

Xcode and Objective-C

Xcode 와 Objective-C 의 기초

Contents

Articles

Xcode4 CurrencyConverter	1
ObjectiveC Linux	45
ObjectiveC Symbole etc	69

References

Article Sources and Contributors	71
Image Sources, Licenses and Contributors	72

Article Licenses

License	73
---------	----

Xcode4 CurrencyConverter

Currency Converter for Xcode 4 Version

개요

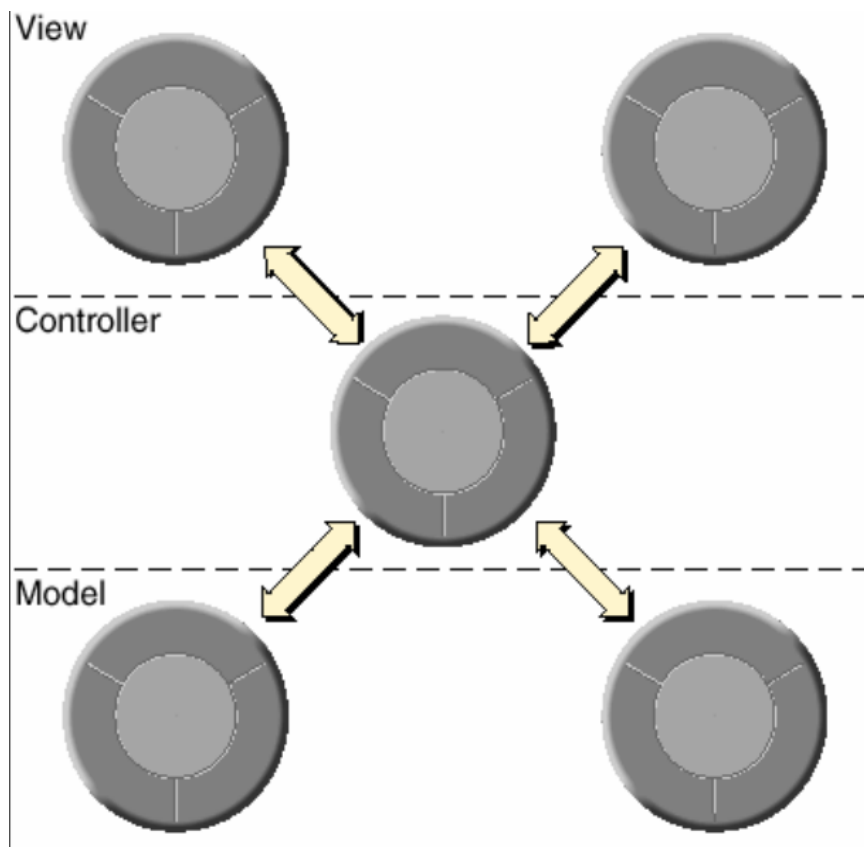
이 강좌는 Xcode 이전의 project builder 시절에 만들어진 apple 의문서를기준하여최신의내용을기존의내용을다시한번만들어나가는데그목적이있습니다. 모자란설명은기존의문서를참고해주시기비랍니다.

이강좌가다른강좌에비해결과물이간단함에도불구하고한번쯤살펴보아야하는이유는다음과같다

1. xib 를이용한 windows UI를다룬다.
2. windows UI 의 **files owner** 를 통한 instance object 의 연결과 **instance object** 와실제코드상의 class 연결을 통한 UI to Objective-C code 의 delegate 를볼수있다.
3. MVC의간단한예제를볼수있다.

MVC 가대체뭔가요?

일반적으로설명되는 MVC 의구조는다음과같습니다.



Controller 외에는서로연결되는부분이전혀없죠. 이론만으로도보자면야객체지향의완벽에가까운보기라할수있을겁니다.

여기원문에서제시되는설명을인용해보겠습니다.(문맥이어색한부분은약간씩수정한부분이있습니다)

▶ Model-View-Controller 는 Smalltalk-80 에서유래하였습니다. MVC 는같은응용프로그램내에있는객체들을 Model, View, Controller 라는추상적인경계로나누어서소통하는구조로되어있습니다.(from Apple)

Model Object(from Apple)

이 객체는 전문적인 지식 또는 기술을 의미합니다. Model 객체는 데이터를 포함하고 있으며, 그 데이터를 조작하는 로직을 정의하고 있습니다. 예를 들자면 '고객'이라는 객체는 비즈니스용 프로그램에서 자주 사용되는 Model 객체입니다. Model 객체는 고객에 관련된 대표적인 정보를 나타내는 데이터를 가지고 있으며, 그 정보로부터 새로운 데이터를 계산하고 액세스해주는 알고리즘에 접근합니다. 더욱 전문화된 Model Class 의 예로는 기상 관측 시스템의 'Front' 라는 Class 를 들 수 있는데, 이 Class 의 객체는 날씨 전말을 나타내는 데이터와 정보를 가지고 있습니다.

Model 객체는 직접적으로 보여지지 않습니다. 또한 이들은 다양한 플랫폼(환경이라는 의미인 거 같습니다)에서 이동, 지속, 분배, 재사용될 수 있습니다.

View Object(from Apple)

View 객체의 Paradigm 은 사용자 인터페이스에서 볼 수 있는 것들(Windows, Button, etc)을 말합니다. View 객체는 표시되는 데이터에 대해서는 알지 못합니다. Application Program Kit 는 Window, TextField, ScrollView, Browser 등 필요한 모든 View 객체를 제공합니다. 또한 새로운 방법(Graph 등)으로 자신만의 View 객체를 만들어 데이터를 표시할 수 있으며 프로그램에 적합한 새로운 방법으로 Window 내에서 View 객체들을 그룹화할 수도 있습니다. 특히 Kit(Application Kit) 내의 View 객체는 재사용성이 매우 높기 때문에 프로그램 간에 일관성을 유지할 수 있습니다.

Controller Object(from Apple)

Controller 객체는 Application Program 내에서 Model 객체와 View 객체 사이의 매개체로 작용합니다. 일반적으로 하나의 프로그램 또는 하나의 Window 에는 1개의 Controller 객체가 있습니다. Controller 객체는 Model 객체와 View 객체 사이에서 데이터를 전달하는 역할을 합니다. 또한 nib(Xcode 4 에서는 xib 입니다) 파일 로드, Window와 프로그램의 대리자(Delegate) 역할 등 프로그램 특유의 잡다한 일들을 진행합니다. 프로그램에 대해 Controller 객체가 하는 일은 매우 구체적이기 때문에 프로그램 코드의 많은 부분을 차지하면서도 일반적으로 재사용은 불가능합니다.(Controller 객체가 모두 재사용될 수 없다는 것은 아닙니다. 설계를 잘 한다면 재사용이 가능하겠죠)

Controller 객체의 핵심적인 역할인 Model 과 View 사이의 중개자 역할 때문에 Model 객체는 Interface(UI)의 상태와 Event 등에 대해 알 필요가 없으며, View 객체는 Model 객체에 대한 Program Interface를 알아야 할 필요가 없습니다. 직접 작성한 View 객체와 Model 객체는 Interface Builder 에 있는 Palette 에서 다른 사용이 사용할 수 있게 할 수 있습니다.

이 강좌에서 사용되는 MVC 에 대하여

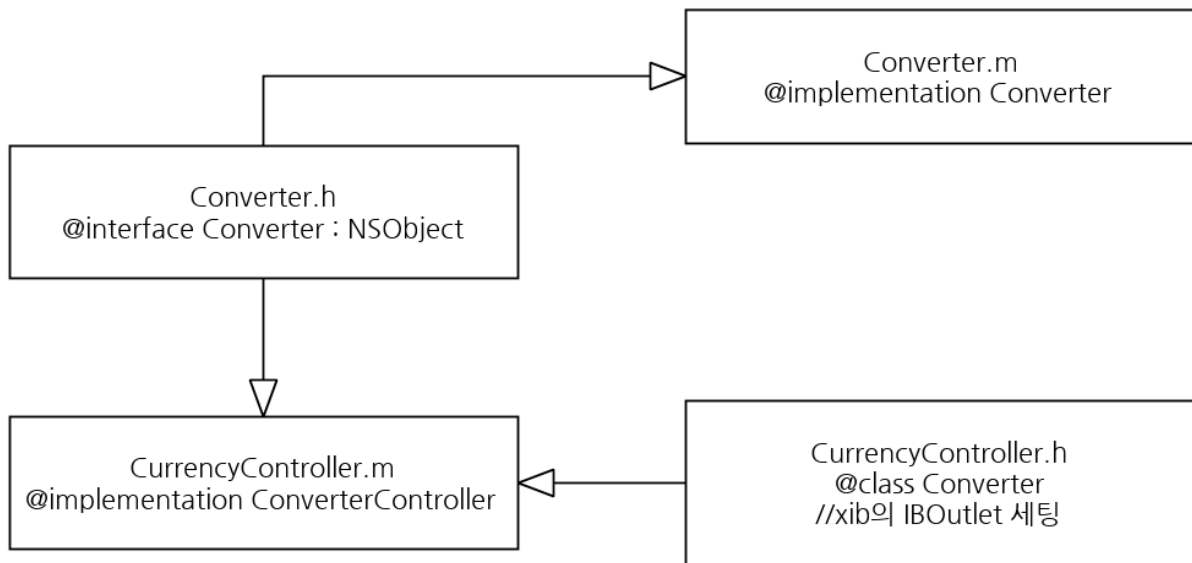
이 강좌에서 제작되는 프로그램은 이런 구조를 조금 더 Extream 하게 가져가는 걸 목적으로 함으로서 원래의 구조와는 약간의 변경이 생기게 되니 관련되는 부분은 참고 부탁드립니다.

Application 의 기본 전제

원래 문서의 Application 에서 설명하는 것들을 몇 가지 중요한 필요조건을 나열해 보도록 하겠습니다.

- MVC(Model-View-Controller)의 구조로 작업한다
- instance object를 사용한다.

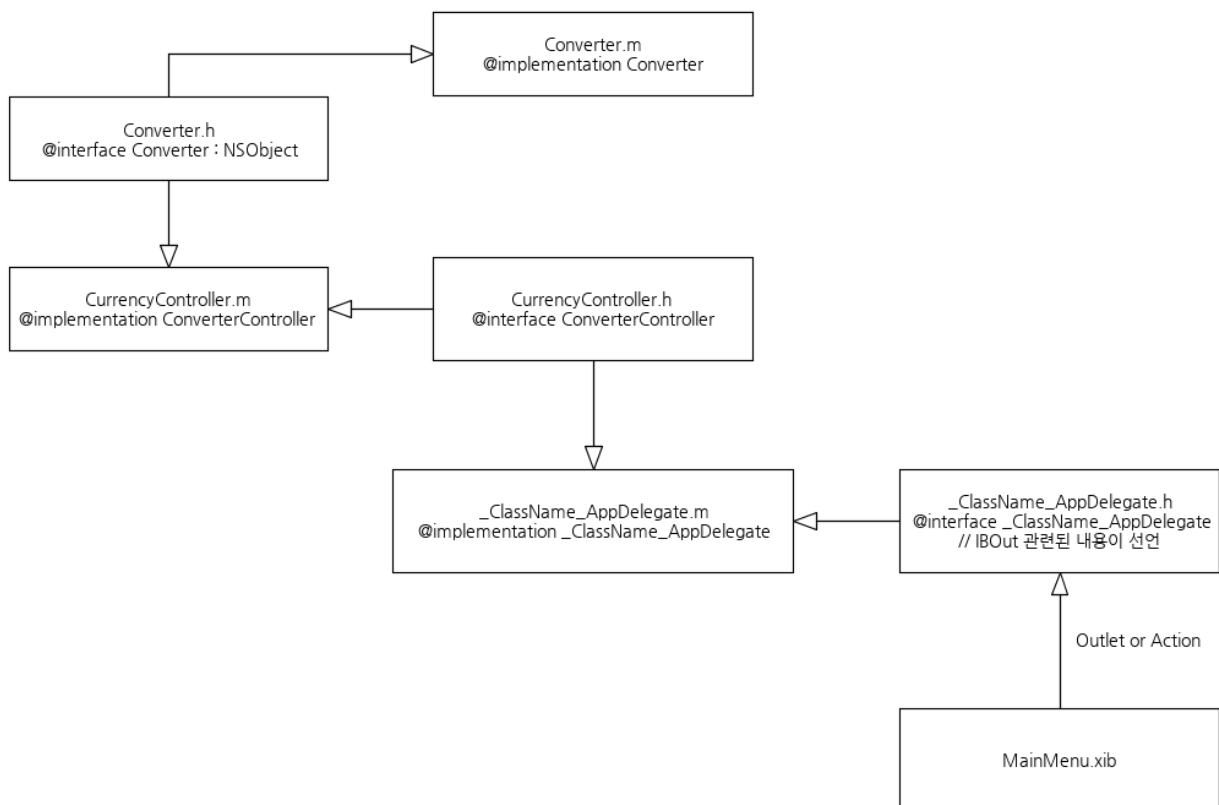
원래 문서에 있는 내용대로 Class Diagram 을 간략하게 표현해보면 다음과 같습니다.



이 어플리케이션은아래와같은전제와객체구조를가지게됩니다.

- Class 명은 **Conv** 로 정의한다
- Converter :: 통화량을 계산해서 값을 반환한다. Model 에 해당됨
- ConverterController :: Converter 객체와 AppDelegate 사이의 활동을 조정하는 클래스. Controller 에 해당됨
- ConvAppDelegate :: xib 파일에서 정의된 UI 객체들에 대한 동작을 정의하는 클래스. View 에 해당됨.
- xib 파일 :: AppDelegate 에 IBOutlet, 및 Action 으로 연결되는 사용자 UI

이러한 내용을 기초로 하면 다음과 같은 관계를 생각해 볼 수 있습니다.



물론굳이이러한구조를가져가야할필요도없고더단순화시킬수있는부분도많겠습
니다만, 여기서는 MVC에대한예제를같이보여주는것을목표로하고있기때문에필요
한것들은다작업하며진행하도록하겠습니다.

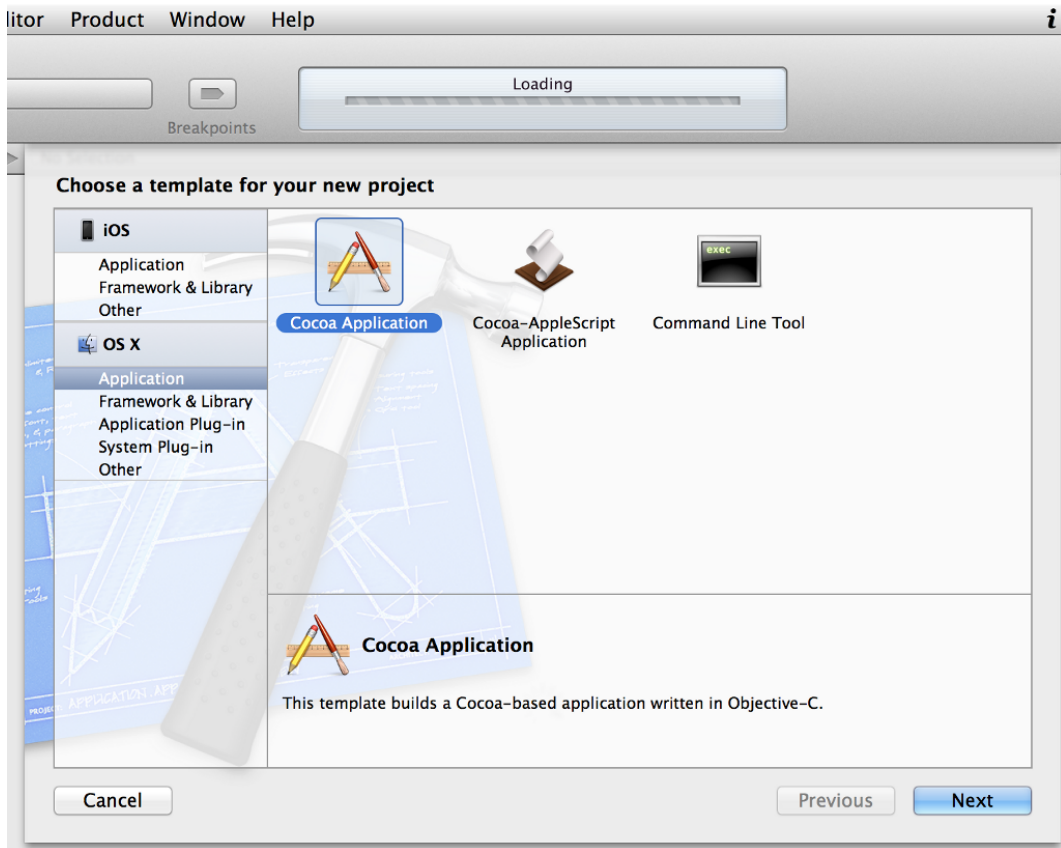
강좌의내용은다음과같은순서로진행됩니다

1. Project의생성
2. 사용자 UI 제작
3. Delegate 파일에 UI 의 element 를연결
4. Converter 클래스구현
5. ConverterController 클래스구현
6. 각클래스간의연결
7. 개발한프로그램에대한디버그및실행

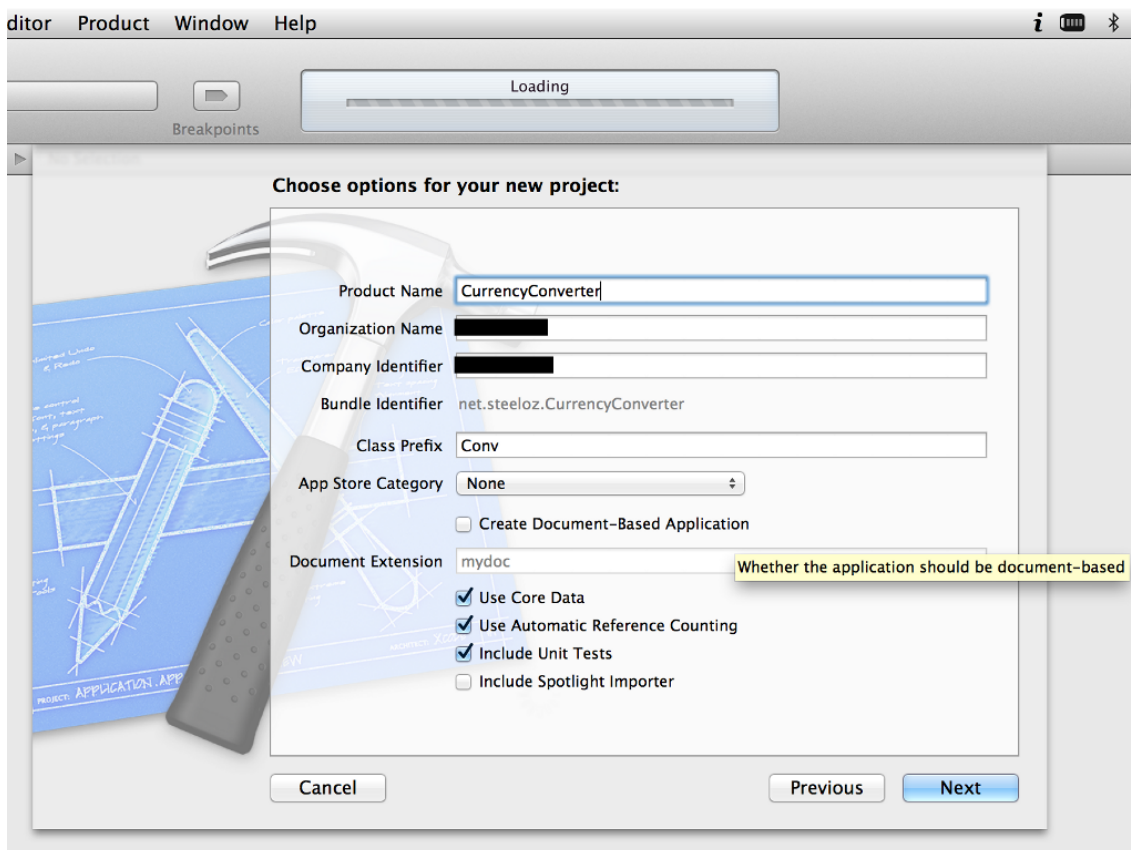
Xcode를이용한 Project의생성



윗그림에서보이듯이 XCode 4 를처음실행하면몇가지를선택할수있는화면이나옵니
다. 우리는여기서새로프로젝트를생성할것이기때문에 **"Create New Xcode Project"** 를선
택해줍니다.



새로운 프로젝트를 위한 template 을선택할수있으며여기서우리가개발하려고하는것은 Foundation Framework 을위한프로그램이기때문에 **"Cocoa Application"** 을선택해줍니다.

