems의 데이터 저장*

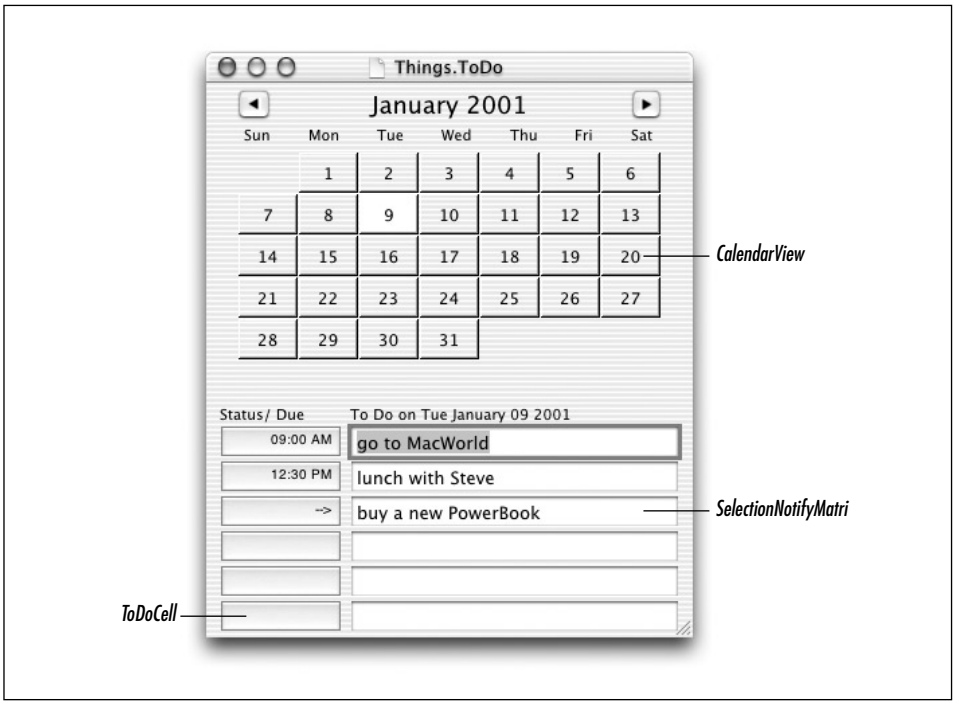
## To Do의 커스팀 뷰

지금까지, Model 객체와 Controller 객체에 관해 설명했지만, Model-View-Controller의 두 번째 객체인 View 객체에 대해서는 거의 설명하지 않았다. "기본적인" View를 사용하는 Travel Advisor와는 달리, To Do의 마지막 인터페이스는 3개의 커스텀 응용 프로그램 Kit 서브클래스에서 객체를 구성한다.

- **CalendarMatrix(NSMatrix 클래스)** 선택된 날짜를 델리게이트에 통보하는 동적 캘린더.
- **ToDoCell 클래스(NSButtonCell 클래스)** 각 상태에 대한 다른 이미지를 가진 3가지 상태(Tristate) 컨트롤. 또한, 아이템이 만료되면 횟수를 제공한다.

- **SelectionNotifyMatrix(NSMatrix 클래스)** 선택 사항이 발생하면 옵저버 객체에게 알려주는 텍스트 필드 리스트.

<그림 12-4>에서 3개의 뷰를 확인한다. 이 장에서는 CalendarMatrix 클래스만을 생성하는 방법에 관해 설명한다. 제13장에서 다른 클래스를 구축하는 방법을 설명한다.




<그림 12-4> To Do의 커스텀 뷰 객체

## Cocoa에서 날짜 및 시간에 관하여

To Do는 스케쥴 응용 프로그램이기 때문에 날짜와 시간을 광범위하게 사용한다. Cocoa는 NSDate에서 상속한 객체로써 날짜와 시간을 표현한다. 날짜와 시간을 객체로 인식했을 때의 장점은 기본 값을 나타내는 모든 객체에 공통적으로 사용할 수 있다는 것이다. 날짜와 시간은 대부분 운영 체제에 있는 일반적인 객체이긴 하지만 특정 운영 체제의 내부와 계층적으로 연결되지 않는다.

NSDate는 NSTimeInterval의 객체로써 날짜와 시간을 제공하고, 초 단위로 이들 값을 나타낸다. NSTimeInterval은 정교하고, 광범위한 날짜와 시간 값의 범위를 제공한다. 예를 들어, 10,000년을 나눠 1,000분의 1초의 정확성을 지원한다.

NSDate와 자신의 서브클래스는 시간을 참조 날짜와 관련된 초 단위로 산출한다.(2001년 1월 1일의 첫번째 순간) NSDate는 이 참조 날짜와 관련된 NSTimeInterval 값으로 모든 날짜와 시간을 전환한다.

NSDate는 NSDate 객체(NSDate로써 현재 시간과 날짜를 리턴하는 date 포함)를 얻고, 날짜를 비교하며, 관련 시간 값을 산출하여 스트링으로 날짜를 표현하는 메소드를 제공한다.

NSDate에서 상속한 NSCalendarDate 클래스는 서양 달력 시스템을 기준으로 날짜를 나타내는 객체를 발생시킨다. NSCalendarDate는 날짜 표현 방식을 조절하여 해당 시간 지역을 반영한다. 이로 인해, 다양한 시간 영역의 NSCalendarDate 객체를 추적할 수 있다. 또한, 시간 영역의 관점에서 날짜 정보를 나타낼 수 있다.

각각의 NSCalendarDate 객체는 캘린더 포맷 스트링을 제공한다. 이 포맷 스트링은 표준 C 라이브러리 기능인 `strftime`에서 사용된 것과 유사한 날짜 전환 지정자(Specifier)를 제공한다. NSCalendarDate는 이 포맷 스트링에 부합하여 사용자가 입력한 날짜를 변환할 수 있다.

NSCalendarDate는 포맷된 스트링과 컴포넌트 시간 값(분, 시간, 1주, 1년)에서 NSCalendarDate 객체를 생성하기 위한 메소드를 제공한다. 또한, 컴포넌트 시간값에 액세스하고, 다양한 포맷, 장소와 시간대로 날짜를 표현하기 위해 메소드를 사용하여 NSDate를 보완한다.

# 응용 프로그램 구축하기

이 섹션은 To Do 응용 프로그램에 대한 기본적인 구조를 생성하는 방법에 관해 설명한다. 먼저, NSDocument서브클래스를 생성한다. 다음, 응용 프로그램에 대한 데이터 모델 클래스를 구현하고, To Do 도큐멘트에 대한 기본적인 사용자 인터페이스를 설정한다. 마지막으로, ToDoDocument 클래스는 입력된 데이터를 허용, 정리, 제공 및 저장하기 위해 사용한 메소드를 구현한다. 이 장을 마치고 나면, 응용 프로그램을 통해 캘린더의 날짜를 클릭하고, To Do 아이템 리스트를 입력할 수 있다. 제13장 및 제14장, *To Do: 마무리 작업*은 To Do 응용 프로그램을 완전한 Mac OS X 응용 프로그램으로 만들기 위해 더 확장된 작업 내용을 설명한다.

## 프로젝트 만들기

Cocoa Document 기반의 응용 프로그램 탬플리트를 사용하여 To Do라는 새로운 Project Builder 프로젝트를 만든다.

Project Builder로 만들어진 모든 Cocoa 응용 프로그램 프로젝트는 동일한 `main` 기능(`main.m` 파일에서)을 제공한다. 사용자가 Finder에서 응용 프로그램이나 도큐멘트 아이콘을 더블 클릭하면, `main`(엔트리 포인트)이 처음으로 호출된다. `main`은 차례로 `NSApplicationMain`을 호출한다.

`NSApplicationMain`은 Cocoa 응용 프로그램을 구동하고, 운용하기 위해 이벤트에 응답하고, 객체 동작을 조정하는 등 필요한 작업을 수행한다. 이 기능은 메시지를 전송하는 응용 프로그램에서 객체 네트워크를 구동한다. `NSApplicationMain`은 다음과 같은 역할을 수행한다.

- 응용 프로그램 랩퍼에 속성리스트로써 저장되어 있는 응용 프로그램 속성을 확보한다. 이 속성 리스트에서 메인 nib 파일과 기본 클래스(응용 프로그램의 경우, NS응용 프로그램의 커스텀 서브클래스이거나 NSApplication이다)의 이름을 얻는다.
- NSApplication의 클래스 객체를 확보하여, `sharedApplication` 클래스를 생성한다. 또한, 전역 변수인 `NSApp`에 저장된 NSApplication의 인스턴스를 생성한다. NS응용 프로그램 객체를 생성하려면, 응용 프로그램을 윈도우 시스템과 Core Graphic 서버에 연결한다.
- NSApp을 소유자로 명시하여 메인 nib 파일을 로딩한다. 로딩은 응용 프로그램 객체를 복구하고, 재생성함으로써 객체간의 연결을 복구한다.
- 메인 이벤트 루프를 구동하여 응용 프로그램을 운용한다. 루프를 통해 응용 프로그램은 다음번 가용 이벤트를 확보하고, 응용 프로그램의 해당 객체로 이를 디스패치한다. 루프는 응용 프로그램 객체가 `stop:` 또는 `terminate:` 메시지를 수신할 때 까지 계속된다. 그 이후 루프는 해제된다.

각자 코드를 `main`에 추가하면 응용 프로그램의 시작 또는 종료 동작을 사용자화할 수 있다.

## *NSDocument의 서브클래스에 이름 바꾸기*

이 응용 프로그램의 NSDocument 서브클래스 이름을 제대로 바꾸려면 새로운 객체를 프로젝트에 추가하기에 앞서 이 단계를 완성해야 한다.

1. Project Builder에서 `MyDocument.nib`을 선택하고, Project 메뉴에서 Rename을 선택한 뒤, nib 파일명을 **`ToDoDocument.nib`**로 변경한다.
2. `MyDocument.h`를 **`ToDoDocument.h`**로 `MyDocument.m`을 **`ToDoDocument.m`**으로 변경한다.

3. Batch find/Replace를 사용하여 서브클래스 MyDocument 이름을 ToDoDocument로 변경한다.
   a. Find tab을 열어, 팝업 메뉴에서 Textual을 선택하였는지 확인한다.
   b. Find 필드에 **`MyDocument`**를 입력하고, Find를 클릭한다.
   c. 모든 히트가 `ToDoDocument.h`와 `ToDoDocument.m` 파일에 있는지 확인한다. 다른 파일이 있다면, 리스트에서 원하는 2개 파일을 선택한다.
   d. Replace 필드에 `ToDoDocument`를 입력하고, Replace를 클릭한다.
4. `ToDoDocument.nib`을 더블 클릭하고, Interface Builder에서 이를 연다.
5. nib 파일 윈도우의 Classes 패인에서 MyDocument를 더블 클릭하고, **`ToDoDocument`**라고 이름을 명명한다.
6. nib 파일 윈도우의 Instances 패인에서 File's Owner를 선택하고, Info 윈도우를 사용하여 클래스가 ToDoDocument인지 확인한다.
7. Project Builder에서 Targets 패인을 선택하고, To Do를 클릭해 응용 프로그램의 Settings 패인을 선택한다.
8. Expert 버튼을 클릭해 속성 리스트 편집기를 나타나게 한다.
9. CFBundleDocumentTypes 어레이를 열면, 0 딕셔네리가 있다.
10. NSDocument 클래스라는 속성을 찾아, 값(MyDocument)을 더블 클릭해 ToDoDocument로 변경한다.

초기 프로젝트 구조를 제대로 배치하였기 때문에 사용자 인터페이스에서 작업을 수행하기 시작한다.

## *Model 클래스(ToDoItem) 만들기*

ToDoItem 클래스는 To Do 응용 프로그램에 Model 객체를 제공한다. 인스턴스 변수는 To Do 리스트에서 수행해야 할 작업이나 지켜야할 의무조항 같은 아이템 요소를 정의하는 데이터를 보유한다. 또한, 이 데이터로 유익한 산출 작업을 수행한다. 따라서, ToDoItem은 데이터에 액세스하는 것 이상의 데이터와 동작을 캡슐화한다.

ToDoItem은 Model 클래스이기 때문에, 사용자 인터페이스로써의 역할을 수행하지 않는다. 그래서, Interface Builder를 사용하지 않고 클래스를 생성한다. 먼저, 프로젝트에 클래스를 추가한다. Project Builder는 탬플리트 소스 코드 파일 생성을 지원한다.

1. File 메뉴에서 New File을 선택한다.
2. Assistant에서 Objective-C 클래스 탬플리트를 선택한다.
3. 새로운 파일을 **ToDoItem.m**이라 명명한다.(ToDoItem.h는 자동으로 생성된다)
4. Finish를 클릭하여, 파일을 생성하고, 이를 프로젝트에 추가한다.

프로젝트를 자주 구축하여 오류를 빨리 찾아내야 한다. 또한, 응용 프로그램이 개발되는 방법에 대한 개념을 파악하고, 코딩을 직접 수행해봐야 한다.

## *인스턴스 변수 선언하기*

앞서 Travel Advisor에서 했듯이 헤더 파일에서 인스턴스 변수와 메소드를 선언하여 시작한다. <예제 12-1>에서 인스턴스 변수와 프로토콜을 **ToDoItem.h**에 추가한다. <표 12-1>은 변수를 상세하게 정의하여 제공한다.

*<예제 12-1> ToDoItem 인스턴스 변수 선언*

```
@interface ToDoItem : NSObject <NSCoding>
{
    NSCalendarDate *day;
    NSString *itemName;
    NSString *notes;
    NSTimer *timer;
    long secsUntilDue;
    long secsUntilNotify;
    ToDoItemStatus status;
}
```

*<표 12-1> ToDoItem 인스턴스 변수*

| 변수 | 설명 |
|---|---|
| day | to-do item의 날짜.(12:00 A.M으로 설정됨) |
| itemName | to-do item의 이름.(도큐멘트 텍스트 필드의 내용) |
| notes | Info 윈도우의 Notes 화면 내용. 이 변수는 회의에서 논의될 의제 등 to-do item 관련 정보로 사용될 수 있다. |
| timer | 통지 메시지의 타이머 |
| secsUntilDue | 아이템이 만료된 시간.(날짜가 만료된 자정이후 초 단위로 제공됨) |
| secsUntilNotify | 신호 통지가 전송된 시간.(만료 시간전에 초 단위로 제공됨) |
| status | INCOMPLETE, COMPLETE 또는 DEFER_TO_NEXT_DAY. |

## 메소드 선언하기

<예제 12-2>에서 **ToDoItem.h**로 메소드 선언을 추가한다. 앞 장에서 튜토리얼을 통해 작업을 수행했기 때문에 이 메소드에 익숙해야 한다.

*<예제 12-2> ToDoItem 메소드 선언*

```
- (id)initWithName:(NSString *)aName andDate:(NSCalendarDate *)aDate;
- (void)dealloc;

- (id)initWithCoder:(NSCoder *)coder;
- (void)encodeWithCoder:(NSCoder *)coder;

- (void)setDay:(NSCalendarDate *)newDay;
- (NSCalendarDate *)day;
- (void)setItemName:(NSString *)newName;
- (NSString *)itemName;
- (void)setNotes:(NSString *)newNotes;
- (NSString *)notes;
- (void)setTimer:(NSTimer *)newTimer;
- (NSTimer *)timer;
- (void)setStatus:(ToDoItemStatus)newStatus;
- (ToDoItemStatus)status;
- (void)setSecsUntilDue:(long)secs;
- (long)secsUntilDue;
- (void)setSecsUntilNotify:(long)secs;
- (long)secsUntilNotify;
```

## 상수 정의하기

상수는 **status** 인스턴스 변수 값이다.

```
typedef enum ToDoItemStatus
{
    INCOMPLETE=0,
    COMPLETE,
    DEFER_TO_NEXT_DAY
} ToDoItemStatus;
```

상수는 임시 값을 처리하는 메소드에서 편리함과 정확성을 제공한다.

```
enum {
    SECS_IN_MINUTE = 60,
    SECS_IN_HOUR = (SECS_IN_MINUTE * 60),
    SECS_IN_DAY = (SECS_IN_HOUR * 24),
    SECS_IN_WEEK = (SECS_IN_DAY * 7)
};
```

### 시간 전환 함수 선언하기

이 함수는 초 단위를 시간과 분으로 전환하고,(사용자 인터페이스에 의해 요구되었듯이) 시간과 분을 다시 초로 전환하는(ToDoItem에 의해 저장되었듯이) 산술화된 서비스를 클라이언트에게 제공한다.

```
long ConvertTimeToSeconds(int hour, int minute, BOOL pm);
BOOL ConvertSecondsToTime(long secs, int *hour, int *minute);
```

### 접근자 메소드 구현하기

이 클래스의 접근자 메소드를 구현할 때 장애가 발생하면 안된다. 한 가지 예외는 **setTimer:** 메소드이다. 이 메소드는 자동 해제하기에 앞서 타이머 기능을 억제하기 위해 **invalidate**를 **timer**로 전송해야만 한다는 점에서 다른 **set accessor** 메소드와 차이가 있다. Timer(NSTimer의 인스턴스)는 런 루프(NSRunLoop의 인스턴스)와 항상 연관이 있다. 추후, 런루프에 관해 좀더 설명한다.

### *Description* 메소드 구현하기

**Description** 메소드는 개발자들이 To Do 응용 프로그램을 디버깅할 때 도움이 된다. ToDoItem를 인수로 사용하여 **po**(print object) 명령어를 **GDB**에 입력할 때, 이 메소드가 발생하며, 중요한 디버깅 정보를 보여준다.

```
- (NSString *)description
{
    NSString *desc = [NSString stringWithFormat:
        @"%@\n\tName:%@\n \
        \tDate: %@\n \
        \tNotes: %@\n \
        \tCompleted: %@\n \
        \tSecs Until Due: %d\n \
        \tSecs Until Notify: %d\n",
        [super description],
        [self itemName],
        [self day],
        [self notes],
        (([self status] == COMPLETE) ? @"Yes":@"No"),
        [self secsUntilDue],
        [self secsUntilNotify]];

    return desc;
}
```

### *init*와 *dealloc* 구현하기

ToDoItem의 **initWithName:andDate:**와 **dealloc** 메소드 구현을 이해하면, 어떤 장애도 제공하지 않아야 한다. **initWithName:** 메소드는 새로운 ToDoItem의 이름과 날짜 속성을 생성하여 초기화한 뒤 설정할 수 있도록 한다.

**dealloc** 메소드는 **release**를 ToDoItem의 인스턴스 변수가 참조하는 객체로 전송한다.

<예제 12-3>으로 부터 **ToDoItem.m**에 코드를 추가한다.

*<예제 12-3> ToDoItem의 init 및 dealloc 메소드 구현*

```
- (id)initWithName:(NSString *)aName
    andDate:(NSCalendarDate *)aDate
{
    if ( self = [super init] ) {
        if ( !aName ) {
            [self release];
            return nil;
        }
        [self setItemName:aName];

        if ( aDate )
            [self setDay:aDate];
        else {
            NSCalendarDate *now = [NSCalendarDate date];
            [self setDay:[NSCalendarDate dateWithYear:[now yearOfCommonEra]
                    month:[now monthOfYear]
                    day:[now dayOfMonth]
                    hour:0 minute:0 second:0
                    timeZone:[NSTimeZone localTimeZone]]];
        }

        [self setStatus:INCOMPLETE];
        [self setNotes:@""];
    }

    return self;
}

- (void)dealloc
{
    [itemName release];
    [day release];
    [notes release];
    [timer invalidate];
    [timer release];

    [super dealloc];
}
```

## 아카이빙과 언아카이빙 구현하기

encodeWithCoder:와 initWithCoder:를 구현하려면, 다음 사항을 염두에 두어야 한다.

- 동일한 순서로 인스턴스 변수를 인코딩 및 디코딩한다.
- 디코딩후에 객체 인스턴스 변수를 복사 또는 보유한다.
- 도큐멘트를 열때 타이머를 재생성하여 리셋하였기 때문에 timer 인스턴스 변수를 아카이빙할 필요는 없다.

인코딩 및 디코딩 객체 유형은 표준 유형에서 수행하는 것과는 약간 차이가 있다. 다음을 참조한다.

```
[coder encodeValueOfObjCType:@encode(long) at:&secsUntilDue];
[coder decodeValueOfObjCType:@encode(long) at:&secsUntilDue];
```

## 시간 전환 기능 구현하기

ToDoItem 클래스를 생성하기 위한 마지막 단계에서 "Value-added" 동작을 제공하는 기능을 구현한다. 이 함수는 클래스 구현 뒤(즉 @end 문장 뒤)에서 바로 ToDoItem.m의 끝에 위치한다. ToDoItem의 시간 전환 기능은 다음과 같이 구현된다.

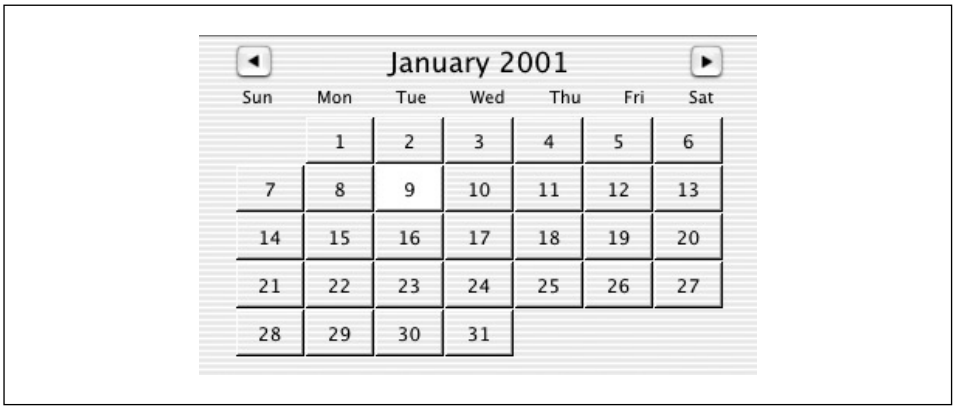
```
long ConvertTimeToSeconds(int hour, int minute, BOOL pm )
{
    if (hour == 12)
        hour = 0;
    if (pm)
        hour += 12;

    return ((hour * SECS_IN_HOUR) + (minute * SECS_IN_MINUTE));
}

BOOL ConvertSecondsToTime(long secs, int *hour, int *minute)
{
    BOOL pm = NO;

    *hour = secs / SECS_IN_HOUR;
    if (*hour > 11) {
        *hour -= 12;
        pm = YES;
    }
    if (*hour == 0)
        *hour = 12;

    *minute = ((secs % SECS_IN_HOUR) / SECS_IN_MINUTE);

    return pm;
}
```

`ConvertSecondsToTime` 함수는 A.M이나 P.M을 가르키기 위해 멀티플 값을 리턴하기 위한 방법으로 우회적인 방식을 사용하여 Boolean을 직접 리턴한다.

이제, 컴파일링을 하여 오류가 해결되었는지 확인한다.

## *Calendar 뷰 구현하기*

<그림 12-5>에서 제공된 To Do 인터페이스 캘린더는 NSMatrix의 커스텀 서브클래스 인스턴스이다. CalendarMatrix는 사용자가 새로운 달을 선택하면 동적으로 업데이트되며, 사용자가 날짜를 선택하면, 델리게이트를 통지하고, 오늘 날짜와 버튼 셀 속성을 설정하여 현재 선택한 날짜를 보여준다.




*<그림 12-5> 캘린더 뷰*

상속 계층에서 클래스의 서브클래스를 생성하면, 개발자는 NSObject의 단순한 서브클래스에서 보다 훨씬 더 많은 어려움에 직면한다. NSMatrix 같은 클래스는 NSObject보다 훨씬 특화되어 있으며, 더 많은 배기지(Baggage)를 운반한다. 또한, NSMatrix 클래스는 NSResponder, NSView, NSControl와 상당히 복잡한 Application Kit에서 상속을 받는다. CalendarMatrix는 NSView에서 상속받기 때문에, 사용자 인터페이스에 나타난다. Model-View-Controller 패러다임의 뷰 객체라 할 수 있다. 따라서, 재사용할 수 있다.

### *매트릭스가 슈퍼클래스일 경우*

특화된 슈퍼클래스를 서브클래스의 베이직으로 선택한 경우, 필요한 것이 무엇인지를 생각하고, 어떤 슈퍼클래스를 제공해야 할 것인지를 파악하는 게 중요하다. To Do의 동적 캘린더는 다음과 같다.

- 가로행과 세로열로 숫자(날짜)를 순서대로 정렬한다.
- 날짜 선택에 응답하고, 이를 전달한다.

- 날짜를 인식한다.
- 달을 검색한다.

Application Kit 클래스에서 참조 도큐멘트를 열어 NSMatrix 섹션을 검토하면, 다음 사항이 나와 있다.

> "NSMatrix는 다양한 방식으로 동작하는 NSCells 그룹을 생성하기 위해 사용되는 클래스이다. 또한, 가로행과 세로열로 NSCells를 정렬하는 메소드를 제공한다. NSMatrix는 매트릭스의 타겟과 액션 뿐만 아니라 각 NSCells의 타겟과 액션을 개별적으로 분리하여 NSControl의 타켓/액션 패러다임에 추가한다."

그래서, NSMatrix는 처음에 나오는 첫 번째 요구 사항과 두 번째 부분(선택에 응답하기)을 위해 상속 기능을 갖는다. 따라서, CalendarMatrix 서브클래스는 슈퍼클래스에서 어떤 것도 변경할 필요가 없다. 단지 날짜를 인식하고, 달을 검색하여, 선택이 되면 델리게이트에게 통보하도록 추가 데이터와 동작으로 NSMatrix를 보충해야 한다.

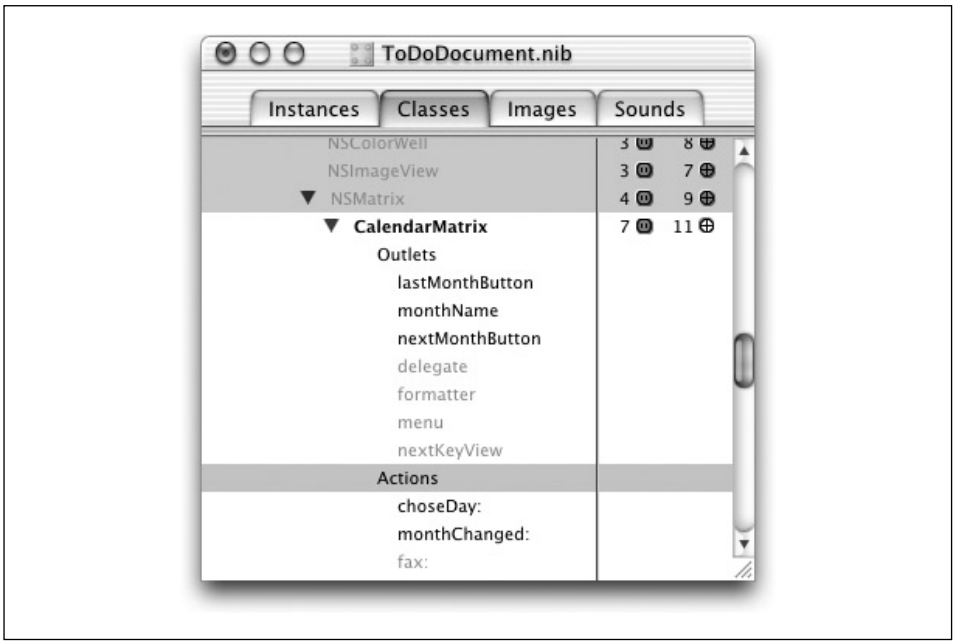
## *인터페이스 구성하기*

이 섹션에서는 CalendarMatrix 서브클래스를 만들고, Interface Builder의 캘린더 뷰 객체를 배치하는 방법을 설명한다.

1. Interface Builder에서 `ToDoDocument.nib`을 연다.
2. Classes 패인에서 NSMatrix를 서브클래스로 하여 CalendarMatrix를 생성한다.
3. <그림 12-6>에서 보이는 바와 같이 아웃렛과 액션을 추가한다. 슈퍼클래스가 정의하는 아웃렛과 액션이 흐릿하게 나타나는지 확인한다.

Currency Converter와 Travel Advisor 프로그램에서 NSObject의 서브클래스를 만들었다면, 그 다음에는 서브클래스를 인스턴스화해야 한다. CalendarMatrix는 뷰(NSView에서 상속됨)이기 때문에 인스턴스 생성 절차는 차이가 있다. 커스텀 NSView 객체를 CalendarMatrix 객체가 되는 사용자 인터페이스에 배치한다. CalendarMatrix 또는 ToDoDocument의 Interface Builder에서 인스턴스를 생성하지 않는다.

1. nib 파일의 메인 윈도우를 연다.
2. Your Document Contents를 삭제한다.
3. 도큐멘트 윈도우의 크기 조정 기능을 불러온다.
4. 가이드의 예제를 참조하여 윈도우를 크기 조정한다. 원한다면, Info 윈도우를 사용하여 도큐멘트 윈도우를 정확히 374×482로 만든다. 이것은 Apple과 O'Reilly 웹 사이트에서 다운로드하기 위해 사용할 수 있는 완성된 예제 응용 프로그램과 일치한다.

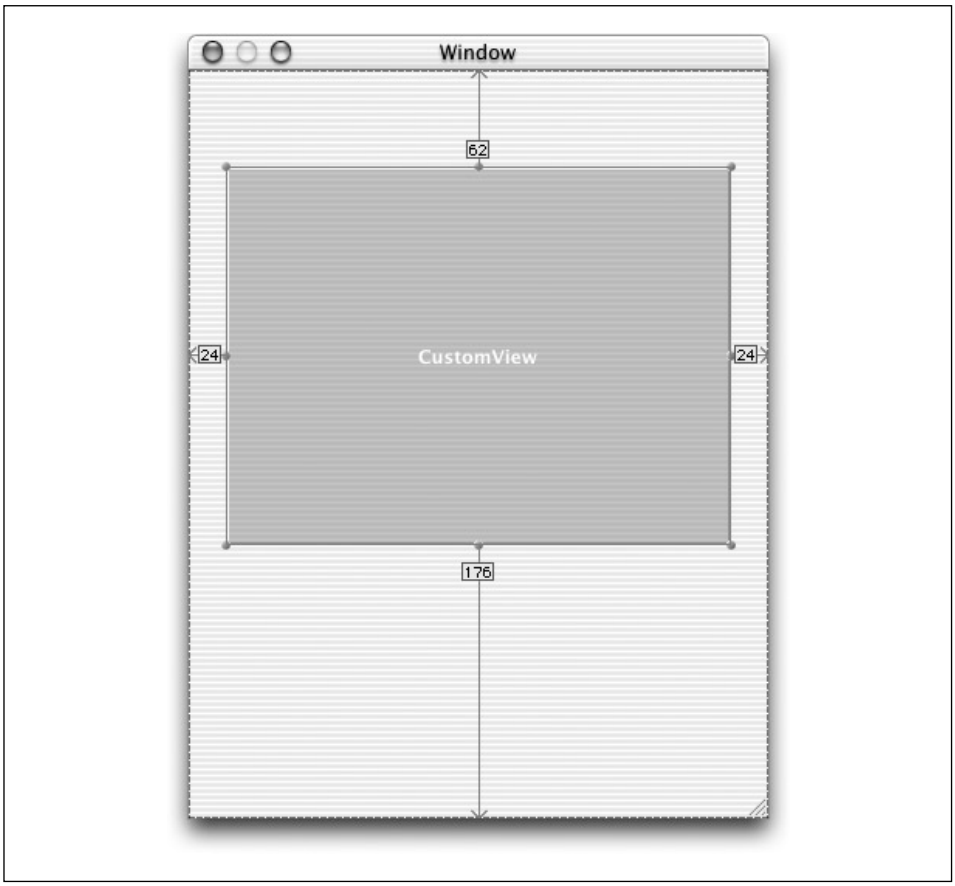



<그림 12-6> ToDoDocument 아웃렛과 액션

5. More Views 팔레트에서 윈도우로 커스텀 View를 드래그한다.
6. 커스텀 View의 크기를 조정하여 배치한다. Interface Builder는 Aqua 레이아웃 가이드라인에 기반한 Snap-to Guide를 제공하여 레이아웃을 지원한다. <그림 12-7>의 그림은 권장할 만한 레이아웃이다. 그러나, 원하는 크기와 위치를 자유롭게 선택해도 좋다.
7. 윈도우에서 커스텀 View를 선택하여 Info 윈도우의 Attributes 화면을 열어, 커스텀 View의 클래스를 CalendarMatrix에 설정한다. 커스텀 View 객체는 인터페이스에서 커스텀 NSView를 나타내는 "프록시" 객체이다. Info 윈도우의 클래스를 선택하여 클래스를 커스텀 View에 할당한다. 커스텀 뷰 프록시 객체와 연결된 커스텀 클래스는 nib 파일에 정의되어야 한다.
8. 마지막으로, 커스텀 View 객체에서 File's Owner 인스턴스로 Control 키를 누른채 드래그하여, CalendarMatrix의 델리게이트 아웃렛을 설정한다. 이 작업은 Calendar Matrix의 델리게이트로 ToDoDocument를 구축하는 것이다.
9. nib 파일을 저장한다.

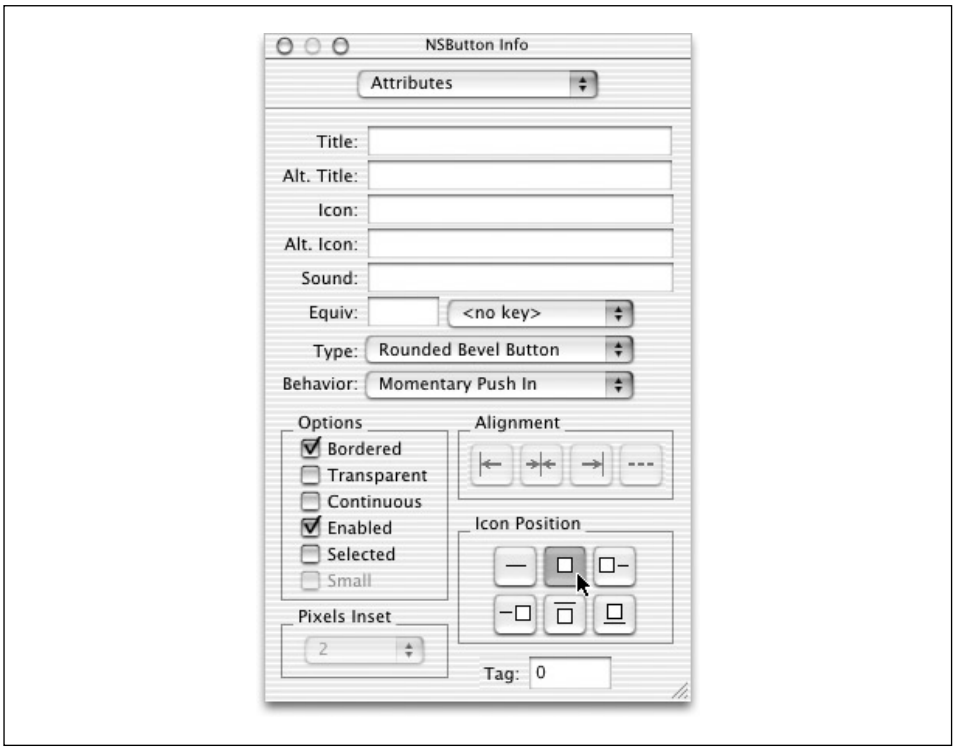
이제, CalendarMatrix에 연결된 컨트롤과 필드를 윈도우에 배치한다.

1. <그림 12-8>에서 보이는 바와 같이, Rounded Bevel Button 버튼을 인터페이스로 드래그하고, Info 윈도우를 사용하여 텍스트 라벨이 없는 Icon Position을 선택한다.




<그림 12-7> 캘린더 뷰 배치하기

2. Finder 윈도우를 열어, 완성된 샘플을 다운로드할 폴더를 검색한다. 제공된 아트 파일을 사용하고 싶지 않으면, 좌,우 화살표 버튼을 위해 자신의 아트를 만든다.
3. 소스를 배치하고, `LeftArrow.tiff`를 클릭하여 `ToDoDocument.nib` 윈도우의 Images 패인으로 드롭한다. 요청시 파일을 Project Builder 프로젝트에 삽입한다.
4. `LeftArrow.tiff`를 클릭하고, 버튼 위로 드롭한다.
5. 화살표에 맞추기 위해 버튼의 크기를 조정한다.
6. `RightArrow.tiff`를 사용하여 동일한 절차를 반복하여 오른쪽면 화살표를 만든다.
7. 요일을 개별 라벨로 입력하여 가로행에 정렬한다. 그리고, 라벨을 커스텀 뷰의 중앙에 놓는다.(이 작업은 시험용으로 수행되며, 일부 오류를 발생할 수 있다) 윈도우는 <그림 12-9>와 같이 보여져야 한다.

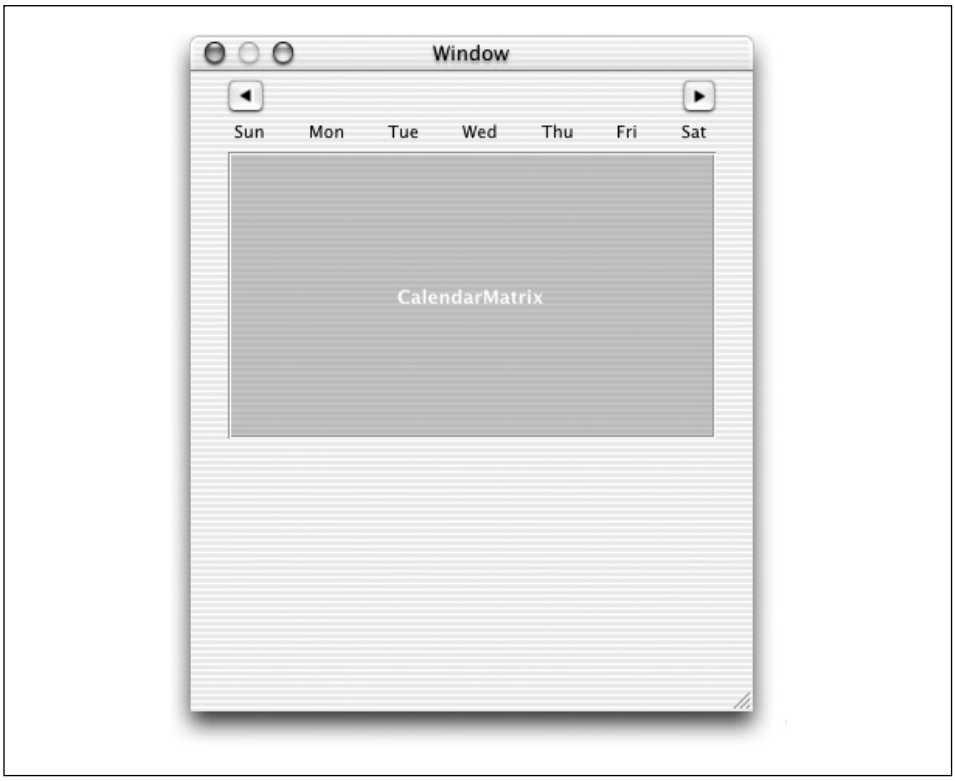



<그림 12-8> *Rounded Bevel Button 속성*

8. 월과 년도를 나타내기 위해 좌,우 화살표 사이에 텍스트 라벨을 추가한다. **`September 9999`**(긴 스트링도 가능)를 입력한 뒤 초기화한다. 텍스트를 Luucida Grande 18 포인트로 설정하고, 중앙에 배치한 뒤 텍스트를 삭제한다. 라벨의 편집과 선택이 불가능한지 확인한다.
9. <표 12-2>의 정보를 사용하여 CalendarMatrix를 위성 객체에 연결한다.

<표 12-2> *CalendarMatrix 연결*

| 이름 | 연결 | 유형 |
|---|---|---|
| monthName | CalendarMatrix에서 라벨 필드로 | 아웃렛 |
| prevMonthButton | CalendarMatrix에서 좌측 포인트 화살표로 | 아웃렛 |
| nextMonthButton | CalendarMatrix에서 우측 포인트 화살표로 | 아웃렛 |
| monthChanged | 양쪽 화살표에서 CalendarMatrix 액션으로 | 액션 |




<그림 12-9> 캘린더 뷰 레이아웃

10. `ToDoDocument.nib`을 저장한다.

11. Classes 화면에서 CalendarMatrix를 선택하고, 이를 위한 파일을 만든다.

좌측에 연결되지 않은 액션 메시지 `choseDay:`가 있는지 확인해야 한다. Interface Builder에서 객체를 연결할 수 없기 때문에 프로그램에서 연결해야한다.

## 선언을 헤더 파일에 추가하기

<예제 12-4>에서 보이는 바와 같이, Project Builder에서 `CalendarMatrix.h`를 변경한다. Interface Builder가 생성한 기존의 선언은 타원 괄호속에 나타난다.

<예제 12-4> CalendarMatrix 인터페이스 추가

```
@interface CalendarMatrix : NSMatrix
{
    @private
    IBOutlet NSButton *lastMonthButton;
    IBOutlet NSTextField *monthName;
    IBOutlet NSButton *nextMonthButton;
```

<예제 12-4> CalendarMatrix 인터페이스 추가 (계속)

```
    NSCalendarDate *selectedDay;
    short startOffset;
}
    /* ... */
- (id)initWithFrame:(NSRect)frameRect;
- (void)setSelectedDay:(NSCalendarDate *)newDay;
- (NSCalendarDate *)selectedDay;
@end

@interface NSObject(CalendarMatrixDelegate)
 - (void)calendarMatrix:(CalendarMatrix *)sender
       didChangeToDate:(NSDate *)date;
@end
```

이 선언은 주의할 몇 가지 흥미로운 사항들이 있다.

- 컴파일러는 `@Private` 키워드를 통해 인스턴스 변수가 다른 객체에 의해 직접 액세스될 수 없도록 한다. 이로써, 네트워크의 다른 객체는 CalendarMatrix와 상호작용할 때 접근자 메소드 사용을 보장받는다.
- CalendarMatrix의 셀은 태그 번호를 좌우상하로 순차적으로 할당한다. 인스턴스 변수인 `startOffset`은 새로운 달의 첫째 날에 해당하는 셀의 태그 번호를 제공한다.
- `CalendarMatrixDelegate`는 델리게이트가 구현한 메소드를 선언한 NSObject 카테고리이다. 이 기법은 델리게이션 메소드에 일반적으로 사용되는 Informal 프로토콜을 생성한다.

## *CalendarMatrix의 Private 메소드 선언하기*

*Private 메*소드는 다른 객체들이 직접 발생시키지 않는 CalendarMatrix의 "내부" 메소드이다. 다른 객체들은 이 메소드를 발생시키려 하지 않기 때문에 클래스의 헤더 파일에 이들을 배치하여 외부에 존재를 노출시키지 않는다. 이 메소드는 독립적인(전용) 헤더 파일이나 클래스의 구현 파일에 선언될 수 있다. 이 클래스에는 Private 메소드가 많지 않기 때문에 구현 파일에 있는 메소드를 사용한다.

1. `CalendarMatrix.m`을 연다.
2. `#import` 및 `@implementation` 사이에 다음 코드를 추가한다.

```
@interface CalendarMatrix (PrivateMethods)
- (void)refreshCalendar;
- (void)highlightTodayIfVisible;
@end
```

이 코드는 CalendarMatrix에서 **Private** 메소드 카테고리를 선언한다. 이 선언 기법은 컴파일러가 메소드를 인식할 수 있도록 한다. 그러나, 외부에서는 확인할 수 없다.

## *initWithFrame:*와 *dealloc* 구현하기

제8장, *이벤트 처리*에서 NSView 서브클래스의 커스텀 초기화를 수행하기 위해 NSView의 **initWithFrame:** 메소드를 오버라이드할 수 있다. CalendarMatrix는 NSView에서 상속한 NSControl의 서브클래스인 NSMatrix의 서브클래스이다. 그래서 동일한 절차를 적용한다.

<예제 12-5>에서 주어진 **initWithFrame:** 구현을 **CalendarMatrix.m**에 추가한다.

*<예제 12-5> CalendarMatrix의 initWithFrame: 메소드 구현*

```
- (id)initWithFrame:(NSRect)frameRect
{
    int i, j;
    int count = 0;
    id cell = [[NSButtonCell alloc] initTextCell:@""];
    NSCalendarDate *now = [NSCalendarDate date];

    [cell setShowsStateBy:NSOnOffButton];

    [super initWithFrame:frameRect
            mode:NSRadioModeMatrix
            prototype:cell
            numberOfRows:6
            numberOfColumns:7];

    // set cell tags
    for (i=0; i<6; i++) {
        for (j=0; j<7; j++) {
            [[self cellAtRow:i column:j] setTag:count++];
        }
    }

    [cell release];

    selectedDay = [[NSCalendarDate dateWithYear:[now yearOfCommonEra]
            month:[now monthOfYear]
            day:[now dayOfMonth]
            hour:0 minute:0 second:0
            timeZone:[NSTimeZone localTimeZone]] retain];

    return self;
}
```

구현 내용은 다음과 같다.

- NSDate이 선언한 클래스 메소드 **date**는 NSCalendarDate로써 현재 날짜(오늘 날짜)를 리턴한다.(NSCalendarDate는 NSDate의 서브클래스이다)
- **super**(NSMatrix) 메시지는 매트릭스 범위를 설정하고, 프로토타입(NSButtonCell)을 사용하여 셀 유형을 식별한 뒤 언제든지 1개의 버튼만을 선택할 수 있다는 것을 의미하는 매트릭스의 일반적 동작(Radio 모드)을 명시한다.
- **for** 루프는 각 셀의 태그 번호를 좌우로 순차적으로 설정한다. 태그는 CalendarMatrix가 셀의 요일 번호를 설정하여, 검색할 수 있도록 하는 메카니즘이다.
- **selectedDay** 인스턴스 변수는 NSCalendarDate 클래스 메소드를 사용하여 초기화된다. 이 메소드는 현재 날짜의 년도, 월 및 요일 요소를 통해 오늘자 자정으로 설정된 NSCalendarDay를 리턴한다. **localTimeZone** 메시지는 Greenwich Mean Time을 기반하여 NSTimeZone 객체를 확보한다.

초기화 코드를 확인했으므로, CalendarMatrix의 **dealloc**을 구현하는데 문제가 없어야 한다.

## *awakeFromNib 구현하기*

**awakeFromNib** 메소드는 추가적인 초기화(일부는 **initWithFrame:**에서 간단하게 수행됨)를 수행한다. 중요한 것은 **self**를 각자 타겟 객체로써 설정하고, Interface Builder에서 수행될 수 없는 타겟인 **choseDay:**의 액션 메소드를 명시한다.

<예제 12-6>에서 주어진 **awakeFromNib** 코드를 추가한다.

*<예제 12-6> CalendarMatrix의 awakeFromNib 메소드 구현*

```
- (void)awakeFromNib
{
    [self setTarget:self];
    [self setAction:@selector(choseDay:)];
    [self setAutosizesCells:YES];
    [self refreshCalendar];
    [self choseDay:self];
}
```

구현 사항은 다음과 같다.

- **setAutosizesCells:**은 다시 드로잉 할때마다 셀 크기를 조정하기 위한 매트릭스를 불러온다.
- **refreshCalendar**(다음번에 작성할것임)는 캘린더를 업데이트하는것이다.

- **choseDay:**(또한 다음에 작성할것임)는 CalendarMatrix 클릭으로 발생된 액션 메소드이다. 이 메소드는 매트릭스 화면을 초기화하기 위해 알맞은 방식이다. 이 메소드를 통해 사용자가 오늘 날짜 셀을 클릭했다는 것을 인식한다.

CalendarMatrix 구현 코드로 작업을 수행한다. 진행하기에 앞서, **selectedDay** 인스턴스 변수의 접근자 메소드를 구현한다.

## *refreshCalendar 구현하기*

**refreshCalendar** 메소드는 클래스의 견인차 역할을 하며, 상당히 길고, 복잡하게 구현된다. 이 섹션에서는 이 메소드에 관해 설명한다.

캘린더를 업데이트하려면, 먼저, 정적 **gNumDaysInMonth[]** 어레이를 초기화하고, **isLeap** 매크로를 작성한다.

```
// "1" based array
containing the number of days in each month
static short gNumDaysInMonth[] =
    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

#define isLeap(year) (((year % 4) == 0 &&
((year % 100) != 0)) || (year % 400) == 0)
```

기능을 정의하고, 지역 변수를 선언한다.

```
- (void)refreshCalendar
{
    NSCalendarDate *firstOfMonth, *selDate;
    int i, currentMonth, dayLabel;
    unsigned int currentYear;
    short daysInMonth;
    id cell;
```

제공된 월에 날짜를 쓰기 전에, CalendarMatrix는 구동하는 셀이 어떤 것이며, 몇 개의 셀을 날짜에 입력할 것인지 확인해야 한다. 이 값을 산출하여 **refreshCalendar** 메소드를 시작한다.

```
    selDate = [self selectedDay];
    currentMonth = [selDate monthOfYear];
    currentYear = [selDate yearOfCommonEra];
```

현재 선택한 달과 년도의 첫째 날(**selectedDay** 인스턴스 변수에서 산출됨)에 대한 NSCalendarDate을 만든다.

```
firstOfMonth = [NSCalendarDate dateWithYear:currentYear
        month:currentMonth
        day:1 hour:0 minute:0 second:0
        timeZone:[NSTimeZone localTimeZone]];
```

월과 년도(예를 들면, 2001년 2월)를 캘린더 위의 라벨에 작성한다.

```
[monthName setStringValue:[firstOfMonth
        descriptionWithCalendarFormat:@"%B %Y"]];
```

gNumDaysInMonth 정적 어레이에서 해당 달의 날짜를 확보한다. 윤달 2월로 설정할 경우, 날짜를 조정한다.

```
/* correct Feb for leap year */
daysInMonth = gNumDaysInMonth[currentMonth];
if ((currentMonth == 2) && (isLeap(currentYear)))
    daysInMonth++;
```

달의 첫째 날 요일을 확보하고, startOffset 인스턴스 변수에 저장한다.

```
startOffset = [firstOfMonth dayOfWeek];
```

날짜를 셀에 작성하고 셀 속성을 설정한다.

```
dayLabel = 1; //start numbering days from the 1st

for (i=0; i < 42; i++) {
    cell = [self cellWithTag:i];
    if (i < startOffset || i >= (daysInMonth + startOffset)) {
        // Blank out unused cells in the matrix
        [cell setBordered:NO];
        [cell setEnabled:NO];
        [cell setTitle:@""];
        [cell setCellAttribute:NSCellHighlighted to:NO];
    } else {
        // Fill in valid days in the matrix
        [cell setBordered:YES];
        [cell setEnabled:YES];
        [cell setFont:[NSFont systemFontOfSize:12]];
        [cell setTitle:[NSString stringWithFormat:@"%d", dayLabel++]];
        [cell setCellAttribute:NSCellHighlighted to:NO];
    }
}
```

for 루프는 달의 날짜의 일부에 해당되지 않은 첫번째 셀과 마지막 셀을 제거하고, startOffset에서 시작하여 daysInMonth도 지속되는 달의 날짜를 작성한다. 그 과정에서 폰트(선택된 날짜는 볼드체로 제공됨)와 다른 셀 속성을 리셋한다.

셀의 매트릭스를 다시 드로잉하고, 요일에 정확하게 라벨을 부착한 후, 현재 선택한 날짜(selectedDay 인스턴스 변수에 저장됨)에 하이라이트를 주기 위해 뷰를 업데이트해야 한다. 셀을 클릭하면, 셀에 하이라이트가 제공되기 때문에 중복될 수 있지만, 이 단계는 "새로운 달"(사용자가 nextMonth 및 lastMonth 버튼을 클릭한 경우)과 "첫 달"(응용 프로그램이 초기에 캘린더 매트릭스를 드로잉한 경우)을 정상적으로 처리하기 위해 필요하다.

달이 바뀌면, 사용자는 셀을 클릭할 기회가 없어진다. 그래서, 매트릭스는 어떤 날짜가 현재인지 파악하지 못한다. **monthChanged:** 액션 메소드의 코드는 **selectedDay**를 첫째날로 설정하여 처리한다. 그러나, 이 방법 단독으로는 첫째날에 해당하는 셀을 뷰에서 선택할 수 없다. **selectCellWithTag:** 메소드는 매트릭스 뷰에서 해당 셀을 선택한다. 여기에 **selectCellWithTag:**를 배치하여(monthChanged:를 직접 배치하는 대신) "첫번째 실행"을 처리한다.

```
        [self selectCellWithTag:([selDate dayOfMonth] + startOffset - 1)];
```

**refreshCalendar** 메소드의 첫 단계에서는 Today 셀 속성을 리셋하면 된다. **highlightTodayIfVisible** 메소드는 현재 나타난 달이 오늘 날짜를 포함하고 있는지, 그래서 오늘자에 해당하는 셀에 하이라이트를 주는지 여부를 결정한다.

```
        [self highlightTodayIfVisible];
    }
```

## *highlightTodayIfVisible 구현하기*

이 "헬퍼" 메소드를 위해 <예제 12-7>에서 **highlightTodayIfVisible** 구현을 추가한다.

*<예제 12-7> CalendarMatrix의 highlightTodayIfVisible 메소드 구현*

```
- (void)highlightTodayIfVisible
{
    NSButtonCell *aCell;
    NSCalendarDate *now = [NSCalendarDate date];
    NSCalendarDate *selDate = [self selectedDay];

    if (([selDate yearOfCommonEra] == [now yearOfCommonEra]) &&
        ([selDate monthOfYear] == [now monthOfYear]) &&
        ([selDate dayOfMonth] != [now dayOfMonth]))
    {
        aCell = [self cellWithTag:([now dayOfMonth] + startOffset - 1)];
        [aCell setHighlightsBy:NSMomentaryChangeButton];
        [aCell setCellAttribute:NSCellHighlighted to:YES];
    }
}
```

## *choseDay:* 액션 메소드 구현하기

<예제 12-8>은 사용자가 캘린더에서 날짜를 선택할 때 발생하는 동작을 명시한다.

*<예제 12-8> CalendarMatrix의 choseDay: 메소드 구현*

```
- (IBAction)choseDay:(id)sender
{
    NSCalendarDate *selDate, *prevSelDate;
    unsigned int selDay;

    prevSelDate = [self selectedDay];

    selDay = [[self selectedCell] tag] - startOffset + 1;
    selDate = [NSCalendarDate dateWithYear:[prevSelDate yearOfCommonEra]
            month:[prevSelDate monthOfYear]
            day:selDay
            hour:0
            minute:0
            second:0
            timeZone:[NSTimeZone localTimeZone]];

    [self setSelectedDay:selDate];
    [self highlightTodayIfVisible];

if ([[self delegate] respondsToSelector:
        @selector(calendarMatrix:didChangeToDate:)])
    {
    [[self delegate] calendarMatrix:self didChangeToDate:selDate];
    }
}
```

이 메소드는 사용자가 캘린더에서 날짜를 클릭하면 발생한다. 먼저, 선택된 셀의 태그 번호를 얻어, 그 태그 번호에서 시작 번호를 가감한 뒤(제로 베이스 인덱싱을 위해 1을 더해 조정), 선택한 날짜를 찾는다. 그 다음, 선택한 날짜를 나타내는 NSCalendarDate를 유도한다. 마지막으로, selectedDay 인스턴스 변수를 새로운 날짜에 설정하고, 오늘자 날짜에 하이라이트를 준다. 그리고, calendarMatrix:didChangeToDate: 메시지를 델리게이트로 전송한다.

## *monthChanged:* 액션 메소드 구현하기

<예제 12-9>는 사용자가 새로운 달을 선택하면 발생하는 동작을 명시한다. CalendarMatrix 위의 화살표 버튼을 클릭하면, **monthChanged:** 메시지를 전송한다. 이 메소드는 달을 검색하기 위해 달력을 앞 뒤로 넘겨볼 수 있도록 한다.

<예제 12-9> CalendarMatrix의 monthChanged: 메소드 구현

```
- (IBAction)monthChanged:sender
{
    NSCalendarDate *thisDate = [self selectedDay];
    int currentYear = [thisDate yearOfCommonEra];
    unsigned int currentMonth = [thisDate monthOfYear];

    if (sender == nextMonthButton) {
        if (currentMonth == 12) {
            currentMonth = 1;
            currentYear++;
        } else {
            currentMonth++;
        }
    } else {
        if (currentMonth == 1) {
            currentMonth = 12;
            currentYear--;
        } else {
            currentMonth--;
        }
    }
    [self setSelectedDay:[NSCalendarDate dateWithYear:currentYear
            month:currentMonth
            day:1 hour:0 minute:0 second:0
            timeZone:[NSTimeZone localTimeZone]]];

    [self refreshCalendar];
    [self choseDay:self];
}
```

먼저, **monthChanged:** 메소드를 통해 어떤 버튼으로 메시지를 전송할 것인지 결정하고, 달을 증가 또는 감소시킨다. 년도의 첫 달 또는 마지막 달을 지나쳤다면, 년도를 증가 또는 감소시켜, 달을 조정한다. 다음은, 새로운 달(및 년도) **selectedDay**로 인스턴스변수를 리셋하고, **refreshCalendar**를 발생시켜, 새로운 달을 나타낸다. 마지막으로, **choseDay:** 메소드를 발생시켜, 뷰를 올바르게 업데이트한다.

## 구축 및 시험하기

축하합니다! CalendarMatrix 작업을 완료했다. 이제, 응용 프로그램을 컴파일링하여, 지금까지 구현한 동작을 시험할 수 있다. 화살표 버튼을 클릭하면, CalendarMatrix는 다음 또는 이전 달을 보여준다. 요일이 하이라이트되며, 현재 날짜 또한 하이라이트된다. 새로운 To Do 도큐멘트를 생성하여 닫을 수 있다.

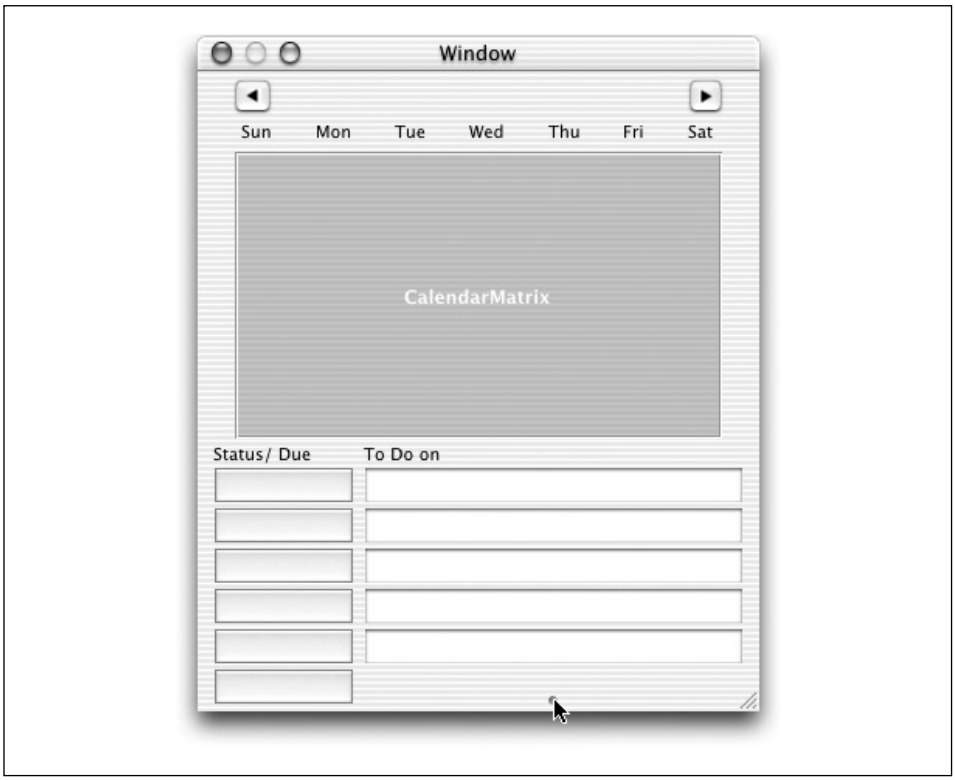
## 도큐멘트 인터페이스 완성하기

To Do 도큐멘트를 위해 남아 있는 사용자 인터페이스 아이템을 구성하기 전에, <예제 12-10>에서 선언을 ToDoDocument.h에 추가한다. 헤더 파일에 추가한 아웃렛은 Interface Builder에서 도큐멘트 윈도우에 추가된 인터페이스 객체에 해당한다. 다른 인스턴스 변수와 메소드 선언은 남은 튜토리얼에서 완성할 ToDoDocument 구현을 용이하게 한다.

*<예제 12-10> ToDoDocument 인스턴스 변수 및 메소드 선언*

```
#import "ToDoItem.h"
#import "CalendarMatrix.h"

@interface ToDoDocument : NSDocument {
    @private
    IBOutlet CalendarMatrix *calendar;
    IBOutlet NSTextField *dayLabel;
    IBOutlet NSMatrix *itemList;
    IBOutlet NSMatrix *statusList;
    NSMutableDictionary *activeDays;
    NSMutableArray *currentItems;
    ToDoItem *selectedItem;
    BOOL selectedItemEdited;
}

- (IBAction)itemStatusClicked:(id)sender;

- (ToDoItem *)selectedItem;

- (void)calendarMatrix:(CalendarMatrix *)matrix
    didChangeToDate:(NSDate *)date;
@end
```

ToDoDocument.nib을 열어, Project Builder에서 Interface Builder의 ToDoDocument.nib 윈도우로 ToDoDocument.h를 드래그한다. 이 작업은 IB가 새로운 아웃렛 및 액션 선언을 읽을 수 있도록 한다. 이제, 남은 작업은 <그림 12-10>에서 보이는 바와 같이, 응용 프로그램의 도큐멘트 인터페이스에 텍스트 필드와 라벨의 메트릭스를 추가하는 일이다. 해당 객체의 핸들을 Option 키를 누른채 드래그하여 매트릭스를 생성한다.




<그림 12-10> 텍스트 필드 매트릭스 생성하기

<표 12-3>에서 보이는 바와 같이 ToDoDocument의 아웃렛과 액션을 연결한다.

<표 12-3> ToDoDocument 아웃렛과 액션

| 이름 | 연결 | 유형 |
|---|---|---|
| calendar | File's Owner에서 CalendarMatrix 객체로 | 아웃렛 |
| dayLabel | File's Owner에서 To Do On 라벨로 | 아웃렛 |
| itemList | File's Owner에서 긴 텍스트 필드의 매트릭스로 | 아웃렛 |
| statusList | File's Owner에서 짧은 텍스트 필드 매트릭스로 | 아웃렛 |
| itemStatusClicked | 짧은 텍스트 필드 매트릭스에서 File's Owner로 | 액션 |

## *init와 dealloc 구현하기*

다음의 `init`와 `dealloc`의 구현이 기준이기 때문에, 더 이상 논의할 필요는 없다. <예제 12-11>의 코드를 `ToDoDocument.m`에 추가한다.

<예제 12-11> ToDoDocument의 init와 dealloc 메소드 구현

```
- (id)init
{
    /* Make sure [super init] is successful before trying to set ivars */
    if (self = [super init]) {
        activeDays = nil;
        currentItems = nil;
        selectedItem = nil;
        selectedItemEdited = NO;
    }

    return self;
}

- (void)dealloc
{
    [activeDays release];
    [currentItems release];

    [[NSNotificationCenter defaultCenter] removeObserver:self];

    [super dealloc];
}
```

## 접근자와 액션 메소드 구현하기

**selectedItem** 접근자 메소드 구현을 추가한다. **selectedItem** 인스턴스 변수의 내용을 리턴해야 한다. 사용자가 사용자 인터페이스에서 ToDo 아이템을 클릭하면 **selectedItem**을 설정하는 코드를 튜토리얼에 작성한다.