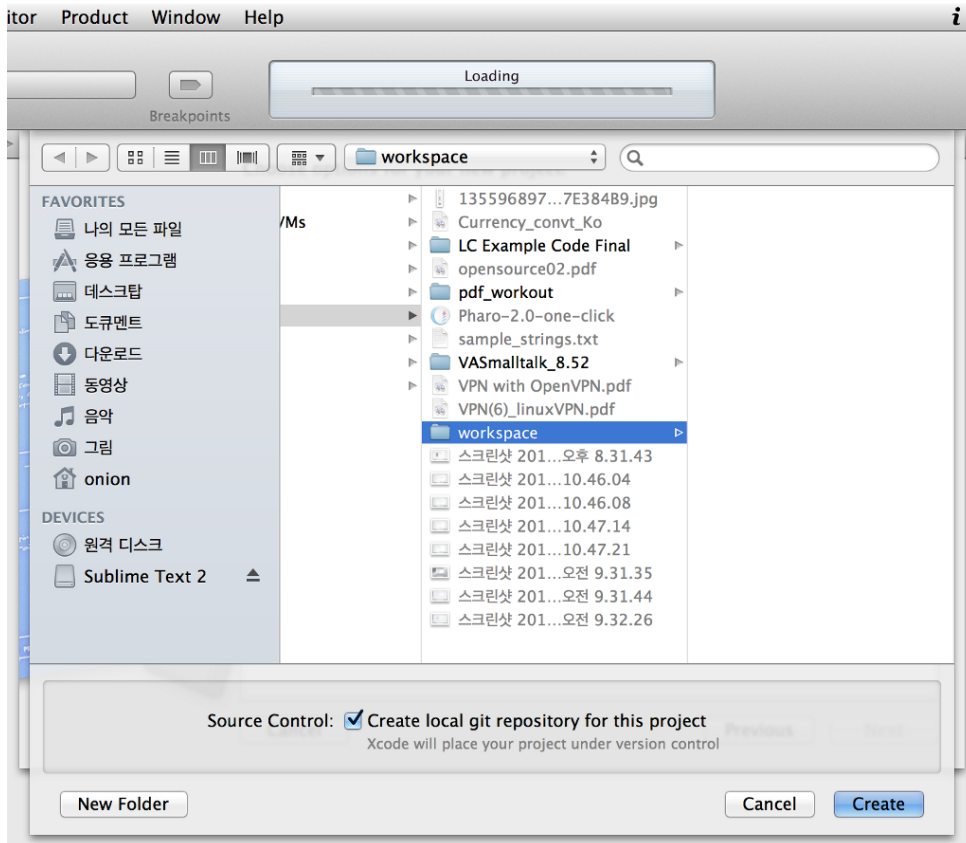


Project 에대한여러가지정보들을기입합니다. 신경써야할정보는다음과같습니다.

- Product Name :: 만들어질프로그램의이름인데당연하겠죠?

- Organization Name :: 저작자정보입니다.
- Company Identifier :: 기본적으로 domain rule을따른다. (ex: com.naver 등의규칙)
- Class Prefix :: 기본적으로생성되는 Class의 prefix를정한다. 여기서는 "**Conv**" 라는이름을 사용한다
- [CheckBox] Create Document-Based Application :: 여기서는 check를하지않는것으로한다. 왜냐 하면 Document 기반의개발방법을사용하지않을것이기때문이다. 때문에 unchecked 하도록한다

위의내용이작성되면 "**Next**" 를눌러서다음단계로진행하도록합니다.

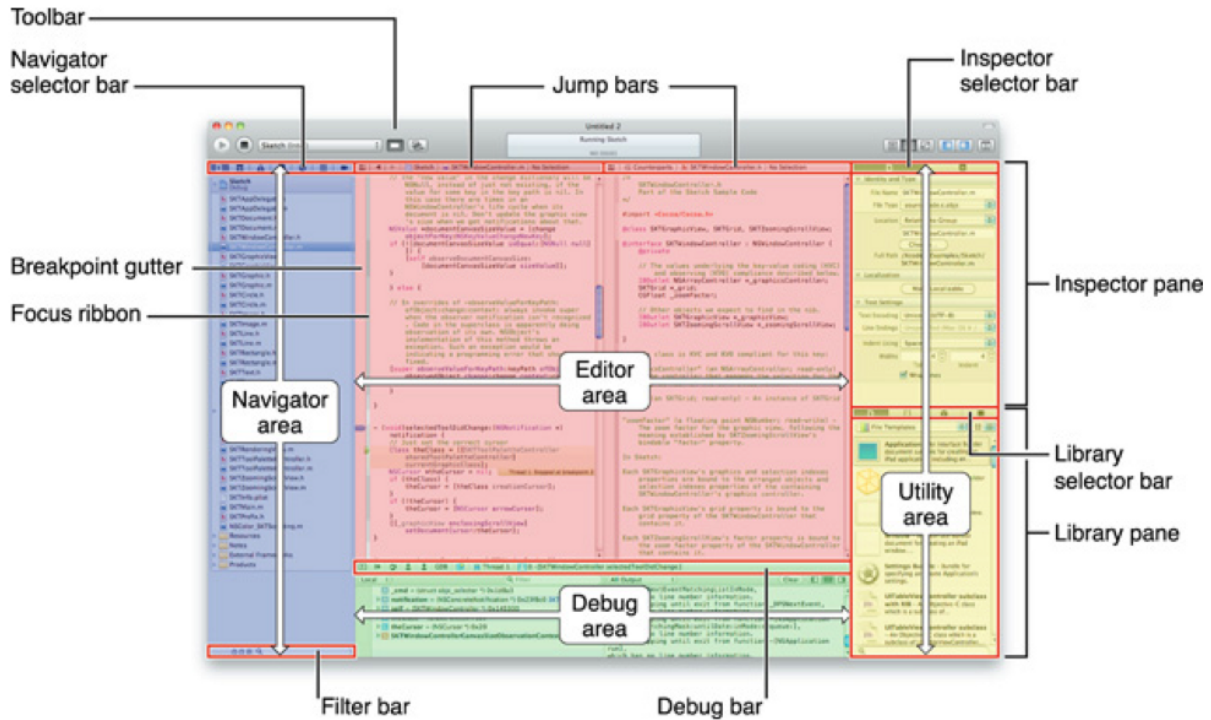


이제설정대로만들어지게될프로젝트를어디저장할것인가를지정하게됩니다. 디렉토리를지정한다음 "**Create**" 버튼을눌러서프로젝트생성을완료하도록합니다.

생성된 Project 의구성요소

프로젝트자체의구성요소에대해알아보기전에 xcode 의 Project 화면을먼저살펴본다음 프로젝트의구성요소들에대해알아보기로하겠습니다. 아래의화면을보겠습니다. 이 화면은 Xcode의시작화면에서 "**Learn about using Xcode**" 를누르면나오는화면입니다.

Xcode 의기본구성요소



위의그림에서설명된부분을하나하나짚어가면서설명하도록하겠습니다. 일단크게 4 가지의영역으로나눌수있습니다.

Navigator area

프로젝트에관련된 File 및 resource 에대한탐색을할수있습니다.

Editor area

프로그램의코드를편집할수있는영역입니다. 사용자 UI(xib등)도편집할수있습니다. iOS 개발의경우는 Phone Application UI 가되겠습니다.

Debug area

디버깅을진행하는경우관련된정보들을참고할수있는영역입니다. 평상시에는사용하지않을수있습니다. 일반적으로 Xcode로작업할때 **View >> Assistant Editor** 를열어놓고주로사용하는영역이되기도합니다.

Utility area

객체또는코드에대한 Inspect 정보들을표시한다거나라이브러리등을표시하게됩니다.

세부분설명

각세부분에대한설명은다음과같습니다.

Toolbar

Xcode의기본적인기능(run 이라던가)과정보등이표시됩니다. Xcode 의실행창에영향을끼칠수있는기능버튼과정보표시창등이있습니다.

Navigator selector bar

Project 에서다루는 File 및 resource 에대한도구들이있습니다. Class 의계층을볼수있기도하고 Class 내에서원하는것을검색할수있는도구를제공하기도합니다. 그외디버깅에관련된항목들을나열할수있는기능도가지고있습니다.

Breakpoint gutter

중단지점을표시할수있는영역입니다. 중단지점을 create/enable/disable 할수있으며생성한중단지점을없애려는경우는 Drag 해서코드편집창쪽으로 Drop 하면됩니다.

Focus ribbon

일반적으로 computer programming language 로프로그래밍을하는경우는개념상의 block 으로 지정되는 "영역(focus)" 라는게존재하게됩니다. depth가깊어질수록짙은색상으로표시되며같은색상끼리는 같은대상영역 이라고판단하면됩니다. 그런영역을색상으로표시해주는영역입니다.

Filter bar

Navigator area 에표시되는내용들을필터링하는조건을설정할수있는부분입니다.

Navigator selector bar 에서선택한내용에따라필터링을지정할수있는조건자체도바뀌게됩니다.

Debug bar

디버깅을할때 step 등을사용할수있는기능들이모여있는곳입니다.

Library pane

Xcode 에서사용할수있는라이브러리들이모여있는곳입니다. Xcode 4부터내장되어버린 Interface Builder 에대응하는것으로서 UI를설계할때사용할수있는 element 들을모아둔 곳이라고보면되겠습니다. 여기서원하는것을 Drag & Drop 하는방식으로작업하게됩니다.

Library selector bar

Library pane 에대해사용할수있는 Library 의종류를변경할수있도록합니다. 다음과같은 종류가있습니다.

- File Template Library :: iOS 용과 Mac 용을고를수있습니다. 여기에보이는내용은 **Navigator area** 로 Drag & Drop 이가능합니다. (아마잘쓸일은없을거같습니다만....)
- Code Snippet Library :: iOS, Mac, User(사용자정의) 등을고를수있습니다. **Editor area** 로 Drag & Drop 이가능하며 Drop 한곳에 Drag 한것에해당되는기본코드템플릿이삽입됩니다.
- Object Library :: Interface Builder 의화면에사용할수있는 element 들의모음입니다.
- Meda Library :: 코드를편집하고있는경우에는 Class 이름등이삽입되고 Interface Builder 화 면에서는해당되는객체를추가할수있습니다.

Inspector pane

현재어떤작업을하고있는가에따라내용이많이달라지는영역입니다. 예를들어코드를편집하고있는경우에는코드가소속된파일의정보가나오게됩니다. Interface Builder 화 면의경우에는선택한객체(element)에대한정보가나오게됩니다.

Inspector selector bar

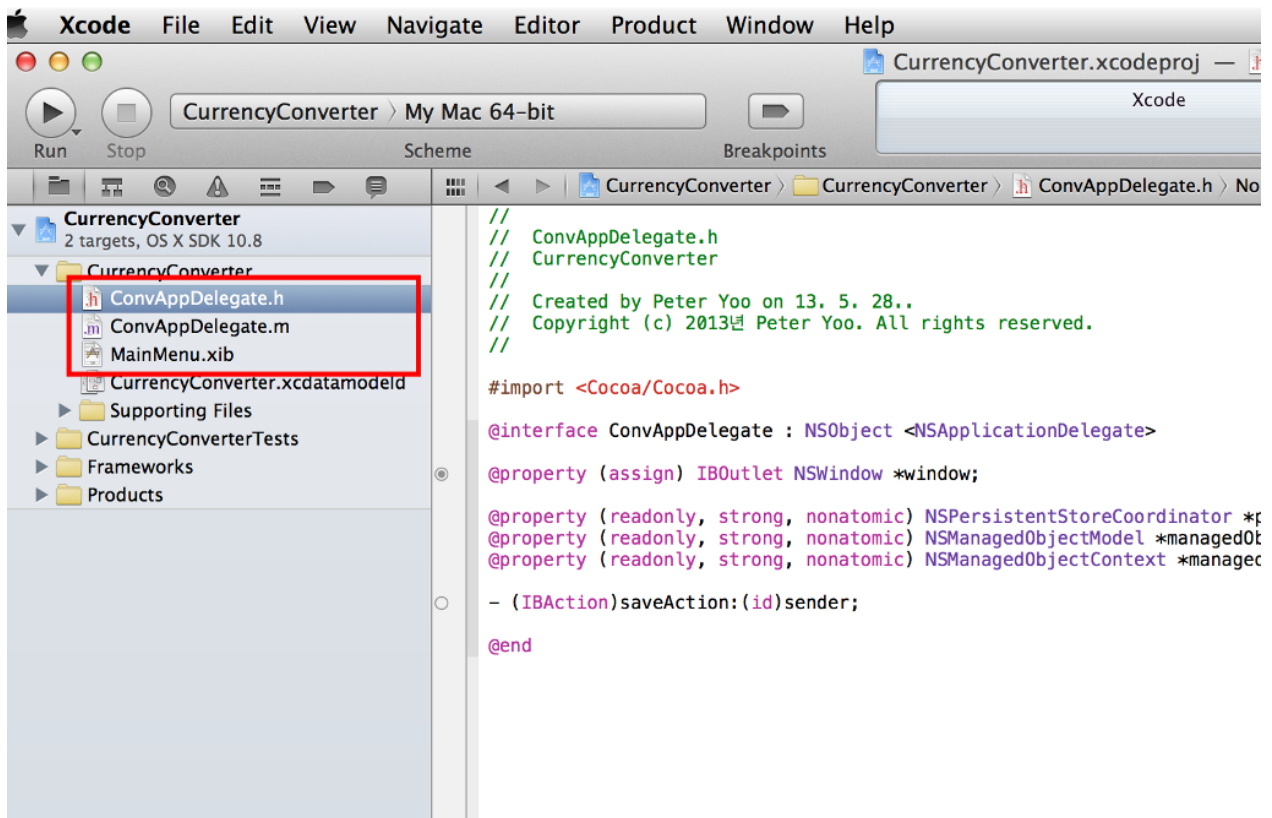
Inspector pane 에서어떤정보를뿌려주고있는가에따라사용할수있는기능이바뀌게됩니다. instance object 의경우는해당객체가참고하는 Class 도설정할수있습니다.

Jump bars

Editor area 에는공통적으로따라다니는기능으로서 **Navigator area** 에서파일을선택하는 것만이나아니라좀더편하게여러개파일및디렉토리를 옮겨다닐수있게해줍니다.

프로젝트의기본구성요소

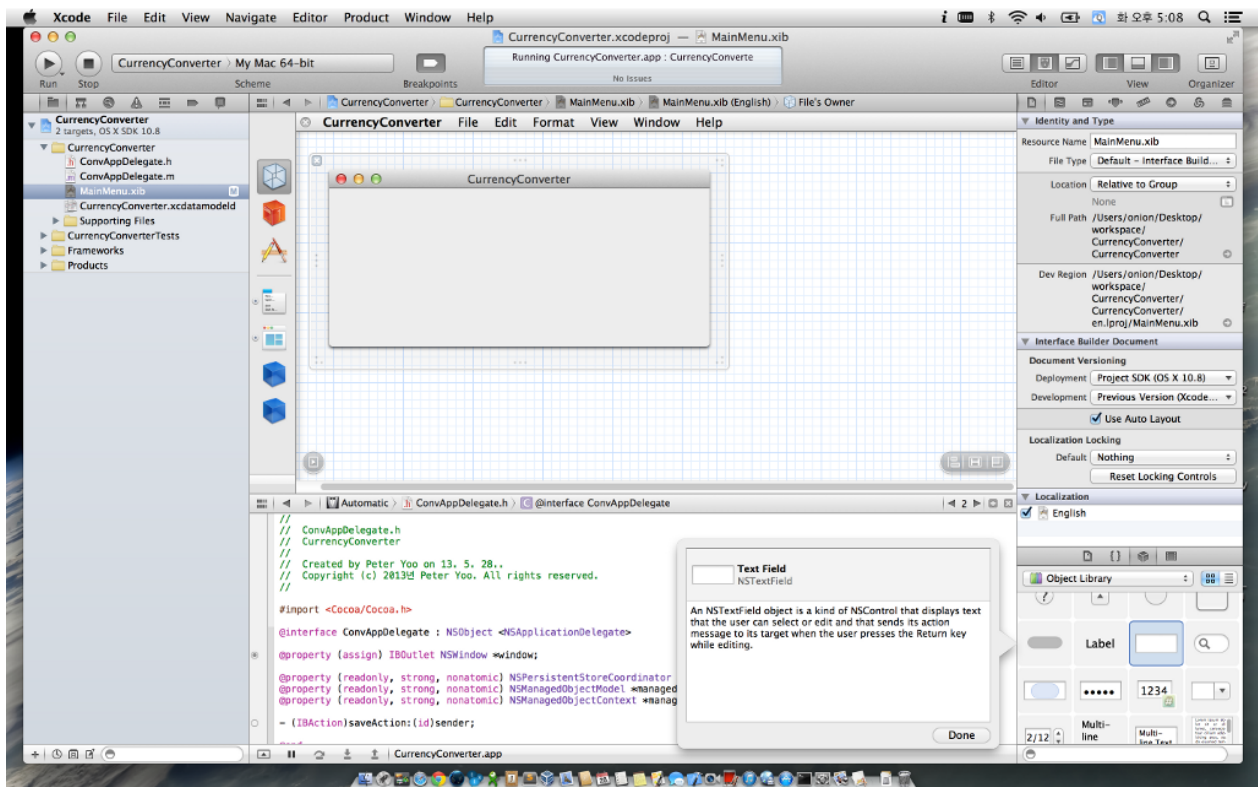
이제부터생성한프로젝트의기본구성요소에대해알아보겠습니다.



위의그림파일에보이는것처럼가장왼쪽의영역에는현재프로젝트에관련된 File 과 resource 파일들이기본적으로나열되어있습니다. 이중프로젝트에신경을써야할만한 파일들몇개만소개하도록하겠습니다.

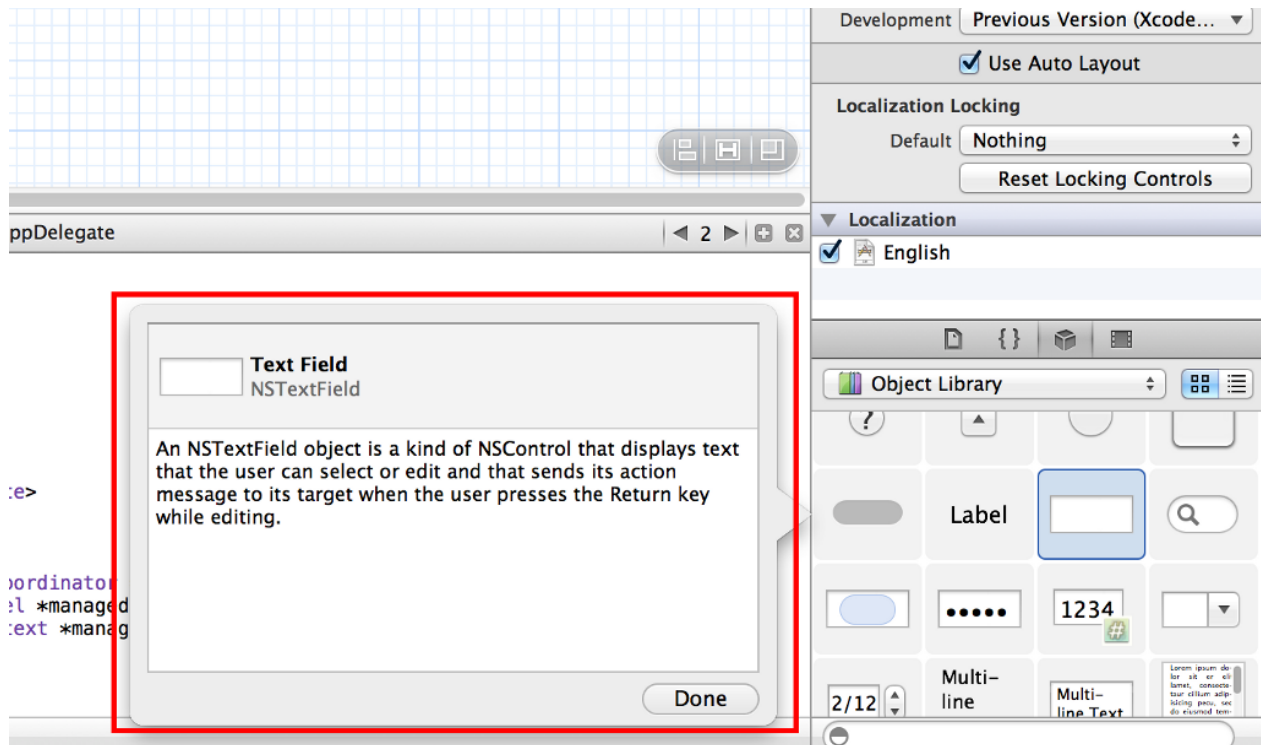
- _ClassName_AppDelegage.h :: 시작할때 Class 이름을 Conv로주었다면 ConvAppDelegate.h 이라는 이름으로파일생성이될것입니다. ConvAppDelegate 에대한 @interface 가선언되어있습니다. 이과정에서 ConvAppDelegate 클래스선언이없는것은이후에 Interface Builder 화면에서 MainMenu.xib 파일을편집할때다시한번알아보도록하겠습니다. 여기서는 interface 를 선언하면서 NSObject 라는 Class 의상속을받아구현하는걸로코드가진행되어있습니다.
- _ClassName_AppDelegage.m :: 시작할때 Class 이름을 Conv로주었다면 ConvAppDelegate.m 이라는이름으로파일생성이될것입니다. ConvAppDelegate 에대한 @implementation 이선언되어 있습니다.
- MainMenu.xib :: Interface Builder 에서사용하는파일입니다. 사용자 UI에대한내용이담겨있 으며해당되는파일을편집하려고 Xcode 내에서클릭하는경우에는, 코드편집창대신 Xcode에 Embed된 Interface Builder 의화면이보이게됩니다.

사용자 Interface의편집



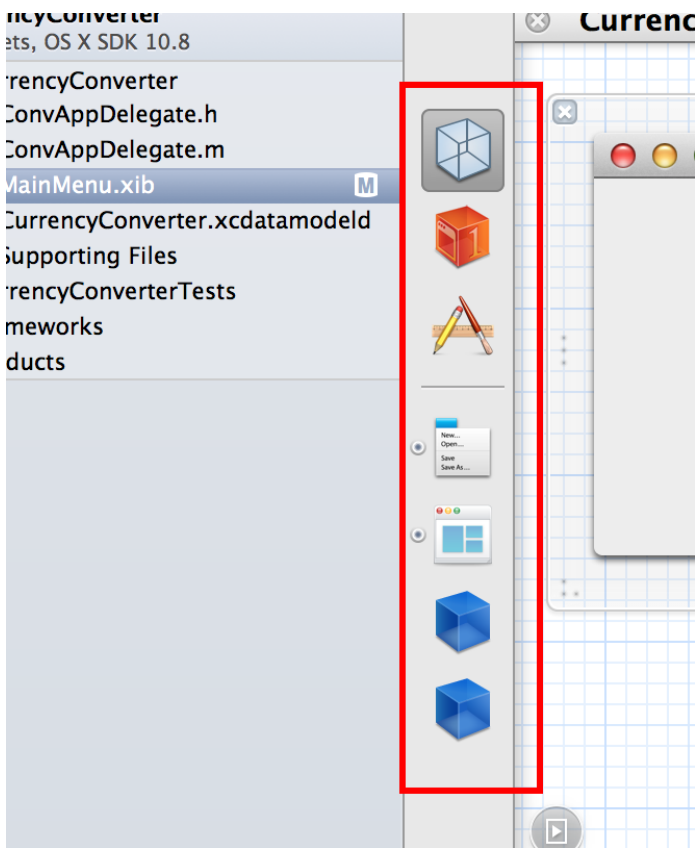
MainMenu.xib 파일을클릭해서화면을위와같이바꿔주시기바랍니다. 물론! 아래쪽의에 디터부분도가능하면위에보이는화면처럼바꿔주실것을권장합니다(View >> Assistant Editor 메뉴부분참고). 화면상에보이는메인창의크기를보이는것처럼줄여주시고편 집하기편한상태로봐주시면되겠습니다. 이제부터몇가지를추가적으로상세히살펴 본뒤 UI 의구성작업을진행하도록하겠습니다.

Interface Builder 에 대한 추가 설명



위의 화면에서 볼 수 있듯이 오른쪽 아래의 영역에서 **Object Library** 를 선택하면 객체를 click 했을 때 조금만 기다리면 객체에 대한 도움말을 볼 수 있습니다. 이제부터 진행되는 내용에서는 이 도움말에서 나오는 Class 이름으로 설명을 하도록 하겠습니다.





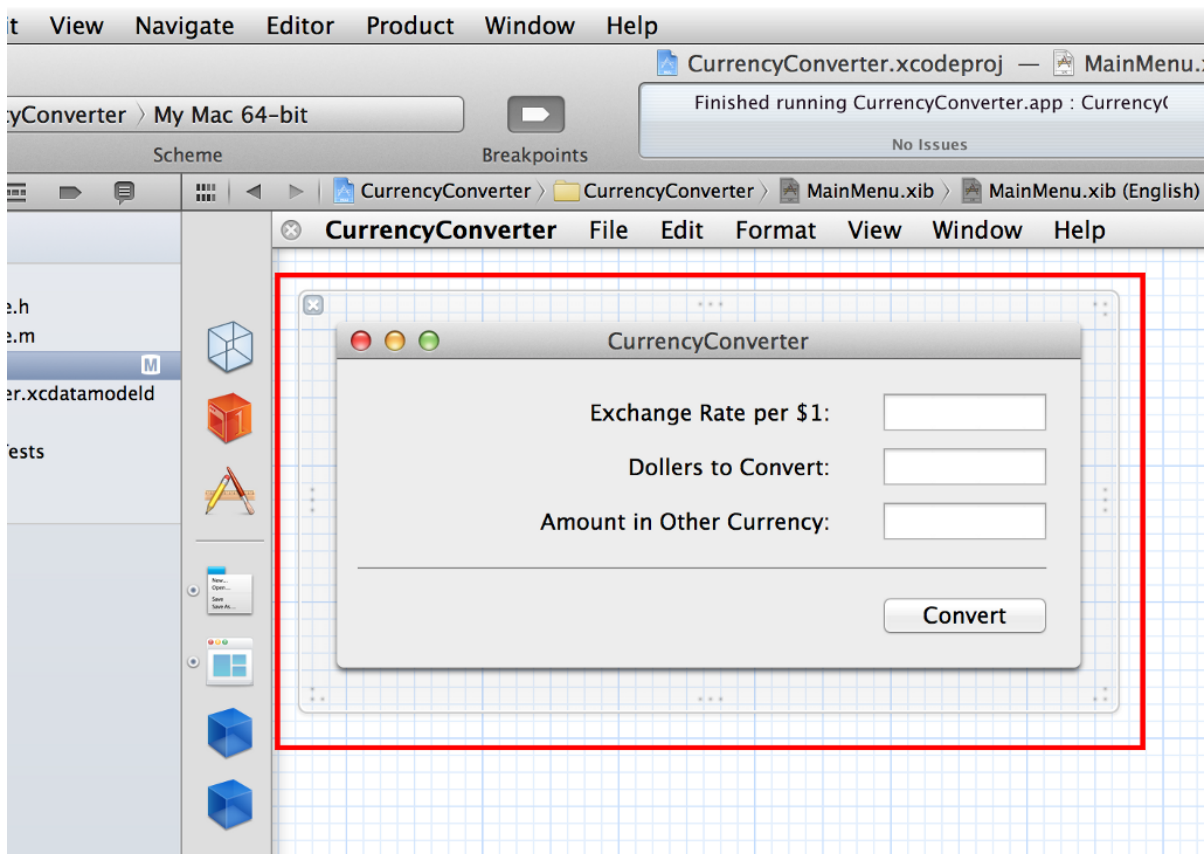
위의부분은예전의 Project Builder 시절에사용되었던팔레트와비슷한용도로쓰이는영역입니다. 이후에설명하게되겠습니다만... Windows의 Delegate 설정과그 Delegate를받는코드와의연계작업등여러가지용도로사용되므로주의해야할필요가있습니다.

기본 Windows에객체를배열하기

일단화면에보이는 Windows에객체를배열합니다. 아래와같은순서로객체를배열하도록합니다.

1. 오른쪽아래부분에서 NSTextField::TextField 를선택한다음 Drag & Drop 으로 Windows에하나배열한다.
2. 배열한객체를선택한다음 ⌘+D(Cmd+D) 를입력한다. 선택된객체가복사된다.
3. 같은방법으로객체를하나더복사한다. 이렇게 TextField 를 3개를놔둔다음보기좋게정렬한다.
4. 제일아래쪽에놓이게될 TextField 한개만선택한다음 Inspector 에서 Attribute 를편집하도록한다. **Behavior** 의값을 **Selectable** 로변경한다.
5. 오른쪽아래부분에서 NSTextField::Label을 Drag & Drop 으로 Windows에하나배열한다.
6. 같은방법으로 Label 을 3개로만들어서 TextField 의좌측에차례대로배열한다. 이후 Label 의 Attribute 에서 **Alignment** 를우측정렬로바꾼뒤 Title 의값을각각아래와같이바꿔준다.
 - Label_01::Title = "Exchange Rate per \$1:"
 - Label_02::Title = "Dollers to Convert:"
 - Label_03::Title = "Amount in Other Currency:"
7. 오른쪽아래부분에서 NSBox::HorizontalLine 을 Drag & Drop 해서이쁘게배치한다.
8. 오른쪽아래부분에서 NSButton::PushButton 을 Drag & Drop 해서이쁘게배치한다.
9. 배치한 PushButton 의 Title 을 Convert 로변경하고버튼의크기를알맞게조절한다
10. 배치한 PushButton 의 Attribute 에서 Key Equivalent 를클릭한다음키보드의 "Enter" 키를눌러준다. 이후로이버튼은 Enter Key 에반응하게된다.

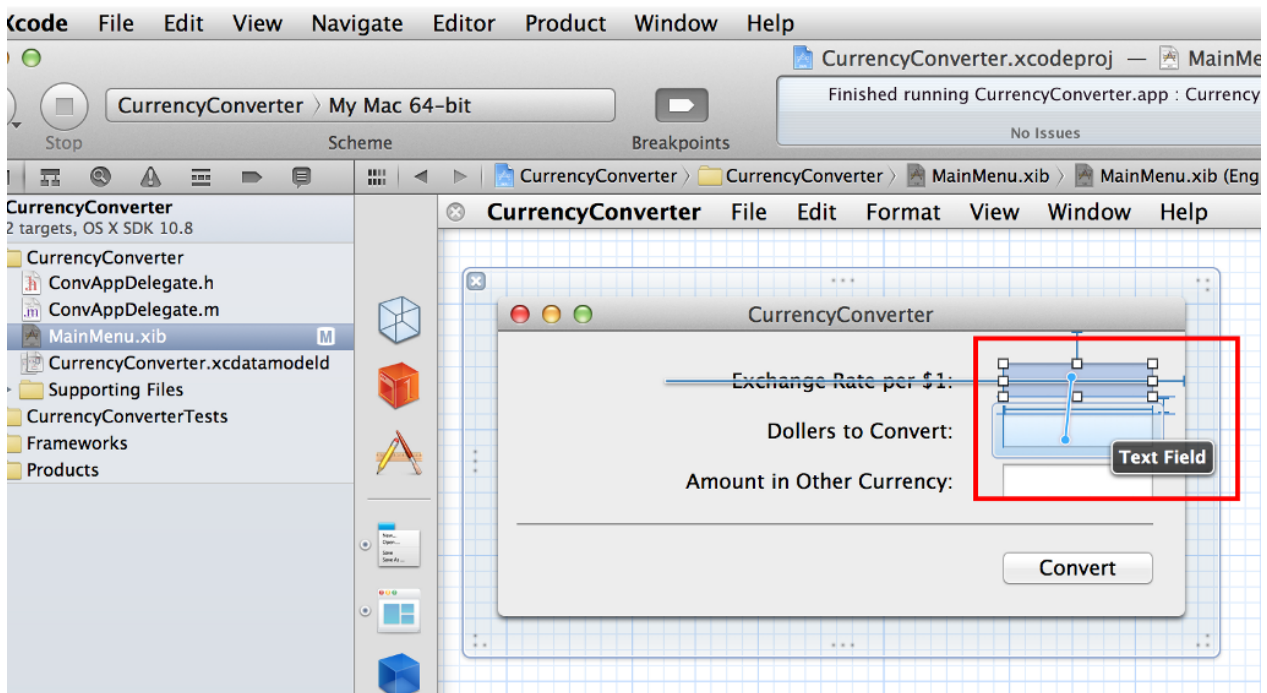
위의과정으로인한결과물의보이는모습이다음과같은모양이되었다면기본배치작업과정은종료입니다.



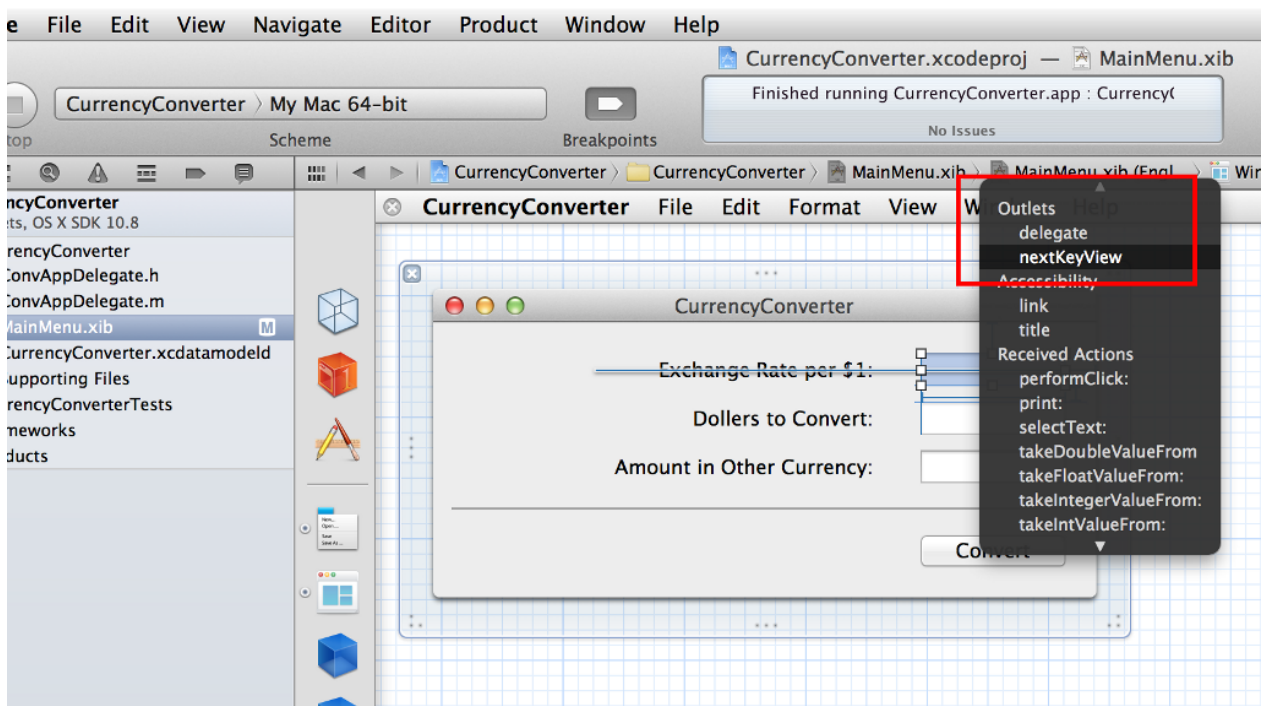
TextField 간에 Tab Order 구성하기

Interface Builder 에서는아래와같은과정을통해서현재객체와다음객체간의관계를설정할수있습니다.

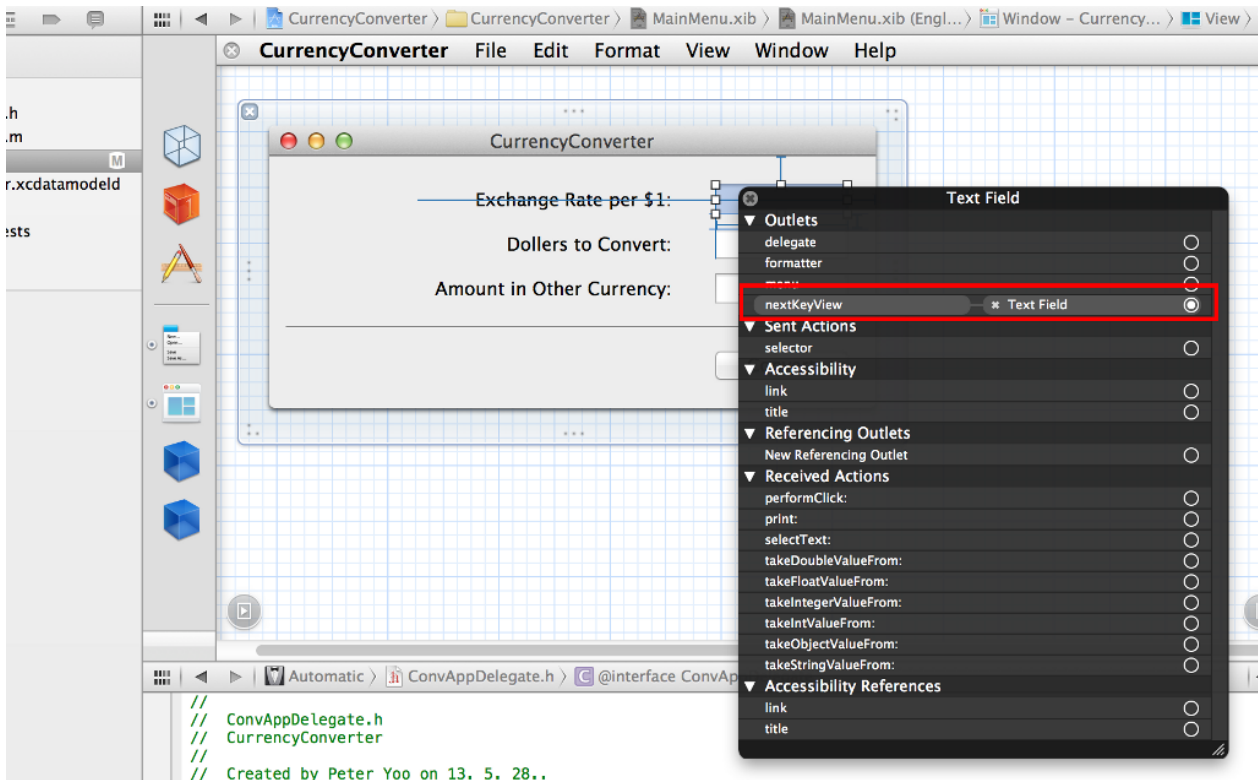
- 화면상에배열한 TextField 중첫번째 TextField 를선택합니다.
- **Ctrl** 키를누른채로첫번째 TextField 에서바로아래의 TextField 로 mouse 커서를드래그합니다. 그림그림과같은화면을볼수있습니다.



- mouse 클릭버튼을때면아래그림에보이는것처럼 popup 메뉴를볼수있습니다. 이중 **Outlets >> nextKeyView** 라는항목을선택해주세요.

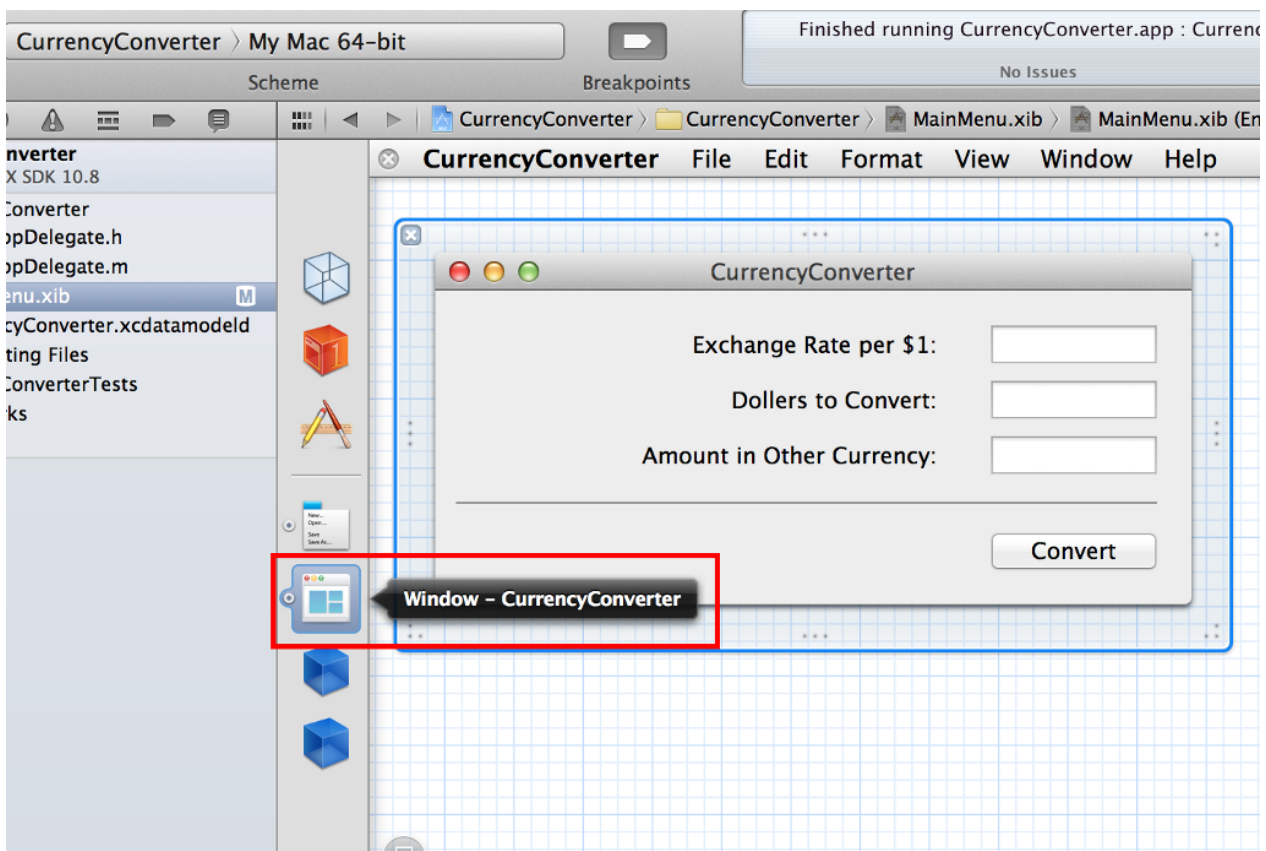


- 이후다시첫번째 TextField를 click 해서선택한다음 mouse 의오른쪽버튼으로 popup menu를불러내면아래처럼객체간의 outlet 이연결된것을확인할수있습니다.