**itemStatusClicked:** 액션 메소드의 비어있는(Stub) 구현을 추가한다. Stub는 헤더 파일에서 선언된 메소드를 클래스가 모두 구현하지 않는다고 해서 컴파일러가 불평하지 못하도록 하는 비어있는 메소드 구현이다. 추후 튜토리얼에서 뭔가 유용한 내용으로 Stub를 채울 것이다.

## ToDoDocument의 델리게이트 메소드 구현하기

상기 언급했듯이, CalendarMatrix는 델리게이트가 동작에 연결될 수 있도록 2개의 메소드를 선언했다. 응용 프로그램의 델리게이트는 ToDoDocument이다. **calendarMatrix:didChangeToDate:** 메소드의 첫번째 버전을 구현한다.

```
- (void)calendarMatrix:(CalendarMatrix *)matrix
    didChangeToDate:(NSDate *)date
{
    [dayLabel setStringValue:[date descriptionWithCalendarFormat:
            @"To Do on %a %B %d %Y" timeZone:[NSTimeZone localTimeZone]
            locale:nil]];
}
```

캘린더는 사용자가 새로운 날짜를 클릭하면, `calendarMatrix:didChangeToDate:`를 전송한다. "CalendarMatrix 델리게이트 메소드 업데이트하기"에서 이 메소드에 대한 더욱 복잡한 버전을 구현할 것이다. 그러나, 여기에서는 단순히 인터페이스의 `dayLabel`를 변경하여 사용자가 클릭한 날짜와 매칭하는 작업을 수행한다.

### *구축 및 시험하기*

이제, 작업을 점검하기로 하자. Calendar Matrix에서 날짜를 클릭하고, `dayLabel` 변경을 확인할 수 있어야 한다.

## *To Do 날짜 관리 및 화면 조정하기*

이 장 초기에서 설명한 To Do 디자인에 관한 내용을 잠시 상기해보면, 응용 프로그램의 진짜 날짜는 Model 클래스인 ToDoItem의 인스턴스로 구성된다는 것을 알 수 있다. To Do는 이들 객체를 어레이에 저장하고, 어레이를 딕셔네리에 저장한다. 또한, 날짜를 특정 어레이에 액세스하기 위한 키로 사용한다.(딕셔네리와 어레이는 수정 가능하다) 또한, 이 디자인은 도큐멘트 인터페이스의 텍스트 필드와 어레이의 "슬롯" 사이에서 위치적인 일치성에 따라 결정된다.

이 섹션에서는 ToDoDocument 클래스가 입력된 데이터를 허용, 정리, 제공 및 저장하는 프로세스 구현에 관해 설명한다. 또한, CalendarMatrix 객체에서 선택한 데이터를 어떻게 보여주고, 조작하는지에 관해서도 설명한다.

### *ToDoDocument의 Private 메소드 선언하기*

이 섹션에서 생성할 고급 ToDoDocument 구현은 여러 "Helper" 메소드를 사용하여 수행된다. 응용 프로그램의 다른 객체가 이 메소드를 굳이 알 필요가 없기 때문에 클래스의 구현 파일에서 이 메소드를 단독으로 선언한다. 이 섹션의 뒷 부분에 구현 내용이 추가되어 있기 때문에 그 부분에서 이 메소드를 자세하게 논의할 수 있다.

<예제 12-12>에서 `ToDoItem.m`의 상단에 있는 `#import` 문장 뒤에 코드를 추가한다.

*<예제 12-12> ToDoDocument의 Private 메소드 선언*

```
@interface ToDoDocument (PrivateMethods)
- (void)setCurrentItems:(NSMutableArray *)newItems;
- (void)updateLists;
- (void)saveDocItems;
- (void)selectItemAtRow:(int)row;
- (void)initDataModelWithDictionary:(NSMutableDictionary *)dict;
@end
```

## *windowControllerDidLoadNib:* 변경하기

앞서 언급했듯이, 이 메소드는 nib 파일을 로딩하였을때 도큐멘트 객체를 요청한다. 그래서, awakeFromNib로써의 기능을 수행한다. 기본적인 구현을 변경하면, <예제 12-13>과 같이 나타난다.

*<예제 12-13> ToDoDocument의 windowControllerDidLoadNib 메소드 구현*

```
- (void)windowControllerDidLoadNib:(NSWindowController *)aController
{
    [super windowControllerDidLoadNib:aController];

    // No undo manager implemented for To Do Documents
    [self setHasUndoManager:NO];

    // Make the document the delegate of the UI's To Do item list
    [itemList setDelegate:self];

    // Init data model
    [self initDataModelWithDictionary:nil];
}
```

위의 내용은 설명할 필요가 없다. 이 메소드는 super 메시지 뒤에 도큐멘트 객체를 구성하여 동작중인 Undo Manager가 없다는 것을 인식한다. 그 다음, 아이템 리스트의 델리게이트라고 선언한다. 그리고, nil을 전송한 내부 데이터 모델을 초기화하여, 객체가 기본적인 초기 데이터 구조로 생성될 수 있도록 한다. 마지막으로 이 메소드는 도큐멘트 윈도우 키를 생성하여 맨 앞에 배열한다.

## *initDataModelWithDictionary:* 구현하기

ToDoDocument.nib이 로딩하면, windowControllerDidLoadNib:이 initDataModelWithDictionary: 메소드를 발생시킨다. 이 메소드는 To Do 도큐멘트 객체의 중앙 데이터 구조인 activeDays 딕셔네리를 관리하는 방식을 제공한다. 이 메소드는 set accessor 메소드로 동작하며, nil 인수를 처리하는 방법을 제외하곤, 다른 메소드와 유사하다. 이 경우, nil에는 딕셔네리가 존재하지 않는다. 따라서, 딕셔네리를 생성해야 한다. <예제 12-14>의 코드를 ToDoDocument.m에 추가한다.

*<예제 12-14> ToDoDocument의 initDataModelWithDictionary: 메소드*

```
- (void)initDataModelWithDictionary:(NSMutableDictionary*)dict
{
    NSDate *date;

    [activeDays autorelease];

    if (dict)
        activeDays = [dict retain];
    else
        activeDays = [[NSMutableDictionary alloc] init];
```

<예제 12-14> ToDoDocument의 initDataModelWithDictionary: 메소드 (계속)

```
    date = [calendar selectedDay];
    [self setCurrentItems:[activeDays objectForKey:date]];
}
```

이 메소드에서 To Do 도큐멘트 데이터 모델을 초기화하면 제13장의 디스크에서 도큐멘트 데이터 로딩을 처리할 때 명확해진다.

## currentItems 접근자 메소드 구현하기

이 메소드는 **set accessor** 메소드로 동작한다. **nil**을 허용하면, **setCurrentItems:**가 어레이를 생성할 뿐만 아니라, 비어있는 스트링 객체를 사용하여 "초기화"한다. 이 같은 작업은 NSmutableArray 메소드가 어레이의 범위내에서 **nil**을 묵인하기 때문에 가능하다. <예제 12-15>를 **ToDoDocument.m**에 추가한다.

<예제 12-15> ToDoDocument의 setCurrentItems: 메소드 구현

```
- (void)setCurrentItems:(NSMutableArray *)newItems
{
    int numRows, numCols;

    [currentItems autorelease];

    if (newItems)
        currentItems = [newItems mutableCopy];
    else {
        [itemList getNumberOfRows:&numRows columns:&numCols];
        currentItems = [[NSMutableArray alloc]
        initWithCapacity:numRows];
        while (numRows--)
            [currentItems addObject:@""];
    }
}
```

## updateLists: 구현하기

<예제 12-16>의 **updateLists:** 메소드는 현재 아이템 어레이를 사용하여 도큐멘트 인터페이스를 업데이트한다. 어레이의 아이템을 변경하면, 다른 ToDoDocument 메소드(응용 프로그램의 다른 객체뿐만 아니라)가 호출되어, 사용자 인터페이스를 업데이트한다.

<예제 12-16> ToDoDocument의 updateLists 메소드 구현

```
- (void)updateLists
{
    int i, numRows;
    ToDoItem *thisItem;
    NSDate *due;

    numRows = [[itemList cells] count];
```

*<예제 12-16> ToDoDocument의 updateLists 메소드 구현 (계속)*

```
    // For each row that has a valid To Do item
    //   update its name, due time, and status.
    for (i = 0; i < numRows; i++) {
        thisItem = [currentItems objectAtIndex:i];
        if ([thisItem isKindOfClass:[ToDoItem class]])
        {
            if ( [thisItem secsUntilDue] ) {
                due = [[thisItem day]
                        addTimeInterval: [thisItem secsUntilDue]];
            } else
                due = nil;

            [[itemList cellAtRow:i column:0]
                    setStringValue:[thisItem itemName]];
            // [[statusList cellAtRow:i column:0] setTimeDue:due];
            // [[statusList cellAtRow:i column:0]
            //          setTriState:[thisItem status]];
        } else  {
            [[itemList cellAtRow:i column:0] setStringValue:@""];
            [[statusList cellAtRow:i column:0] setTitle:@""];
            [[statusList cellAtRow:i column:0] setImage:nil];
        }
    }
}
```

**updateLists** 메소드는 **currentItems** 어레이의 아이템(ToDoItems)명을 **itemList** 텍스트 필드에 작성한다. 또한, **itemList** 옆의 매트릭스(**statusList**) 셀의 비쥬얼 상태를 업데이트한다. 이들 셀은 제13장에서 생성될 NSButtonCell의 커스텀 서브클래스 인스턴스이다. 이제, <예제 12-16>에 모든 코드를 입력한다. 추후, 커스텀 셀 클래스(ToDoCell)를 생성할 때, 이 예제를 참조할 수 있다.

기본적으로, 이 메소드는 아이템 어레이를 통해 다음 작업을 수행한다.

- 어레이의 객체가 ToDoItem라면, 아이템 명을 어레이 슬롯에 고정된 텍스트 필드에 쓰고, 필드 옆의 버튼 셀을 업데이트한다.
- 객체가 ToDoItem이 아니라면, 해당 텍스트 필드와 셀을 비운다.

이 메소드 버전에서 상기 사항이 언급되어 있는지 확인한다. 이 작업은 응용 프로그램이 완전히 종료되기 전에 데이터 모델을 구축 및 시험할 수 있도록 한다. **setTimeDue:**와 **setTriState:**를 참조한 2개의 메소드는 아직 구현되지 않았다. 제13장에서 이들을 구현하면, 상기 사항을 언급하지 않아도 된다.

## *saveDocItems: 구현하기*

이 메소드는 **currentItems** 어레이를 점검하여, 최소 1개의 ToDoItem가 배치되어 있으면, 날짜에 해당하는 키를 사용하여 **activeDays** 딕셔네리에 어레이를 배치한다.

이 메소드는 생성된 아이템을 저장할 수 있도록 현재 선택된 날짜가 변경될 때마다 발생한다. <예제 12-17>을 ToDoDocument.m에 추가한다.

*<예제 12-17> ToDoDocument의 saveDocItems 메소드 구현*

```
- (void)saveDocItems
{
    ToDoItem *anItem;
    int i, cnt = [currentItems count];

    // save day's current items (array) to document dictionary
    for (i=0; i<cnt; i++) {
        if ( (anItem = [currentItems objectAtIndex:i]) &&
            ([anItem isKindOfClass:[ToDoItem class]]))
        {
            [activeDays setObject:currentItems forKey:[anItem day]];
            break;
        }
    }
}
```

## *selectItemAtRow: 구현하기*

이 메소드는 가로행 번호에 제공된 To Do 아이템을 선택하기 위해 사용된 편리한 메소드이다. 이 메소드는 제13장에서 설명할 통지에 응답하기 위해서 발생한다.

```
- (void)selectItemAtRow:(int)row
{
    [itemList selectCellAtRow:row column:0];
}
```

## *controlTextDidBeginEditing: 구현하기*

사용자가 텍스트 필드에 입력할 때 컨트롤은 controlTextDidBeginEditing:을 델리게이트에 전송한다. 이 통지를 통해 현재 아이템 상태를 추적할 수 있다. 삽입점이 텍스트 필드에 있다면, 컨트롤은 사용자가 입력할 때 마다 응답해야 하는 삽입점에서 controlTextDidEndEditing:을 전송한다.

```
- (void)controlTextDidBeginEditing:(NSNotification *)notif
{
   selectedItemEdited = YES;
}
```

## *controlTextDidEndEditing: 구현하기*

이 메소드는 도큐멘트 데이터 모델을 관리한다. 어떻게 도큐멘트 관리가 수행되는지를 이해하려면, 일단 응용 프로그램을 사용하고 있다고 가정한다. ToDoItem을 currentItems 어레이에 추가시키는 사용자 이벤트는 무엇인가? To Do는 아이템 입력이 신속하게 "수행"될 수 있도록 한다.

그래서, ToDoItem을 어레이에 추가하기 위해 사용자가 버튼을 클릭할 필요가 없도록 한다. 사용자가 아이템을 입력하고, 다음 중의 1가지 사항을 수행하면, 아이템이 추가된다.

- Tab 키를 누른다.
- Enter 키를 누른다.
- 텍스트 필드 바깥을 클릭한다.

**controlTextDidEndEditing:** 통지는 다음과 같은 작업을 수행한다. 편집 가능한 필드 매트릭스(**itemList**)는 커서가 텍스트 필드에 있을 때 이 메소드를 발생시킨다. 아이템이 인스턴스에 입력되면, 이 메소드는 ToDoItems를 내부 저장 장치에 추가하여, 이들을 삭제 또는 변경한다. <예제 12-18>을 **ToDoDocument.m**에 추가한다.

*<예제 12-18> ToDoDocument의 controlTextDidEndEditing: 메소드 구현*

```
- (void)controlTextDidEndEditing:(NSNotification *)notif
{
    id newItem;
    int row = [itemList selectedRow];
    NSString *newName = [[itemList selectedCell] stringValue];
    NSString *prevNameAtIndex;

    if (selectedItemEdited == NO)
        return;

   if ([[currentItems objectAtIndex:row] isKindOfClass:[ToDoItem class]]) {
        prevNameAtIndex = [[currentItems objectAtIndex:row] name];

        if ([newName isEqualToString:@""]) {
            [currentItems replaceObjectAtIndex:row withObject:@""];
        } else if (![prevNameAtIndex isEqualToString:newName])
        {
            [[currentItems objectAtIndex:row] setItemName:newName];
        }
    } else if (![newName isEqualToString:@""])
    {
        newItem = [[ToDoItem alloc] initWithName:newName
                andDate:[calendar selectedDay]];
        [currentItems replaceObjectAtIndex:row withObject:newItem];
        [newItem release];
    }

    selectedItem = [currentItems objectAtIndex:row];

        if (![selectedItem isKindOfClass:[ToDoItem class]])
        selectedItem = nil;

    [self updateLists];
```

<예제 12-18> ToDoDocument의 controlTextDidEndEditing: 메소드 구현 (계속)

```
    selectedItemEdited = NO;

    [self updateChangeCount:NSChangeDone];
}
```

컨트롤은 삽입점이 텍스트 필드에 있으면, controlTextDidEndEditing:을 델리게이트로 전송한다. controlTextDidEndEditing:의 구현은 새로운 ToDoItems를 생성할 뿐만 아니라, ToDoItems를 어레이에서 제거하고, 아이템 텍스트를 변경한다. 여기서 수행하는 작업은 사용자가 수행하는 작업과 일치한다. 아이템이 편집되지 않았다면, 진행할 이유가 없기 때문에 코드를 리턴한다.

사용자가 기존 아이템의 텍스트를 삭제하면, 코드는 삭제된 텍스트의 가로행에 해당하는 ToDoItem을 삭제한다. 필드에 입력된 텍스트가 currentItems 어레이의 해당 아이템 명과 일치하지 않으면, 아이템 명을 변경한다. 상기 2개의 조건 중에 어느 것도 적용하지 않고, 텍스트를 입력하면, 새로운 ToDoItem을 생성하여, currentItems 어레이에 삽입한다. 마지막으로, 도큐멘트 인터페이스의 아이템 리스트를 업데이트하여, 편집 상태를 리셋한 뒤, NSDocument의 updateChangeCount: 메소드를 사용하여 도큐멘트를 변경된 대로 표시한다.

## CalendarMatrix 델리게이트 메소드 업데이트하기

<예제 12-19>에서 완성된 구현을 통해 CalendarMatrix의 델리게이트 메소드를 업데이트한다. 그래서, 응용 프로그램이 캘린더의 사용자 액션에 응답할 수 있도록 한다.

<예제 12-19> ToDoDocument의 calendarMatrix:didChangeToDate: 메소드 구현하기

```
- (void)calendarMatrix:(CalendarMatrix *)matrix
    didChangeToDate:(NSDate *)date
{
    [self saveDocItems];
    [self setCurrentItems:[activeDays objectForKey:date]];

    [dayLabel setStringValue:[date descriptionWithCalendarFormat:
            @"To Do on %a %B %d %Y" timeZone:[NSTimeZone defaultTimeZone]
            locale:nil]];

    [self updateLists];
    [self selectItemAtRow:0];
}
```

사용자가 새로운 날짜를 클릭하면, 캘린더는 calendarMatrix:didChangeToDate:를 전송한다. 이 구현을 통해 현재 아이템을 activeDays 딕셔너리에 저장한다. 그리고, 현재 아이템을 선택된 날짜(그 날짜에 해당하는 아이템이 없다면, objectForKey: 메시지는 nil을 리턴하고, currentItems 어레이는 비어있는 스트링으로 초기화한다)에 해당하는 아이템에 설정한다. 마지막으로, 새로운 데이터로 매트릭스를 업데이트하고, 리스트의 첫번째 To Do 아이템을 선택한다.

## 시험하고 구축하기

응용 프로그램을 컴파일하고, 데이터 모델을 시험한다. `ToDoDocument.m`의 키 메소드에 브레이크포인트를 정하고, 델리게이션과 통지 메소드가 정상적으로 발생하였는지 확인한다. To Do 아이템 리스트에서 아이템을 추가 및 삭제할 수 있어야 한다.

기본적인 응용 프로그램 프레임워크를 배치하였으므로, To Do 기능을 완전히 구현하기 위해 좀 더 고급 기능을 추가하도록 하자. 다음 장은 만료 날짜와 알람, 완성 상태, 기술적 Note를 비롯하여 ToDoItem의 속성, 점검 및 검토할 수 있도록 Info 윈도우를 생성하는 방법을 설명한다.

# 13

# *To Do: 확장*

제11장, *Cocoa의 멀티플 도큐멘트 아키텍처*에서 Cocoa의 멀티 도큐멘트 아키텍처에 관한 기본 원리를 설명했으며, 제12장, *To Do: 기본 원리*에서 아키텍처를 사용하여 To Do 응용 프로그램의 코어 기능을 구축했다. 이 장에서는 To Do 응용 프로그램에 기능과 특성을 추가하는 방법을 설명한다. 기능은 다음과 같다.

- 아이템과 만료 시간을 나타내는 각 아이템의 커스텀 버튼을 사용자는 아이템의 상태를 변경하기 위해 클릭할 수 있다.
- ToDo 아이템의 속성을 점검 및 변경할 수 있도록 하는 Info 윈도우
- 아이템이 만료되기 전에 응용 프로그램이 경보를 발할 수 있도록 하는 타이머
- To Do 도큐멘트를 디스크에 저장할 수 있는 능력

마지막으로, Finder의 To Do 도큐멘트를 더블 클릭하여 사용자가 응용 프로그램을 자동으로 실행할 수 있도록 제14장, *To Do: 마무리 작업*에서 도큐멘트 입력, 응용 프로그램 및 도큐멘트 아이콘의 기능을 추가적으로 제공한다.

## *Info 윈도우 생성 및 관리하기*

Info 윈도우는 사용자가 객체 속성을 검토 및 설정할 수 있도록 하는 필드 그룹과 컨트롤을 제공한다. 객체는 여러 속성을 가지고 있고, 사용자가 이 같은 속성을 간단하게 설정할 수 있기 때문에, Info 윈도우는 1개 이상의 화면을 제공한다. 사용자는 팝업 메뉴를 통해 멀티플 화면에 액세스한다. 사용자는 이 화면에서 다음 작업을 수행한다.

- 아이템이 만료된 시간을 명시한다.
- 만료시간 이전에 지정된 간격으로 통보를 요청한다.
- Note를 아이템에 연결한다.
- 아이템을 Completed 또는 Deferred로 표시한다.
- 완성되지 않은 아이템의 스케줄을 다시 조정한다.

To Do 응용 프로그램은 사용자가 새롭게 선택한 ToDoItem의 속성을 점검 및 설정할 수 있는 Info 윈도우를 제공한다. Info 윈도우는 독립적인 컨트롤러인 InfoWindowController를 제공한다. 이 장에서는 Info 윈도우와 InfoWindowController를 생성하는 방법을 설명한다. 또한, 다음 작업을 중점으로 설명한다.

- 사용자 선택에 따라 화면을 관리한다.
- 새로운 ToDoItem을 확보한다.
- 새롭게 선택된 화면을 업데이트한다.
- 사용자가 ToDoItem을 변경할 때 새로운 ToDoItem을 업데이트한다.

## 응용 프로그램의 메뉴 사용자화

Info 윈도우을 구축하기에 앞서, Info 윈도우를 불러온 응용 프로그램의 메인 윈도우에 메뉴 아이템을 추가한다.

1. `MainMenu.nib`를 연다.
2. Revert 및 Page Setup 아이템간의 File 메뉴에 구분선을 추가로 배치한다.
3. 이들 구분선 사이에 Show Info....라는 새로운 메뉴 커맨드를 만든다.
4. 이 커맨드에 Command-Shift-I 키를 추가한다.

## 응용 프로그램 델리게이트 생성 및 구성하기

To Do Info 윈도우는 포괄적이며, 동적이다. 다시 말하면, ToDoDocuments은 1개의 Info 윈도우를 제공하고, 응용 프로그램의 File 메뉴에서 이 윈도우를 발생시킨다. 사용자가 Info 윈도우를 제공하기 위해 선택한 메뉴 아이템은 nib 파일(`MainMenu.nib`)에 있는 반면에 다음 섹션에서 생성할 Info 윈도우는 다른 nib 파일에 들어 있기 때문에 복잡한 디자인 문제를 발생시킨다. 이 작업을 수행하려면, Show Info...메뉴 아이템의 액션 메소드의 타켓이 될 수 있는 `MainMenu.nib`에서 객체를 가지고 있어야 한다.

이 섹션에서는 Info 윈도우를 만들어 제공할 수 있는 단순한 클래스인 ToDoApp델리게이트를 만드는 방법을 설명한다. 이 디자인에서는 Info 윈도우가 느리게 로딩된다는 장점이 있다.

"느린" 로딩은 Info 윈도우의 컨트롤러 객체가 Show Info... 아이템이 File 메뉴에서 실제로 발생할 때까지, 다시 말하면, 시스템 자원을 필요로 할 때 까지 nib 파일을 생성하지도 로딩하지도 않는다는 것을 의미한다.

이 디자인 패턴을 증명하기 위해 단순한 응용 프로그램으로 시험해 보려면, TestInfoWindow라는 예제 응용 프로그램을 사용한다.

ToDoApp델리게이트를 생성하려면 다음 단계를 따른다.

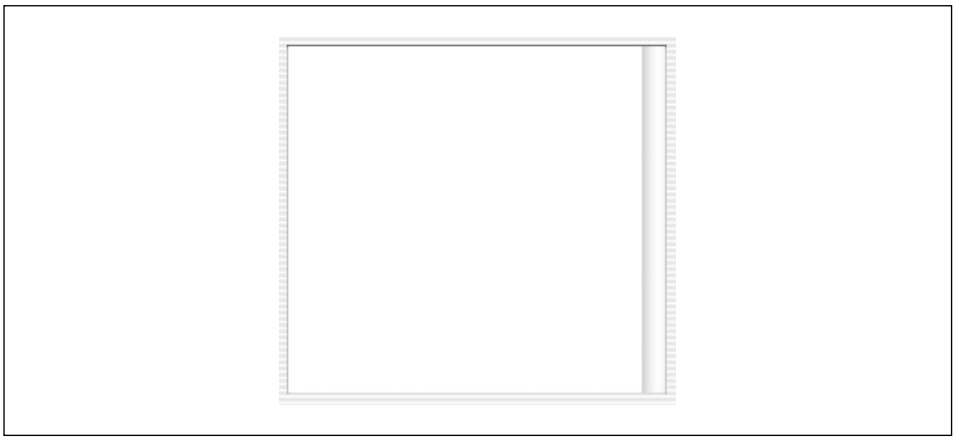
1. `MainMenu.nib` 윈도우의 Classes 패인에서 NSObject의 서브클래스인 ToDoAppDelegate를 생성한다.
2. `showInfo:` 액션을 추가한다.
3. ToDoAppDelegate 인스턴스를 만든다.
4. Show Info... 메뉴 아이템을 ToDoAppDelegate에 연결하고, `showInfo:` 액션을 선택한다.
5. ToDoAppDelegate를 File's Owner의 델리게이트(런타임시, NSApp인 경우)로 만든다.
6. ToDoAppDelegate 파일을 만들고, 이들을 프로젝트에 추가한다.
7. `ToDoAppDelegate.m`을 열어, `InfoWindowController.h`를 위해 임포트할 코드를 추가한다. InfoWindowController 클래스 파일은 아직 존재하지 않지만, 다음 섹션에서 이 파일을 만들 것이다. 따라서, 이 섹션에서 필요한 코드를 추가해야 한다.
8. 다음 코드에서 보여진 바와 같이, 액션 메소드 정의를 작성한다. 이 클래스 메소드는 InfoWindowController 클래스가 Info 윈도우 컨트롤러의 공유 인스턴스를 만들고,(만들어지지 않은 경우) 제공할 수 있도록 한다.

```
- (IBAction)showInfo:(id)sender {
[[InfoWindowController sharedInfoWindowController]
        showWindow:sender];
}
```

## *To Do Info 윈도우의 nib 파일 생성하기*

이 섹션에서는 Info 윈도우의 사용자 인터페이스 생성 방법을 설명한다. 윈도우는 ToDoItem 속성의 다양한 세트에 액세스할 수 있도록 3개의 패인을 제공한다. 완성된 Info 윈도우에서, 표준 팝업 메뉴를 통해 어떤 패인을 시각화할 것인지 선택할 수 있다. 3개의 패널을 통해 다음을 수행할 수 있다.

- 아이템과 관련된 기술적 텍스트 Note를 생성한다.(<그림 13-1>참조)



<그림 13-1> Info 윈도우의 Notes 패인

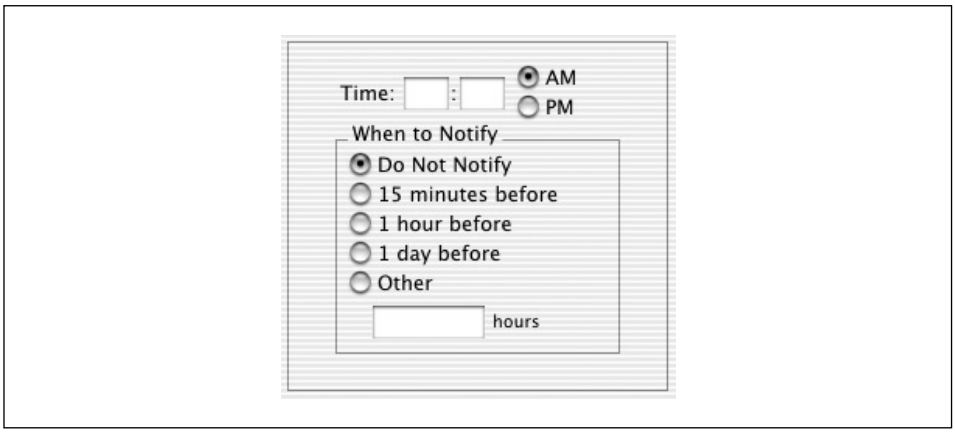
- Task Completed라 표시하거나, 다른 시간으로 일정을 변경한다.(<그림 13-2>참조)


<그림 13-2> Info 윈도우의 Rescheduling 패인

- 아이템의 만료 날짜가 다가오면, 경보를 발하는 통지 알람과 함께 아이템의 만료 날짜 및 시간을 설정한다.(<그림 13-3>참조)

다음은 윈도우와 구성 패인을 만드는 단계이다.

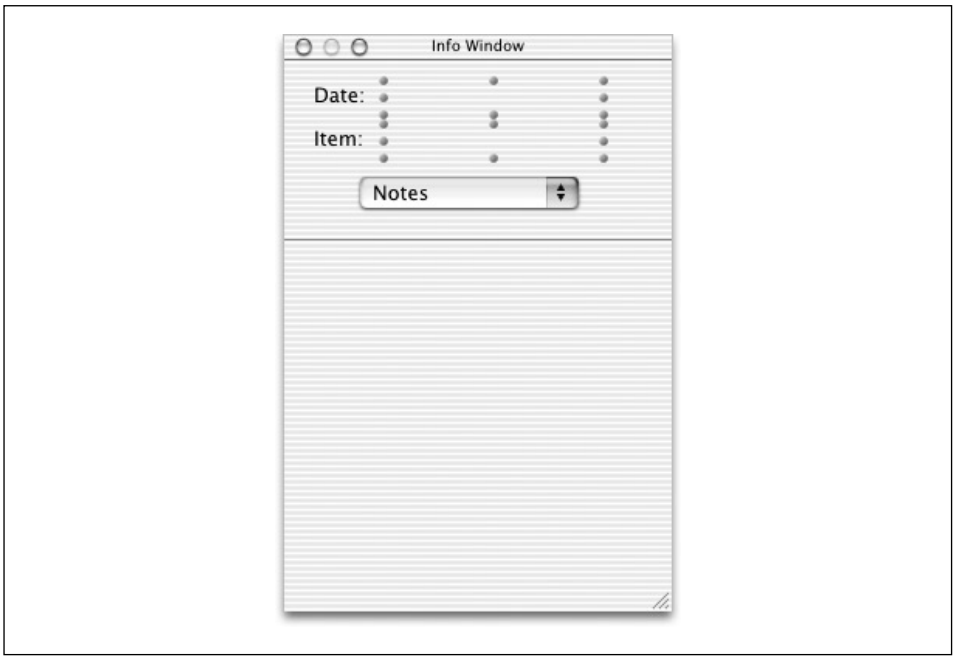
1. Interface Builder의 File 메뉴에서 New를 선택한다.
2. 시작점 리스트에서 Empty를 선택하여, nib 파일을 생성한다.
3. nib 파일을 `ToDoInfoWindow.nib`로써 저장하고, 프로젝트에 삽입한다.
4. Windows 팔레트에서 `ToDoInfoWindow.nib` 윈도우의 Instances 패인으로 NSPanel 객체를 드래그한다.




<그림 13-3> Info 윈도우의 Notification 패인

5. 패널 인스턴스를 **Info window**로 변경한다.
6. **Info window** 패널의 타이틀을 만든다.
7. 패널을 Utility 윈도우로 만든다.
8. <그림 13-4>에서 보이는 바와 같이, 라벨, 텍스트 필드 및 팝업 버튼을 패널에 배치한다. 텍스트 필드는 경계가 없어야 하며, 편집 또는 스크롤을 하지 않아야 한다. 또한, 바탕 화면도 나타나지 않아야 한다.
9. 팝업을 더블 클릭하여, 플로팅 윈도우에 3개의 기본적인 아이템(Item1, Item2, Item3)이 나타난다. 이들 아이템 명칭을 **Notification**, **Reschedule** 및 **Notes**로 변경한다.(아이템 타이틀을 더블 클릭하여 선택한다)
10. 태그를 0, 1, 2로 할당하여, 버튼 셀을 팝업한다.
11. 팝업 버튼 아래 구분선을 만든다.
12. 비어있는 상자 객체를 패널의 하단에 놓는다. 상자의 Title 속성을 해제하고 객체의 크기를 조정한다. 그래서, 패널 하단 내부에 매칭시킨다. 크기를 조정한후 보더를 해제한다.
13. nib 파일을 저장한다.

패널 하단의 비어있는 상자 객체가 무엇인지 궁금할 것이다. 이 상자는 객체 속성을 나타내지 않는다. 그러나, Info 윈도우를 운용하는데 있어 중요한 역할을 한다. Views 팔레트에서 드래그한 상자는 Content 뷰라는 1개의 서브뷰를 갖고 있다. NSBox의 Content 뷰는 상자의 영역과 완전히 일치한다. 이 메소드를 사용하여, Info 윈도우가 제공한 내용을 변경한다.