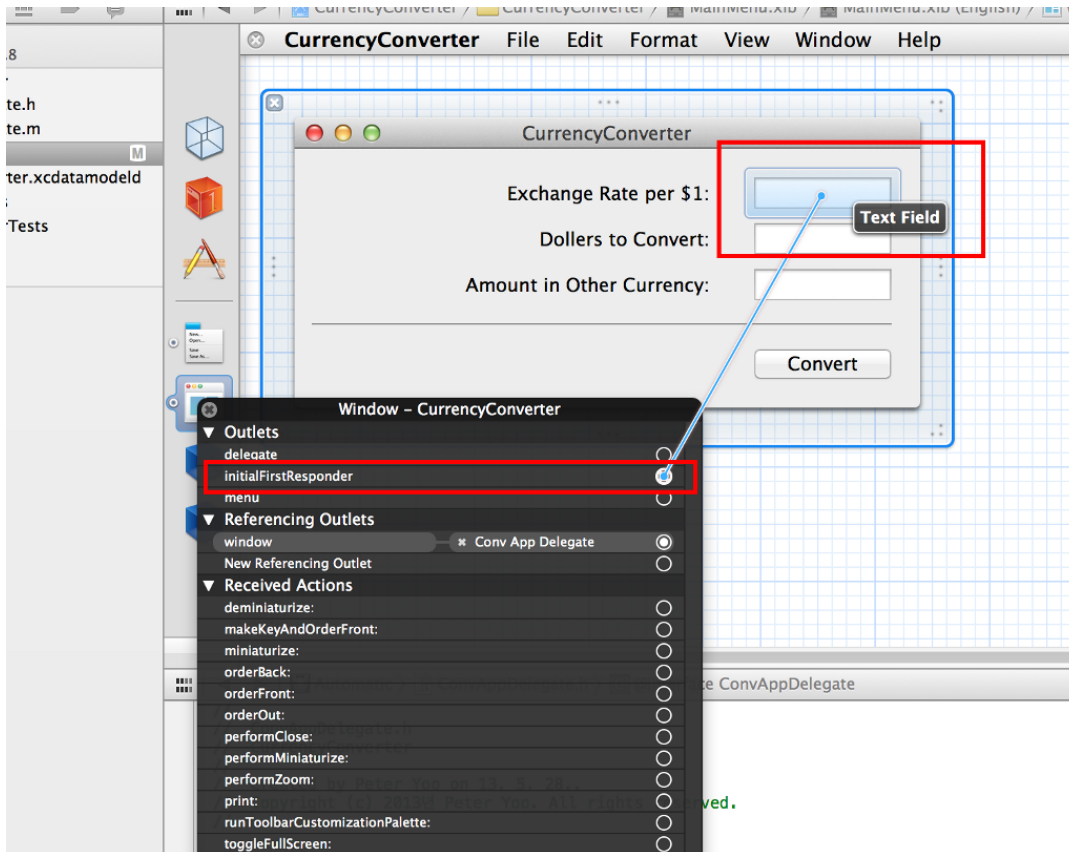
위의과정을기초로해서두번째 TextFiled 의 nextKeyView 를첫번째 TextFiled 로연결해주세
요.

Windows 에대한 initialFirstResponder 설정하기

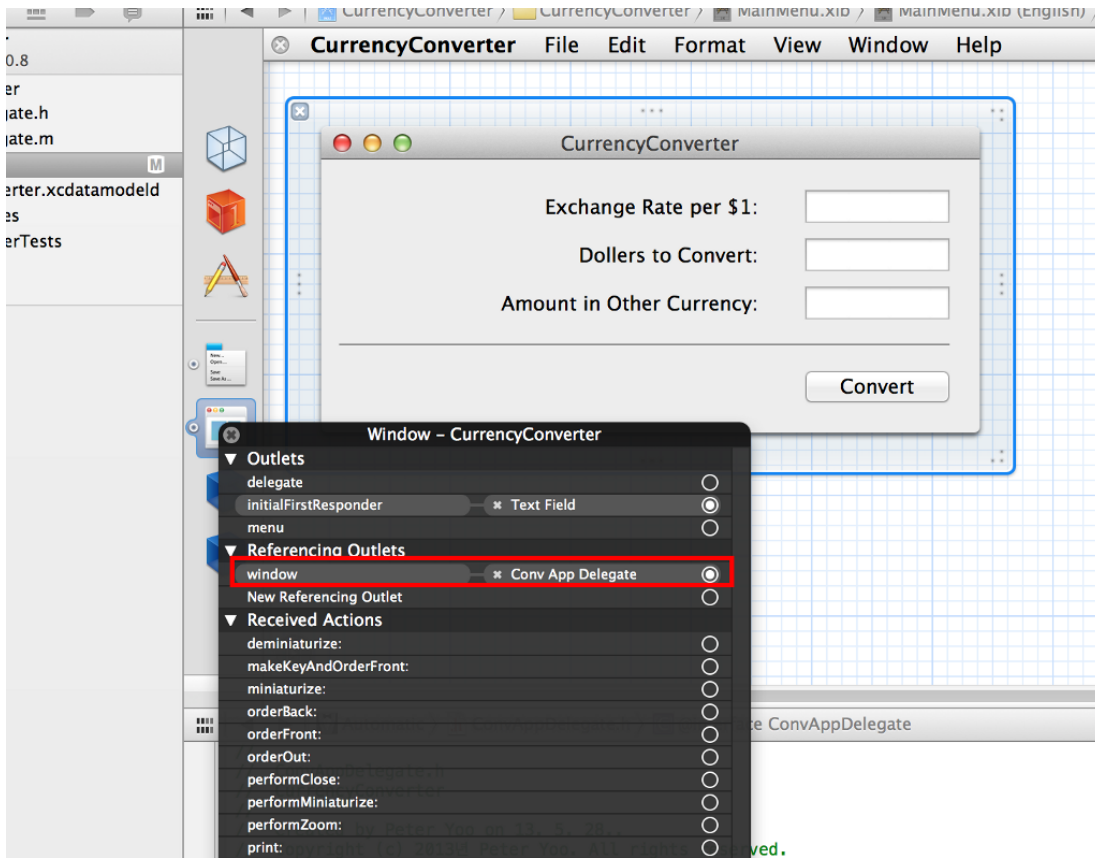


위의그림에보이는부분을주의해주시기바랍니다. 그리고 **Window - CurrencyConverter** 라
고나와있는부분에서 mouse 의오른쪽버튼을클릭해서 popup menu를불러주시기바랍니

다.



등장한 popup menu 에서 **Outlets >> initialFirstResponder** 항목을 봐주시기 바랍니다. 오른쪽에 보면 동그라미가 있죠? mouse 커서를 가져가면 + 기호로 변하게 됩니다. 이때 mouse click & drag 로 화면에 보이는 것처럼 선을 가장 첫 번째의 TextFiled 로 가져가서 drop 해주세요. 그러면 해당되는 outlet 이 연결됩니다.

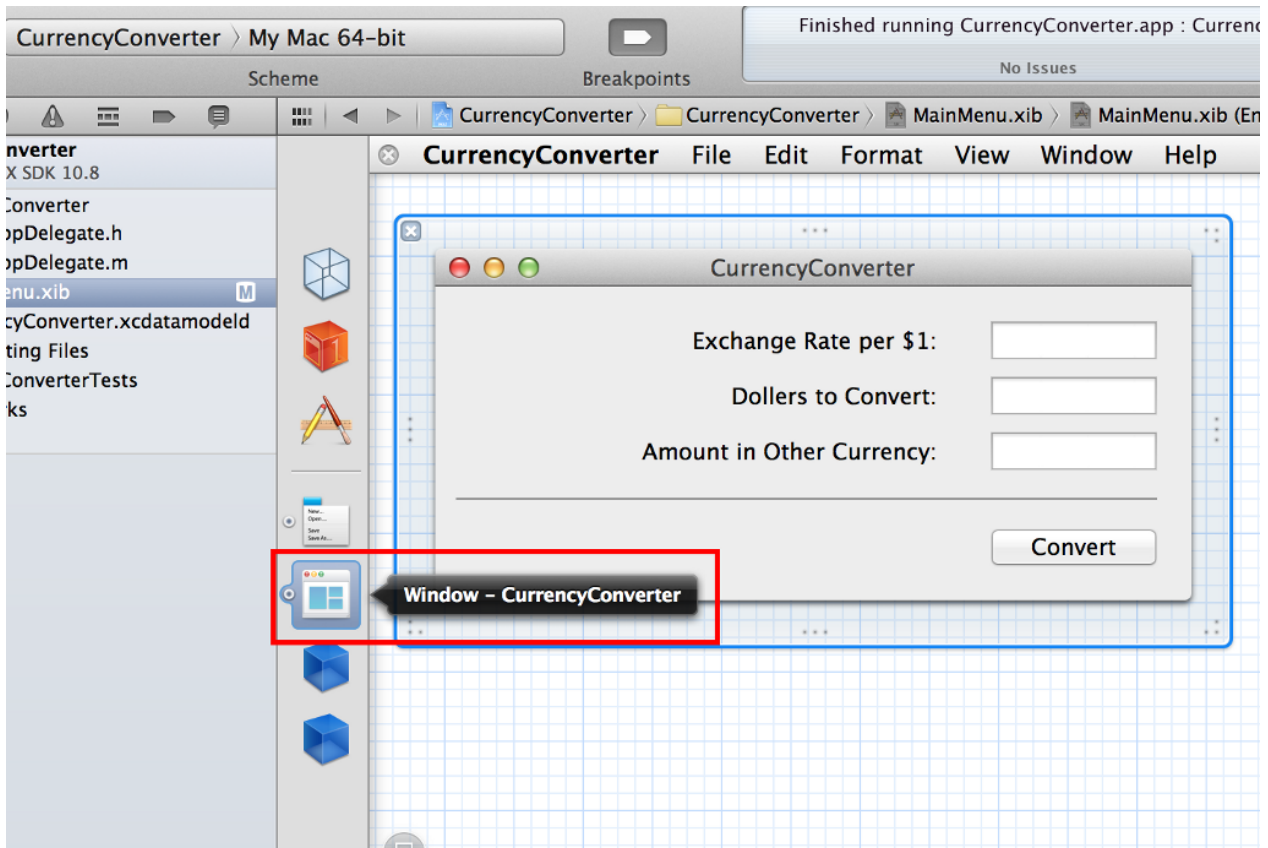


이후화면을보면 popup 에서 outlet 이새로연결된것을확인할수있습니다.

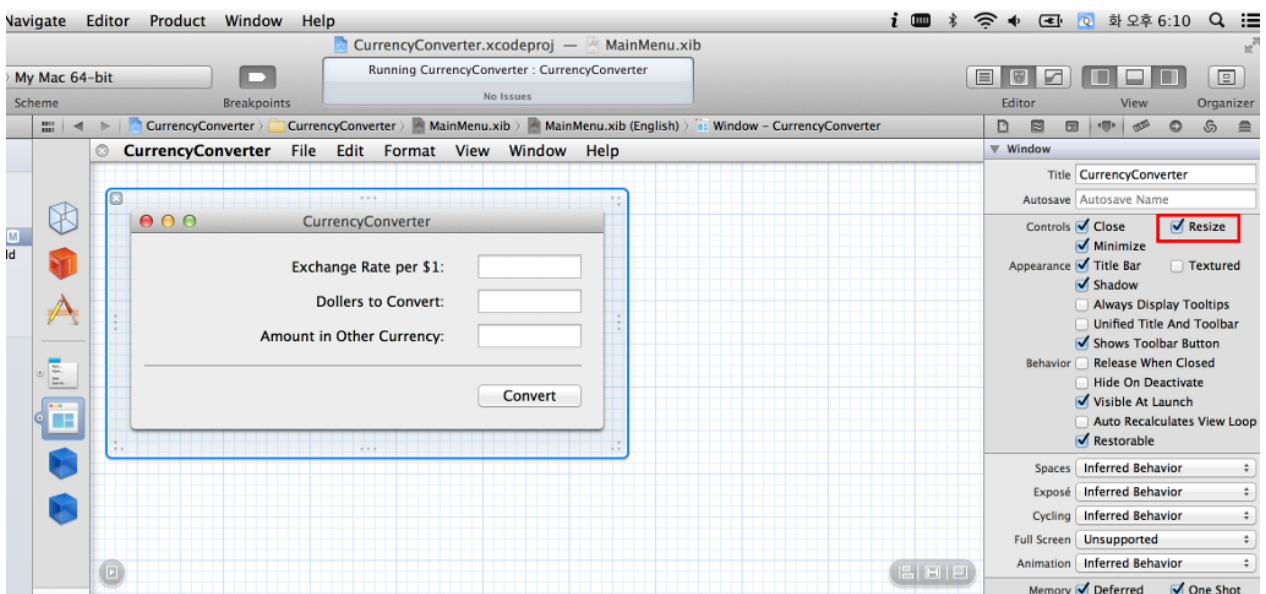
지금작업은처음 application 이구동되었을때입력의 focus 를어디로놓을까에대한부분입니다. 고로이작업이후에 application 을구동하게된다면자연히첫번째의 TextFiled 에뭔가를입력할수있는상태로자동으로준비가됩니다.

Windows 의 Attribute 조정

이렇게만들어진 Window의크기를조정하게하면나중에 resize 시 Layout 이깨지는현상을 경험하게됩니다. 일단은 resize 가안되게기능을제한하도록하겠습니다.

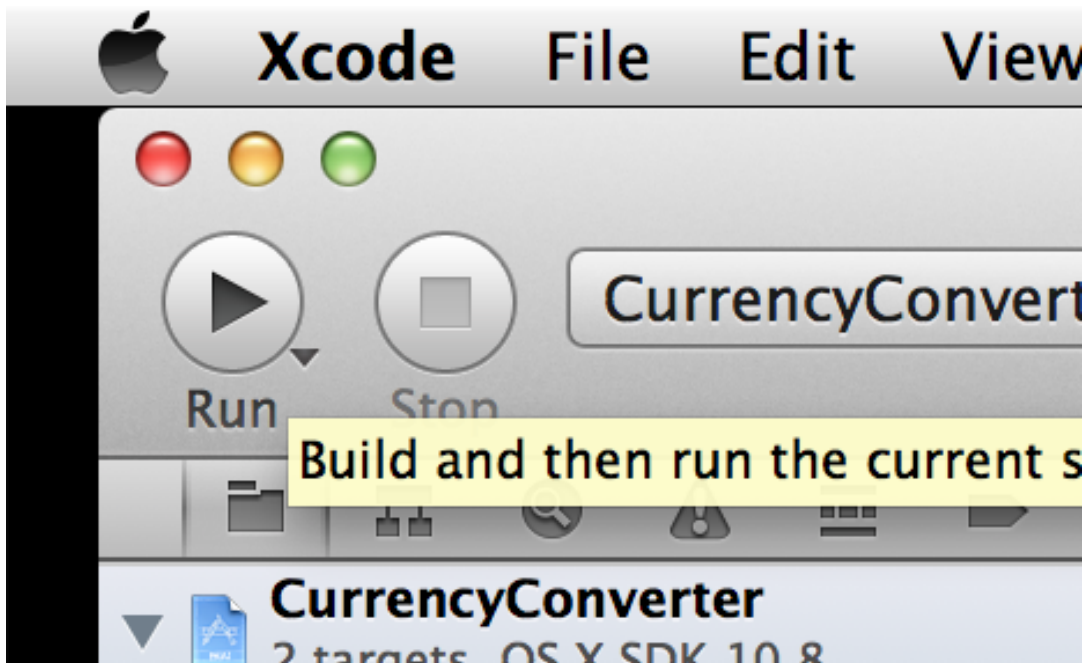


위에보이는것처럼 windows 를 click 하고오른쪽의 Inspector 부분의 Attribute 부분을확인합니다. 이후아래화면처럼 Attribute 에서 resize 를 uncheck 합니다.

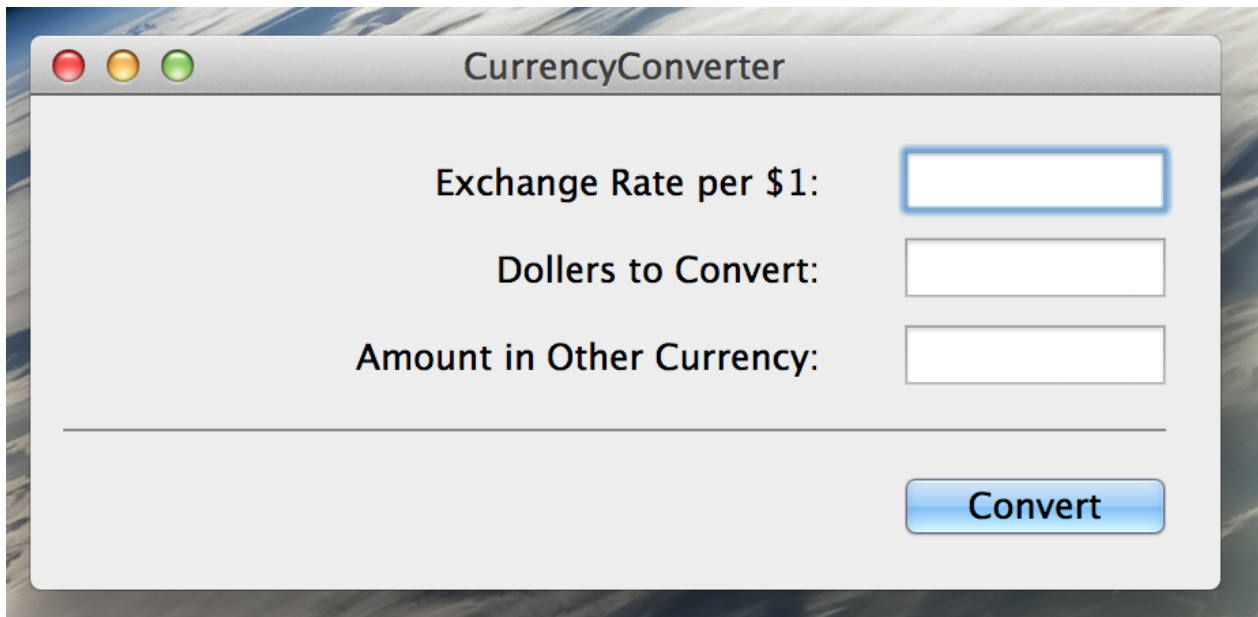


작업된 **Interface** 의 테스트

Xcode의 화면 왼쪽 상단을 보면 Run 이라는 버튼이 있습니다.



버튼을 눌러서 만들어진 interface 가 정상이라면 아래와 같은 구동 화면을 확인할 수 있습니다. 화면을 확인했으면 ⌘+q(Cmd+q) 를 눌러서 application 을 종료합니다.

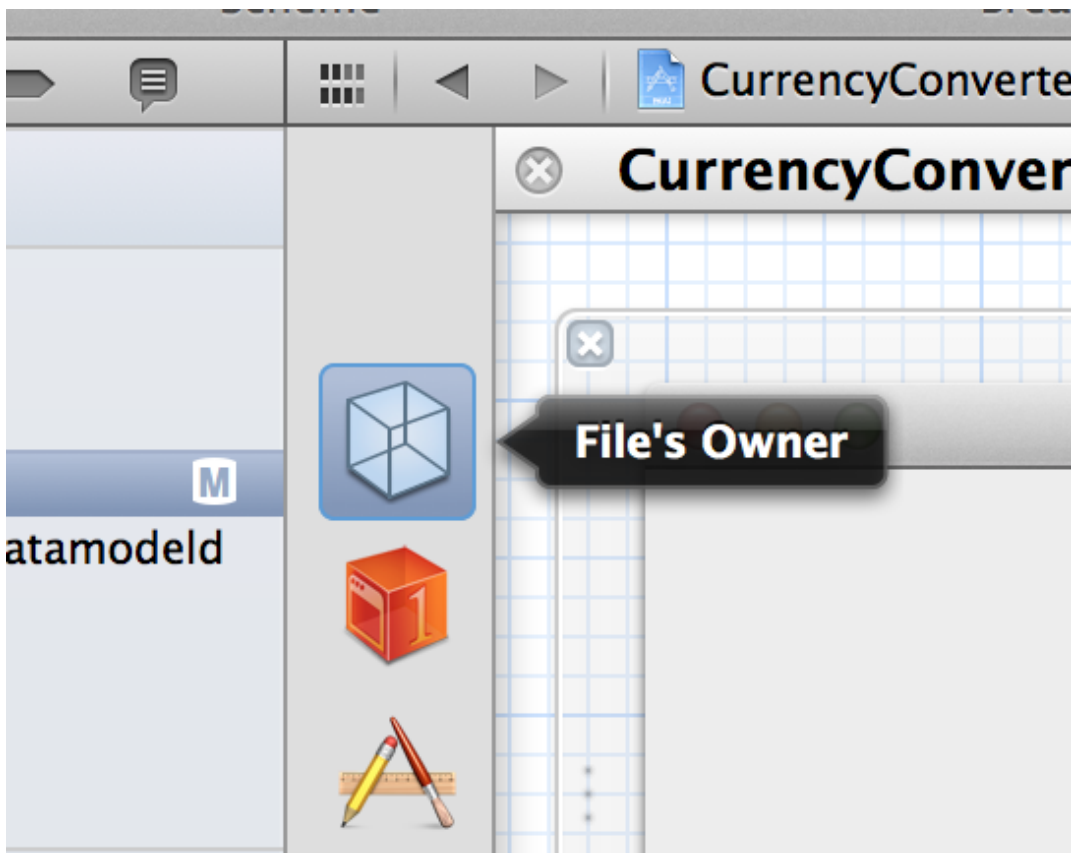


Interface Builder 에서 생성한 UI 와 instance Object. 그리고 delegate 의 관계

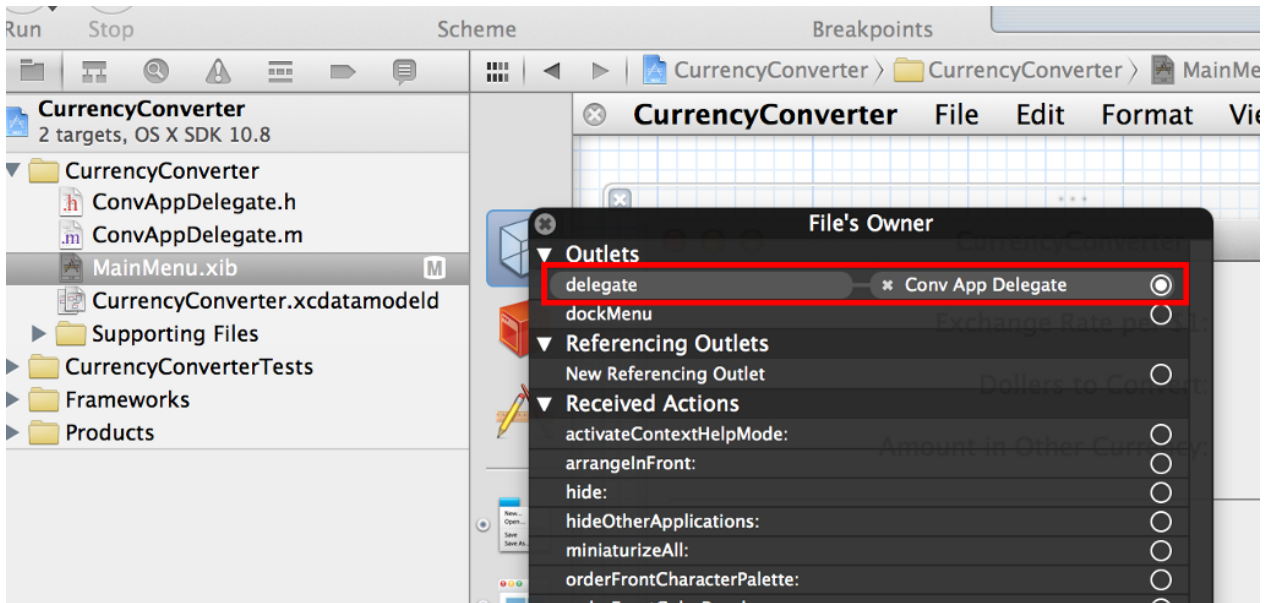
이부분이 기존 문서에는 없던 부분이라 좀 생소할 것 같아서 세부적인 설명이 필요한 부분이라 생각되어 별도로 내용을 추가하도록 합니다. (명확한 설명이 없이는 좀 헷갈리는 부분이기도 합니다)

▶ 지금이 과정에 대한 작업 부분은 과정이 끝나는 이후로 사용하지 않습니다. 고로 현재 시점까지의 소스코드에 대한 백업을 권장합니다

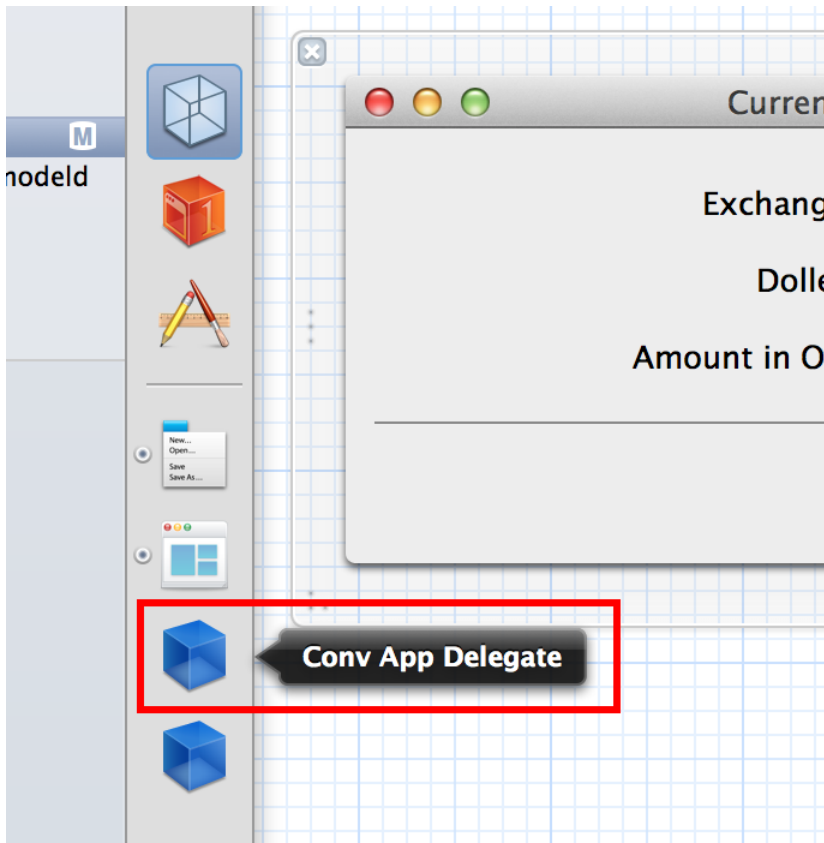
기존의 문서에서는 Interface Builder 의 Palette 에서 객체를 생성하고 이를 Program Code 로 연결하는 작업등을 진행했었지만 Xcode 4에는 그런 작업을 할 필요는 없습니다. 사실 Xcode 4에서 Project 를 **Cocoa Application** 으로 생성한 시점부터 지금부터 설명하려는 작업이 이미 기본적으로 되어 있기 때문입니다. 이제부터 Interface Builder 에서 작업된 UI 가 어디로 어떻게 연결되어 있는지에 대한 부분을 조금만 살펴보도록 하겠습니다.



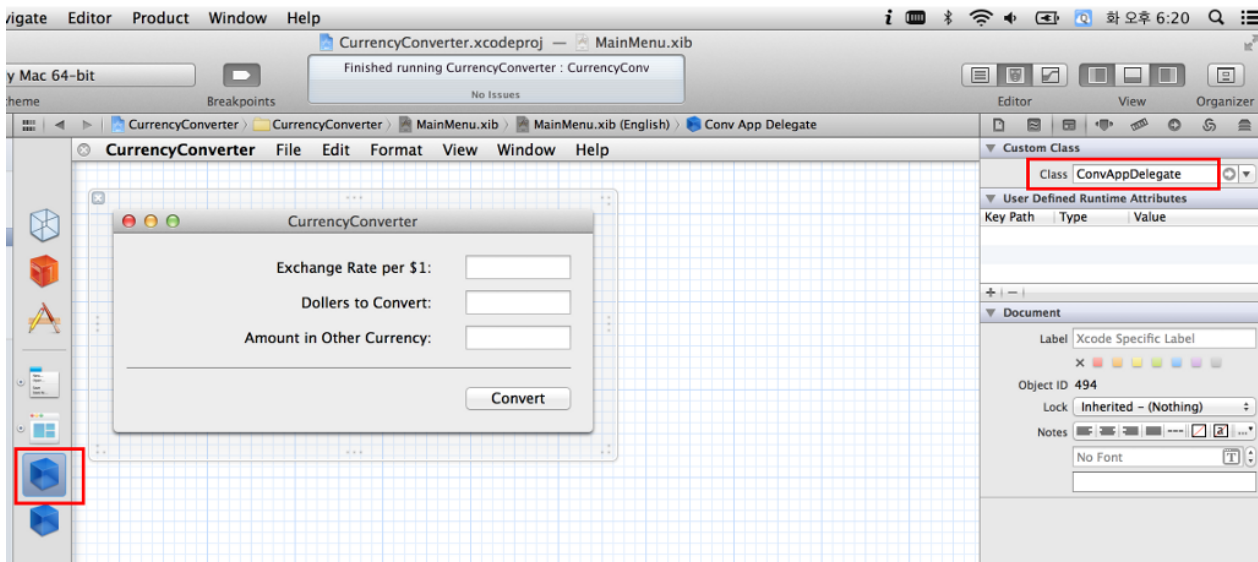
그림에 보이는 부분에서 mouse 의 우클릭을 하도록 합니다. 그러면 popup menu 를 확인할 수 있습니다. popup menu 를 좀 더 자세히 살펴보도록 할까요?



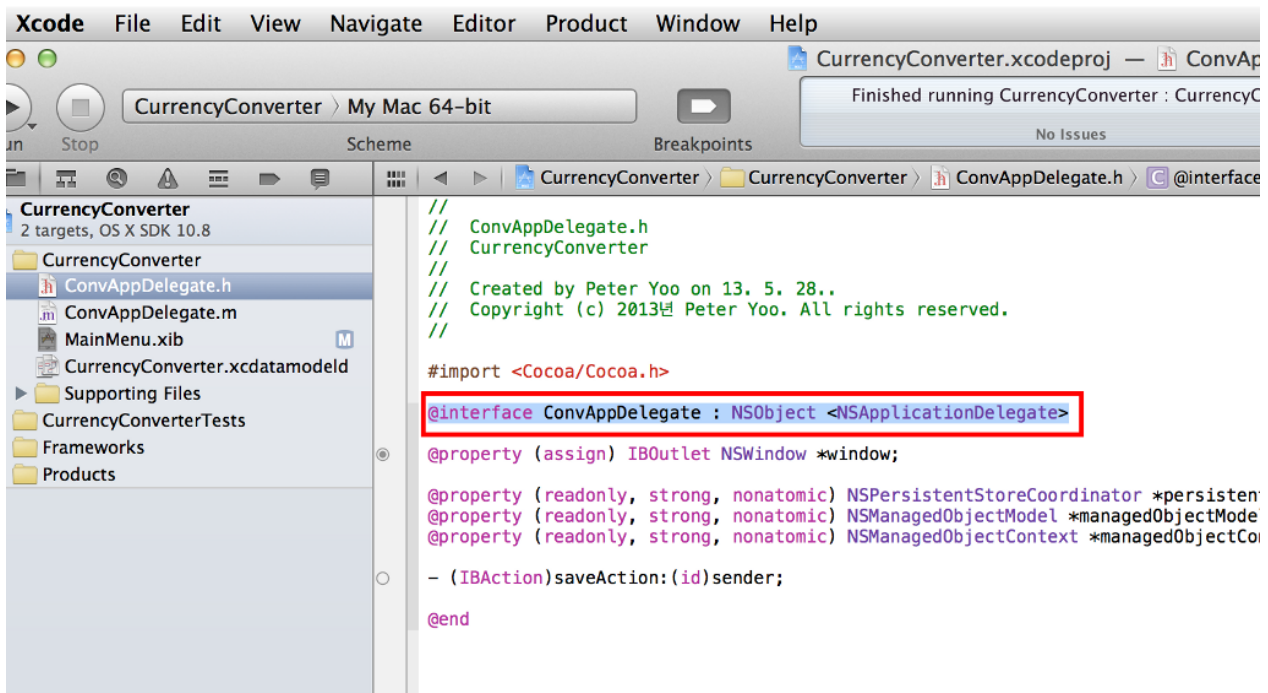
popup menu 에는 이미 delegate 가 Conv App Delegate 로 연결되어 있는걸 확인할 수 있습니다. 흠.. 그래요. 연결된 건 좋은데 대체 이걸 어디로 연결된 걸까요?



이야? 관련되었을 거 같은 이름을 찾다 보니 생각 외로 가까운 데 있었네요? 그럼 이 객체를 클릭해 보도록 하겠습니다. 이후 Inspector 쪽의 Attribute 부분을 보도록 하겠습니다.



호오.. 이객체의 Class 는 ConvAppDelegate 로 지정되어있네요?... 그럼이녀석은분명어딘가에있을겁니다.. 어디있는지한번찾아보기로할까요?



역시.. ConvAppDelegate.h 파일에이미선언되어있는 @interface ConvAppDelegate 로 연결되어있었군요!.

이강좌의처음부분에서제가 ConvAppDelegate.h 파일을언급하면서 Class 선언에대해서는 아래부분에서살펴보자는얘기한걸기억하신분이있을까요? 맞습니다. 굳이 Code 에서 Class 를선언하지않아도이미 xib 파일에서 instance class 형태로존재를하고있기때문에 해당되는 instance class 의구현(implementation)만 Delegate 되어있으면되는거였습니다.

그렇다면대체 Delegate 는뭔까요? 그것은 Objective-C 에서의 Delegate 부분에서좀더자세히 보도록하고여기서는대략만설명하도록하겠습니다.

Interface Builder 에서배열되고놓여지는객체들은전부 **Inspector >> Identity** 부분에서해당되는객체가가리키는 Class 를지정할수있습니다. 그리고 Xcode 는이런정보들을기초로 program code 에서특정객체(Class)를불렀을때 UI 상의어떤객체가동작하게될지알수있는거죠. Xcode 에서는이런연결들을통해프로그램을제어합니다. 이렇게 UI 객체의 Class 등을정의한것을 **Delegate(위임)** 라고하는것같습니다.

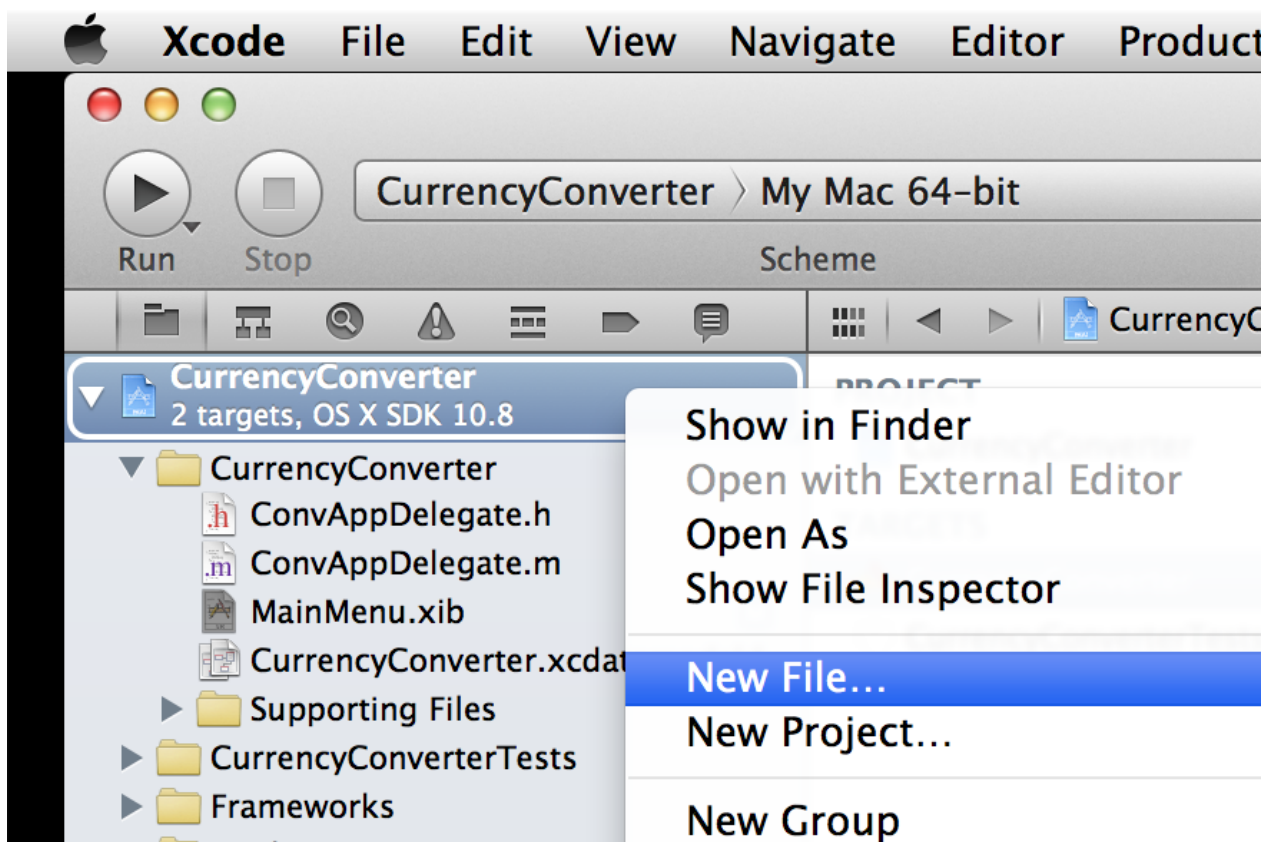
그렇다면..... 내가별도의 Class 를만들어서이것을 Delegate Class 로지정할수는없을까요?
 그래요.. 안될리가없겠죠. 이제부터는.. 크게쓸모는없지만어떤구조로작동되는지를
 알아보기위해번외적인작업을진행함으로서어떤식으로 Delegate 를연결하는지를아
 주조금더알아보도록하겠습니다.