<그림 13-4> Info 윈도우

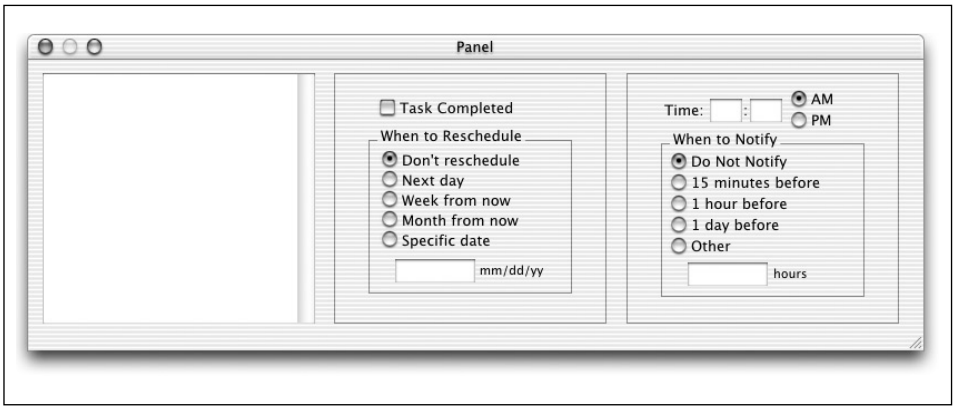
## Offscreen 패널 생성하기

Info 윈도우의 화면을 제공하는 Offscreen 패널을 생성한다.

1. Windows 팔레트에서 `ToDoInfoWindow.nib`의 Instances 패인으로 패널 객체를 드래그한다.
2. <그림 13-5>를 참조하여 패널 크기를 조정한다.
3. <그림 13-5>에서 보이는 바와 같이, 라벨, 텍스트 필드, 스크롤 뷰, 스위치 및 라디오 버튼 매트릭스를 패널에 배치한다.
4. When to Reschedule 및 When to Notify groupings(상자)를 만든다.
5. Notes, Reschedule 및 Notification의 3개 화면과 일치하는 3개의 그룹을 만든다.
6. 아우터 상자를 Info 윈도우의 견본 뷰와 동일한 크기로 만든다.

이제, Info 윈도우가 어디에서 화면을 확보하고, 어떻게 자리에 배치되는 지를 확인할 수 있다. 먼저, Info 윈도우를 열어(ToDoInfoWindow.nib이 로딩되면), 차례로 Info 윈도우 컨트롤러, InfoWindowController를 열면, Info 윈도우에 비어있는 상자(`dummyView`)의 Content 뷰를 Offscreen 패널에서 Notification의 Content 뷰로 대체한다.

그리고, 사용자가 Info 윈도우에서 새로운 팝업 버튼을 선택할 때마다 InfoWindowController는 현재 나타난 Content 뷰를 해당 Offscreen 상자의 Content 뷰로 대체한다.




<그림 13-5> *Info 윈도우 패널*

앞서, 이 책의 예제 파일과 함께 나온 예제 응용 프로그램의 TestInfoWindow를 시험하지 않았다면, 지금 시험해볼 수 있다. 이 같은 매우 단순한 응용 프로그램은 팝업 버튼을 통해 교환할 수 있는 2개의 뷰로 공유 Info 윈도우를 구현한다. Info 윈도우를 보면, 어떤 작업이 수행되는지 알 수 없다. 그러나, 코드를 검토해보면 이 디자인 패턴이 어떻게 운용되는지 명확하게 확인할 수 있을 것이다.

## *포매터 적용하기*

이 섹션에서는 데이터가 더욱 시각적인 포맷으로 제공될 수 있도록 포매터를 Info 윈도우의 텍스트 필드에 적용하는 방법을 설명한다. 또한, 포매터는 사용자가 유효하지 않은 시간과 분 값을 입력하지 못하도록 시간 필드의 범위 점검 메카니즘으로 동작한다.

1. DataViews 팔레트에서 번호 포매터 객체를 드래그하여 Notification 화면의 시간 필드(Time 뒤의 첫 번째 필드)에 드롭한다.
2. Integer Format을 선택한다.
3. Info 윈도우의 Formatter 화면에서 최소 1에서 최대 12로 필드값을 설정한다.
4. 번호 포매터를 Minutes 필드(Time 뒤의 두 번째 필드)에 적용하고, Integer Format을 선택한다.
5. 최소 0에서 최대 59로 필드 값을 설정한다.

6. DataViews 포매터에서 Rescheduling 화면(mm/dd/yy 필드)의 데이터 필드로 포매터 객체를 드래그한다.
7. Info 윈도우의 Formatter 화면 테이블에서 %m/%d/%y 포맷을 선택한다.

## *InfoWindowController 클래스 정의하기*

InfoWindowController는 Info 윈도우의 nib 파일을 로딩하고, Info 윈도우의 뷰 상태를 관리한다. Info 윈도우 컨트롤러는 다른 응용 프로그램 객체에서 통지를 수신하기 위해 설정된다. 그래서, 현재 ToDo item이나 To Do document가 변경되면, 뷰를 동적으로 업데이트한다. 다음은 컨트롤러 클래스를 생성하는 단계이다.

1. Interface Builder에서 infoWindowController 클래스를 NSWindowController의 서브클래스로 만든다.
2. <표 13-1> 및 <표 13-2>의 좌측 세로열에 있는 아웃렛과 액션을 추가한다. window 아웃렛은 이미 있기 때문에 다시 만들 필요는 없다. 4단계에서 이 아웃렛을 연결하는 작업을 빠뜨리지 않도록 테이블에 추가한다.
3. ToDoInfoWindow.nib 윈도우의 Instances 패인에서 File's Owner를 선택하고, File's Owner Info 윈도우를 사용하여 커스텀 클래스를 InfoWindowController로 변경한다.
4. InfoWindowController의 프록시인 File's Owner를 사용하여 <표 13-1> 및 <표 13-2>의 오른쪽 세로열에 수록된 대로 연결한다.

*<표 13-1> InfoWindowController 아웃렛 및 연결*

| Outlet | 연결 대상 |
|---|---|
| window | Info 윈도우(Instances 패널에서 아이콘이나 윈도우의 타이틀바로 연결한다.) |
| dummyView | Info 윈도우의 비어 있는 상자 객체 |
| infoWindowViews | 교환 가능 뷰가 있는 Offscreen 윈도우(Instances 패널에서 아이콘이나 윈도우의 타이틀 바로 연결한다) |
| notesView | 텍스트 뷰를 내장한 Offscreen 윈도우 상자 |
| notifyView | 긴급 아이템 통지와 관련된 필드 및 컨트롤을 제공하는 Offscreen 윈도우 상자 |
| reschedView | 일정 변경 아이템과 관련된 필드 및 컨트롤을 제공하는 Offscreen 윈도우 상자 |
| infoPopUp | Info 윈도우의 팝업 버튼 |
| infoDate | Date 라벨 옆의 편집 불가능 텍스트 필드 |
| infoItem | Item 라벨 옆의 편집 불가능 텍스트 필드 |

<표 13-1> InfoWindowController 아웃렛 및 연결 (계속)

| Outlet | 연결 대상 |
|---|---|
| infoNotes | 스크롤 텍스트 뷰 내부 |
| infoNotifyHour | Time 라벨 옆의 텍스트 필드 |
| infoNotifyMinute | Time 라벨 뒤의 두번째 필드 |
| infoNotifyAMPM | A.M.과 P.M. 라디오 버튼을 홀딩시키는 매트릭스 |
| infoNotifyOtherHours | When to Notify 상자의 텍스트 필드 |
| infoNotifySwitchMatrix | When to Notify 상자의 라디오 버튼 매트릭스 |
| infoSchedComplete | Task Completed 스위치 |
| infoSchedDate | When to Reschedule 상자의 텍스트 필드 |
| infoSchedMatrix | When to Reschedule 상자의 라디오 버튼 매트릭스 |

<표 13-2> InfoWindowController 액션 및 연결

| 액션 | 연결 발생 |
|---|---|
| swapInfoWindowView: | Info 윈도우의 팝업 버튼 |
| switchClicked: | "When to Notify" 상자 스위치의 매트릭스, AM-PM 매트릭스, Task Completed 스위치 및 When to Reschedule 스위치의 스위치 매트릭스 |

5. 각 텍스트 필드의 델리게이트 아웃렛을 InfoWindowController로 연결한다.
6. 양쪽 윈도우를 닫고, ToDoInfoWindow.nib을 저장한다.
7. InfoWindowController의 소스 코드 파일을 생성하여, 프로젝트에 추가한다.

## 아웃렛 입력하기

제6장, *중요한 Cocoa 패러다임*에서 Cocoa 프로그래밍을 실행하여 정적으로 유형화된 아웃렛을 사용하는 방법을 설명했다. 이 단계가 반드시 필요한 것은 아니지만, 컴파일러가 입력을 엄격하게 점검할 수 있도록 수 많은 디버깅을 간단하게 처리한다.

InfoWindowController.h의 모든 아웃렛을 검토하려면 약간의 시간이 소요되며, ID에서 인터페이스 객체로 입력하여 입력 선언을 변경한다. 인터페이스 객체의 클래스 멤버가 확실하지 않다면, Interface Builder의 nib 파일을 열어, 해당 객체를 선택한 뒤, Info 윈도우를 불러온다. Info 윈도우의 타이틀바는 인터페이스 객체의 클래스에 해당된다.

## InfoWindowController.h에 선언 추가하기

이 섹션에서는 Interface Builder가 생성한 헤더 파일에 선언을 추가하는 방법을 설명한다. <예제 13-1> 상단에 있는 enum은 Notify 패인에서 다양한 팝업 메뉴 아이템과 라디오 버튼을 식별하기 위해 클래스에 의해 사용되며, 추가된 인스턴스 변수는 현재의 ToDo 도큐멘트에 대한 참조를 가지고 있다.

<예제 13-1>에서 선언을 `InfoWindowController.m`에 추가한다.

*<예제 13-1> InfoWindowController 인터페이스에 추가*

```
@class ToDoDocument;

enum { NOTIFY_TAG = 0, RESCHEDULE_TAG, NOTES_TAG };
enum { NotifyLengthNone = 0, NotifyLengthQuarter, NotifyLengthHour,
    NotifyLengthDay, NotifyLengthOther };

@interface InfoWindowController : NSObject
{
    @private
    /* ... */
    ToDoDocument *_inspectingDocument;
}
/* ... */
+ (id)sharedInfoWindowController;

@end
```

## 기본적인 메소드 구현하기

이 섹션에서는 InfoWindowController의 기본적인 메소드 구현을 추가하는 방법을 설명한다. 이 메소드는 nib 파일을 로딩하고, Offscreen 패널 뷰를 Info 윈도우로 교환하면서 공유 Info 윈도우 인스턴스를 생성한다.

### 헤더 파일 임포트하기

InfoWindowController가 ToDoItem 데이터를 제공 및 변경하고, ToDoDocument와 효과적으로 상호작용하려면, 클래스가 구현하는 메소드에 관해 알아야 한다. 클래스의 공용 헤더 파일에서 메소드 관련 내용을 참조한다. `InfoWindowController.m`을 열어, `ToDoItem.h` 및 `ToDoDocument.h`를 임포트한다.

## Private 선언 추가하기

다음의 헬퍼 메소드 선언을 InfoWindowController.m에 추가한다.

```
static void clearButtonMatrix(id matrix);

@interface InfoWindowController (PrivateMethods)
- (void)updateInfoWindow;
- (void)setMainWindow:(NSWindow *)mainWindow;
@end
```

이 선언은 헬퍼 기능으로 추후 튜토리얼에서 자세하게 설명한다.

## sharedInfoWindowController 클래스 메소드 구현하기

이 클래스 메소드는 컨트롤러 객체를 공유 참조하기 위해 정적 변수인 _sharedInfoWindowController를 사용한다. 이 메소드는 공유한 InfoWindowController 인스턴스가 존재하는지 여부와 존재하지 않으면, 생성할 것인지 여부만을 점검한다.

```
+ (id)sharedInfoWindowController
{
    static InfoWindowController *_sharedInfoWindowController = nil;

    if (!_sharedInfoWindowController) {
        _sharedInfoWindowController = [[InfoWindowController
                allocWithZone:[self zone]] init];
    }
    return _sharedInfoWindowController;
}
```

## init, windowDidLoad 및 dealloc 구현하기

<예제 13-2>는 InfoWindowController의 기본적인 초기화 및 클린업 메소드를 구현한다. 추후 버전에 이 구현을 추가할테지만, 초기 버전에서 복잡한 구성을 추가하기에 앞서 Info 윈도우를 시험할 수 있도록 한다.

*<예제 13-2> InfoWindowController의 init, windowDidLoad, 및 dealloc 메소드 구현하기*

```
- (id)init
{

    self = [self initWithWindowNibName:@"ToDoInfoWindow"];
    if (self)
        [self setWindowFrameAutosaveName:@"Info"];

    return self;
}

- (void)windowDidLoad
```

*<예제 13-2> InfoWindowController의 init, windowDidLoad, 및 dealloc 메소드 구현하기 (계속)*

```
{
    [super windowDidLoad];

    [notifyView retain];
    [notifyView removeFromSuperview];

    [reschedView retain];
    [reschedView removeFromSuperview];

    [notesView retain];
    [notesView removeFromSuperview];

    [infoWindowViews release];
    infoWindowViews = nil;

    [infoNotes setDelegate:self];

    [self swapInfoWindowView:self];
    [self setMainWindow:[NSApp mainWindow]];
    [self updateInfoWindow];

    [[NSNotificationCenter defaultCenter] addObserver:self
            selector:@selector(mainWindowResigned:)
            name:NSWindowDidResignMainNotification object:nil];
}

- (void)dealloc
{
    [notifyView release];
    [reschedView release];
    [notesView release];

    [[NSNotificationCenter defaultCenter] removeObserver:self];

    [super dealloc];
}
```

nib 파일의 Offscreen 윈도우에 **windowDidLoad** 및 **dealloc**, **retain** 및 **release** 뷰가 있는지 확인한다. Info 윈도우의 안 밖에서 뷰를 교환하려면 뷰 객체의 소유권을 변경해야 하기 때문에 이 작업은 수행되어야 한다. Info 윈도우 컨트롤러는 nib 파일을 로딩하는 순간 뷰를 보유함으로써, 뷰 객체가 뷰 외부에 있는 동안에는 해제되지 않도록 한다.

교환 가능 뷰를 보유하면, **windowDidLoad** 메소드는 Notes 뷰의 델리게이트가 되도록 한다. 이 방식을 통해, Info 윈도우에 Notes 뷰 텍스트 변경을 통보하는 델리게이트 메소드를 수신한다. 그 다음에는 기본적인(Notify) 뷰에서 메소드를 교환하여, **setMainWindow:** 메시지를 전송한다. 마지막으로, 메소드는 InfoWindowController를 설정하여, 응용 프로그램의 메인 윈도우가 변경되었거나 닫힐 때 통지를 수신한다.

이 통지는 컨트롤러가 Info 윈도우의 뷰를 정상적으로 업데이트할 수 있는 기회를 제공한다.

### *updateInfoWindow 메소드의 Stub 추가하기*

Stub 메소드는 비어있는 Placeholder 구현이므로, 다른 메소드가 이를 참조할 수 있다. 또한, 컴파일러와 런타임을 충족하여 현재, 부분적으로 완성된 상태의 응용 프로그램을 컴파일링하고 시험할 수 있도록 한다. 이 장의 뒷 부분에서는 Info 윈도우 인터페이스를 업데이트하여, 사용자가 메인 ToDoDocument 윈도우에서 새로운 ToDoItem으로 변경한 내용을 반영한 뒤 완성된 구현을 제공하는 방법을 설명한다.

```
- (void)updateInfoWindow
{
}
```

### *setMainWindow: 구현하기*

컨트롤러는 사용자가 현재 확인한 ToDoDocument를 추적하기 위해 이 메소드를 사용한다. **windowDidLoad**에서 Info 윈도우는 기본적인 통지 센터에 등록하여 메인 윈도우가 변경되면, 통지를 수신한다. 이 메소드는 통지를 수신하면, 발생한다.

```
- (void)setMainWindow:(NSWindow *)mainWindow
{
    NSWindowController *controller = [mainWindow windowController];

    if ([[controller document] isKindOfClass:[ToDoDocument class]])
        _inspectingDocument = [controller document];
    else
        _inspectingDocument = nil;

    [self updateInfoWindow];
}
```

### *mainWindowChanged: 구현하기*

이 메소드는 **NSWindowDidBecomeMainNotification** 통지가 게재되면 호출되기 위해 통지 센터에 등록된 헬퍼 메소드의 구현이라 할 수 있다.

```
- (void)mainWindowChanged:(NSNotification *)notification
{
    [self setMainWindow:[notification object]];
}
```

### *swapInfoWindowView: 구현하기*

<예제 13-3>에서 보여진 바와 같이, 팝업 메뉴의 사용자 선택을 기반으로 하여 Info 윈도우의 화면을 변경한다.

*<예제 13-3> InfoWindowController의 swapInfoWindowView: 메소드 구현하기*

```
- (IBAction)swapInfoWindowView:(id)sender
{
    int selected;
    NSBox *newView;

    selected = [[infoPopUp selectedItem] tag];

    switch (selected) {
        case NOTIFY_TAG:
            newView = notifyView;
            break;
        case RESCHEDULE_TAG:
            newView = reschedView;
            break;
        case NOTES_TAG:
            newView = notesView;
            break;
    }

    if ([dummyView contentView] != newView) {
        [dummyView setContentView:newView];
        // more code to follow later...
    }
}
```

이 메소드는 사용자가 선택한 팝업 버튼에 따라 현재 Info 윈도우의 화면을 변경한다. **dummyView**의 Content 뷰를 교체하여 이 같은 변경을 수행한다. 이 메소드는 다음을 수행한다.

- 선택된 팝업 버튼의 태그를 얻어 **newView** 로컬 변수를 태그에 해당하는 Offscreen 상자 객체에 할당한다.
- 선택한 화면이 Info 윈도우에 있다면 리턴한다.

## *구축 및 시험하기*

이제, 코딩을 멈추고, 응용 프로그램을 구축한다. 기본적인 Info 윈도우의 운용을 시험한다. File 메뉴에서 Info 윈도우을 발생시켜, 팝업 버튼을 통해 Info 윈도우 하단에 있는 뷰를 교환할 수 있어야 한다.

## 기능이 강화된 InfoWindowController 메소드 구현하기

이 섹션에서는 Info 윈도우의 화면을 현재 선택한 ToDoItem과 동기화하여 Info 윈도우를 완성하는 방법을 설명한다.

Info 윈도우를 정상적으로 구동하기 위해 Info 윈도우에서 사용자 액션에 응답하는 ToDoItem을 변경해야 한다. Info 윈도우는 활성화되어 있는 ToDoItem 윈도우의 현재 아이템 변경을 추적해야 한다.

Info 윈도우는 `_inspectingDocument` 인스턴스 변수에서 새로운 ToDoDocument의 내부를 지속적으로 참조하면서 첫번째 문제를 해결한다. InfoWindowController는 활성화되어 있는 윈도우가 변경된 시스템에서 통지를 수신할 때 마다 참조 내용을 업데이트한다. 이런 방식으로, Info 윈도우는 올바른 정보를 제공한다. 사용자가 Info 윈도우에서 컨트롤을 클릭하면, InfoWindowController는 새로운 ToDoItem으로 변경을 수행하며, 아이템이 변경된 ToDoDocument로 메시지를 전송한다. 이 작업은 Info 윈도우에서 초기화된 변경을 반영하는 인터페이스를 업데이트할 기회를 제공한다.

Info 윈도우는 통지를 수신한 도큐멘트 윈도우에서 새로운 ToDo 아이템으로 변경된 내용을 추적한다. 이 장의 뒷부분에서 아이템이 변경되었을 때, 새로운 ToDo 아이템을 추적하고, 통지를 게재하는 메소드를 구현하는 방법을 설명한다. InfoWindowController는 이 통지를 수신하여 인터페이스를 정상적으로 업데이트한다.

### updateInfoWindow 구현하기

이 메소드는 새로운 ToDoItem가 활성화되어 있는 To Do 도큐멘트 윈도우에서 선택되었을 때 새로운 Info 윈도우 화면을 업데이트한다. `updateInfoWindow`는 긴 메소드이기 때문에 단계적으로 접근해야 한다. 첫 단계에서는 공통 데이터 요소(아이템, 명, 날짜)를 업데이트한다. Info 윈도우의 새로운 패인이 Notification 패인이라면, 메소드는 이 화면을 업데이트한다.

<예제 13-4>에서 보여진 바와 같이 메소드의 첫 단계는 Stub 구현을 입력하는 것이다.

*<예제 13-4> InfoWindowController의 updateInfoWindow 메소드 시작하기*

```
- (void),pdateInfoWIndow
{
    int minute=0, hour=0, selected=0;
    long notifySecs, dueSecs;
    BOOL pmFlag;
    ToDoItem *selectedItem;
```

<예제 13-4> InfoWindowController의 updateInfoWindow 메소드 시작하기(계속)

```
    selected = [[infoPopUp selectedItem] tag];
    selectedItem = [_inspectingDocument selectedItem];

    if ([selectedItem isKindOfClass:[ToDoItem class]])
{
        [infoItem setStringValue:[selectedItem itemName]];
        [infoDate setStringValue:[[selectedItem day]
                descriptionWithCalendarFormat:@"%a, %b %d %Y"
                timeZone:[NSTimeZone localTimeZone] locale:nil]];

        switch (selected) {
            case NOTIFY_TAG:
                dueSecs = [selectedItem secsUntilDue];
                pmFlag = ConvertSecondsToTime(dueSecs, &hour, &minute);

                [[infoNotifyAMPM cellAtRow:0 column:0] setState:!pmFlag];
                [[infoNotifyAMPM cellAtRow:1 column:0] setState:pmFlag];
                [infoNotifyHour setIntValue:hour];
                [infoNotifyMinute setIntValue:minute];

                notifySecs = [selectedItem secsUntilNotify];

                clearButtonMatrix(infoNotifySwitchMatrix);

                switch(notifySecs) {
                    case 0:
                        [[infoNotifySwitchMatrix
                                cellAtRow:NotifyLengthNone column:0]
                                setState:NSOnState];
                        break;
                    case (SECS_IN_HOUR/4):
                        [[infoNotifySwitchMatrix
                                cellAtRow:NotifyLengthQuarter column:0]
                                setState:NSOnState];
                        break;
                    case (SECS_IN_HOUR):
                        [[infoNotifySwitchMatrix
                                cellAtRow:NotifyLengthHour column:0]
                                setState:NSOnState];
                        break;
                    case (SECS_IN_DAY):
                        [[infoNotifySwitchMatrix
                                cellAtRow:NotifyLengthDay column:0]
                                setState:NSOnState];
                        break;
                    default:  // other
                        [[infoNotifySwitchMatrix
                                cellAtRow:NotifyLengthOther column:0]
                                setState:NSOnState];
                        [infoNotifyOtherHours setIntValue:
                                (notifySecs/SECS_IN_HOUR)];
                        break;
```

*<예제 13-4> InfoWindowController의 updateInfoWindow 메소드 시작하기(계속)*

```
                }
            break;
        case RESCHEDULE_TAG:
            // left as an exercise
            break;
        case NOTES_TAG:
            [infoNotes setString:[selectedItem notes]];
            break;
    } //switch
```

<예제 13-4>에서 보여진 바와 같이 메소드의 첫 단계는 다음과 같다.

- 선택한 팝업 버튼으로 할당된 태그를 얻는다.
- ToDoItem인지 확인하기 위해 `selectedItem` 인수를 시험한다. 이 시험은 `selectedItem`이 `nil`이기 때문에 중요하다. 이 메소드는 기존의 데이터 화면(다음 예제)을 제거한다.
- selectedItem이 ToDoItem이라면, `updateInfoWindow:`는 먼저 Item 및 Date 필드를 업데이트한다.
- 선택된 팝업 버튼의 태그가 `NOTIFY_TAG` 라면, 관련 Info 윈도우의 화면을 업데이트한다. 이 태스크는 초 단위에서 시간, 분 및 P.M 값으로 만료 시간을 변경하고, 이들 값으로 해당 필드 및 버튼 매트릭스를 설정하여 시작된다. 그리고, When to Notify 매트릭스에서 해당 스위치를 설정한다. `clearButtonMatrix`를 호출하여 모든 스위치를 해제하고,(초 단위에서 제공된) 통지 길이에 해당하는 스위치를 설정한다.

이 튜토리얼은 Reschedule 화면을 업데이트한 메소드 코드를 비롯하여 To Do 응용 프로그램의 Rescheduling 로직을 생략한다. ToDoItems의 Rescheduling은 이 장의 마지막 부분에서 선택적으로 실행하기 위해 여기서는 유보한다.

<예제 13-5>에서와 같이 인수가 `nil`이라면, 모든 디스플레이를 재설정하여 `updateInfoWindow:`의 구현을 완료한다.

*<예제 13-5> InfoWindowController의 updateInfoWindow 메소드 마무리하기*

```
} else { // selectedItem is not a ToDoItem
[infoItem setStringValue:@""];
        [infoDate setStringValue:@""];
        [infoNotifyHour setStringValue:@""];
        [infoNotifyMinute setStringValue:@""];
        [[infoNotifyAMPM cellAtRow:0 column:0] setState:NSOnState];
        [[infoNotifyAMPM cellAtRow:1 column:0] setState:NSOffState];
        clearButtonMatrix(infoNotifySwitchMatrix);
        [[infoNotifySwitchMatrix cellAtRow:NotifyLengthNone column:0]
                setState:NSOnState];
        [infoNotifyOtherHours setStringValue:@""];
```

*<예제 13-5> InfoWindowController의 updateInfoWindow 메소드 마무리하기 (계속)*

```
        [infoNotes setString:@""];
    }
}
```

updateInfoWindow 메소드는 스위치 매트릭스에서 모든 버튼, 셀, 상태를 NO로 재설정하는 clear-ButtonMatrix 함수를 호출한다.

## *clearButtonMatrix 구현하기*

clearButtonMatrix 유틸리티 함수 코드를 추가한다.

```
    void clearButtonMatrix(id matrix)
    {
        int i, rows, cols;
        [matrix getNumberOfRows:&rows columns:&cols];
        for(i=0; i<rows; i++)
            [[matrix cellAtRow:i column:0] setState:NO];
    }
```

matrix의 셀인 첫 번째 인수에서 우회하여 getNumberOfRows:columns: 메시지를 리턴한다.

## *switchClicked: 구현하기*

<예제 13-6>에서 제공된 이 메소드는 Info 윈도우에서 입력된 새로운 값으로 현재의 아이템을 업데이트한다.

*<예제 13-6> InfoWindowController의 switchClicked: 메소드 구현하기*

```
- (IBAction)switchClicked:(id)sender
{
    long dueSecs = 0;
    int idx = 0;

    ToDoItem *theItem = [_inspectingDocument selectedItem];

    if (sender == infoNotifyAMPM) {
        if ([infoNotifyHour intValue]) {
            BOOL pmFlag = ([infoNotifyAMPM selectedRow]==1);
            dueSecs = ConvertTimeToSeconds([infoNotifyHour intValue],
                    [infoNotifyMinute intValue], pmFlag);
            [theItem setSecsUntilDue:dueSecs];
        }
    } else if (sender == infoNotifySwitchMatrix) {
        idx = [infoNotifySwitchMatrix selectedRow];
        switch(idx) {
            case NotifyLengthNone:
                [theItem setSecsUntilNotify:0];
                break;
```

<예제 13-4> InfoWindowController의 switchClicked: 메소드 구현하기 (계속)

```
            case NotifyLengthQuarter:
                [theItem setSecsUntilNotify:(SECS_IN_HOUR/4)];
                break;
            case NotifyLengthHour:
                [theItem setSecsUntilNotify:SECS_IN_HOUR];
                break;
            case NotifyLengthDay:
                [theItem setSecsUntilNotify:SECS_IN_DAY];
                break;
            case NotifyLengthOther:
                [theItem setSecsUntilNotify:
                        ([infoNotifyOtherHours intValue] * SECS_IN_HOUR)];
                break;
            default:
                NSLog(@"Error in selectedRow");
                break;
        }
    } else if (sender == infoSchedComplete) {
        [theItem setStatus:COMPLETE];
    } else if (sender == infoSchedMatrix) {
         // left as an exercise
    }

    [self updateInfoWindow];
    [_inspectingDocument selectedItemModified];
}
```

사용자가 Info 윈도우의 스위치 버튼을 클릭하거나 AM-PM 라디오 버튼 중 하나를 클릭하면, **switchClicked:** 메소드가 발생한다. 이 메소드는 **sender** 인수를 파악하여 작업을 수행한다.

**sender**가 라디오 버튼 매트릭스(AM-PM)라면, 코드는 새로운 값을 갖기 위해 현재의 아이템을 설정하는 **ConvertTimeToSeconds;**를 호출하여 새로운 만료 시간을 설정한다. **sender**가 When to Notify 매트릭스라면, 선택된 셀의 인덱스를 확보하고, 스위치를 사용하여 현재 아이템의 새로운 **secsUntilNotify** 값을 설정한다. **sender**가 Task Completed 스위치라면, 현재 아이템 상태를 **COMPLETE**로 설정한다. 상기에서 언급했듯이, Rescheduling 블록의 구현은 마지막으로 실행하기 위해 여기서는 유보한다.

마지막으로, 메소드는 Info 윈도우 뷰를 업데이트하고, 선택한(점검된) 아이템이 변경되었다고 통보한다. 이 작업은 ToDoDocument가 Info 윈도우에서 수행된 변경 내용을 기반으로 상태를 업데이트할 기회를 제공한다.

텍스트 필드는 컨트롤이기 때문에, 사용자가 Return을 누를 경우, **sendsActionOnEndEditing**을 설정할 경우, 첫 번째 리스폰더 상태를 손실할 경우에 타켓/액션 메시지를 전송할 수 있다. 따라서, 데이터를 필드에 입력하면, **switchClicked:** 응답을 받을 수 있다.

대안적인 디자인을 제공하려면 텍스트 엔트리를 처리하기 위한 델리게이션 메시지에 따라 결정된다.

## 통지 메소드 구현하기

**textDidEndEditing:** 메소드는 Info 윈도우의 Notes 뷰가 편집 텍스트를 완성했다는 통지를 게재하면 발생한다. 현재의 ToDo 아이템의 Notes 속성을 재설정하여 메시지가 변경되었다는 내용을 통지하는 메시지를 전송한다. <예제 13-7>에서와 같이 **controlTextDidEndEditing:** 메소드는 Info 패인에서 텍스트 필드의 내용이 변경되면 새로운 ToDoItem을 업데이트한다.

*<예제 13-6> InfoWindowController의 텍스트 통지 메소드 구현*

```
- (void)textDidEndEditing:(NSNotification *)notification
{
    if ([notification object] == infoNotes) {
        [[_inspectingDocument selectedItem] setNotes:[infoNotes string]];
        [_inspectingDocument selectedItemModified];
    }
}

- (void)controlTextDidEndEditing:(NSNotification *)notification
{
    long dueSecs = 0;
    ToDoItem *theItem = [_inspectingDocument selectedItem];

    if ([notification object] == infoNotifyHour ||
        [notification object] == infoNotifyMinute) {
        dueSecs = ConvertTimeToSeconds([infoNotifyHour intValue],
                [infoNotifyMinute intValue],
                [[infoNotifyAMPM cellAtRow:1 column:0] state]);
        [theItem setSecsUntilDue:dueSecs];
    } else if ([notification object] == infoNotifyOtherHours)
    {
        if ([infoNotifySwitchMatrix selectedRow] == NotifyLengthOther)
            [theItem setSecsUntilNotify:
                    ([infoNotifyOtherHours intValue] * SECS_IN_HOUR)];
        else
            return; // nothing changed
    } else if ([notification object] == infoSchedDate)
    {
        // left as an exercise
    }

    [_inspectingDocument selectedItemModified];
}
```

**textDidEndEditing:** 및 **controlTextDidEndEditing:** 통지 메소드는 편집을 수행한 후, 커서가 텍스트 객체나 필드에 있다면 델리게이트(다른 옵저버)로 전송된다.

Notes 텍스트 객체를 편집하면, `textDidEndEditing:`이 발생한다. `textDidEndEditing:`은 텍스트 객체의 컨텐트로 ToDoItem의 Notes 인스턴스 변수를 재설정하여 응답한다.