Form 의 Delegate 변경하기

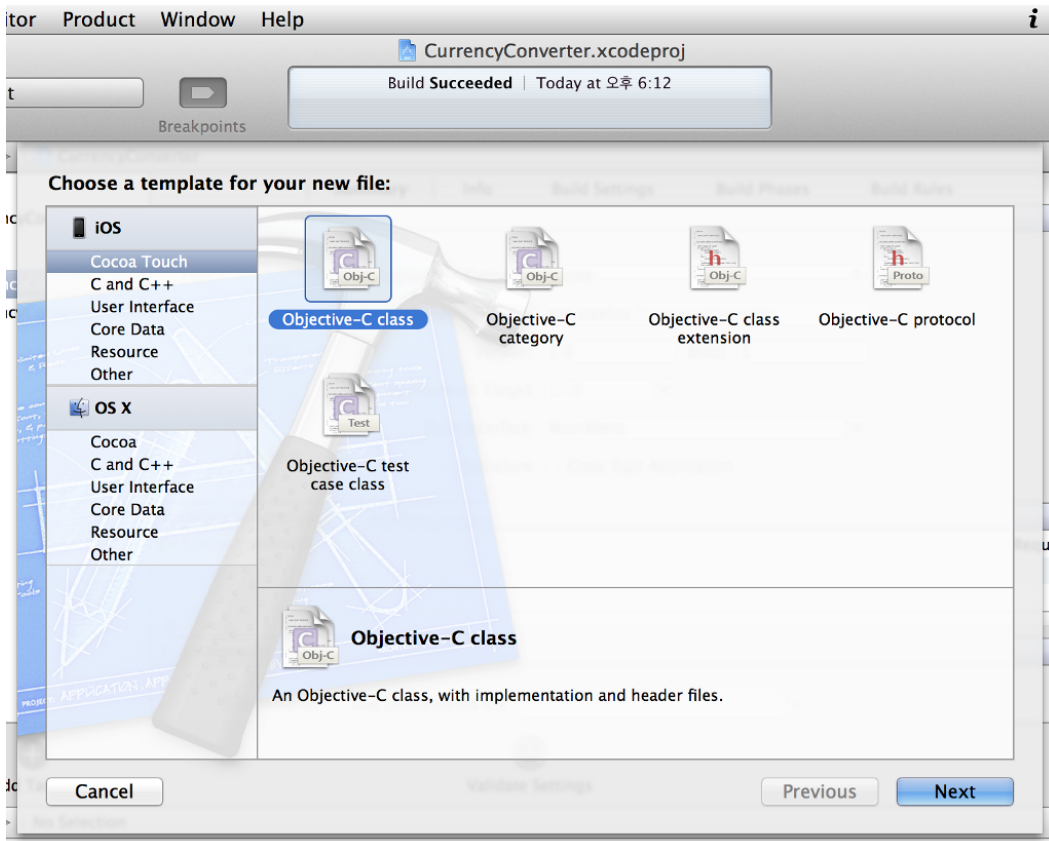
일단 Delegate 를변경하기위해서는 Windows의 Delegate 를받아줄뭔가가필요합니다. 그럼
 어디서부터시작을해야하는게맞는걸까요?

Delegate 의대상이될 Class 생성

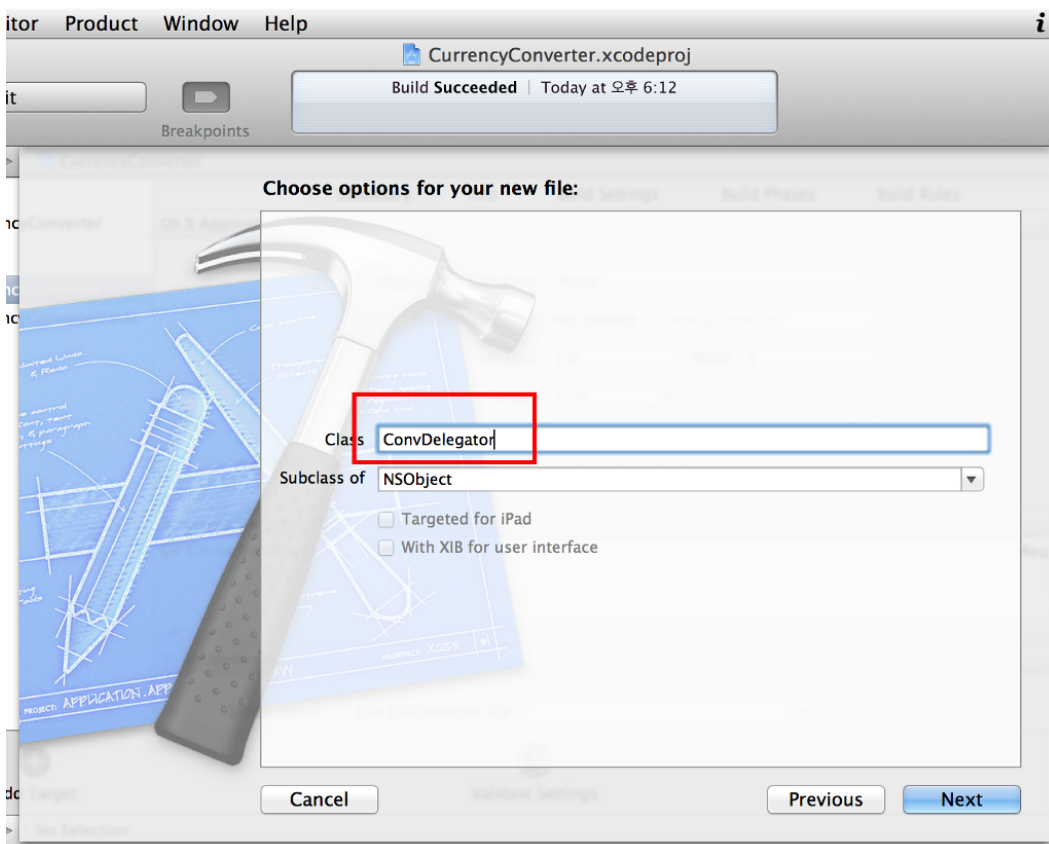
Delegate 에는타겟이필요합니다. 누군가나의대리자가되어야할필요가있다면없는존
 재를대리자로내세울수는없겠죠. 일단 Delegate 의교환에는교환대상이될새로운 Class
 가필요합니다.



위화면에서보이는것처럼 Project 위에서 popup 메뉴를이용해서새로운파일을생성합니
 다. (사실나중에안됩니다만.. CurrencyConverter Folder 아이콘위에서파일을생성했다면아
 래에설명하는과정중하나가줄었을겁니다)

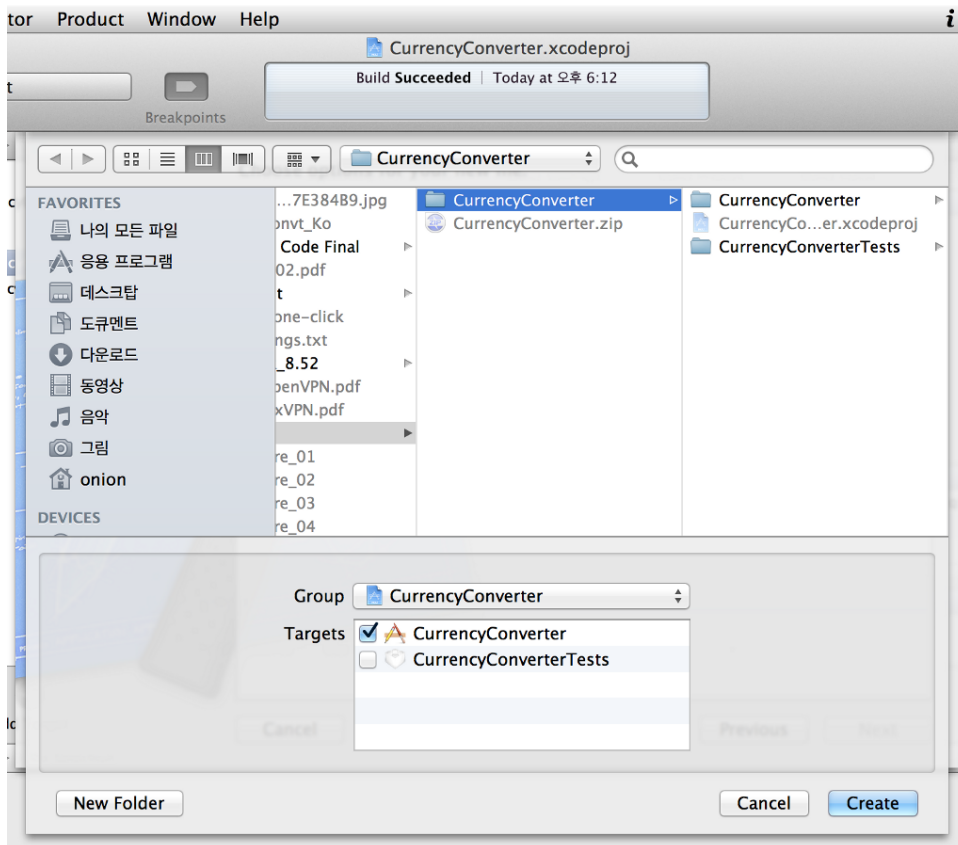


New File 메뉴를 눌렀을 경우 나오는 화면입니다. 우리는 새로운 Class 를 추가하려고 하는 것이기 때문에 **Objective-C Class** 를 선택해줍니다.

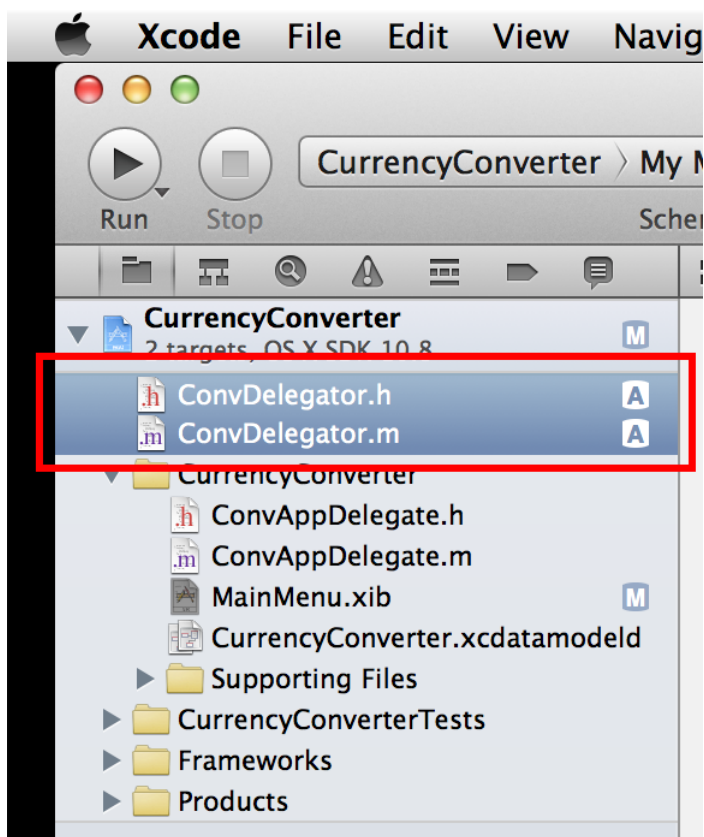


새로 만들게 될 Class 의 이름입니다. 어차피 Delegate 의 테스트에 사용할거고 본 강좌에서는 **Conv** 라는 prefix 를 사용하기로 이미 정했기 때문에 Class 의 이름은 ConvDelegator 로 정해줍니다.

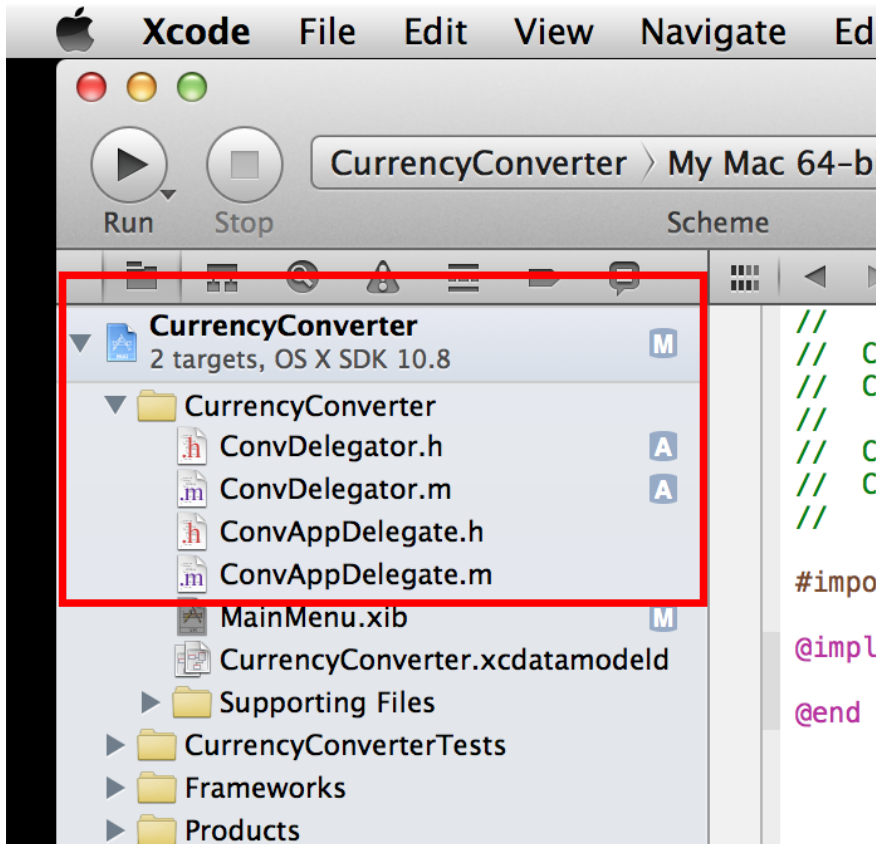
니다.



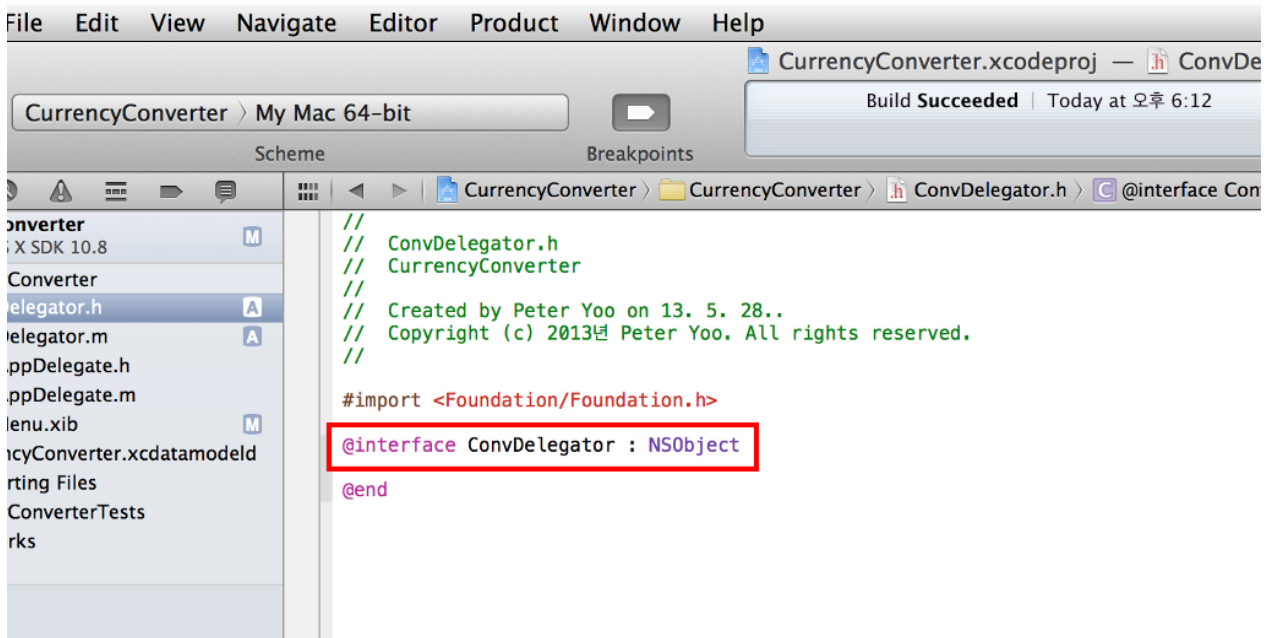
이제 Class 파일을 저장할 경로를 설정할 때입니다. 대략 프로젝트 디렉토리 아무곳에나 생성하기로 합니다. (어차피 테스트하고 지울거니깐요. 이전단계까지 백업은 미리 받아놓으셨겠죠?)



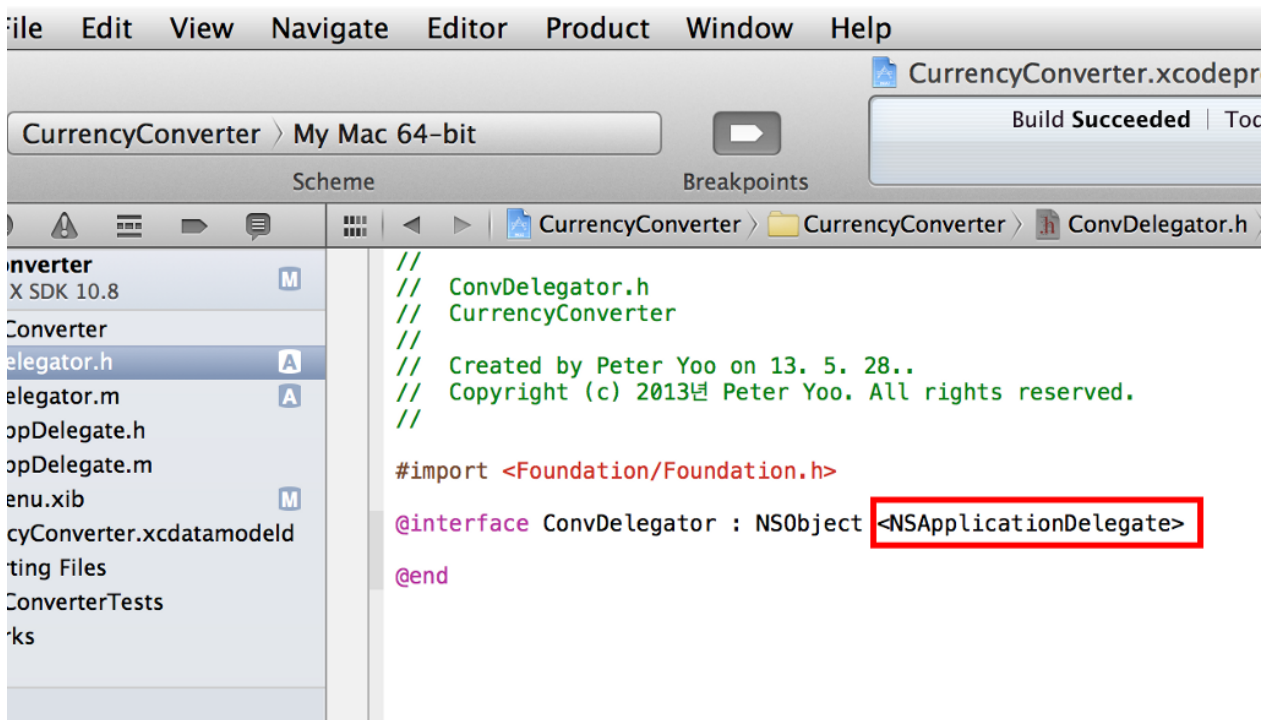
오호원하는 Class 의내용을담을파일이새로생성되었습니다. 어라? 그런데원하는 Folder 바깥쪽에있군요. 관리의편의성을위해 2 개파일을선택한후바로아래쪽의 Folder 로 Drag 해줍니다.



위와같은파일의목록이보이게되면일단 Class 파일의생성과배치는성공적으로진행이된겁니다.



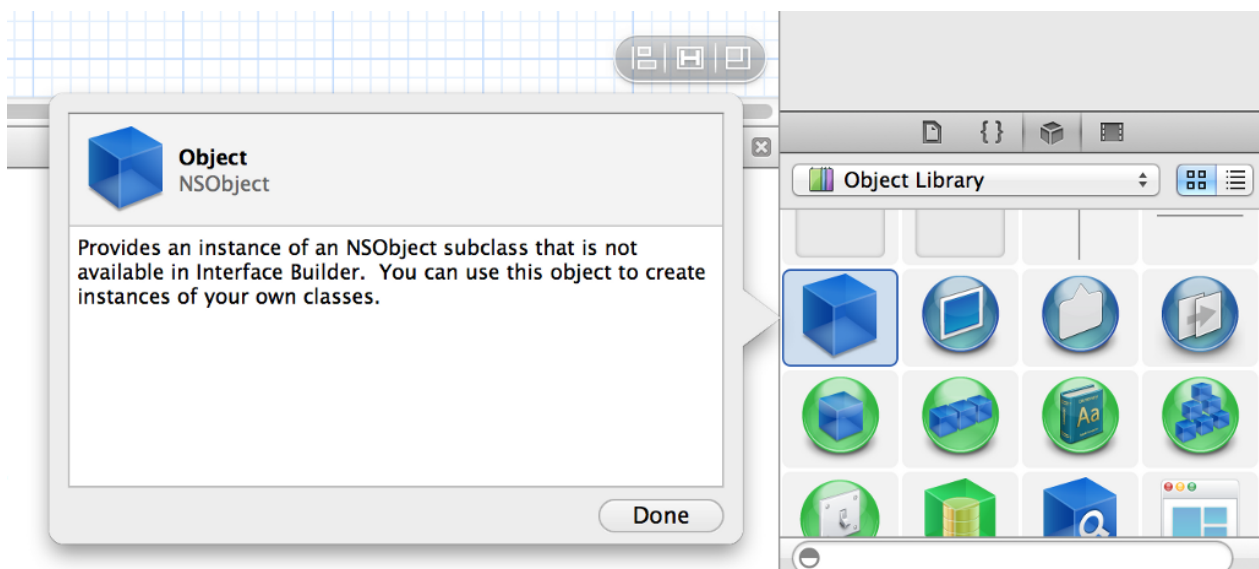
생성된파일중 .h 라고되어있는 header 파일을보겠습니다. @interface 부분이선언되어있는게보이는군요. 이제이선언부에 protocol 관련내용을조금적어주겠습니다.