통지를 발생시킨 객체가 Notifications 화면의 시간 또는 분 필드라면, `controlTextDidEndEditing:`은 새로운 값을 갖기 위해 새로운 만료 시간을 산출하고, 현재의 아이템을 설정한다.

통지를 발생시킨 객체가 When to Notify 상자의 Other...시간 텍스트 필드라면, 메소드는 Other 스위치를 점검하고, 새로운 값으로 ToDoItem을 설정했는지 확인한다. 마지막으로, 비어있는 Rescheduling 블록은 추후에 실행하기 위해 여기서는 유보한다.

메소드가 끝날 무렵에, 도큐멘트는 `selectedItemModified` 메시지를 전송한다.

응용 프로그램을 컴파일하여, 자르기 또는 붙이기 같은 입력을 통해 오류가 발생했는지 확인해야 한다. 수신할 수 있는 유일한 컴파일러 경고는 ToDoDocument 클래스가 `selectedItemModified`에 응답하지 않았다는 내용이다. 이 장의 뒷부분에서 이 메소드를 구현한다.

# *SelectionNotifyMatrix 만들기*

마지막 섹션에서 응용 프로그램의 Info 윈도우를 구현하고, 인프라를 설정하여 새로운 ToDo 아이템으로 변경한다. 사용자가 필드를 클릭하거나 탭 키를 눌러, ToDoDocument에서 새로운 ToDo 아이템을 선택하면, Info 윈도우는 화면과 동기되어야 한다. 이 섹션은 이 같은 기능을 구현할 방법을 설명한다.

제12장에서 언급했듯이, ToDoDocument의 윈도우에서 나타난 ToDo 아이템 리스트는 텍스트 셀의 NSMatrix이다. 응용 프로그램의 Info 윈도우와 새로운 ToDo 아이템이 정상적으로 동기화하려면, 새로운 선택 사항의 변경 내용을 감지할 수 있고, 여기에 관심있는 옵저버(즉 InfoWindowController)를 위해 통지를 게재할 수 있는 NSMatrix의 서브클래스를 생성해야 한다.

이 섹션에서는 응용 프로그램 Kit 클래스 동작을 오버라이드하는 방법을 설명한다. 코드를 슈퍼클래스로 추가하여 서브클래스를 생성한 뒤 객체 동작에서 원하는 변경을 만들 수 있다. 생성해야할 서브클래스는 SelectionNotifyMatrix이다.

## SelectionNotifyMatrix 파일 만들기

Project Builder에서 새로운 클래스의 인터페이스 및 구현 파일을 생성하고, 프로젝트에 추가한다. 헤더 파일의 다음 선언을 추가한다.

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

extern NSString *RowSelectedNotification;

@interface SelectionNotifyMatrix : NSMatrix {
}
@end
```

스트링 정의를 구현 파일에 추가한다.

```
@implementation SelectionNotifyMatrix

NSString *RowSelectedNotification =  @"RowSelectedNotification";
/*...*/
```

**RowSelectedNotification**은 새롭게 선택한 가로행을 변경할 때 게재될 통지를 식별하는 스트링 상수이다.

## mouseDown: 오버라이드하기

제8장, *이벤트 처리*에서 생성한 Dot View 응용 프로그램은 NSView의 서브클래스를 생성하고, **mouseUp:** 메소드를 오버라이드하여 뷰에서 마우스 클릭을 추적하는 방법을 증명했다. 동일한 기법을 NSMatrix의 서브클래스를 생성하는 방법에 적용한다. 이 섹션에서는 NSMatrix의 **mouseDown:**을 오버라이드하여 이벤트 처리를 사용자화하는 방법을 설명한다.

<예제 13-8> **mouseDown:** 구현을 **SelectionNotifyMatrix.m**에 추가한다.

*<예제 13-8> SelectionNotifyMatrix의 mouseDown: 메소드 구현*

```
- (void)mouseDown:(NSEvent *)theEvent
{
    int row;
    [super mouseDown:theEvent];

    row = [self selectedRow];
    if (row != -1) {
        [[NSNotificationCenter defaultCenter]
                postNotificationName:RowSelectedNotification
                object:self
                userInfo:nil];
    }
}
```

mouseDown: 버전은 mouseDown:의 NSMatrix 구현을 발생시켜, 이벤트가 정상적으로 처리될 수 있도록 하며, 클릭한 셀의 가로행을 확보한다. 이 가로행이 확보되면, 이 메소드는 RowSelectedNotification을 기본적인 통지 센터에 게재한다. 완성된 응용 프로그램에서 Info 윈도우의 Controller 객체는 통지를 수신하여 새롭게 선택된 ToDo 아이템에서 데이터와 함께 Info 윈도우의 뷰를 업데이트한다.

## selectCellAtRow:column: 오버라이드하기

탭을 사용하여 새로운 아이템 선택을 처리하려면, selectCellAtRow:column: 메소드를 오버라이드해야 한다. mouseDown:의 상기 구현에서와 같이, 유일한 변경 사항은 RowSelectedNotification을 게재한 코드를 추가하는 것이다. 그러면 Info 윈도우는 새로운 선택을 인식하게 된다.

```
- (void)selectCellAtRow:(int)row column:(int)col;
{
    [super selectCellAtRow:row column:col];

    [[NSNotificationCenter defaultCenter]
            postNotificationName:RowSelectedNotification
            object:self
            userInfo:nil];
}
```

## itemList Matrix 클래스 재할당하기

SelectionNotifyMatrix 클래스를 생성하였으므로, 인터페이스 객체의 클래스 멤버를 재할당해야 한다. Interface Builder에서 이 작업을 간단하게 수행할 수 있다.

1. ToDoDocument.nib.를 연다.
2. Project Builder에서 SelectionNotifyMatrix.h를 클릭하고, nib 파일 윈도우로 드래그한다.
3. 텍스트 셀의 itemList 매트릭스를 선택한다.
4. Info 윈도우의 Custom 화면에서 SelectionNotifyMatrix를 선택한다.
5. nib 파일을 저장한다.

## 통지에 응답하기

선택을 추적할 인프라를 구축하였으므로, 사용자가 itemList에서 ToDoItem을 클릭할 때 게재된 통지에 응답할 코드를 작성해야 한다.

1. ToDoDocument.h의 상단에서(#import 뒤에) ToDoItemChangedNotification을 선언한다.

```
extern NSString *ToDoItemChangedNotification;
```

2. ToDoDocument.m을 연다.
3. 통지 정의를 추가한다.

```
NSString *ToDoItemChangedNotification =
@"ToDoItemChangedNotification";
```

4. 다음 메소드 구현을 추가한다.

```
- (void)rowSelected:(NSNotification *)notification
{
    int row = [[notification object] selectedRow];

    selectedItem = [currentItems objectAtIndex:row];

    if (![selectedItem isKindOfClass:[ToDoItem class]])
        selectedItem = nil;

    [[NSNotificationCenter defaultCenter]
            postNotificationName:ToDoItemChangedNotification
            object:selectedItem
            userInfo:nil];
}
```

5. ToDoDocument를 SelectionNotifyMatrix가 게재한 RowSelectedNotification의 옵저버로 만든다. 이 통지는 사용자가 ToDo 아이템을 클릭하거나 탭 키를 눌렀을 때 발생한다. ToDoDocument의 windowControllerDidLoadNib: 메소드에 코드를 추가한다. 힌트: rowSelected:는 옵저버 셀렉터 및 2개의 addObserver:를 위해 사용되는 메소드이다. 2개의 addObserver:는 다름아닌 itemList 및 statusList으로 호출을 수행한다.

rowSelected: 메소드는 새로운 아이템이 어떤 것인지 파악하기 위해 통지를 통해 전달된 객체에서 획득한 가로행 번호를 사용하여, ToDoItemChanged 통지를 게재한 뒤 Info 윈도우가 새로운 아이템이 변경되었다는 것을 알 수 있도록 한다.

ToDoDocument.m에서 상태 리스트가 정상적으로 나타날 수 있도록 updateLists 메소드에서 2줄의 코드 라인의 주석을 해지한다.

## 데이터 동기화

이제, ToDoDocument 클래스에 기능을 추가하여 Info 윈도우와 데이터 화면을 간단하게 동기화한다. Info 윈도우는 ToDoDocument에서 새롭게 선택된 ToDoItem에 관한 정보를 제공하여, 새로운 도큐멘트가 열리면, 뷰를 업데이트해야 한다.

또한, ToDoDocument는 Info 윈도우에서 아이템을 변경할때 화면을 업데이트한다. 이 섹션에서는 이 같은 동기화 작업을 설명한다.

**ToDoDocument.m**의 **controlTextDidEndEditing:** 메소드는 ToDoItems를 추가, 삭제 또는 변경한다. 그래서, Info 윈도우가 새로운 ToDoItem에서 변경이 이루어지는 시기를 알 수 있도록 한다. 통지와 관련된 객체인 **selectedItem**을 넘겨주어 **ToDoItemChangedNotification**을 게재할 수 있도록 메소드를 변경한다.

1. **ToDoDocument.m**을 연다.
2. **InfoWindowController.h**를 임포트한다. **RowSelectedNotification** 선언을 파악하기 위해 **SelectionNotifyMatrix.h** 도 임포트한다.
3. 다음 코드를 **controlTextDidEndEditing:** 메소드에 추가한다.

```
/* ... */
[self updateChangeCount:NSChangeDone];

[[NSNotificationCenter defaultCenter] postNotificationName:
        ToDoItemChangedNotification object:selectedItem
        userInfo:nil];
```

4. **InfoWindowController.m**을 열어, **windowDidLoad** 메소드를 변경한 뒤 **ToDoItemChanged-Notification**의 옵저버로서 InfoWindowController를 등록한다.

```
[[NSNotificationCenter defaultCenter] addObserver:self
        selector:@selector(selectedItemChanged:)
        name:ToDoItemChangedNotification object:nil];
```

5. **selectedItemChanged:** 메소드 정의를 **InfoWindowController.m**에 추가한다.

```
- (void)selectedItemChanged:(NSNotification *)notification {
    [self updateInfoWindow];
}
```

새로운 ToDoItem으로 변경되면, Info 윈도우가 전송한 **selectedItemModified** 메시지에 ToDoDocument가 정상적으로 응답할 수 있도록 작업해야 한다.

1. **ToDoDocument.h**를 열어, 메소드 선언을 추가한다.

```
- (void)selectedItemModified;
```

2. **ToDoDocument.m**을 열어, 구현을 추가한다.

```
- (void)selectedItemModified
{
    if (selectedItem)
```

```
        [self setTimerForItem:selectedItem];
        [self updateLists];
        [self updateChangeCount:NSChangeDone];
    }
```

이 메소드는 먼저 선택된 아이템의 타이머를 업데이트한 뒤 뷰를 업데이트하여 Info 윈도우에서 이루어진 변경을 반영하고, 변경된 대로 도큐멘트에 표시한다.

응용 프로그램의 1개의 객체에서 다른 객체로 변경 사항을 전달하는 통지는 1개의 객체가 다른 객체에 관한 특정 정보를 보유할 필요성을 배제하기 때문에 훌륭한 디자인이라 할 수 있다. 또한, 어떤 객체는 변경 사항이 발생하면 옵저버가 될 수 있기 때문에 응용 프로그램이 더욱 확장될 수 있도록 한다.

## *ToDoItem 상태를 나타내기 위해 커스팀 뷰 생성하기*

To Do 응용 프로그램의 경우, 3가지 상태 버튼을 구현하여 ToDoItem의 상태를 나타낸다. 버튼은 3단계를 이미지로 보여준다. 3가지 상태는 Not done(이미지가 없음), Done(*X*) 및 Deferred(*O*)이다. 이 상태는 ToDoItem 상태와 일치한다.

이 섹션에서 구현할 ToDoCell 클래스는 3가지 상태 버튼으로 동작하는 셀을 발생시킨다. 이들 버튼은 만료 시간을 제공한다. 다음은 셀 이미지를 프로젝트에 추가하는 방법을 제공한다.

1. Project Builder에서 Resources 그룹 내 Images 그룹을 생성한다.
2. Project 메뉴에서 Add Files을 선택한다.
3. Add Images에서 `DoneMark.tiff` 파일을 선택한다.
4. 독립적인 프로젝트를 생성하고자 한다면, 옵션을 선택하여 파일을 프로젝트 디렉토리에 복사한다.
5. OK를 클릭한다.
6. 동일한 위치에 있는 `DeferredMark.tiff` 파일의 경우에도 동일한 단계를 반복한다.
7. Images 그룹을 생성하였으므로, `LeftArrow.tiff`와 `RightArrow.tiff`를 Images 그룹으로 이동한다.

## NSButtonCell를 슈퍼 클래스로 선택한 이유

ToDoCell의 슈퍼클래스는 NSButtonCell이다. ToDoCell를 생성할 경우, 데이터 및 동작을 NSButtonCell에 추가하고, 기존 동작을 오버라이드한다. 이 작업은 다음과 같은 의문을 제기한다.

- 왜 버튼 자체가 아니라 버튼 셀이어야 하는가?
- 왜 이 슈퍼클래스여야 하는가?

NSCell은 상태를 인스턴스 변수로 정의한다. 그래서, 모든 셀은 인스턴스 변수를 상속받는다. 셀은 효율성을 위해 상태 정보를 보유한다. 컨트롤(매트릭스)은 각각의 상태가 설정된 셀 콜렉션을 관리한다. NSButton은 상태 값을 확보 및 설정하기 위한 메소드를 제공한다. 그러나, 포함하고 있는 셀(NSButtonCell)의 상태값에 액세스한다.

NSButtonCell은 버튼 셀이 원하는 상당량의 동작을 보유하고 있기 때문에 ToDoCell의 슈퍼클래스라 할 수 있다. NSActionCell에서 상속으로 인해, 버튼 셀은 타켓과 액션 정보를 보유할 수 있다. 또한, 이미지와 텍스트를 동시에 제공할 수 있는 고유 기능을 제공한다. 이 같은 기능은 ToDoCell에 필요한 동작이다.

Cocoa 클래스에서 특화된 서브클래스가 필요하다고 생각되면 클래스뿐만 아니라 클래스의 슈퍼클래스와 자매 클래스에 관한 헤더 파일과 참조 도큐멘테이션을 검토하는데 얼마간의 시간을 소요해야한다.

## ToDoCell 인터페이스 구현하기

이 섹션에서는 ToDoCell 클래스 파일을 생성하고, 클래스 인터페이스 선언을 제공하는 방법을 설명한다.

1. Project Builder에서 ToDoCell의 인터페이스 및 구현 파일을 생성한다.
2. NSButtonCell의 슈퍼클래스를 생성한다.
3. <예제 13-9>에서 제공된 상수, 인스턴스 변수 및 메소드 선언을 추가한다.

*<예제 13-9> ToDoCell 인터페이스*

```
typedef enum ToDoButtonState {NOT_DONE=0, DONE, DEFERRED} ToDoButtonState;

@interface ToDoCell : NSButtonCell
{
    ToDoButtonState triState;
    NSImage *doneImage, *deferredImage;
    NSDate *timeDue;
}
- (void)setTriState:(ToDoButtonState)newState;
- (ToDoButtonState)triState;
- (void)setTimeDue:(NSDate *)newTime;
```

*<예제 13-9> ToDoCell 인터페이스 (계속)*

```
- (NSDate *)timeDue;
@end
```

triState 인스턴스 변수는 ToDoButtonState를 값으로써 할당한다. NSImage 변수는 "O"를 보유하고, Completed 및 Deferred(즉 다음날로 일정 변경)로 표시되는 마크 이미지를 확인한다. timeDue 인스턴스 변수는 아이템이 NSDate로써 만료된 시간을 전송한다. 이 객체는 스트링으로 전환되어, 제공된다.

## *Private 메소드 선언하기*

ToDoCell.m 상단에 다음 선언을 추가하여 ToDoCell의 Private 헬퍼 메소드를 선언한다.

```
@interface ToDoCell (PrivateMethods)
- (void)updateImage;
@end
```

## *init과 dealloc 구현하기*

ToDoCell의 init 메소드는 버튼 유형, 이미지 및 텍스트 위치, 텍스트 폰트, 보더 등 슈퍼클래스(NSButtonCell) 속성을 설정한다.

1. <예제 13-10>에서와 같이 init을 구현한다.
2. dealloc을 구현한다.

*<예제 13-10> ToDoCell의 init 메소드 구현*

```
- (id)init
{
    NSString *path;
    [super initTextCell:@""];

    triState = NOT_DONE;
    [self setType:NSToggleButton];
    [self setImagePosition:NSImageLeft];
    [self setBezelStyle:NSShadowlessSquareBezelStyle];
    [self setFont:[NSFont userFontOfSize:10]];
    [self setAlignment:NSRightTextAlignment];

    doneImage=[[NSImage imageNamed:@"DoneMark"] retain];

    deferredImage=[[NSImage imageNamed:@"DeferredMark"] retain];

    return self;
}
```

## 접근자 메소드 구현하기

상태 정보로의 액세스는 ToDoCell에서 듀얼 패스 태스크를 제공한다. 이 작업은 새로운 상태 인스턴스 변수인 triState를 설정하여 확보할 뿐만 아니라 슈퍼클래스 접근자 메소드를 오버라이드하여 상속된 인스턴스 변수를 정상적으로 처리한다.

1. triState 인스턴스 변수를 확보 및 설정하는 메소드를 작성한다.

```
- (void)setTriState:(ToDoButtonState)newState
{
    if (newState > DEFERRED)
        triState = NOT_DONE;
    else
        triState = newState;
    [self updateImage];
}

- (ToDoButtonState)triState { return triState; }
```

2. 상태를 확보 및 설정하는 슈퍼클래스 메소드를 오버라이드한다.

```
- (void)setState:(int)val
{
}

- (int)state
{
    if (triState == DEFERRED)
        return (int)DONE;
    else
        return (int)triState;
}
```

triState의 새로운 값이(DEFERRED)보다 클 경우, 0(NOT_DONE)으로 설정한다. 그렇지 않으면, 값을 할당한다. 그 이유는, 사용자가 ToDoCell를 클릭할 때 setTriState:는 현재 값보다 큰 1로 발생하기 때문이다. 이런 방식은 사용자가 ToDoCell의 3개 상태를 반복적으로 사용할 수 있도록 한다.

setState:는 널(null) 메소드로 오버라이드된다. 그 이유는, 버튼을 클릭하면, NSCell가 상태를 0(NO)으로 설정하면서 중재하기 때문이다. 이 오버라이드는 결과를 무효화시킨다.

## 셀 이미지 설정하기

이 코드는 3개 상태 인수를 파악하고, 셀의 이미지를 적절하게 설정하여 셀 이미지 화면을 처리한다.(setImage:는 NSButtonCell 메소드이다) 그리고, updateCell:을 셀 컨트롤(매트릭스)의 Control 뷰로 전송하여 셀을 다시 드로잉한다.

<예제 13-11>에서와 같이 updateImage: 메소드를 구현한다.

*<예제 13-11> ToDoCell의 updateImage 메소드 구현*

```
- (void)updateImage
{
    switch ([self triState]) {
        case NOT_DONE:
            [self setImage:nil];
            break;
        case DONE:
            [self setImage:doneImage];
            break;
        case DEFERRED:
            [self setImage:deferredImage];
            break;
    }

    [(NSControl *)[self controlView] updateCell:self];

}
```

## *Mouse 클릭 추적하기*

독립적인 셀 서브클래스를 생성하고 나면, 셀 동작 고유의 메소드를 오버라이드 할 수 있다. NSCell에서 상속된 마우스 추적 메소드는 이 같은 메소드이다. 이 메소드를 오버라이드하여 마우스로 셀을 클릭하거나 셀 위에 드래그할 때 특화된 동작을 통합할 수 있다. ToDoCell은 이 메소드를 오버라이드하여 triState 값을 증대시킨다.

<예제 13-12>에서 보여진 바와 같이 2개의 NSCell 마우스 추적 기능을 오버라이드한다.

*<예제 13-12> ToDoCell의 마우스 추적 메소드 구현*

```
- (BOOL)startTrackingAt:(NSPoint)startPoint inView:
(NSView *)controlView
{
    return YES;
}

- (void)stopTracking:(NSPoint)lastPoint at:(NSPoint)stopPoint
    inView:(NSView *)controlView mouseIsUp:(BOOL)flag
{
    if (flag == YES)
        [self setTriState:([self triState]+1)];
}
```

이 경우, startTrackingAt:inView:를 오버라이드하여 YES를 리턴한다. 그리고, ToDoCell이 마우스를 추적한다고 컨트롤에 신호를 보낸다. stopTracking:at:inView:mouseIsUp:을 오버라이드하여, 플래그를 파악하고, 이것이 Mouse-up 이벤트인지 확인한다. Mouse-up 이벤트라면, 메소드는 triState 인스턴스 변수를 증대시킨다.

setTriState: 메소드가 2(DEFERRED)보다 클 경우, 증대된 값을 0(NOT_DONE)으로 "랩"한다.

## 시간 만료 설정하기

setTimeDue: 메소드는 저장하고 있는 NSDate 인스턴스 변수의 해석과 출력을 처리한다는 것만을 제외하고는 다른 세트 접근자 메소드와 유사하다. timeDue 접근자 메소드는 표준이다.

1. 다음과 같이 setTimeDue:를 구현한다.

```
- (void)setTimeDue:(NSDate *)newTime
{
    [timeDue autorelease];

    if (newTime) {
        timeDue = [newTime retain];
        [self setTitle:[timeDue descriptionWithCalendarFormat:
                @"%I:%M %p" timeZone:[NSTimeZone localTimeZone] locale:nil]];
    } else {

        timeDue = nil;
        [self setTitle:@"-->"];
    }
}
```

2. timeDue를 구현하여 NSDate를 리턴한다.

newTime이 유효한 객체라면, NSDate 메소드인 decriptionWithCalendarFormat:timeZone:locale:를 사용하여 setTitle:로 결과를 제공하기 전에 데이터 객체를 해석 및 포맷한다. newTime이 nil이라면, 만료 시간은 명시되지 않는다. 그래서, 메소드는 타이틀을 "→"로 설정한다.

이제, ToDoCell에 필요한 모든 코드를 완성했다. 그러나, To Do 인스턴스에 이 클래스의 인스턴스를 설치해야 한다.

## 커스텀 셀 생성 및 설치하기

이 섹션에서는 새로운 도큐멘트를 생성할 때 ToDoDocument를 변경하여 매트릭스에서 커스텀 셀을 생성 및 설치하는 방법을 설명한다.

1. ToDoDocument.m을 연다.
2. ToDoCell.h를 임포트한다.
3. 다음 코드를 [itemList setDelegate:self]; 뒤의 windowControllerDidLoadNib에 삽입한다.

```
int index;
/*...*/
index = [[statusList cells] count];
while (index--) {
    ToDoCell *aCell = [[ToDoCell alloc] init];
    [aCell setTarget:self];
    [aCell setAction:@selector(itemStatusClicked:)];
    [statusList putCell:aCell atRow:index column:0];
    [aCell release];
}
```

코드의 이 블록은 To Do 인터페이스를 위해 생성했던 좌측 매트릭스(**statusList**)에서 각각의 셀을 ToDoCell로 대체한다. NSMatrix의 **putCell:atRow:column:** 메소드를 발생시켜, ToDoCell을 생성하고, 타켓과 액션 메시지를 설정한 뒤 **statusList**로 삽입한다.

마지막으로, ToDoCells 매트릭스를 클릭할 때 전송된 동작 메시지를 구현해야 한다. (ToDoCell 외부 객체를 위해 Mouse-down에 응답하는 반면, **ToDoCell.m**의 마우스 추적 메소드는 상태를 내부적으로 관리한다)

## 마우스 클릭에 응답하기

**itemStatusClicked:** 메소드는 클릭한 ToDoCell 및 해당 텍스트 필드의 객체를 확보한다. 그 객체가 ToDoItem라면, 메소드는 상태를 업데이트하여 ToDoCell 상태를 반영한다. 그리고, 윈도우에 편집 도큐멘트가 포함되어 있다고 표시한다.

<예제 13-13>에서와 같이 **ToDoDocument.m**에서 **itemStatusClicked:** 구현을 완성한다.

*<예제 13-12> ToDoDocument의 itemStatusClicked: 메소드 구현*

```
- (IBAction)itemStatusClicked:(id)sender
{
    int row = [sender selectedRow];
    ToDoCell *cell = [sender cellAtRow:row column:0];
    id item;

    item = [currentItems objectAtIndex:row];
    if (item && [item isKindOfClass:[ToDoItem class]]) {
        [item setStatus:[cell triState]];

        [self setTimerForItem:item];
        [self updateLists];
        [self updateChangeCount:NSChangeDone];
```

<예제 13-12> ToDoDocument의 itemStatusClicked: 메소드 구현 (계속)

```
        [[NSNotificationCenter defaultCenter]
                postNotificationName:ToDoItemChangedNotification
                object:item
                userInfo:nil];
    }
}
```

# 타이머 설정하기

To Do의 특성 중 하나는 긴급 만료 시간을 사용자 아이템에 통보하는 것이다. 사용자는 Alert 윈도우에서 메시지를 구성하여 이들 통지의 만료 시간 전에 시간 간격을 다양하게 지정할 수 있다. 이 섹션은 To Do의 통지 기능을 구현하는 방법을 설명한다. 이 과정을 통해, 타이머를 생성, 설정 및 응답하는 방법에 대한 기본 원리를 배울 수 있다.

다음은 타이머가 어떻게 동작하는지 보여준다. Info 윈도우에서 선택된 When to Notify Switch(Do Not Notify가 아님)를 가진 각 ToDoItem은(`secsUntilNotify` 값을 가지고 있음) When to Notify Switch를 위한 타이머를 설정한다. 사용자가 Do Not Notify를 선택하여 통지를 취소하면, 도큐멘트 컨트롤러는 타이머를 무효하게 한다. 타이머가 발생하면, Alert 윈도우를 보여주는 메소드를 발생시켜, Do Not Notify 스위치를 선택하고, `secsUntilNotify`를 0에 설정한다.

타이머를 구현하는 것은 Project Builder에서 전적으로 구현된다.

## 아이템의 타이머 설정하기

아이템의 타이머는 날짜, 만료 시간, 통지 시간 또는 상태가 변경될 때 마다 재설정되어야 한다. 아이템의 접근자 메소드가 이 작업을 처리한다. 그러나, 몇 개의 인스턴스 변수가 순서대로 설정될 때가 있다. 이 때, 개별적으로 변경할 때는 타이머를 재설정하지 않는게 낫다. 그래서, 개별적 업데이트 단계를 구현하는 대신 `setTimerForItem:`을 ToDoDocument에서 구현한다.

도큐멘트에서 ToDoItem의 모든 타이머 관련 값은 InfoWindowController에 의해 설정 및 변경될 수 있다. 각각 변경한 뒤에 InfoWindowController는 `selectedItemModified` 메시지를 전송한다. 이 메소드가 선택된 아이템으로 `setTimerForItem:`을 호출하면, Info 윈도우에서 변경할 것은 없다고 보면 된다.

또한, ToDoDocument는 여러 위치에서 ToDoItems를 변경한다. 아이템의 상태 버튼을 클릭하면, ToDoDocument의 `itemStatusClicked:` 메소드는 아이템 상태를 업데이트한다. 이 경우는 이미 구현되었다. 아이템 타이머가 발생한 후, `secsUntilNotify`는 제거되어야 하며, 타이머를 꺼야 한다. 이 섹션에서는 이 기능을 구현하는 방법을 설명한다.

마지막으로, 파일에서 ToDoItems를 해독할 때 각 아이템은 아이템이 초기화된 뒤에 타이머 설정을 가져야 한다. "Archiving 및 Unarchiving(Save 및 Open)구현하기" 색션에서 이 작업을 구현한다.

## *setTimerForItem: 구현하기*

이 메소드는 ToDoItem의 통지 및 상태 값에 따라 타이머를 설정 또는 무효화한다. 그 방법은 다음과 같다.

1. **ToDoDocument.h**를 열어, **setTimerForItem:** 선언을 추가한다.

```
- (void)setTimerForItem:(ToDoItem *)anItem
```

2. **ToDoDocument.m**을 연다.
3. <예제 13-14>에서 보여준 대로 **setTimerForItem:** 메소드를 구현한다.

*<예제 13-14> ToDoDocument의 setTimerForItem: 메소드 구현*

```
- (void)setTimerForItem:(ToDoItem *)anItem
{
    NSDate *notifyDate;
    NSTimer *aTimer;

    if ([anItem secsUntilNotify] && [anItem status] == INCOMPLETE) {
        notifyDate = [[anItem day] addTimeInterval:
                ([anItem secsUntilDue] - [anItem secsUntilNotify])];
        aTimer = [NSTimer scheduledTimerWithTimeInterval:
                [notifyDate timeIntervalSinceNow]
                target:self
                selector:@selector(itemTimerFired:)
                userInfo:anItem
                repeats:NO];
        [anItem setTimer:aTimer];
    } else
        [anItem setTimer:nil];
}
```

먼저, 이 메소드는 ToDoItem이 제로 이외의 **secsUntilNotify** 값과 **INCOMPLETE**의 **status**를 갖고 있는지 확인하기 위해 시험한다. 그 값을 가지고 있을 경우, 메소드는 통지를 전송해야 할 시점을 구성한다. 그리고, 타이머를 생성하여, 일정을 조정한 뒤 적절한 시기에 발생시킨다. 타이머가 발생하면, **itemTimerFired:** 발생한다. 또한, ToDoItem에서 타이머를 설정한다. **secsUntilNotify** 변수가 0이라면, 아이템 타이머를 무효화한다.

## *타이머에 응답하기*

ToDoItem 타이머가 발생하면, **itemTimerFired:** 메소드를 생성한다.(타이머의 일정을 조정할 때 이 메소드를 지정했다는 것을 기억한다)

<예제 13-15>에서 **itemTimerFired:** 구현하기.

*<예제 13-15> ToDoDocument의 itemTimerFired: 메소드 구현*

```
- (void)itemTimerFired:(id)timer
{
    id anItem = [timer userInfo];
    NSDate *dueDate = [[anItem day] addTimeInterval:[anItem secsUntilDue]];

    NSBeep();
    NSRunAlertPanel(@"To Do", @"%@ on %@", nil, nil, nil,
            [anItem itemName],
            [dueDate descriptionWithCalendarFormat:@"%b %d, %Y at %I:%M %p"
            timeZone:[NSTimeZone defaultTimeZone] locale:nil]);

    [anItem setSecsUntilNotify:0];

    [self setTimerForItem:anItem];
    [self updateLists];

    [[NSNotificationCenter defaultCenter]
            postNotificationName:ToDoItemChangedNotification
            object:anItem
            userInfo:nil];
}
```

이 메소드는 만료 시간(NSDate)에 알람 소리를 구성하고, 만료된 ToDoItem 명과 시간을 명시하여 Alert 윈도우를 제공한다. ToDoItem의 **secsUntilNotify** 인스턴스 변수를 0으로 설정하여, 통지가 더 발생하지 않도록 한다. 마지막으로, 화면을 업데이트하고, 아이템이 변경되었다는 통지를 게재한다.

## 구축 및 시험하기

이제, 응용 프로그램을 시험해보기로 하자. 구현되지 않은 유일한 기능은 파일 시스템에서(으로) To Do 도큐멘트를 저장 및 로딩하는 것이다. 다음 순서를 검토하여 To Do 동작을 관찰한다.

1. Document 도큐멘트에서 New를 선택하면, 응용 프로그램은 새로운 To Do 도큐멘트를 생성하여 현재 날짜를 선택한다.
2. 몇 개의 아이템을 입력한다. 캘린더에서 새로운 날짜를 클릭하고, 몇 개의 아이템을 더 입력한다. 이전 날짜를 클릭하고, 입력한 아이템이 어떻게 다시 나타나는지 확인한다.
3. File 메뉴에서 Show Info를 선택한다. Info 윈도우가 나타나면, 아이템을 클릭하고, 아이템의 이름과 날짜가 Info 윈도우의 상단에서 어떻게 나타나는지 확인한다. 2개 정도 아이템과 일부 관련 Note의 만료 시간을 입력한다. 입력하고 난 뒤, 시간이 To Do 도큐멘트의 Status/Due 열에서 어떻게 나타나는지 확인한다. 몇 개의 아이템을 다시 클릭하고, Notifications 및 Notes의 화면이 어떻게 변경되는지 확인한다.

4. Status/Due 버튼을 클릭한다. 이미지는 3개 상태를 토글한다. 만료 시간을 제공하는 아이템에서 통지 시간을 선택한다. 응용 프로그램은 통지 메시지를 통해 Alert 윈도우를 즉시 제공한다. 이 윈도우를 닫으려면, 통지 옵션을 Do Not Notify에 설정한다.

# *Archiving 및 Unarchiving(Save 및 Open) 구현하기*

To Do의 주요 기능을 구현하고, 디버깅했기 때문에 도큐멘트의 저장 및 실행기능을 **ToDoDocument.m**에 추가하여 마무리한다.

## *dataRepresentationOfType 구현하기*

앞서 언급했듯이, 이 메소드는 도큐멘트 데이터를 저장하고, NSData 인스턴스 형태로 이를 리턴한다. 기능은 단순하다.

```
- (NSData *)dataRepresentationOfType:(NSString *)aType
{
    [self saveDocItems];
    return [NSArchiver archivedDataWithRootObject:activeDays];
}
```

이 메소드는 현재 날짜 아이템을 **activeDays** 딕셔네리에 저장하여 보관한다.

## *loadDataRepresentation:ofType 구현하기*

이 메소드는 진행하는 방법을 결정하기 전에 **calendar**가 존재하는지 여부를 점검한다. **calendar**가 존재하면, nib 파일이 이미 로딩되고 도큐멘트 컨트롤러가 데이터를 제공할 것을 요청하는(Revert가 File 메뉴에서 선택되었을때 발생한다)것을 의미한다. **calendar**가 아직 존재하지 않는다면,(nib 파일이 아직 로딩되지 않았음) 메소드는 **dataFromFile** 인스턴스 변수의 도큐멘트 데이터 참조를 저장한다. **windowControllerDidLoadNib**을 호출하면, 메소드는 데이터가 긴급 데이터인지 확인하고, 긴급 데이터라면 도큐멘트에 로딩한다.

1. 인스턴스 변수인 **dataFromFile**을 **ToDoDocument.h**에 추가한다. 변수를 **NSData***로써 정적으로 입력한다.
2. 다음 메소드 정의를 **ToDoDocument.m**에 추가한다.

```
- (BOOL)loadDataRepresentation:(NSData *)data ofType:(NSString *)aType
{
    if (calendar)
```

```
            [self loadDocWithData:data];
        else
            dataFromFile = [data retain];

        return YES;
    }
```

## windowControllerDidLoadNib 변경하기

**windowControllerDidLoadNib:** 메소드로 가서, **[self initDataModelWithDictionary:nil];**를 다음 내용으로 대체한다.

```
    if (dataFromFile)
    {
        [self loadDocWithData:dataFromFile];
        [dataFromFile release];
        dataFromFile = nil;
    } else {
        // A new ToDoDocument is being created so there's no data to load.
        [self loadDocWithData:nil];
    }
```

이 코드는 긴급 데이터를 로딩한다. 긴급 데이터가 없다면, 비어있는 딕셔네리로 도큐멘트를 초기화한다.

## loadDocWithData: 구현하기

다음은 To Do 도큐멘트의 복구 작업을 보여준다. <예제 13-16>에서 보여진 바와 같이, 메소드의 프로토타입을 Private 메소드의 ToDoDocument 리스트와 새로운 인스턴스 변수인 **NSData *dataFromFile;**에 추가하고, 구현을 입력한다.

*<예제 13-16> ToDoDocument의 loadDocWithData: 메소드 구현*

```
- (void)loadDocWithData:(NSData *)data
{
    NSEnumerator *dayEnum, *itemEnum;
    ToDoItem *anItem = nil;
    NSArray *itemArray;
    NSDate *itemDate, *now, *due;
    NSMutableDictionary *dict;
    NSTimeInterval elapsed;

    if (data) {
        dict = [NSUnarchiver unarchiveObjectWithData:data];

        [self initDataModelWithDictionary:dict];

        dayEnum = [activeDays keyEnumerator];
        now = [NSDate date];
        while ((itemDate = [dayEnum nextObject])) {
```

<예제 13-16> ToDoDocument의 loadDocWithData: 메소드 구현 (계속)

```
            itemArray = [activeDays objectForKey:itemDate];
            itemEnum = [itemArray objectEnumerator];

            while ((anItem = [itemEnum nextObject]))
                if ([anItem isKindOfClass:[ToDoItem class]] &&
                    [anItem secsUntilNotify] &&
                    [anItem status] == INCOMPLETE)
                {
                    due = [[anItem day] addTimeInterval:
                           [anItem secsUntilDue]];
                    elapsed = [due timeIntervalSinceDate:now];
                    if (elapsed > 0)
                        [self setTimerForItem:anItem];
                    else {
                        NSBeep();
                        NSRunAlertPanel(@"To Do", @"%@ on %@ is past due!",
                            nil, nil, nil, [anItem itemName],
                        [due descriptionWithCalendarFormat:
                                @"%b %d, %Y at %I:%M %p"
                                timeZone:[NSTimeZone localTimeZone]
                                locale:nil]);

                        [anItem setSecsUntilNotify:0];
                    }
                }
        }
    } else
        [self initDataModelWithDictionary:nil];

    [self selectItemAtRow:0];
    [self updateLists];

    [dayLabel setStringValue:[[calendar selectedDay]
            descriptionWithCalendarFormat:@"To Do on %a %B %d %Y"
            timeZone:[NSTimeZone defaultTimeZone] locale:nil]];
}
```

이 메소드는 **activeDays** 딕셔네리를 복구한 뒤, 이를 사용하여 초기에 구현한 유틸리티 기능으로 컨트롤러의 데이터 모델을 초기화한다. 그리고, 도큐멘트의 모든 ToDo 아이템을 검토하여, 만료 날짜와 알람 세트가 있는지 확인한다. 만료 날짜가 향후에 있다면, 메소드는 새로운 타이머를 생성하여 적절한 시간에 발생할 수 있도록 설정한다. 아이템의 만료 날짜가 지났다면, 메소드는 아이템이 종료되었다고 사용자에게 알린다. 마지막으로, 이 메소드는 리스트에서 ToDo 아이템을 먼저 선택하여, 뷰를 업데이트하고, 도큐멘트 윈도우에서 오늘자 날짜를 작성한다.

# 구축 및 시험하기

이제, 기능적으로 완전히 구현된 멀티 도큐멘트 응용 프로그램이 완성되었다. 일부 아이템을 생성하고, 도큐멘트를 저장해본다. 도큐멘트를 실행시켜 타이머가 정상적으로 재생성되었는지 확인한다.

## 옵션 실행

다른 기능과 동작을 추가하여 To Do 응용 프로그램을 보충할 수 있어야 한다. 다음을 시험해본다.

- **동작중인 ToDo 아이템으로 요일 나타내기** 시각적으로 다르게 보일 수 있도록 캘린더를 변경한다.
- **응용 프로그램 환경설정 구현하기** 패널을 생성한다. 패널이 멀티 화면이라면, InfoWindowController를 위해 수행한 작업을 그대로 이행한다. Preferences는 아이템(리스트의 마지막 아이템 참조)을 로깅 및 퍼징하기 전에 ToDoItems이 종료된 시간, 실행할 기본적인 도큐멘트와 기본적인 Rescheduling간격(다음 아이템 참조) 설정을 제공한다. 사용자 디폴트로 설정된 Preferences를 저장 및 검색한다. 자세한 내용은 NSUserDefaults 규격서를 참조한다.
- **리스케줄링 구현하기** To Do의 Info 윈도우는 Rescheduling 화면을 제공한다. 현재, Rescheduling 화면은 어떤 작업도 수행하지 않고 있다. 특정 기간 별로 아이템 일정을 변경하는 기능을 구현한다.
- **로깅과 퍼징 구현하기** 특정 기간 이후(Preferences을 통해 설정) 종료된 ToDoItems(포맷된 텍스트대로)을 로그에 추가하고, 응용 프로그램에서 ToDoItems를 삭제한다.

# 14

# *To Do: 마무리 작업*

제12장, To Do: 기본 원리에서 To Do 응용 프로그램의 기본적인 구조를 개발하였고, 제13장, To Do: 확장에서 기본적인 프레임워크를 구축하여 고급 기능을 구현했다. 이 장은 To Do 응용 프로그램을 완성하기 위해 마무리 작업을 수행하는 방법을 설명한다.

- Project Builder에서 마지막 응용 프로그램 설정을 구성한다.
- 응용 프로그램 아이콘을 추가한다.
- 도큐멘트 유형 정보(및 도큐멘트 아이콘)를 추가한다.
- 컴파일러를 최적화하여 응용 프로그램 성능을 향상시킨다.

이 장에서는 예제로 To Do 응용 프로그램을 사용하지만, 모든 Cocoa 응용 프로그램(멀티 도큐멘트 응용 프로그램에만 연결된 도큐멘트 유형 정보는 예외)에 동일하게 적용된다.

## *응용 프로그램 설정 구성하기*

응용 프로그램 동작의 다양한 기능을 조정할 수 있는 Project Builder는 여러 응용 프로그램에 특수한 설정값을 제공한다. 이 섹션에서는 모든 옵션을 자세하게 설명하지는 않지만,(자세한 내용은 Project Builder의 온라인 헬프를 참조) 정상적인 응용 프로그램 동작에 중요한 값을 설정하기 위해 수행해야 할 옵션에 대해 설명한다.

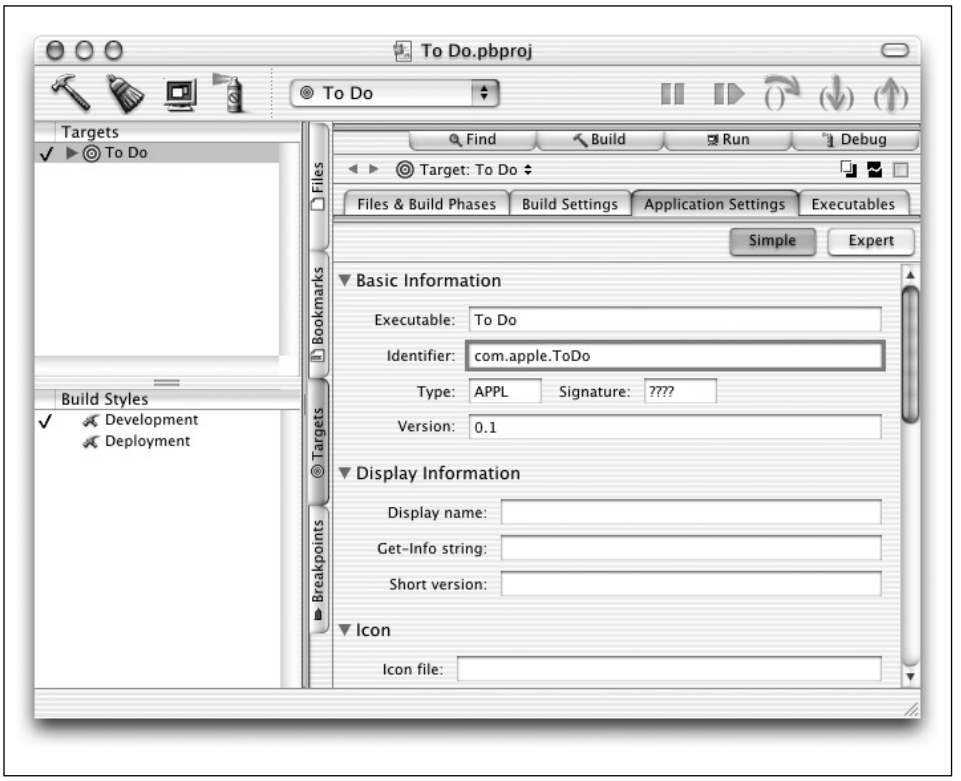
### *번들 식별자 지정하기*

번들 식별자는 Mac OS X에서 응용 프로그램 및 번들을 식별하기 위해 사용하는 메소드이다. 어떤 면에서, 번들 식별자는 응용 프로그램을 식별하기 위해 Mac OS 9이 사용한 크리에이터 코드와 상당히 유사하다.

Mac OS X의 응용 프로그램(Finder 포함)은 프레임워크인 Launch Services를 사용하여 등록된 응용 프로그램, 도큐멘트, URL 스킴 및 관련 정보를 검색한다. Launch Services는 도큐멘트를 열고, 응용 프로그램을 구동하기 위해 API를 제공한다. 예를 들어, Finder에서 도큐멘트를 더블 클릭하면(또는 Dock에서 가상본을 클릭) Launch Services는 번들 식별자를 사용하여 도큐멘트 유형과 파일 확장자를 검토하고, 도큐멘트를 포함하고 있는 응용 프로그램을 배치한다.

Mac OS X 응용 프로그램을 위해 올바른 번들 식별자를 정의하는 것은 중요하다. 이 식별자는 고유한 기능을 제공해야 하기 때문에 Apple은 자바 스타일 네이밍을 채택했다. 이들 이름은 이름뒤에 인터넷 도메인 명의 특성을 제공한다. 예를 들어, ToDo 응용 프로그램의 Apple 버전은 번들 식별자로 com.apple.ToDo를 사용한다. 개인이나 회사 도메인 명을 기반으로 하여 응용 프로그램에 적합한 식별자를 선택해야 한다.

번들 식별자를 지정하려면, <그림 14-1>에서 보이는 바와 같이, Project Builder의 메인 윈도우에서 Target 옵션의 응용 프로그램 Settings 탭을 클릭한다. 식별자 스트링을 Basic Information 영역에 입력한다.




*<그림 14-1> Project Builder의 응용 프로그램 Settings 패인*

## 응용 프로그램 시그너쳐 지정하기

번들 식별자를 비롯하여 응용 프로그램에 *응용 프로그램 시그너쳐*(또한 *크리에이터 코드*라고 지칭)를 지정할 수 있다. 응용 프로그램 시그너쳐는 Launch Services가 도큐멘트를 소유하고 있는 응용 프로그램을 식별하기 위해 사용할 수 있는 고유한 4개의 문자 코드이다. 시그너쳐는 Mac OS 9의 `OSType`과 동일하다. 응용 프로그램의 시그너쳐를 Classic 응용 프로그램과 Classic 데스크탑 데이터베이스에 적용하면 Mac OS 9에서와 동일하게 사용된다. 도큐멘트를 열기 위해 사용할 응용 프로그램을 검색하면 Launch Services는 응용 프로그램의 시그너쳐보다 먼저 제공된다.

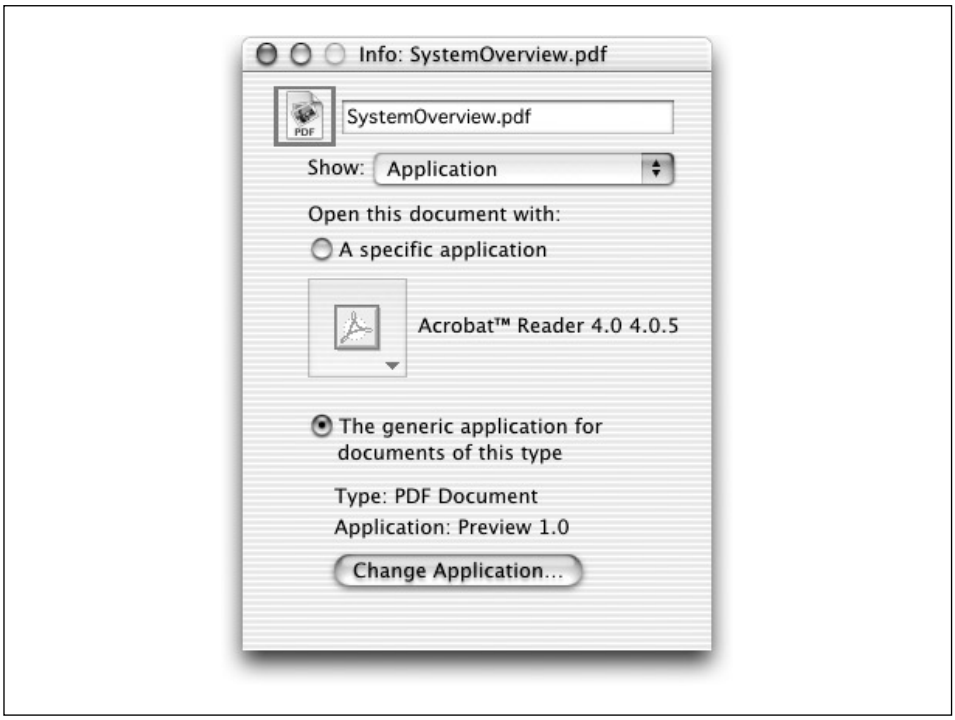
응용 프로그램이 응용 프로그램 시그너쳐를 제공하고, 도큐멘트가 크리에이터를 공유하면, 도큐멘트는 응용 프로그램에 *밀접하게 연결*된다고 말한다. 도큐멘트 파일명 확장자를 제거하는 작업은 도큐멘트를 열기 위해 응용 프로그램을 선택하는데 별 문제가 되지 않는다. 도큐멘트가 응용 프로그램 시그너쳐를 보유하고 있는 한, 도큐멘트를 처리하려면 응용 프로그램을 불러와야 한다. 응용 프로그램에 도큐멘트가 밀접하게 연결되지 않도록 하려면, 응용 프로그램 시그너쳐를 도큐멘트에 제공하지 않으면 된다.

실제로, X의 경우 우선적으로 취해야 하는 작업은 도큐멘트에 크리에이터 코드를 작성하지 않는 것이다. 여기서 도큐멘트란 사용자가 선호하는 응용 프로그램(Finder의 Info 윈도우에서 선택함)에서 열리는 도큐멘트를 의미한다. 이 작업은 이미지, 텍스트 파일등 광범위하게 사용되는 도큐멘트 유형에 적합하다. 사용자는 원한다면 <그림 14-2>에서 보여진 바와 같이 Finder에서 도큐멘트에 밀접하게 연결될 수 있다.

사용자는 Finder에서 제공된 유형의 도큐멘트를 선택하고, Info 윈도우의 응용 프로그램 패인을 불러와 특정 응용 프로그램이나 일반적인 응용 프로그램에서 실행될 수 있는 도큐멘트를 선택해야 한다. 도큐멘트가 응용 프로그램에 밀접하게 연결되어 있는 경우 이런 방식으로 도큐멘트의 실행 바인딩을 변경한다면 효과가 없다.

일반적으로, 응용 프로그램 시그너쳐는 이를 필요로 하는 서브시스템이 일부 존재한다 하더라도 OS X 전용 응용 프로그램에서는 더 이상 필요하지 않다. 이 같은 서브시스템은 InternetConfig이다. 응용 프로그램이 도큐멘트를 열어, InternetConfig에 배치한 경우, 응용 프로그램의 시그너쳐가 있어야 한다. 또한, OS 9에서 크리에이터 코드를 정상적으로 실행하는 도큐멘트를 생성한 응용 프로그램에는 크리에이터 코드를 작성해야 한다.

Apple은 응용 프로그램 시그너쳐의 데이터베이스를 제공한다. 개발자는 사용하고 있는 응용 프로그램이 Apple의 *http://developer.apple.com/dev/cftype/register.html*에 등록되어 있는지 확인한다.




*<그림 14-2> Finder에서 도큐멘트의 실행 바인딩 변경하기*

### 화면 정보 지정하기

Display Name은 응용 프로그램의 메뉴 바에 나타난 이름이다. 이 이름은 메뉴바의 공간을 너무 차지하지 않도록 최대 16문자로 제한한다. 이에 비춰 봤을 때, To Do는 사용하기 적합한 이름이다.

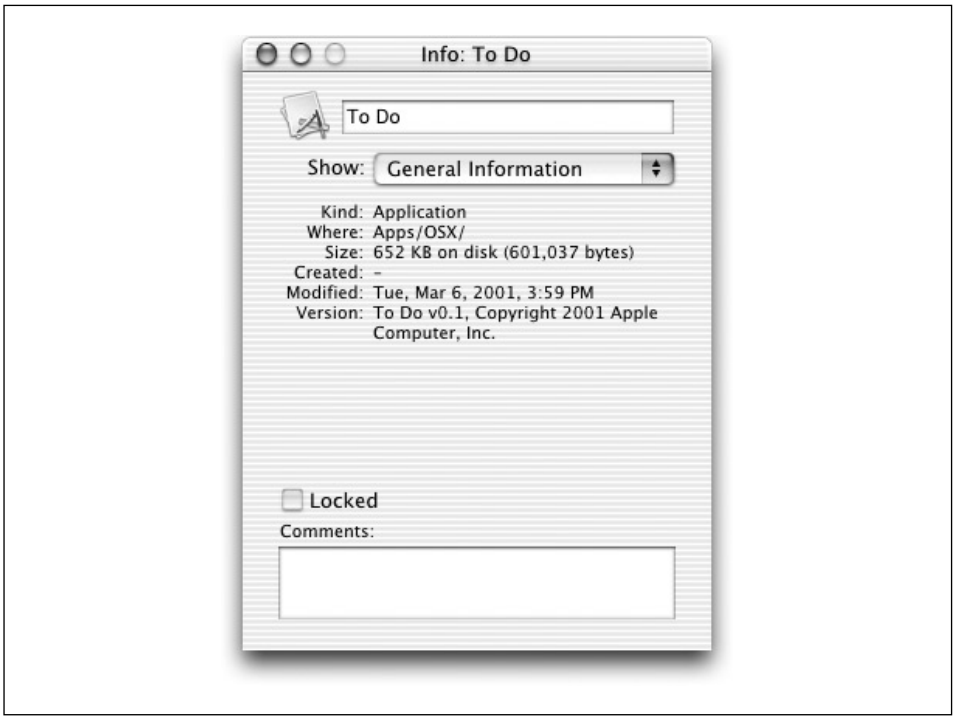
Get-Info 스트링은 Finder의 Info 윈도우(<그림 14-3> 참조)에 나타나야 하는 정보이다. 일반적으로, 번들의 풀 네임, 버전 번호 및 저작권 정보로 구성된다.

Short Version은 버전 번호(예를 들면, 1.0)이다.

## 응용 프로그램 아이콘 추가하기

Mac OS X의 개발자 툴 패키지는 Mac OS X 스타일 아이콘 파일로 이미지를 전환시키는 단순한 유틸리티인 Icon Composer를 제공한다. 표준 그래픽 응용 프로그램을 사용하여, 아이콘의 아트를 생성할 수 있으며, TIFF, PICT 또는 포토샵 포맷에서 32비트 이미지로 저장하고, Icon Composer로 임포트한다. 이미지가 임포트되면, Icon Composer는 Mac OS X이 사용하는 `icns` 포맷에 파일을 저장한다. 따라서, 아이콘 그룹이 저장된다.

Aqua Human Interface Guidelines 도큐멘트(*http://developer.apple.com/techpubs/macosx*)는 Mac OS X의 아이콘 생성에 관한 중요한 정보를 제공한다.

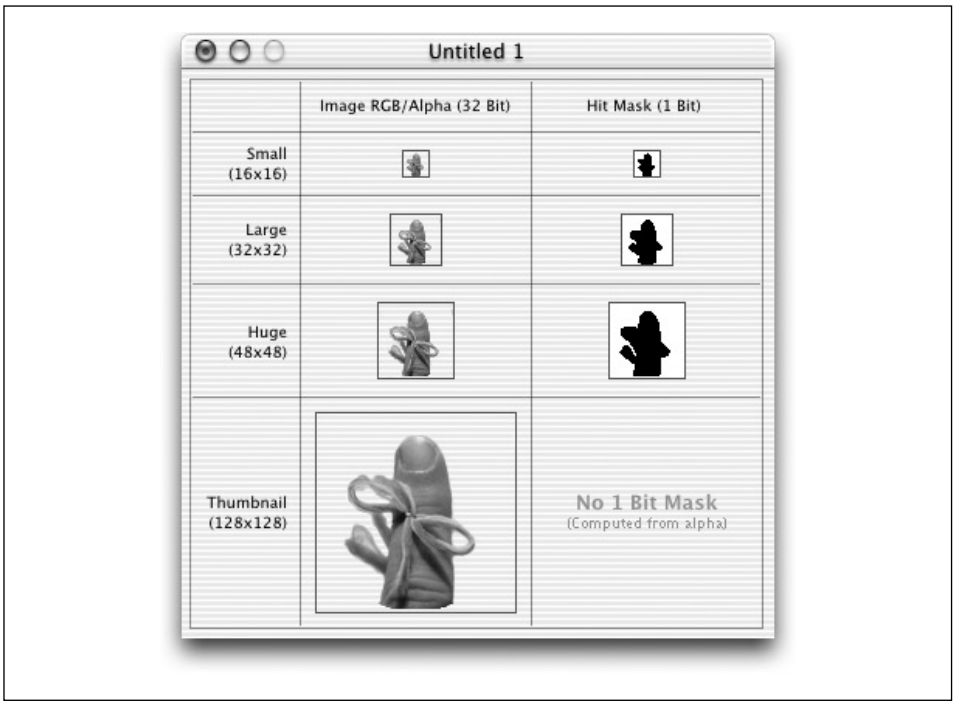



*<그림 14-3> To Do의 Get-Info 스트링을 보여주는 Finder Info 윈도우*

이 섹션에서는 Adobe Photoshop으로 편리하게 작업하기 위해 응용 프로그램을 사용하여 몇 초내에 아이콘의 소스 아트를 생성하는 방법을 설명한다. 포토샵과 친숙하지 않다면, 샘플 파일에 포함된 `ToDo.tiff`를 사용하고, 다음 단계를 건너뛰어도 된다.

1. 포토샵을 구동하고, 백지 환경에 새로운 128×128 픽셀 이미지를 생성한다.
2. 아트를 도큐멘트로 드로잉하거나 베끼기/붙이기한다.
3. Channels 탭을 사용하여 새로운 이미지 채널을 생성한다. 이 새로운 채널은 알파 마스크로 동작한다. Finder 및 Dock에서 백지 마스크 영역을 페인트칠 한다.
4. 파일을 TIFF 이미지로써 저장한다. Cocoa에서 이미지 파일로 권장하는 포맷이다.

5. `/Developer/Applications`에서 Icon Composer 응용 프로그램을 구동한다.
6. File 메뉴에서 Import Image를 선택한다. Icon Composer의 Open 대화 상자는 아이콘에 관한 설명을 명시하여 임포트된 이미지를 생성할 수 있도록 하는 팝업 메뉴의 기능을 한다. 썸네일 아이콘이 필요하면, 대화 상자 하단의 Import To 팝업 메뉴에서 Thumbnail 32Bit Data를 선택하고, TIFF 도큐멘트를 실행한다.
7. Thumbnail 가로행에서 Huge, Large 및 Small 가로행(<그림 14-4> 참조)으로 이미지를 드래그하여 드롭한다. Icon Composer는 이미지를 자동적으로 각각의 크기에 맞춰 조절한다. 아이콘 이미지가 잘 조정되지 않으면, 변화를 줘서 사용자가 응용 프로그램의 아이콘을 명확하게 식별할 수 있도록 해야 한다. "응용 프로그램 아이콘 추가하기" 섹션은 Icon Composer의 메인 윈도우에서 이용 가능한 모든 크기로 아이콘을 생성하여 완성된 `ToDo.icns` 파일을 보여준다.




*<그림 14-4> 완성된 To Do 아이콘*

8. Icon Composer에서 파일을 `ToDoApp.icns`로 저장한다.
9. 아이콘 파일을 Project Builder 프로젝트에 추가한다. 파일을 프로젝트에 추가하려면 프로젝트 디렉토리로 복사해야 한다.

10. 응용 프로그램 Settings 패인에서 아이콘 파일 명을 텍스트 필드 라벨이 붙은 Icon에 입력한다.
11. 프로젝트를 구축한다. Finder에서 새로운 아이콘을 확인해야 하며, 응용 프로그램이 구동될 때 Dock에서 출력되어야 한다.

## To Do의 도큐멘트 유형 정의하기

제11장, *Cocoa의 멀티플 도큐멘트 아키텍처*에서 도큐멘트 유형에 관하여 배웠다. 그래서, To Do 도큐멘트 유형을 설정하는 작업에 익숙해져야 한다.

1. Project Builder의 Document Types 리스팅에서 기본적인 도큐멘트 유형을 선택한다. 명칭을 **ToDoDocument**로 변경한다.
2. Editor에 설정된 역할을 수행한다.
3. 도큐멘트 확장자를 tdo로 한다.
4. "응용 프로그램 시그너쳐 지정하기" 섹션에서 "밀접하게 연결된 도큐멘트"를 생성하려면 응용 프로그램 Signature와 도큐멘트가 관련이 있다는 것을 배웠다. 원한다면, To Do 응용 프로그램 시그너쳐를 To Do 도큐멘트 유형에 추가할 수 있다. OS 타입 리스트에 단순히 **ToDo**를 입력한다. 강력하게 구속된 응용 프로그램을 생성하지 않으려면, 빈칸(기본적인 엔트리 "????"를 삭제한다)을 그대로 두면 된다.
5. 아이콘을 도큐멘트 유형에 연결하려면, 상기 섹션에서 사용한 동일한 절차를 수행하여 응용 프로그램 아이콘을 생성한다. 그리고, 도큐멘트 아이콘 파일 명을 지정한다.

## Compiler Optimization 사용하기

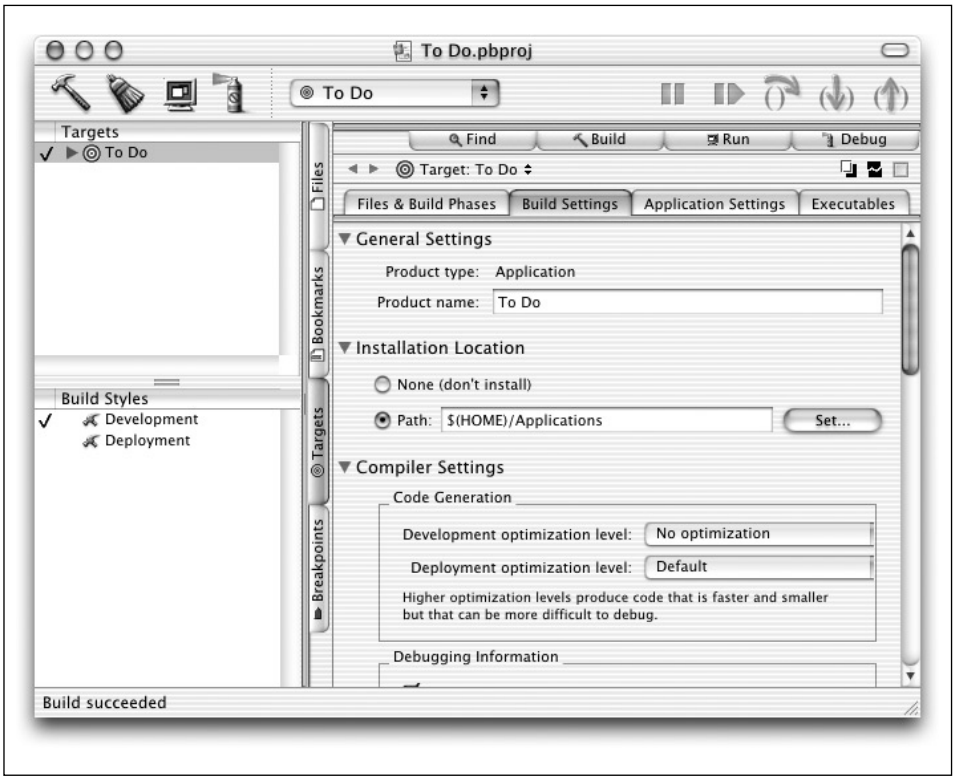
이 섹션에서는 Compiler Optimization을 불러와 응용 프로그램의 마지막 마무리를 하는 작업에 관해 설명한다. 여러 Optimization Level을 선택할 수 있다. 일반적으로, Optimization Level가 높아질수록 컴파일링에 소요되는 시간은 길어지고, 응용 프로그램을 디버깅하는 작업은 어려워진다. 응용 프로그램 최적화에 관한 자세한 내용은 Project Builder에서 Developer Tools Help를 선택하면 된다.

최적화하려면,

1. <그림 14-5>에서 보이는 바와 같이, Project Builder에서 Target 옵션 화면의 Build Settings 패인을 선택한다.
2. Deployment Optimization Level을 선택한다.

응용 프로그램의 최종 버전을 구축하려면, Build Styles 패인에서 Deployment를 선택한다. 이 작업을 통해 선택한 Optimization Level을 사용할 수 있다.

원한다면, 설치 위치를 선택한다. 이 작업을 통해 Build를 완성한 경우 응용 프로그램을 복사할 폴더를 선택할 수 있다. Project Builder 이외의 완성된 응용 프로그램을 시험할 수 있도록 한다.




*<그림 14-5> Project Builder의 Build Settings 윈도우*

# IV

## 참조

# A

# Cocoa에서 드로잉하기

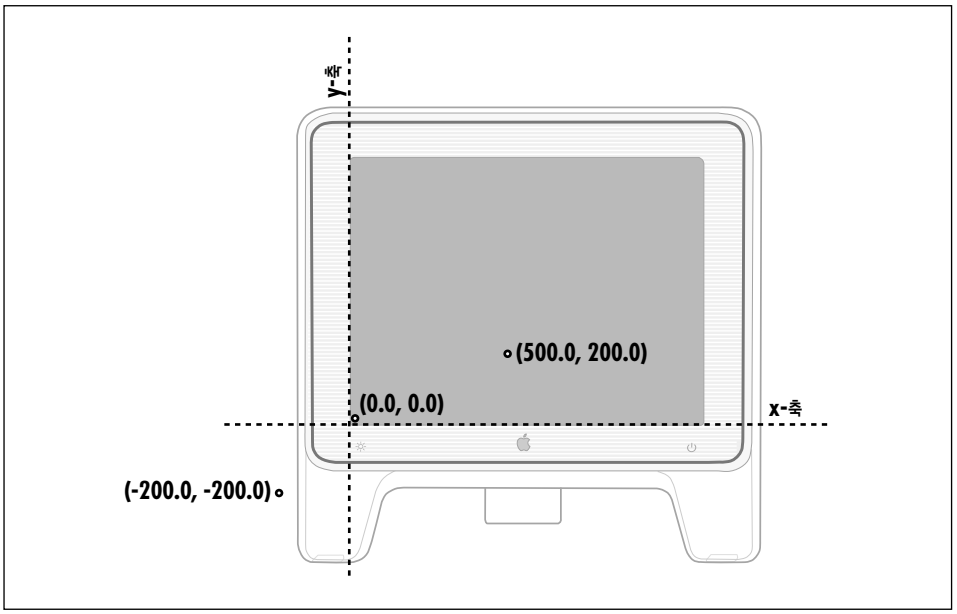
Cocoa의 Application Kit는 광범위한 사용자 인터페이스 객체를 제공하지만, 대부분의 복잡한 응용 프로그램은 최소한 몇 개의 커스텀 인터페이스 요소만을 필요로 한다. 여러 응용 프로그램에서 데이터를 제공하려면 커스텀 드로잉을 수행해야 한다. 이 부록은 Cocoa 이미지 모델의 기본 원리에 관해 설명하고, 커스텀 그래픽 생성 방법을 제공한다.

Cocoa의 드로잉 장치를 최대한 활용하려면, Mac OS X 그래픽을 뒷받침하는 이미지 모델인 Quartz에 관해 배워야 한다. Quartz는 Adobe사의 Portable Document Format(PDF)를 기반으로 한다. PDF는 Adobe사의 PostScript 이미지 모델을 기반으로 한다. Quartz, PostScript, PDF에 관한 설명은 이 부록에 나와 있지는 않다. 따라서, Adobe의 PostScript Language Reference("Red Book"으로 통용됨)와 PDF 1.3 규격서(*http://www.pdfzone.com/resources/pdfspec13.html*)를 참조하면 된다.

## 좌표계

화면 좌표계는 배치, 크기조정, 드로잉 및 이벤트 처리에 사용되는 다른 좌표계의 기준이 된다. <그림 A-1>에서 보이는 바와 같이 2차원 좌표 그리드가 화면 전체를 점유하고 있는 것으로 보인다. 사용자에게는 보이지 않지만, 3개의 사분면이 *x*-축, *y*-축, 또는 이 2개 축선의 음수 값을 취한다. 사분면의 시작점은 좌측 하단 코너에 있다. x-축은 수평선으로, y-축은 수직선으로 표현된다. 또한, 각 선을 픽셀이라 한다.

<그림 A-1>은 1개의 디스플레이 장치를 예로 들어 화면 좌표계를 보여주지만, 화면 좌표계는 컴퓨터에 있는 모든 물리적 프레임 버퍼의 화면 사각형이 결합한 논리적인 사각형 영역이다. 좌표계의 시작점은 좌측 하단 코너에 있다.



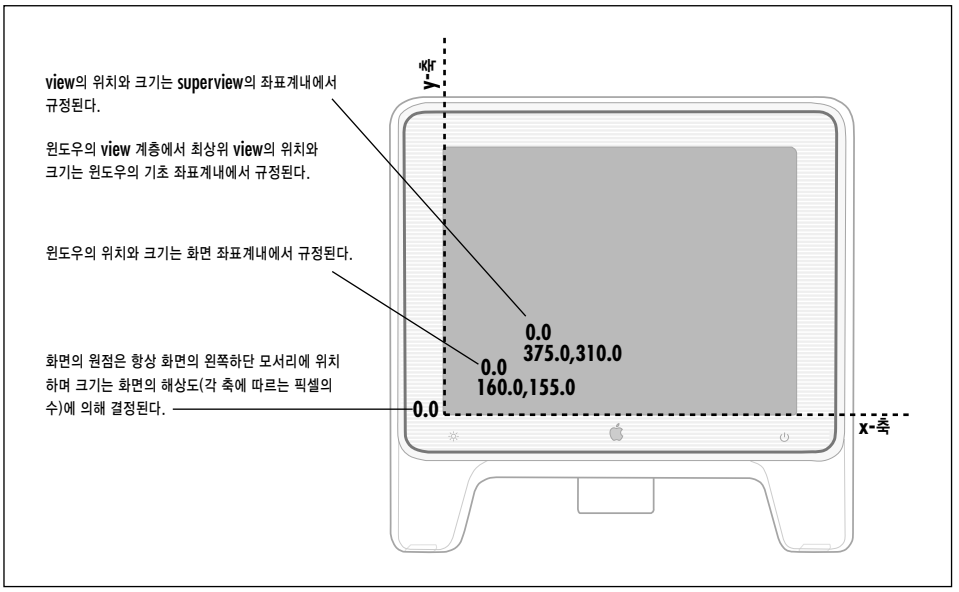

*<그림 A-1> 스크린 좌표계*

화면 좌표계는 단 1개의 함수를 제공한다. 그 함수는 다름아닌 화면에 윈도우를 배치하는 것이다. 응용 프로그램에서 새로운 윈도우를 만드려면, 화면상에 윈도우의 크기와 위치를 지정해야 한다.

윈도우의 참조 좌표계<그림 A-2>는 *기준 좌표계*라고도 한다. 화면 좌표계는 2가지의 측면에서 화면 좌표계와 다르다.

- 화면 좌표계는 특정 윈도우에만 적용된다. 각 윈도우는 각자 기준 좌표계를 가지고 있다.
- 좌표계의 시작점은 화면의 좌측 하단 코너가 아닌 윈도우의 좌측 하단 코너에 있다. 윈도우가 움직이면, 시작점과 전체 좌표계는 윈도우를 따라서 움직인다.

각 NSView(제8장, *이벤트 처리* 및 이 장 후반부 "NSView"에서 설명됨)는 기준 좌표계나 슈퍼뷰의 좌표계에서 변형된 좌표계를 사용한다. 좌표계는 NSView의 좌측 하단에서 시작점을 제공한다. 이로써 드로잉 기능을 더욱 편리하게 사용할 수 있다. NSView는 기준 좌표계와 지역 좌표계를 변경하기 위해 여러 메소드를 제공한다. 드로잉 작업을 할 때, 응용 프로그램의 좌표계는 좌표를 제공한다.




<그림 A-2> 윈도우 좌표계

특정 NSView로 드로잉하기에 앞서, 포커스를 뷰(NSView의 lockFocus 메소드 사용. 이 메소드는 이 장의 뒷 부분에서 상세히 설명됨)에 "고정"해야 한다. 이로써, NSView의 좌표계가 드로잉 커맨드를 모두 적용할 수 있는 새로운 좌표계가 될 수 있다.

## 변환행렬

변환행렬은 1개의 좌표 영역에서 다른 좌표 영역으로 포인트를 맵핑한다. Cocoa와 Quartz는 스케일링, 회전, 이동같은 표준 그래픽 화면을 제공하기 위해 변환행렬을 사용한다.

2차원 변환을 수행하기 위해 사용되는 행렬은 3×3이다. <그림 A-3>은 3×3 행렬의 예제를 보여준다. u 및 v는 항상 0.0이며, w는 항상 1이다.

3×3 행렬 내용은 다음 등식에 의해 포인트 $(x, y)$를 포인트 $(x', y')$로 변환한다.

$$x' = ax + cy + t_x$$

$$y' = bx + dy + t_y$$

예를 들면, <그림 A-4>에서 보여준 행렬은 이동 기능을 수행하지 않는다. 이를 *항등 행렬*이라 한다.

상기 언급한 공식을 통해, 이 행렬이 기존의 포인트 $(x, y)$와 동일한 새로운 포인트 $(x', y')$를 생성한다는 것을 확인할 수 있다.

$$\begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} a & b & u \\ c & d & v \\ t_x & t_y & w \end{bmatrix} = \begin{bmatrix} x' & y' & 1 \end{bmatrix}$$

<그림 A-3> 3×3 행렬로 변환된 포인트

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

<그림 A-4> 항등 행렬

$$x' = 1x + 0y + 0$$

$$y' = 0x + 1y + 0$$

$$x' = x \text{ and } y' = y$$

특정 변위로 이미지를 움직이려면, 이동 기능을 사용해야 한다. 이 기능은 명시된 합계에 의해 각 포인트의 $x$ 및 $y$ 좌표를 변경한다. <그림 A-5>에서 제공된 행렬은 이동 기능에 관해 설명한다.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

<그림 A-5> 이동 기능을 표현한 행렬

스케일링 기능을 수행하여 이미지를 확대 또는 축소할 수 있다. 이 기능은 스케일링 요소에 의해 $x$와 $y$좌표를 변경한다. $x$와 $y$요소는 새로운 이미지를 원래의 이미지보다 크거나 작게 할 것인지 결정한다. $x$를 음수로 만들어 $x$-축의 이미지를 신속하게 이동할 수 있다.

마찬가지로, *y*를 음수로 만들어 *y*-축 이미지를 수평으로 신속하게 이동할 수 있다. <그림 A-6>의 행렬은 스케일링 기능에 관해 설명한다.

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

*<그림 A-6> 스케일링 기능을 설명한 행렬*

마지막으로, 회전 기능을 수행하여 특정 각도로 이미지를 회전할 수 있다. *x*와 *y*의 요소를 명시하여 회전의 크기 및 방향을 지정한다. <그림 A-7>의 행렬은 $\theta$ 각도에 의해 이미지를 시계 반대 방향으로 회전하는 기능에 관해 설명한다.

$$\begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & -\cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

*<그림 A-7> 회전 기능을 설명한 행렬*

여러 변환 행렬을 단일 행렬로 정의할 수 있도록 행렬을 결합할 수 있다. 그 결과로 발생된 행렬은 2개의 변환 속성을 보유한다. 예를 들면, <그림 A-8>의 행렬과 유사한 행렬을 정의하면 이미지를 동시에 스케일링 하거나 이동할 수 있다.

$$\begin{bmatrix} s & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

*<그림 A-8> 스케일링 및 이동 기능을 설명한 행렬*

2개의 행렬을 연결하여 결합할 수 있다. 수학적으로, 2개의 행렬은 곱셈을 통해 결합된다. 행렬을 연결하는 순서는 행렬 기능이 가환적이지 않기 때문에 상당히 중요하다.

변환 행렬을 처리하는 함수는 Application Kit의 `NSAffineTransform.h`와 Core Graphic에서 찾아볼 수 있다. 이 장의 뒷 부분의 "Quartz로 드로잉하기"는 이미지를 크기 조정하기 위해 변환 행렬을 사용하는 예제를 제공한다.

## *NSView*

NSView에서 상속한 모든 객체는 화면에 나타날 수 있다. 화면에 나타나려면, NSView는 NSWindow에 배치되어야 한다. NSViews는 Core Graphics(Quartz) 프로그래밍 인터페이스로 호출하여 드로잉한다.

NSView는 그래픽하에서 프레임 캔버스로 취급될 수 있다. 프레임은 NSView를 슈퍼뷰에 배치하고, 크기를 정의하며, 드로잉을 경계면에 고정한다. 반면에 캔버스는 NSView의 내부 좌표계를 정의하여 실제 드로잉을 수행한다. 프레임을 이동하거나, 크기를 조정하고, 슈퍼뷰에서 회전시킬 수도 있다. 그러면, NSView의 이미지도 따라서 움직인다. 마찬가지로, 캔버스를 움직이거나, 확대 및 회전시킬 수 있다. 그러면, 이미지가 프레임내에서 움직인다.

NSView는 드로잉 작업을 수행할 수 있는 영역내에서 컨텍스트를 제공한다. 이 컨텍스트는 3개의 구성 요소를 가지고 있다.

- 드로잉을 윈도우에 고정할 수 있는 사각형 프레임
- 좌표계
- 현재 그래픽 상태

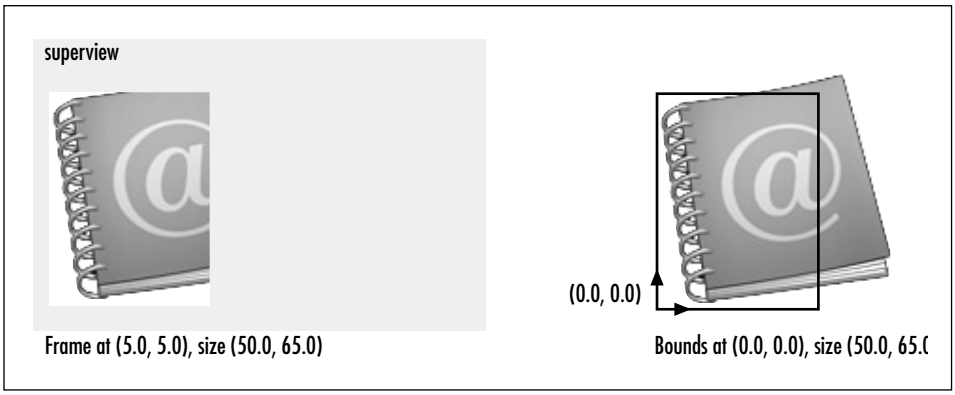
NSView는 `display` 메시지(또는 `display`의 Variant)를 수신하거나 자동 디스플레이 발생을 통해 드로잉 작업을 한다. `display` 메시지는 NSView의 `drawRect:` 메소드 및 NSView의 모든 서브뷰에 있는 `drawRect:` 메소드를 발생시킨다. `drawRect:` 메소드는 NSView를 완전히 다시 드로잉하기 위해 필요한 모든 코드를 제공해야 한다.

NSView 객체는 다음 경우에 자동으로 나타날 수 있다.

- 사용자가 스크롤할 경우(스크롤링 기능이 지원되고 있다는 가정 하에)
- 사용자가 NSView 윈도우의 크기를 조정하거나 노출시킬 경우
- 윈도우가 디스플레이 메시지를 수신하거나 자동으로 업데이트된 경우
- 속성이 Application Kit 객체를 위해 변경된 경우

## 프레임 및 바운드

NSView 슈퍼뷰의 좌표계 측면에서 NSView의 프레임은 NSView의 위치와 크기를 명시한다. <그림 A-9>에서 보이는 바와 같이 NSView를 둘러싸고 있는 사각형이 NSView이다.




*<그림 A-9> 프레임 및 바운드*

NSView를 프레임(`setFrameOrigin:`, `setFrameSize:` 등)을 참조하여 슈퍼뷰내에서 이동, 크기조정 및 회전할 수 있다.

NSView는 효과적으로 드로잉하기 위해 좌표계로 전환된 사각형 프레임을 제공해야 한다. 드로잉에 적합한 이 사각형 프레임을 *바운드*라 한다. 사각형 바운드는 사각형 프레임과 동일한 영역을 제공한다. 그러나, 사각형 바운드는 다양한 좌표계에서 그 영역을 명시한다. 기본적인 좌표계에서 NSView의 바운드는 프레임에 배치된 포인트가 바운드의 시작점($x$=0.0, $y$=0.0)이 된다는 것을 제외하고는 프레임과 동일하다. 기본적인 좌표계의 $x$-축과 $y$-축은 프레임의 경계면과 평행한다. 예를 들면, 프레임을 회전시킬 경우 기본적인 좌표계는 프레임을 따라서 회전한다. 프레임과 바운드의 관계는 드로잉과 구성면에서 중요한 관련을 갖는다.

- 각 NSView의 좌표계는 슈퍼뷰가 변환된 것이라 할 수 있다.
- 드로잉 명령어는 화면이나 좌표에서 NSView의 위치를 일일히 설명할 필요는 없다.
- 슈퍼뷰의 좌표계에서 변경이 이루어지면 서브뷰로 전달된다.

NSView는 좌표계를 신속하게 이동시키며(따라서, 양수 $y$-축이 아래로 이동할 수 있다) 좌표계를 변경한다.

## 포커스

display... 메소드가 NSView의 drawRect:를 발생시키기 전에 그래픽 컨텍스트, 좌표계, 클리핑 패스 및 기타 그래픽 상태 정보를 비롯한 뷰에 관한 정보를 사용하여 Core Graphics을 설정한다. 이 작업을 수행하기 위해 lockFocus 메소드가 사용된다. lockFocus를 취소할 수 있는 메소드는 unlockFocus 메소드이다. 포커스는 현재 그래픽 상태를 다음과 같이 변경한다.

- NSView 윈도우를 새로운 그래픽 컨텍스트로 만든다.
- NSView 프레임 주위에 클리핑 패스를 생성한다.
- Core Graphics 좌표계를 NSView의 좌표계와 일치시킨다.

NSView이 생성한 모든 드로잉 코드는 정상적인 결과를 산출하기 위해 이들 메소드 발생을 괄호로 묶어야 한다. display... 메소드를 수행하지 않으면서 뷰에서 드로잉해야 하는 메소드를 정의할 경우, Core Graphics으로 명령어를 전송하기에 앞서 드로잉 작업을 하는 뷰에 lockFocus를 전송하고, 작업이 끝나는 순간 unlockFocus를 전송해야 한다.

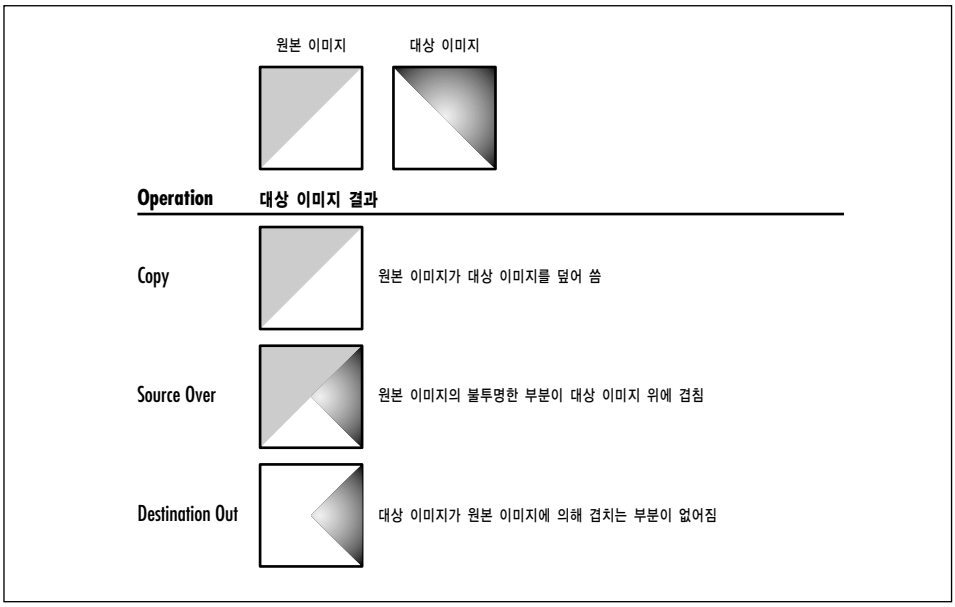
단 1개의 NSView만이 포커스를 제공할 수 있다. lockFocus가 발생했을 때 포커스가 다른 NSView에 고정되어 있으면, 포커스는 스택에 배치된다. 그래서, unlockFocus가 호출될 때 포커스를 복구할 수 있다.

모든 lockFocus가 1개의 unlockFocus와 정확히 균형을 이루어야 한다.

## 합성 이미지

NSViews의 또 다른 기법으로는 *이미지 합성*이 있다. 이미지 합성은 이전에 드로잉한 이미지를 오버레이하여 새로운 이미지를 만들 수 있을 뿐만 아니라 복사도 할 수 있다. 사진사처럼 2개의 음화에서 이미지를 프린트한다. 다양한 합성 오퍼레이터는 소스 및 목적지 이미지가 합성되는 방법을 결정할 수 있다. <그림 A-10>은 3개의 기본적인 오퍼레이터를 나타낸다.

NSImage를 사용하여 이미지를 사용자 인터페이스에 복사할 수 있다. NSImageRep의 색상, 회색음영, TIFF, PDF 등 다양한 서브클래스를 사용하여 동일한 이미지의 다양한 표현을 저장하고, 제공된 화면 유형에 적합한 표현을 선택한다. 또한, 번들(응용 프로그램의 메인 번들 포함) 파일, 페이스트보드 또는 NSData 객체에서 이미지 데이터를 읽을 수 있다. NSViews는 이미지를 합성하여 (SourceOver 오퍼레이터 사용) 프레임내에 이미지를 나타낼 수 있다. 일반적으로, NSImage의 compositeToPoint:operation:(또는 관련 메소드)를 사용하여 이미지를 합성한다. 초기 이미지를 부분적으로 투명한 색으로 페인트칠하면 흥미로운 합성 효과를 저장할 수 있다.



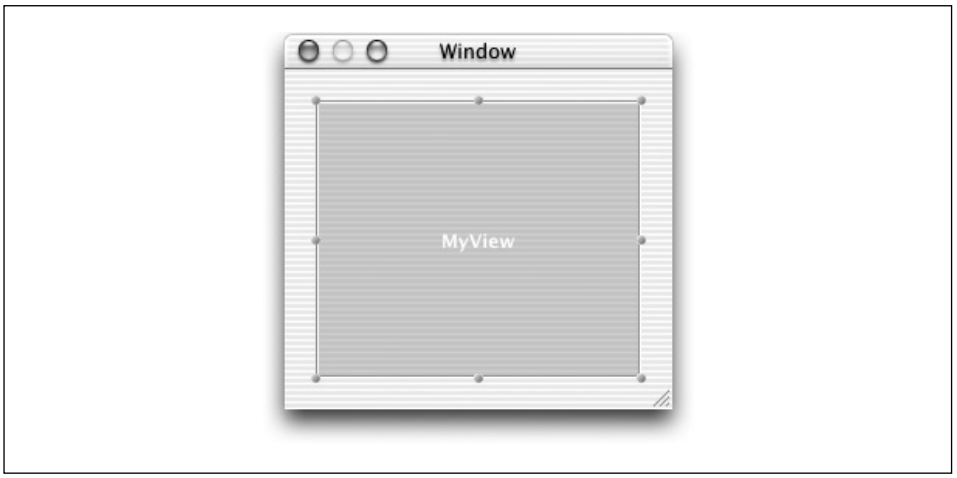

*<그림 A-10> 기본적인 합성 오퍼레이터*

`/Developer/Examples/AppKit`의 CompositeLab 응용 프로그램은 Cocoa에서 합성 작업을 수행하는 방법에 관한 예제를 제공한다.

## *NSStrings 드로잉하기*

이 섹션에서는 상당히 간단한 드로잉을 제공하여 커스텀 NSView 서브클래스의 좌측 상단 코너에 스트링을 표현할 응용 프로그램 생성하는 방법을 설명한다.

1. 새로운 Cocoa 응용 프로그램 프로젝트인 Simple Draw를 만든다.
2. 메인 nib 파일을 연다
3. NSView의 서브클래스인 `MyView`를 만든다.
4. MyView 파일을 만들고, 프로젝트에 추가한다.
5. <그림 A-11>에서 보이는 바와 같이, 응용 프로그램 윈도우의 비트를 더 작게 만들고, More Views 팔레트에서 윈도우로 커스텀을 드래그한다.
6. CustomView를 선택하고, Info 윈도우를 불러와, 뷰 클래스를 MyView로 변경한다.




<그림 A-11> 커스텀 뷰를 인터페이스에 추가하기

7. MyView.h를 열어, 다음 인스턴스 변수와 메소드 선언을 추가한다.

   이들 인스턴스 변수 및 세터 메소드는 뷰가 현재 스트링 및 현재 폰트를 지속적으로 유지할 수 있도록 한다. 스트링이나 폰트를 변경하면 뷰는 새로운 객체를 사용해 다시 드로잉 작업을 한다.