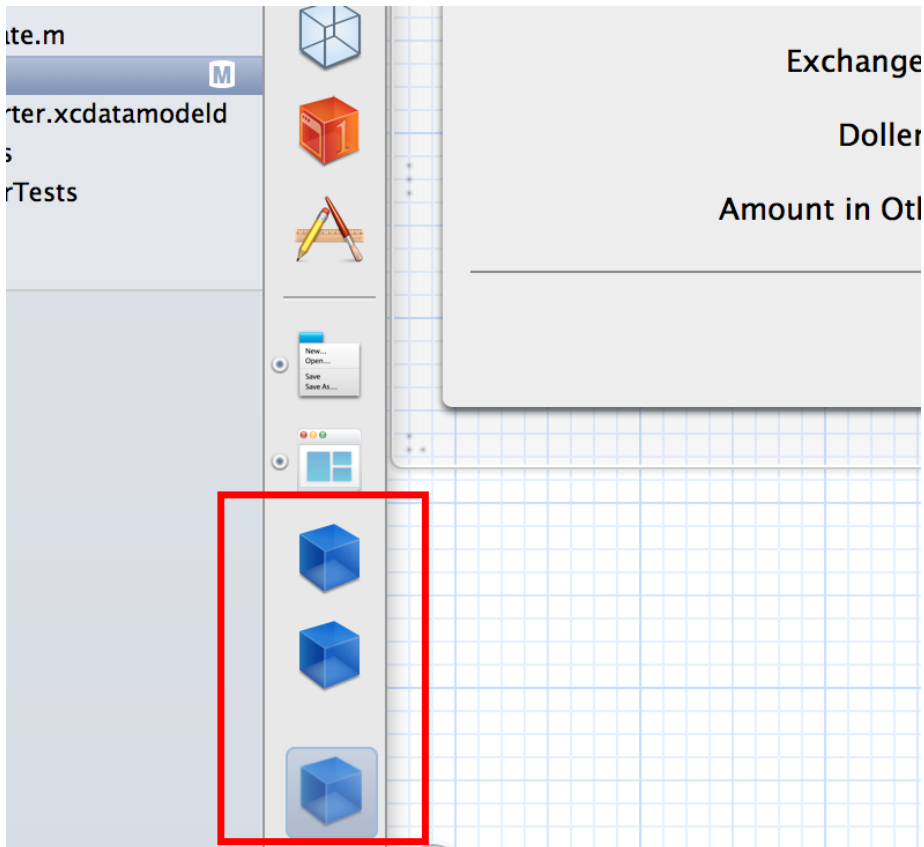
위와같이 **NSApplicationDelegate** 부분을추가해줍니다. 이작업의의미는 나는이제 *Delegate* 를받을수있어요' 라는의미라느군요(아.. 출처가기억이.....)

자이제 Class 관련부분에서코딩으로해야할부분은끝났습니다. 이제다음과정으로진행해볼까요?

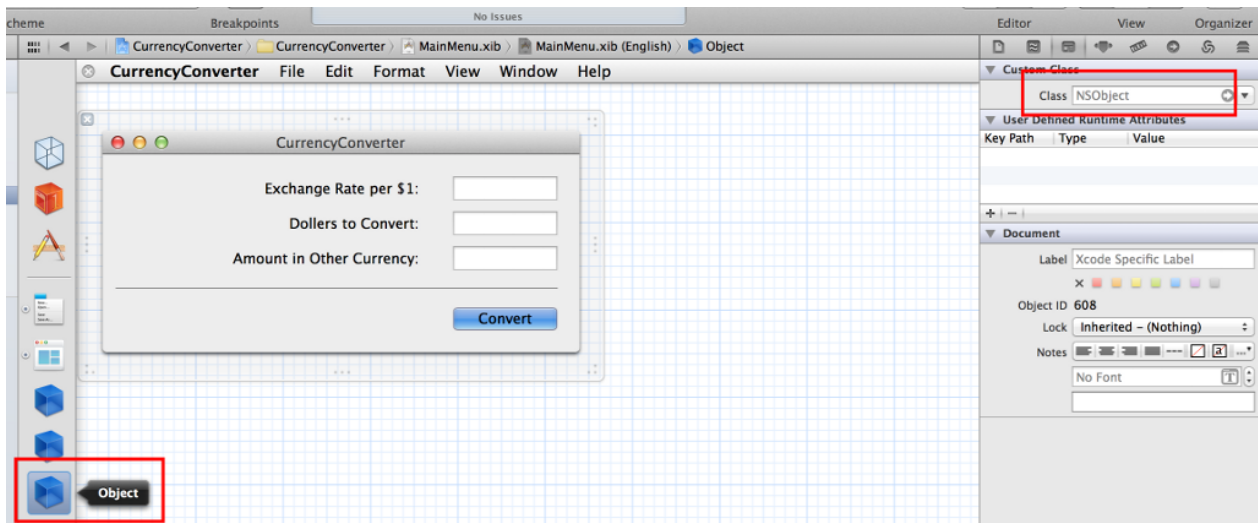
Interface Builder 에서 instance Object 설정하기



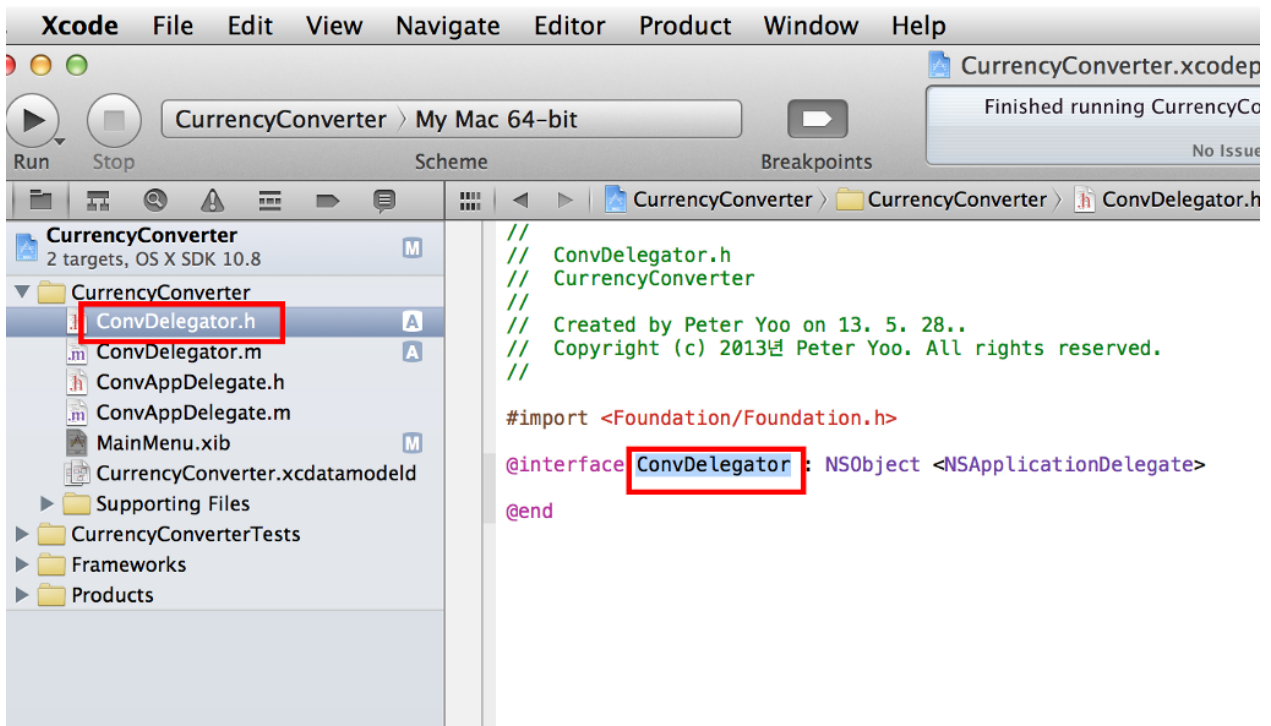
Interface Builder 화면에서 NSObject::Object를선택합니다.



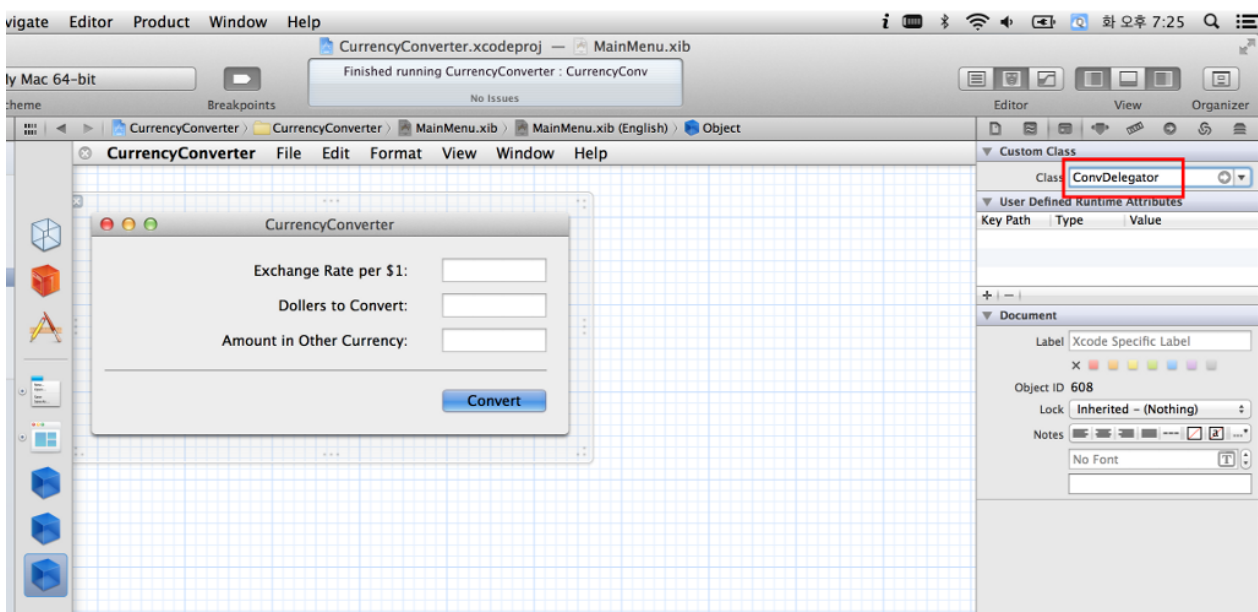
화면처럼선택한 Object 를 Interface Builder 의팔레트에엮어줍니다.



엮은 Object 에대한기본적인내용을잠시살펴보겠습니다. 일단 instance Object 의현재이름은 **Object** 입니다. 그리고현재 Delegate 되어있는 Class 는없다는걸확인할수있습니다. 이제 Delegate 할 Class 를찾으러가볼까요.



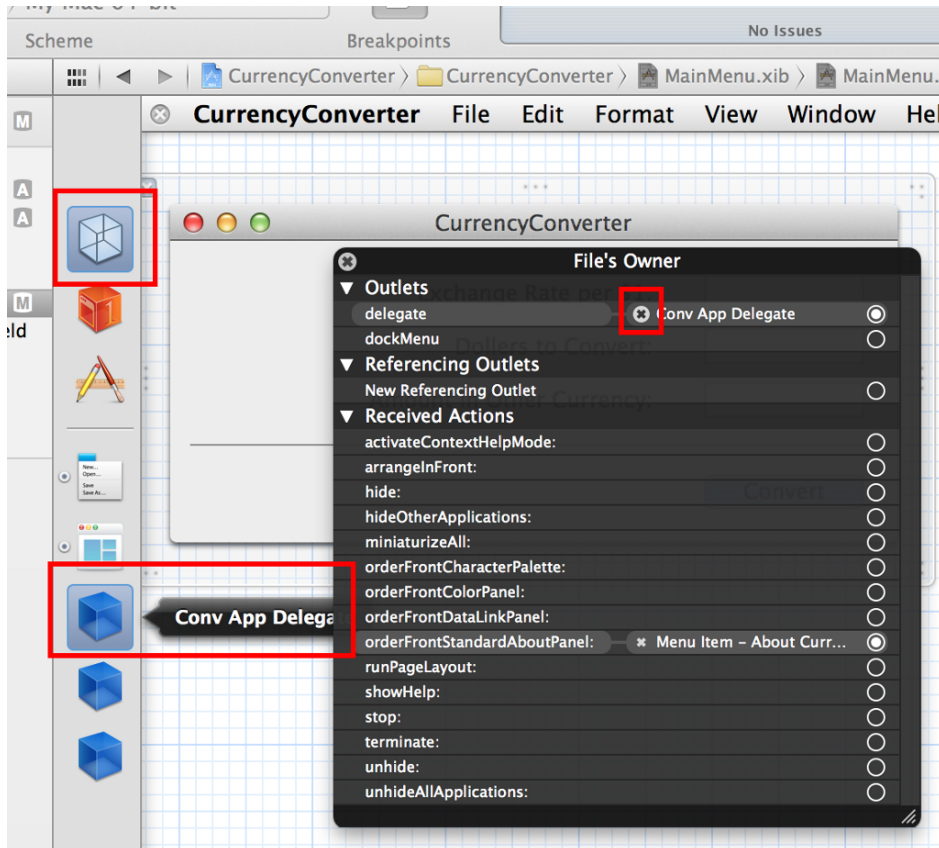
아까생성한 Class 의 header 파일을보겠습니다. 오호라... ConvDelegator 라는게이미있군요. 저너석을쓸겁니다.



위의화면에서봤던 **Object** 라는이름의 instance object 가가리키는건 ConvDelegator 라고지정 을해보겠습니다. 자이제 instance object 는 interface 를가질수있게되었습니다 :D

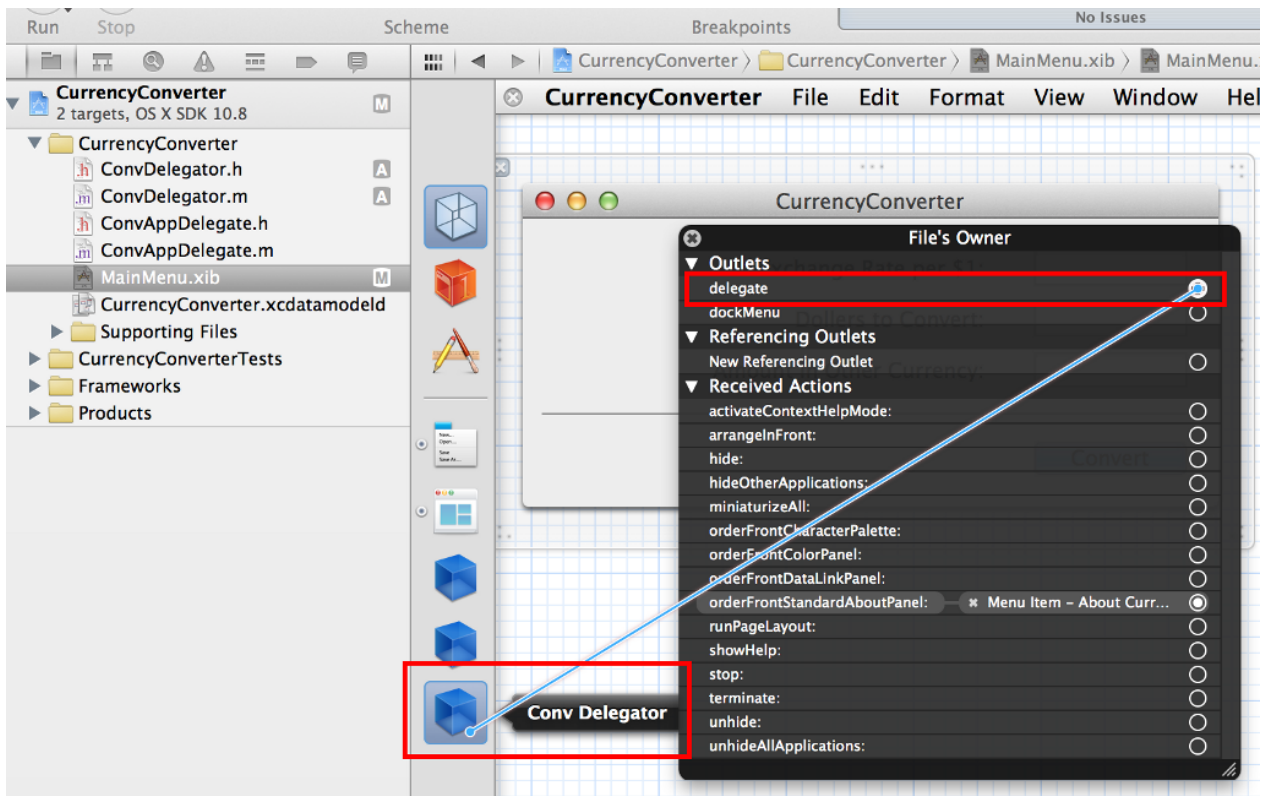
Windows의 Delegate 재지정

이제부터는 기존에 지정되어 있던 Delegator 에 대한 지정을 해제하고 새로 만든 instance object 로 새로운 Delegate Mapping 을 진행해보겠습니다.

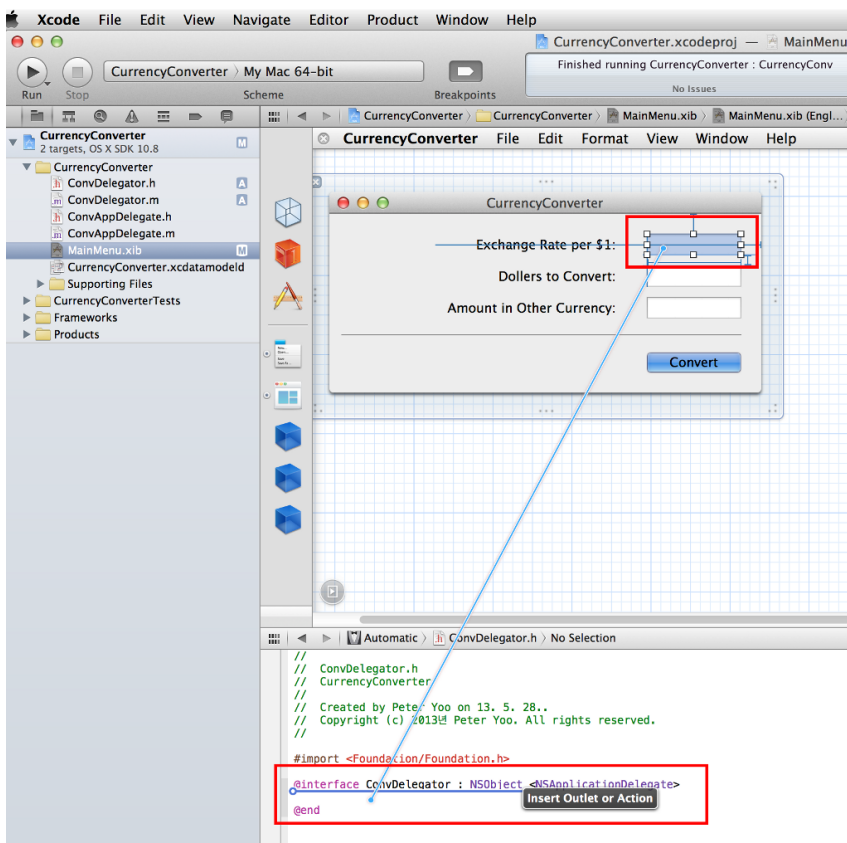


화면에 보이 다시 팔레트의 제일 위쪽에는 **File's Owner** 라는게 있습니다. 여기서 popup 메뉴를 불러보도록 하죠. 그러면 현재 xib 파일의 Delegate 로 지정된 instance object 를 알 수 있습니다. 오호.. 운 좋게도 한 화면에 같이 있었군요. 해당되는 객체 위에 **Conv App Delegate** 라는 문구가 표시되는 게 보입니다.^[1] 사실 잘 찾아보면 **ConvAppDelegate.h** 파일 안에 있는 interface 부분이 바로 이 부분입니다. 이제 조금 감이 오기 시작하시나요?