```
@interface MyView: NSView
{
    NSString *string;
    NSFont *font;
}

- (void)setString:(NSString *)value;
- (void)setFont:(NSFont *)value;
- (BOOL)isFlipped;

@end
```

8. MyView.m을 열어, isFlipped 메소드를 추가한다. 이 메소드는 좌표계의 시작점을 뷰의 좌측 상단 코너로 신속하게 이동시킨다.

```
- (BOOL)isFlipped
{
    return YES;
}
```

9. **initWithFrame:** 메소드를 오버라이드하여 뷰의 기본적인 스트링과 폰트를 설정한다.

```
- (id)initWithFrame:(NSRect)frame
{
    [super initWithFrame:frame];
    [self setString: @"Hello World"];
    [self setFont: [NSFont systemFontOfSize: 12]];
    return self;
}
```

10. **setString:** 메소드를 구현한다.

```
- (void)setString:(NSString *)value
{
    [string autorelease];
    string = [value copy];
    [self setNeedsDisplay: YES];
}
```

11. **setFont:** 메소드를 구현한다.

```
- (void)setFont:(NSFont *)value
{
    [font autorelease];
    font = [value retain];
    [self setNeedsDisplay: YES];
}
```

12. **drawRect:** 메소드를 구현한다. **drawAtPoint:** 메소드는 뷰 상단에서 5픽셀 아래로 뷰 폭의 4분의 1지점에서 시작하는 스트링을 제공한다.

```
- (void)drawRect:(NSRect)rect
{
    NSRect myBounds = [self bounds];
    NSMutableDictionary *attrs = [NSMutableDictionary dictionary];

    [attrs setObject: font forKey: NSFontAttributeName];
    [string drawAtPoint:
         NSMakePoint((myBounds size.width/4.0), 5) withAttributes: attrs];
}
```

이 예제는 **rect** 대신에 **[self bounds]**를 사용하여 어디로 드로잉할 것인지 결정한다. **drawRect:** 메소드로 전환된 **rect** 파라미터는 업데이트에 필요한 뷰의 서브영역이며, 스크롤 뷰에서 뷰를 천천히 스크롤하면 변경될 수 있다. 뷰를 윈도우에 배치하면, 이들 차이점을 알아차릴 수 없다. 그러나, 코드를 재사용하는 데 있어 이 차이점은 중요한 역할을 한다. **[self bounds]**를 사용하지 않을 경우, 뷰를 스크롤 뷰에 배치하거나 **setNeedsDisplayInRect:**를 사용해 드로잉을 최적화하면, 큰 장애가 발생한다.

# NSBezierPath

NSBezierPath 객체는 PostScript-스타일 커맨드를 사용하여 패스를 만든다. 패스는 직선 및 곡선을 결합하여 구성된다. 그리고, 사각형, 타원, 원, 기호 등 식별가능한 모양을 구성할 수 있다. 또한, 직선이나 곡선을 사용하여 복합적인 다각형을 구성할 수 있다. 한 개 패스에서 양끝점을 연결할 수도 있고, 연결하지 않을 수도 있다.

NSBezierPath 객체는 연결여부에 관계없이 여러 분리된 패스를 제공한다. 각각의 패스는 NSBezierPath 객체의 "서브패스"라 할 수 있다. NSBezierPath 객체의 서브패스는 그룹으로 취급해야 한다. 서브패스를 개별적으로 취급하려면 NSBezierPath 객체를 각각 분리하면 된다.

제공된 NSBezierPath 객체의 경우, 패스의 아웃라인을 스트로크하거나 패스가 점유한 영역을 객체로 채울 수 있다. 또한, 뷰나 다른 영역의 클리핑 영역으로 패스를 사용할 수 있다. NSBezierPath의 메소드를 사용하여 스트로크되었거나 객체로 채워진 패스에서 히트 디렉션을 수행할 수 있다. 히트 디렉션은 러버 밴딩과 드래깅 기능에서처럼, 인터렉티브 그래픽 구현에 사용된다.

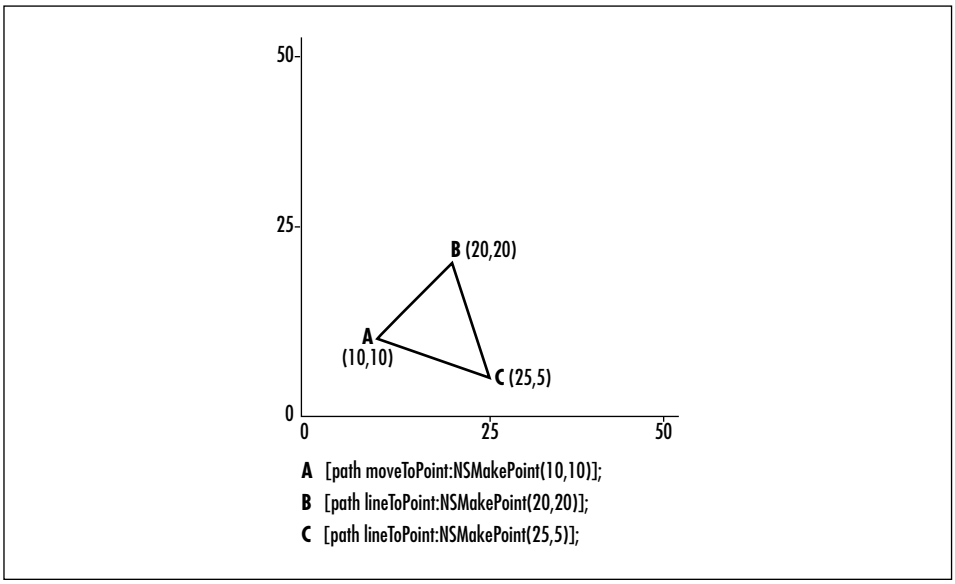
## 패스 구축하기

`bezierPath` 또는 `bezierPathWithRect:`를 사용하여 NSBezierPath 인스턴스를 생성할 수 있다. `bezierPath` 메소드는 새로운 Bezier-패스 객체를 비어있는 패스에 초기화한다. 반면에, `bezierPathWithRect:` 메소드는 특정한 사각형으로 패스를 생성한다.(NSBezierPath의 새로운 인스턴스에 메모리를 할당하거나 기본적인 초기화기인 `init` 메소드를 사용하여 비어있는 패스로 초기화할 수 있다)

패스 정보를 NSBezierPath 객체에 추가하려면, `moveToPoint:`, `lineToPoint:` 및 `curveToPoint:controlPoint1:controlPoint2:` 같은 패스 구축 메소드의 시퀀스를 생성해야 한다. 예를 들면, 다각형 패스를 생성하려면, `lineToPoint:` 메시지에 이어 `moveToPoint:` 메시지를 NSBezierPath 객체에 전송한다. 패스에 포인트를 추가하여, Closed 패스를 구성하고자 할 경우, `closePath` 메시지를 전송하여 끝점에서 시작점을 연결한다.(<그림 A-12> 참조)

패스 구축 메소드가 발생하는 순서는 상당히 중요하다. 라인을 연속적으로 정의할 경우에만 연결된다. 패스는 한 개 이상의 분리된 서브패스로 구성된다. 현재의 포인트는 가장 최근에 추가된 포인트이다.

대부분의 구축 메소드는 현재의 포인트를 다음 세그먼트의 시작점으로 사용한다. 새로운 서브패스를 생성하고자 할 경우, 먼저 **moveToPoint:**를 발생시켜야 한다. 예를 들면, **lineToPoint:**은 현재 포인트에서 특정 포인트로 라인 세그먼트를 추가한다. 일부 메소드는 **moveToPoint:**를 발생시킨다. 따라서, 새로운 서브패스가 자동으로 생성된다. 자세한 내용은 메소드에 관한 설명을 참조한다.




*<그림 A-12> 간단한 패스 구축하기*

이 메소드는 기존 패스에 새로운 패스를 추가하고, 동일한 모양을 제공한다. **appendBezierPathWithPoints:count:** 메소드가 특정 포인트에 인접한 패스를 생성한다 하더라도 새로운 패스는 리시버의 원래 패스에서 멀리 떨어져 생성된다. **appendBezierPath...** 메소드를 사용하여 패스를 NSBezierPath 객체에 추가한다.

```
NSRect aRect = NSMakeRect(0.0, 0.0, 50.0, 50.0);
aPath = [[NSBezierPath bezierPath] appendBezierPathWithOvalInRect:aRect];
```

## 패스 요소

구축 메소드를 사용한다 하더라도 모든 패스는 **moveToPoint:**, **lineToPoint:**, **curveToPoint:controlPoint1:controlPoint2:** 및 **closePath**에 해당하는 데이터 포인트의 시퀀스 및 공통 요소 유형으로 변경된다.

이들 요소 유형은 열거형 유형인 NSBezierPathElement에서 다음 상수로 명시될 수 있다.

- * NSMoveToBezierPathElement
- * NSLineToBezierPathElement
- * NSCurveToBezierPathElement
- * NSCloseBezierPathElement

NSCloseBezierPathElement를 제외한 모든 요소는 최소 한 개의 관련 데이터 포인트를 제공한다. 1개 이상의 데이터 포인트를 제공하는 유일한 요소는 NSCurveToBezierPathElement(곡선 모양을 정의하기 위해 컨트롤 포인트를 추가한다)이다. NSBezierPath는 elementAtIndex: 및 elementTypeAtIndex:associatedPoints:을 비롯하여 패스 요소 정보(및 관련 포인트)를 직접 얻기 위한 여러 메소드를 정의한다. 이들 메소드를 사용하여 패스를 삭제하고, 새로운 패스를 다시 구축한다.

## 패스 드로잉하기

일반적으로, NSView의 drawRect: 메소드 내에서 NSBezierPath 객체를 표현한다. drawRect: 메소드를 발생시키면, 포커스는 뷰에 고정되고, 모든 드로잉 기능도 그 뷰에 고정된다. 따라서, 뷰를 구축하면 뷰 포인트가 뷰 좌표계에 명시된다. 또한, 임의 좌표계를 사용하여 패스를 구축하고, NSAffineTransform 객체를 사용하여 패스를 뷰의 좌표계로 변환한다. NSAffineTransform 객체는 패스를 이동, 스케일링 및 회전할 수 있다.(온라인의 Cocoa 도큐멘트에서 NSAffineTransform 클래스 규격서를 참조한다)

NSBezierPath 객체를 채우거나 스트로크하기 전에 그래픽 속성을 설정하여 패스에 사용해야 한다. NSBezierPath의 set... 메소드를 사용하여 라인 캡 스타일, 라인 조인 스타일, 라인 폭, 미터 제한, 곡선 평면화, 망점 단계 등의 속성을 설정할 수 있다. 다른 속성은 해당 객체를 사용하여 설정해야 한다. 예를 들면, set를 NSColor 객체에 전송하여 현재 그래픽 컨텍스트에서 색상을 설정한다.

Stroke 또는 Fill 메소드를 사용하여 패스를 표현할 수 있다. Stroke 메소드는 현재 그래픽 컨텍스트에서 컬러, 라인 폭, 캡 및 조인 스타일, 곡선 평면화 드로잉 속성을 사용하여 리시버의 패스를 따라 라인을 드로잉한다. Fill 메소드는 패스가 둘러싸고 있는 영역을 페인트칠하여 패스를 표현하고, 색상 및 곡선 평면화 속성을 사용한다. 또한, 패스가 닫히지 않았다면, 닫기 기능(closePath을 생성)을 수행한다.(끝점이 시작점에 연결되어 있으면, 서브패스는 닫힌다 반대의 경우, 서브패스는 열린다)

NSBezierPath 객체를 생성하지 않고, 모양을 드로잉하기 위해 일부 클래스 메소드를 제공한다. 예를 들어, `fillRect:` 및 `strokeRect:` 클래스 메소드를 사용하여 사각형을 채우거나 사각형의 아웃라인을 드로잉한다. 그리고, `strokeLineFromPoint:toPoint:` 클래스 메소드를 사용하여, 라인 세그먼트를 드로잉한다.

# NSBezierPath로 드로잉하기

이 섹션에서는 "NSStrings 드로잉하기" 섹션에서 생성한 Simple Draw 응용 프로그램을 변경하여 NSBezierPath를 사용한 뷰에서 간단한 모양을 드로잉하는 방법을 설명한다.

## 라인과 사각형 드로잉하기

Simple Draw의 `drawRect:` 메소드를 확대하여 NSBezierPath를 검토한다.

- 뷰를 불투명한 흰색 바탕으로 채운다.
- 뷰를 밝은 회색 크로스헤어로 양분한다.
- 뷰 둘레에 옅은 검은색 보더를 드로잉한다.

다음과 같이 작업이 수행된다.

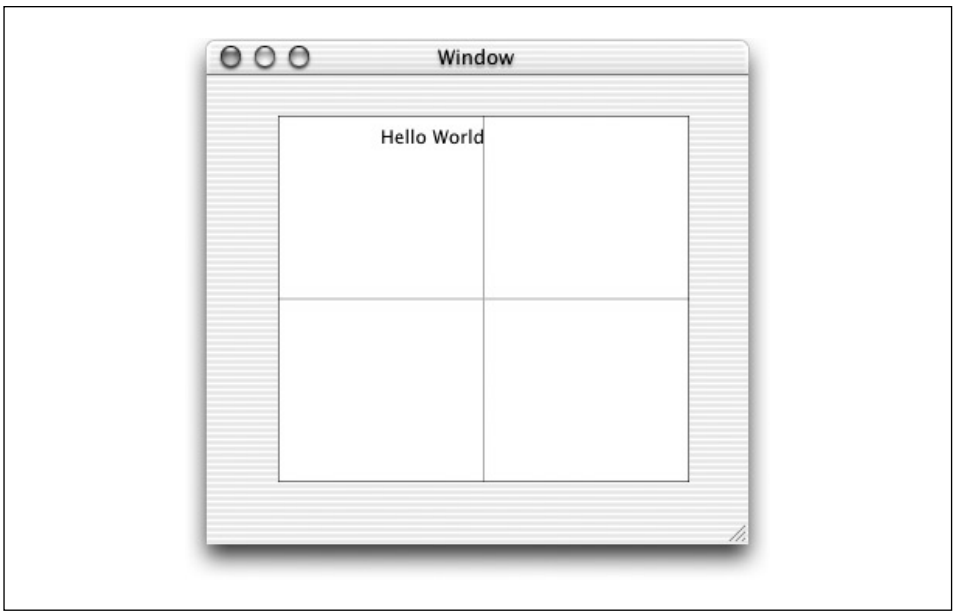
`MyView.m`을 열어, 다음과 같이 `drawRect:` 구현을 확대한다.

```
- (void)drawRect:(NSRect)rect
{
    NSRect myBounds = [self bounds];
    NSMutableDictionary *attrs = [NSMutableDictionary dictionary];

    // Paint a white background.
    [[NSColor whiteColor] set];
    [NSBezierPath fillRect:myBounds];

    // Draw some crosshairs on the view.
    [[NSColor lightGrayColor] set];

    [NSBezierPath strokeLineFromPoint:
            NSMakePoint(0,(myBounds.size.height/2.0))
            toPoint:NSMakePoint(myBounds.size.width,
(myBounds.size.height/2.0))];

    [NSBezierPath strokeLineFromPoint:
            NSMakePoint((myBounds.size.width/2.0), 0)
            toPoint:NSMakePoint((myBounds.size.width/2.0),
myBounds.size.height)];

    // Draw a black border around the view.
    [[NSColor blackColor] set];
    [NSBezierPath strokeRect:myBounds];
```

```
        // Render a string.
        [attrs setObject: font forKey: NSFontAttributeName];
        [string drawAtPoint:
                NSMakePoint((myBounds.size.width/4.0), 5) withAttributes: attrs];
    }
```

**drawRect:** 메소드의 새로운 버전에서는 흰색 컬러로 초기화된 NSColor 객체를 생성하고, **set** 메소드에 이를 전송할 수 있다. **set**은 그래픽 컨텍스트의 현재 색상을 변경한다. 그래서, 이어지는 드로잉은 특정 색상을 사용하여 표현된다. 따라서, **drawRect:**는 NSBezierPath 클래스 메소드인 **fillRect:**를 사용하여 MyView를 흰색으로 채운다. 그 다음, **drawRect:**는 현재 색상을 밝은 회색으로 바꾸고, 뷰에 2개의 라인을 드로잉하여 4등분한다. 크로스헤어를 드로잉한 뒤, 메소드는 NSBezierPath의 **strokeRect:**를 사용하여 뷰 주위에 검정색 보더를 드로잉한다. 마지막으로, **drawRect:**는 상기와 같이 스트링을 표현한다.

응용 프로그램을 구축하여 운용한다. <그림 A-13>과 동일한 그림이 나타나야 한다.




*<그림 A-13> 불투명 바탕과 이등분된 크로스헤어를 사용한 MyView*

## 복잡한 모양 드로잉하기

이 섹션에서는 십자 모양의 기호로 화살표와 원을 드로잉하여 클래스를 한층 더 확대하는 방법을 설명한다. 여기서 제공되는 예제에서는 흥미로운 기법이 사용된다. 이 예제를 통해 NSBezierPath를 사용하여 모델을 생성할 수 있으며, 변환 행렬을 사용하여 뷰에 모델을 표현할 수 있다.

이 과정을(모델이 도장 모양인) 도장툴과 유사하다고 생각할 수 있다. 그러나, 도장툴은 변환행렬을 사용하여 이동, 확대 및 회전이 가능하며, 종이는 뷰의 역할을 한다.

다음과 같이 작업이 수행된다.

1. **MyView.h**를 열어, 화살표와 원의 인스턴스 변수 선언을 추가한다.

```
NSBezierPath *arrowPath;
NSBezierPath *circlePath;
```

2. **awakeFromNib** 및 **isOpaque** 선언을 추가한다.

```
- (void)awakeFromNib;
- (BOOL)isOpaque;
```

3. 모델에 대한 접근자 메소드 선언을 추가한다.

```
- (NSBezierPath *)arrowPath;
- (void)setArrowPath: (NSBezierPath *)newPath;

- (NSBezierPath *)circlePath;
- (void)setCirclePath: (NSBezierPath *)newPath;
```

4. 모델을 만들고 드로잉하는 메소드 선언을 추가한다.

```
- (NSBezierPath *)createArrowPath;
- (NSBezierPath *)createCirclePath;

- (void)drawArrow;
- (void)drawCircle;
```

5. **MyView.m**을 열어, **isOpaque**와 **awakeFromNib** 구현을 추가한다. **awakeFromNib** 메소드는 메소드들을 발생시켜, 클래스의 화살표와 원 모델을 만든다. **isOpaque**는 드로잉 서브시스템에 대한 성능을 암시한다.

```
- (void)awakeFromNib
{
    [self setArrowPath: [self createArrowPath]];
    [self setCirclePath: [self createCirclePath]];
}

- (BOOL)isOpaque
{
    return YES;
}
```

6. 드로잉 메소드를 호출하여 화살표와 원을 표현할 수 있도록 **drawRect:**을 변경한다. 이 드로잉 메소드를 구현의 끝부분에 추가한다.

```
/* ... */
    [self drawArrow];
    [self drawCircle];
```

7. 모델의 접근자 메소드를 구현한다.

```
- (NSBezierPath *)arrowPath
{
    return arrowPath;
}

- (void)setArrowPath:(NSBezierPath *)newPath
{
    [newPath retain];
    [arrowPath release];
    arrowPath = newPath;
}

- (NSBezierPath *)circlePath
{
    return circlePath;
}

- (void)setCirclePath:(NSBezierPath *)newPath
{
    [newPath retain];
    [circlePath release];
    circlePath = newPath;
}
```

8. 이제, 흥미로운 작업을 시작해보기로 하겠다.

   <예제 A-1>에서 보이는 바와 같이 **createArrowPath** 구현을 추가한다. 이 작업을 통해 화살표의 NSBezierPath 모델이 생성된다. 이 예제의 주석은 과정을 설명한다.

*<예제 A-1> createArrowPath 메소드 구현*

```
- (NSBezierPath *)createArrowPath
{
    // An NSPoint that is reused to create the arrow model.
    NSPoint point;

    // The ratio of arrowhead to arrow length
    float headLengthRatio = 0.30;

    // The ratio between the base midpoint and the actual point.
    // at which the arrowhead baselines meet.
    float headBaseRatio = 0.70;

    // The half-angle width of the arrowhead in radians.
    float arrowHeadWidth = 0.3;

    // The Bezier path used to draw the arrow.
    NSBezierPath *model = [NSBezierPath bezierPath];

    // Start the drawing process by moving to the origin at (0,0).
    point.x = 0;
```

<예제 A-1> createArrowPath 메소드 구현(계속)

```
    point.y = 0;
    [model moveToPoint:point];

    // Draw a line to the middle point of the base of the arrowhead.
    point.x = 1 - (headLengthRatio*headBaseRatio);
    [model lineToPoint:point];

    // Draw line to the upper base point of the arrowhead.
    point.x = 1 - headLengthRatio;
    point.y = headLengthRatio* tan(arrowHeadWidth);
    [model lineToPoint:point];

    // Draw line to the tip of the arrowhead.
    point.x = 1;
    point.y = 0;
    [model lineToPoint:point];

    // Draw line to the lower base point of the arrowhead.
    point.x = 1 - headLengthRatio;
    point.y = -(headLengthRatio* tan(arrowHeadWidth));
    [model lineToPoint:point];

    // Draw line to the middle point of the base of the arrowhead.
    point.x = 1 - (headLengthRatio*headBaseRatio);
    point.y = 0;
    [model lineToPoint:point];

    // Finish drawing.
    [model closePath];

    return model;
}
```

9. 이제, <예제 A-2>에서 보이는 바와 같이 원 모양을 만들기 위한 메소드를 구현한다. 진행 과정은 화살표 모델을 생성하는 상기 단계와 유사하다.

<예제 A-2> createCirclePath 메소드 구현

```
- (NSBezierPath *)createCirclePath
{
     // An NSPoint that is reused to create the arrow model.
     NSPoint point;

    NSBezierPath *model = [NSBezierPath bezierPath];

    // Create simple circle with its center at (0,0).
    point.x = 0;
    point.y = 0;
    [model appendBezierPathWithArcWithCenter:point
            radius:1.0 startAngle:0 endAngle:360];
```

*<예제 A-2> createCirclePath 메소드 구현(계속)*

```
    // Draw a plus sign.
    // Start with a horizontal line.
    point.x = .5;
    point.y = 0;
    [model moveToPoint:point];

    point.x = -.5;
    [model lineToPoint:point];

    // Draw the vertical line.
    point.x = 0;
    point.y = .5;
    [model moveToPoint:point];

    point.y = -.5;
    [model lineToPoint:point];

    return model;
}
```

10. 이제, <예제 A-3>에서 보이는 바와 같이 화살표 모델을 뷰에 드로잉할 수 있는 메소드 구현을 추가한다. 화살표를 뷰에 표현하기 전에 변환 행렬을 사용하여 화살표를 변경한다.

*<예제 A-3> drawArrow 메소드 구현*

```
- (void)drawArrow
{
    // A reference to the model that will be drawn.
    NSBezierPath *arrowToDraw;

    // Transformation matrices to modify the arrow model.

    NSAffineTransform *scaleMatrix;
    NSAffineTransform *rotationMatrix;
    NSAffineTransform *translationMatrix;

    // This matrix will hold the combined transformations above.
    NSAffineTransform *arrowTransformMatrix;

    // Point onscreen (offset from NSView origin) to start drawing.
    NSPoint screenLocation = NSMakePoint(100, 100);

    // The colors to use for the arrow's outline and fill.
    NSColor *arrowStrokeColor = [NSColor blueColor];
    NSColor *arrowFillColor = [NSColor greenColor];

    // The size of the arrow to draw relative to the model. Change this
    // to draw bigger or smaller arrows.
    float scaleFactor = 100.0;
```

<예제 A-3> drawArrow 메소드 구현(계속)

```
    // Create and configure the scaling matrix.
    scaleMatrix = [NSAffineTransform transform];
    [scaleMatrix scaleBy: scaleFactor];

    // Create and configure the rotation matrix.
    rotationMatrix = [NSAffineTransform transform];
    [rotationMatrix rotateByDegrees:45];

    // Create and configure the translation matrix.
    translationMatrix = [NSAffineTransform transform];
    [translationMatrix translateXBy: screenLocation.x
            yBy: screenLocation.y];

    // Combine scaling, rotation, and translation into
    // one matrix operation.
    arrowTransformMatrix = [NSAffineTransform transform];
    [arrowTransformMatrix appendTransform:scaleMatrix];
    [arrowTransformMatrix appendTransform:rotationMatrix];
    [arrowTransformMatrix appendTransform:translationMatrix];

    // Use the combined transformations to modify the model for drawing.
    arrowToDraw = [arrowTransformMatrix transformBezierPath: [self arrowPath]];

    // Draw the arrow outline.
    [arrowStrokeColor set];
    [arrowToDraw setLineWidth:2.0];
    [arrowToDraw stroke];

    // Fill in the arrow.
    [arrowFillColor set];
    [arrowToDraw fill];
}
```

10. 마지막으로, <예제 A-4>에서 보이는 바와 같이 **drawCircle** 메소드 구현을 추가한다.

<예제 A-4> drawCircle 메소드 구현

```
- (void)drawCircle
{
    // The local copy of the circle that will be drawn.
    NSBezierPath *circleToDraw;

    // Transformation matricies to use.
    NSAffineTransform *translationMatrix;
    NSAffineTransform *scaleMatrix;

    // Draw circle at this location.
    NSPoint screenLocation = NSMakePoint(50, 50);
```

<예제 A-4> drawCircle 메소드 구현(계속)

```
    // Colors for the circle outline and fill.
    NSColor *circleStrokeColor = [NSColor blackColor];
    NSColor *circleFillColor = [NSColor redColor];

    // The size of the circle.
    float circleRadius = 10.0;

    // Create and configure the translation matrix.
    translationMatrix = [NSAffineTransform transform];
    [translationMatrix translateXBy: screenLocation.x
            yBy: screenLocation.y];

    // Create and configure the scaling matrix.
    scaleMatrix = [NSAffineTransform transform];
    [scaleMatrix scaleBy: circleRadius];

    // Combine the scaling and translation operations.
    [scaleMatrix appendTransform:translationMatrix];

    // Transform the circle path for drawing.
    circleToDraw = [scaleMatrix transformBezierPath:[self circlePath]];

    // Fill the circle.
    [circleFillColor set];
    [circleToDraw fill];

    // Stroke (outline) the circle.
    [circleStrokeColor set];
    [circleToDraw setLineWidth:2.0];
    [circleToDraw stroke];
}
```

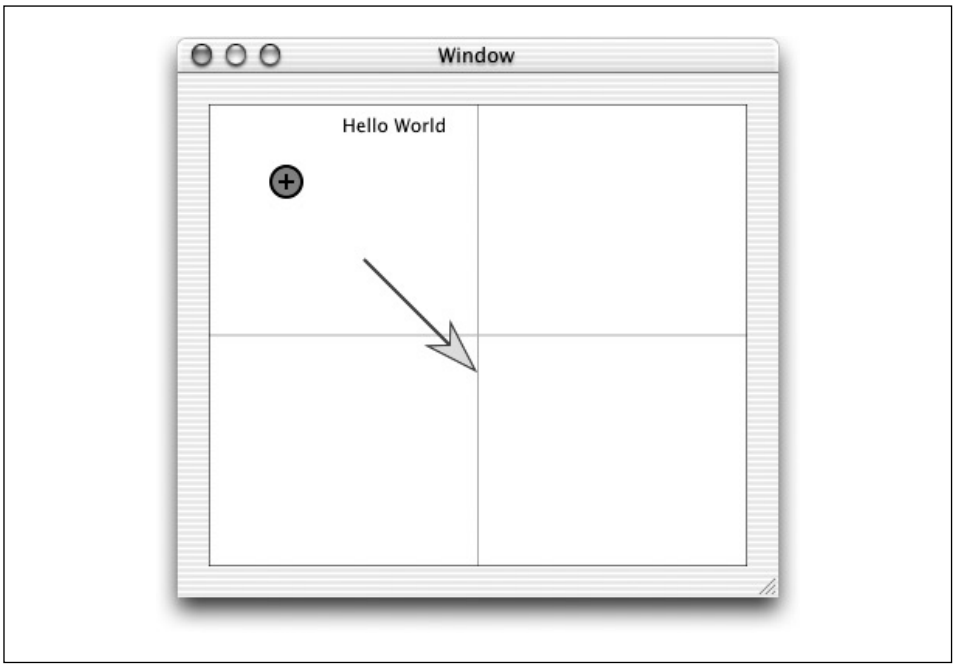
이제, 프로그램을 구축하여 시험한다. <그림 A-14>와 동일한 그림이 나타나야 한다.

# *Quartz로 드로잉하기*

일반적으로, 응용 프로그램에서 드로잉을 하려면 Cocoa의 클래스와 기능을 사용할 수 있어야 한다. Cocoa가 제공하지 않는 기능을 필요로 할 경우, Core Graphics API를 사용할 수 있다. 대부분의 경우, 실제 화면 드로잉의 오버헤드와 비교해보면, Cocoa의 드로잉 기능과 객체는 오버헤드가 거의 없다고 볼 수 있다. 성능 측정의 결과로써 오버헤드가 필요하면, Core Graphics API를 사용한다.

Cocoa의 기능과 객체는 Core Graphics을 사용하는 것 보다 더 원활하게 Cocoa의 화면 장치와 상호작용한다. 예를 들어, Core Graphics을 사용하여 현재 변환 행렬을 변경하면, 잘못 표현된 컨텍스트에서(예를 들면, 텍스트 서브시스템에 의해 드로잉된 회전된 텍스트) Cocoa 객체가 생성될 수 있다.

NSView를 사용해 현재 변환 행렬을 변경하면, 예상대로 작업이 수행된다.




*<그림 A-14> NSBezierPath을 사용하여 드로잉한 모양으로 Simple Draw 출력*

이 섹션에서는 NSView에서 Core Graphics을 사용하기 위해 상기 섹션에서 설명한 Simple Draw 응용 프로그램을 변경하는 방법을 설명한다. 이 프로그램의 새로운 버전은 MyView의 상단 우측 코너에서 멀티컬러로 나선 패턴을 드로잉하는 방법을 제공한다.

1. Project Builder에서 Simple Draw 프로젝트를 연다.
2. Project 메뉴에서 Add Frameworks 커맨드를 선택하고, Core Services 프레임워크를 Simple Draw 프로젝트에 추가한다. Core Services 프레임워크는 Core Graphics 프레임워크를 포함한 엄브렐러(Umbrella) 프레임워크이다. Core Graphics 프레임워크에 링크하여 드로잉을 사용해야 한다.
3. `MyView.m`을 열고 Application Services 헤더를 임포트한다.

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

4. 다음 지역 변수를 **drawRect:** 메소드 상단에 추가한다. Core Graphics 드로잉에 필요한 Core Graphics 컨텍스트를 참조하기 위해 이들 변수를 루프 인덱스와 앵글로써 각각 사용한다.

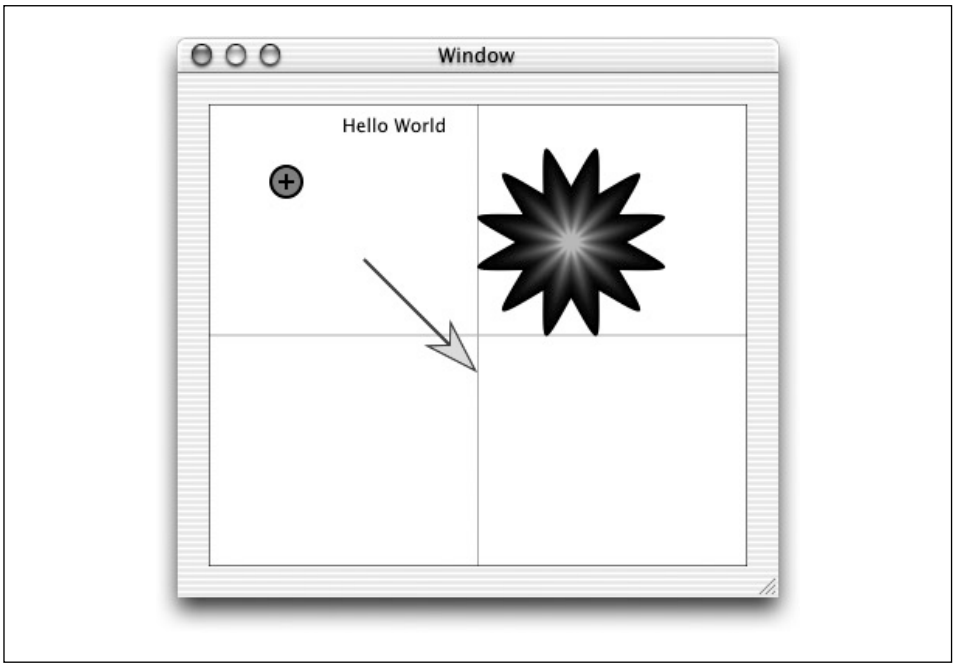
```
- (void)drawRect:(NSRect)rect {
int i, j;
    float a0, a1;
    CGContextRef context;
    NSRect myBounds = [self bounds];
    /*...*/
```

5. <예제 A-5>에서와 같이 **drawRect:** 하단에 코드를 추가한다.

*<예제 A-5> Quartz Primitives를 사용한 drawRect:구현*

```
    /*...*/
    // Get the Core Graphics context from MyView's window.
    context = [[NSGraphicsContext
            graphicsContextWithWindow:[self window]] graphicsPort];

    // Translate to the upper right corner of the view.
    // Remember CG doesn't see the isFlipped setting so it's
    // still using a coordinate system with the origin in the lower left.
    CGContextTranslateCTM(context,
        NSMidX(myBounds) + 80.0, NSMidY(myBounds) + 80.0);

    // Scale the transformation matrix to shrink the size of the drawing.
    CGContextScaleCTM(context, 40.0, 40.0);

    // Draw a sequence of Bezier curves.
    for (i = 0; i < 20; i++) {
        CGContextSetRGBFillColor(context, i/19.0, i/30.0, 0.0, 1.0);
        CGContextScaleCTM(context, 0.9, 0.9);
        CGContextMoveToPoint(context, 1.0, 0.0);
        for (j = 0; j < 12; j++) {
            a0 = 2 * M_PI * (j + 0.5) / 12;
            a1 = 2 * M_PI * (j + 1.0) / 12;
            CGContextAddCurveToPoint(context, 2*cos(a0), 2*sin(a0),
                 2*cos(a0), 2*sin(a0), cos(a1), sin(a1));
        }

        CGContextClosePath(context);
        CGContextFillPath(context);
    }
}
```

다시 한번, 프로그램을 구축하여 시험한다. <그림 A-15>와 동일한 그림이 나타나야 한다.




<그림 A-15> Core Graphics 호출을 추가한 후 Simple Draw 출력

Core Graphics의 헤더 파일을 몇 분 동안 검토하여, API에서 사용 가능한 기능과 친숙해진다. Project Builder의 Groups and Files 리스트에서 Application Services 프레임워크내 Core Graphics 프레임워크의 헤더를 검색한다. 일부 흥미로운 헤더를 찾아볼 수 있다.

- `CGGeometry.h`. 이 헤더는 CGPoint, CGSize 및 CGRect와 같은 기본적인 데이터 유형과 이들을 조작하기 위한 기능을 정의한다.
- `CGContext.h` 이 헤더는 드로잉 오퍼레이터를 포함한다.
- `CGAffineTransform.h`. 이 헤더는 변환 행렬을 조작하는 기능을 제공한다.

이들 예제를 변경하는 데 시간을 좀 더 투자하면 Cocoa에서 드로잉하는 방법을 훨씬 더 자세하게 배울 수 있다. 좀더 시험을 해보려면, `NSView.h`, `NSBezierPath.h`, `NSGeometry.h`, `NSAffineTransform.h`, `NSFont.h`, `NSColor.h`, `NSGraphics.h`, `NSImage.h` 등 Application Kit 헤더를 참조한다.

O'REILLY®

# Learning Cocoa

*Learning Cocoa*는 단순히 눈으로만 읽어보는데 그치지 않고, 직접 프로그램을 만들어봄으로써 Mac OS X의 주요 응용 프로그램을 손쉽게 개발할 수 있도록 합니다. 이 책의 초반부에서는 Interface Builder(사용자 인터페이스를 디자인하기 위한 Apple의 그래픽 편집기)와 Project Builder(Apple의 통합 개발 환경)를 다루는 한편 Cocoa 프로그래밍의 기본 요소에 친숙해질 수 있도록 간단한 튜토리얼을 제시합니다.

이 책은 여러분이 Objective-C를 사용하여 점점 더 복잡해지는 예제 응용 프로그램을 개발할 수 있도록 안내합니다. 하나의 튜토리얼에서 배운 기술과 개념은 그 다음에 나오는 튜토리얼의 더욱 향상된 기술과 개념의 초석이 됩니다.

이 책에서는 다음과 같은 내용이 다뤄집니다.

- Cocoa의 주요 패러다임
- 이벤트 처리
- 데이터 기능성
- Cocoa의 다중 도큐멘트 아키텍쳐
- Cocoa 및 Core Graphics (Quartz) APIs.

이 책에서 제시한 예제 프로그램을 완성시키기 위해 프로그래밍에 대한 풍부한 경험이 필요한 것은 아닙니다. 하지만, C 프로그래밍 언어에 대한 지식은 도움이 됩니다. 여러분이 Java나 Smalltalk과 같은 객체 지향 프로그래밍 언어와 친숙하다면, 이 책의 3장 *Objective-C 프리머*에서 설명하는 Objective-C를 금방 익힐 수 있습니다.

이 책은 Mac OS X을 개발한 Apple 컴퓨터사에서 작성하였으며, 여러분께 어디서도 얻을 수 없는 귀중한 정보를 제공해드립니다.

ADC

Apple Developer Connection

**Recommended Title**

Apple Computer사는 세계에서 가장 훌륭하고 안정적인 운영 체제인 Mac OS X을 개발하기 위해 오픈 소스 기술과 Apple만의 독자적인 프로그래밍 기술을 결합하였습니다. 이와 같은 정신으로 Apple은 중요한 기술 서적 출판을 위해 O'Reilly & Associates와 제휴하였습니다. ADC 로고는 본 서적이 Apple 엔지니어들이 기술적으로 검증하고, Apple Developer Connection이 추천한 서적임을 나타냅니다.

http://www.applecomputer.co.kr/developer

***O'Reilly사의 웹 www.oreilly.com를 방문하십시오.***