일단 지금 진행하는 부분에서는 지정되어 있는 Delegate 를 해제하는 것이 첫 번째 과정이어야 하기 때문에 popup 메뉴에 강조된 것처럼 **X** 표시를 눌러서 지정되어 있는 Delegate 를 삭제해줍니다.

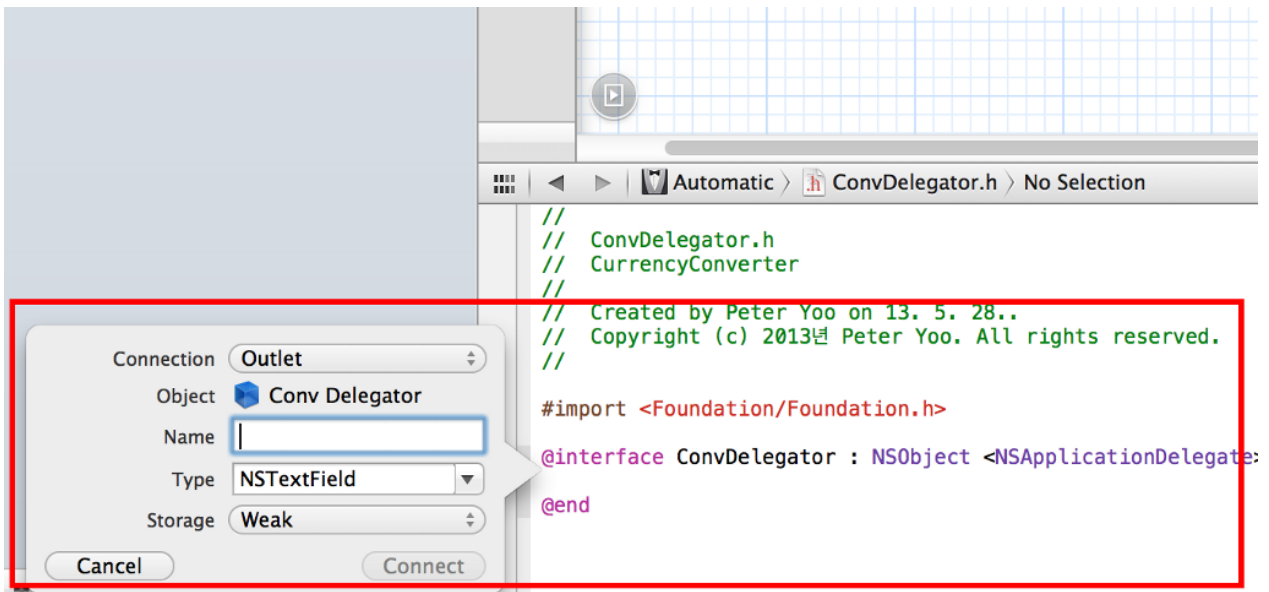


Delegate 를삭제했다면새로만들어진 instance object 를다시재지정해줘야겠죠. 화면대로 popup 메뉴에있는 + 버튼을마우스로눌러서아래쪽에우리가새로만든 **ConvDelegator** instance object 로지정해줍니다. 이렇게하면이제우리가만든 instance object 가현재 xib 파일의 Delegate 객체로연결이되는겁니다. 확인은어떻게하냐구요?



일단아래쪽의 **Assistant Editor** 에우리가만든 Class 에관련된 **ConvDelegator.h** 파일을표시합니다. 그다음 Windows 안에서원하는객체를하나마음속으로고른후 Ctrl 버튼을누르면

서아래쪽으로 획! Drag 해주는거죠. 이야... 뭔가표시가나오네요? 참고로 제대로 **Delegate** 설정이되어있지않다면 **Drag** 했을때삽입을할수있는위같은화면을볼수없습니다.



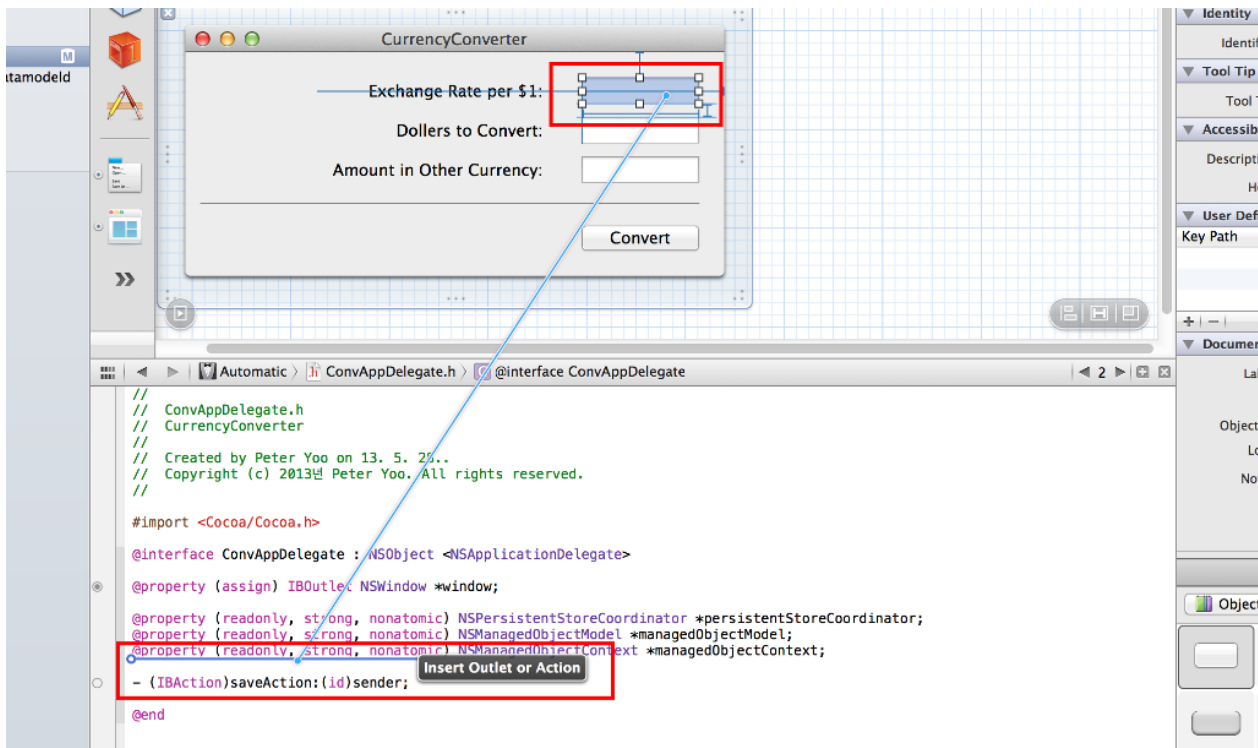
mouse 의커서키를떼면다음과같은 **Outlet** 또는 **Action** 을삽입할수있는 Help Window 가뜩니다. 이렇게까지되면내가만든 Class 를 Delegate 로등록해서사용할수있게되겠죠?

위의과정에서잘본것처럼 xib 파일은별개로동작하는것도아니며분명필요한만큼의 설정값을들가지고있다는사실을아실수있을겁니다. 지금한작업은기본적으로는전혀필요가없는작업일수있겠습니다만, 어떤식으로구조가돌아가는지를알기위해서는필요한과정이라생각합니다. 이것으로 palate 와 interface builder, 그리고 Delegate 에대한부분이조금은명확해진것같습니다.

Xcode 4 의 Interface Builder 에서 IBOutlet 코드생성하기

자.. 이제다시원점으로돌아와서원래해야할일을하도록하겠습니다. (이바로위에서 했던 Delegate는작업안한걸로할겁니다. 그러니코드를원복해주세요) Interface Builder 에서 만들어진 UI 를실제코드와연결시킴으로서나중에 UI 를사용가능하게만드는것이이 부분에서진행해야할부분입니다.

일단 MainMenu.xib 파일을선택해서 Interface Builder를열도록합니다. 그리고화면의아래쪽에는 **Assistant Editor** 를준비해주시구요. 에디터에는 **ConvAppDelegate.h** 파일을열어주세요. 자... 이제진행해야할부분은간단합니다... 가볍게끌어다놓기만하면되는거죠 :D



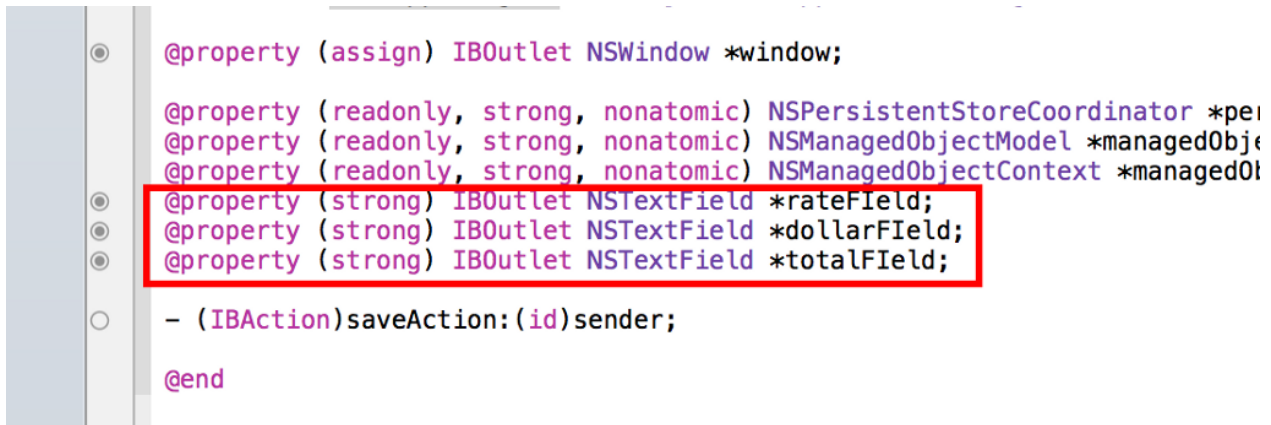
Ctrl 키를 누른 상태에서 Windows 의 첫번째 TextField 를 Drag 해서 위 의 스샷에서 보이는 것처럼 아래쪽의 **Assistant Editor** 쪽으로 끌어다줍니다. 그럼 삽입이 가능한 부분에서 어떻게 할 수 있는 표시가 뜨게 됩니다. 이제 mouse 의 클릭을 해제해 보도록 할까요?



오호 왼쪽에 Help Window 가 떴습니다. IBOutlet 코드를 편하게 삽입하기 위한 도움창 같은 건데요. 우리는 Action 이 아닌 Outlet 을 삽입할 거라서 Connection 은 **Outlet** 으로 설정해 줍니다(기본값이죠). 위의 TextField 를 프로그램 코드에서 사용하기 위한 이름을 정할 수 있고 Storage Type 을 결정할 수 있습니다(이전의 Project builder 시절에는 없는 부분입니다만). 여기서 저는 Strong 을 선택하도록 하겠습니다. 아래의 코드에서 확인할 수 있겠지만 사실 이렇게 지정되는 값은 @property 로 지정됩니다. 좀 더 자세한 내용은 아래쪽에서 다루기로 하겠습니다.

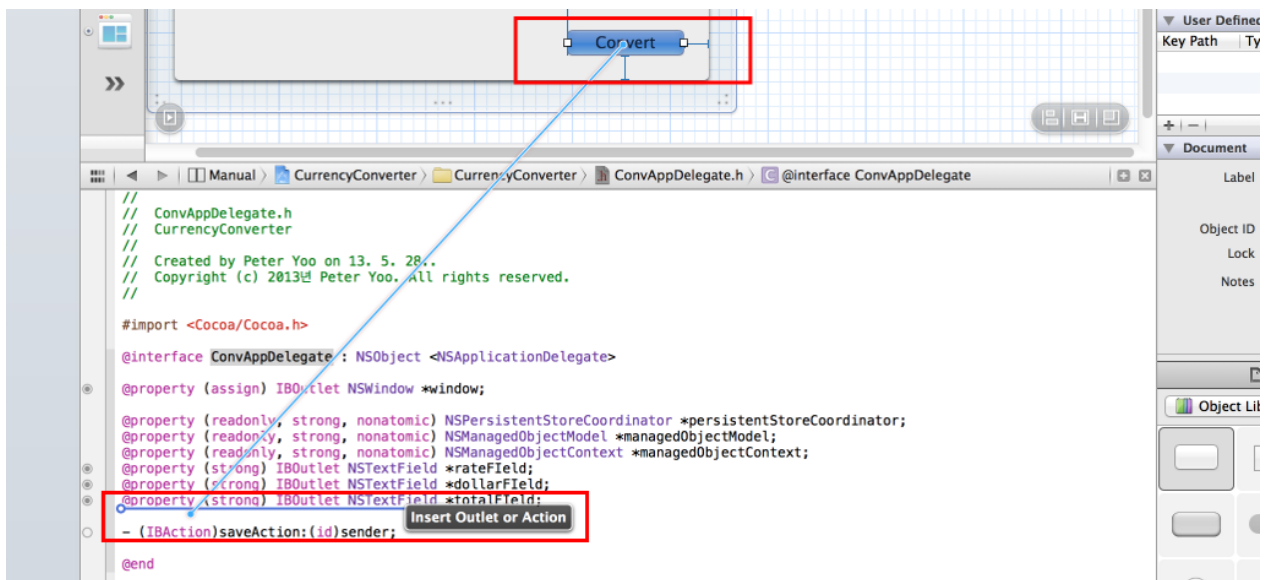
입력값이 정리되면 Connect 버튼을 눌러서 코드를 삽입하도록 하겠습니다.

위와 같은 방법으로 3개의 TextField 를 전부 삽입해 주시면 되겠습니다. 그러면 코드 는 다음과 같은 모양이 될 겁니다.

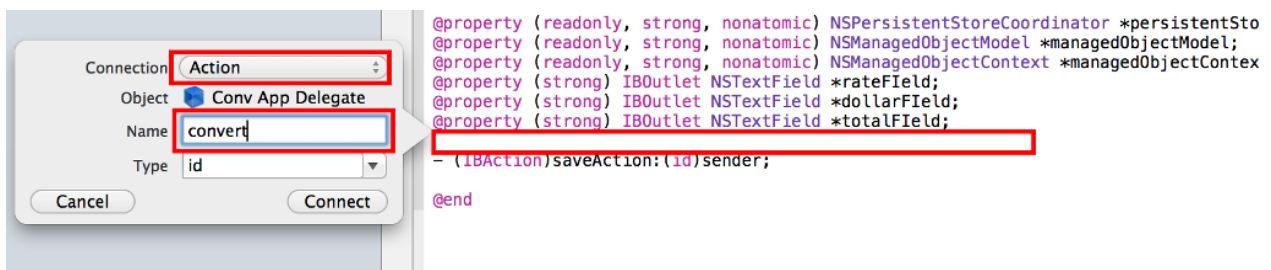


보이시나요? 이 코드의 모양을 잘 보고 TextField 를 입력할 때 이름을 지정해주시는 부분을 참고해주시면 되겠습니다.

자 이제 Main Window 에 있는 Connect Button 을 연결하는 걸로 하겠습니다. Button 은 Action 으로 연결할 것입니다.



TextField 를 Outlet 코드 삽입할 때처럼 Ctrl 키를 눌러서 삽입을 원하는 곳으로 Drag 합니다. 그럼 Outlet 을 연결할 때와 같은 진행이 됩니다.



이번에 삽입할 것은 **Action** 입니다. Connection 을 **Action** 으로 바꾸고 해당되는 객체를 연결할 이름을 입력합니다. 그 다음 connect 버튼을 눌러줍니다.

이렇게 내용을 삽입하고 나서 코드를 약간 정리하면 다음과 같은 모양이 될 겁니다.

```

// ConvAppDelegate.h
// CurrencyConverter
//
// Created by Peter Yoo on 13. 5. 28..
// Copyright (c) 2013년 Peter Yoo. All rights reserved.
//

#import <Cocoa/Cocoa.h>

@interface ConvAppDelegate : NSObject <NSApplicationDelegate>

    @property (assign) IBOutlet NSWindow *window;

    @property (readonly, strong, nonatomic) NSPersistentStoreCoordinator *persistentStoreCoordinator;
    @property (readonly, strong, nonatomic) NSManagedObjectModel *managedObjectModel;
    @property (readonly, strong, nonatomic) NSManagedObjectContext *managedObjectContext;

    @property (strong) IBOutlet NSTextField *rateField;
    @property (strong) IBOutlet NSTextField *dollarField;
    @property (strong) IBOutlet NSTextField *totalField;
    - (IBAction)convert:(id)sender;

    - (IBAction)saveAction:(id)sender;

@end

```

새로삽입한코드를조금보기 좋게 위쪽이랑 띄워놨습니다. 화면에 보이는 것처럼 정리
가된다면 기본코드는 성공입니다.

이번부분에서삽입된코드는 다음과 같습니다.

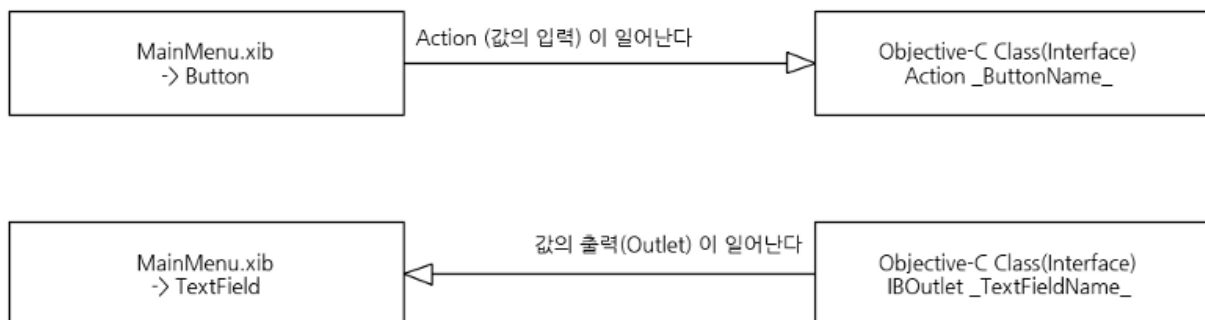
```

@property (strong) IBOutlet NSTextField *rateField;
@property (strong) IBOutlet NSTextField *dollarField;
@property (strong) IBOutlet NSTextField *totalField;
- (IBAction)Convvert:(id) sender;

```

대체 **Outlet** 과 **Action**은 뭔가요?

Window의 입장에서 Outlet 은 출력, Action은 입력입니다. TextField 또는 Button 등은 값을 출력하
고 입력할 수 있죠. 그렇기 때문에 Outlet과 Action 양쪽 다 이용이 가능합니다. 굳이 흐름을 보
자면 아래처럼 되겠군요.



코드에서 값을 Windows 로 보내는 경우에는 **Outlet**, Windows 에서 코드로 값을 보내는 경우
는 **Action** 이라고 생각하면 됩니다. 그리어렵지는 않습니다.

간단한 예를 들어보자면 Windows 에 배치한 객체가 Label 인 경우는 Outlet 을 삽입하려 할 때
아예 **Outlet** 만 선택할 수 있도록 되어 있습니다. Label 은 입력을 받을 방법이 없기 때문이죠.

궁금하신 분들은 실험을 해보시는 것도 괜찮을 거 같습니다 :D

storage는 대체 어떤 의미를 가지는 건가요?

자세한 설명은 xcode 4.2 ARC retain, weak, strong 부분을 참고해 주시기 바랍니다.

일단 위의 부분에서 일부만 발췌하면 다음과 같은 부분이 있는 걸 볼 수 있습니다.

▶ strong은 그냥 retain과 같은 의미라고 생각하면 된다. 그러니까 기존에 property 선언 시 사용했던 retain 키워드는 모두 strong으로 교체 해주면 된다.

▶ weak의 경우 조금 설명이 더 붙어 있는데 일단 weak는 기존의 assign과 같은 개념이지만 dealloc 되었을 경우 대기포인터로 남아 있지 않고 바로 nil값으로 처리된다는 점이 다르다고 한다.

▶ 일단 깊숙한 부분에 대해서는 아직 이해가 부족한 만큼 그냥 retain -> strong, assign -> weak로 대체해서 사용하면 될 것 같다.

오호.. 새로 추가된 부분인가 봅니다. 사실 저는 그냥 strong 으로 사용하는 게 편할 거 같기는 합니다만.... 객체의 해제라는 점에서는 weak 가 더 유용한 거 같군요 :D

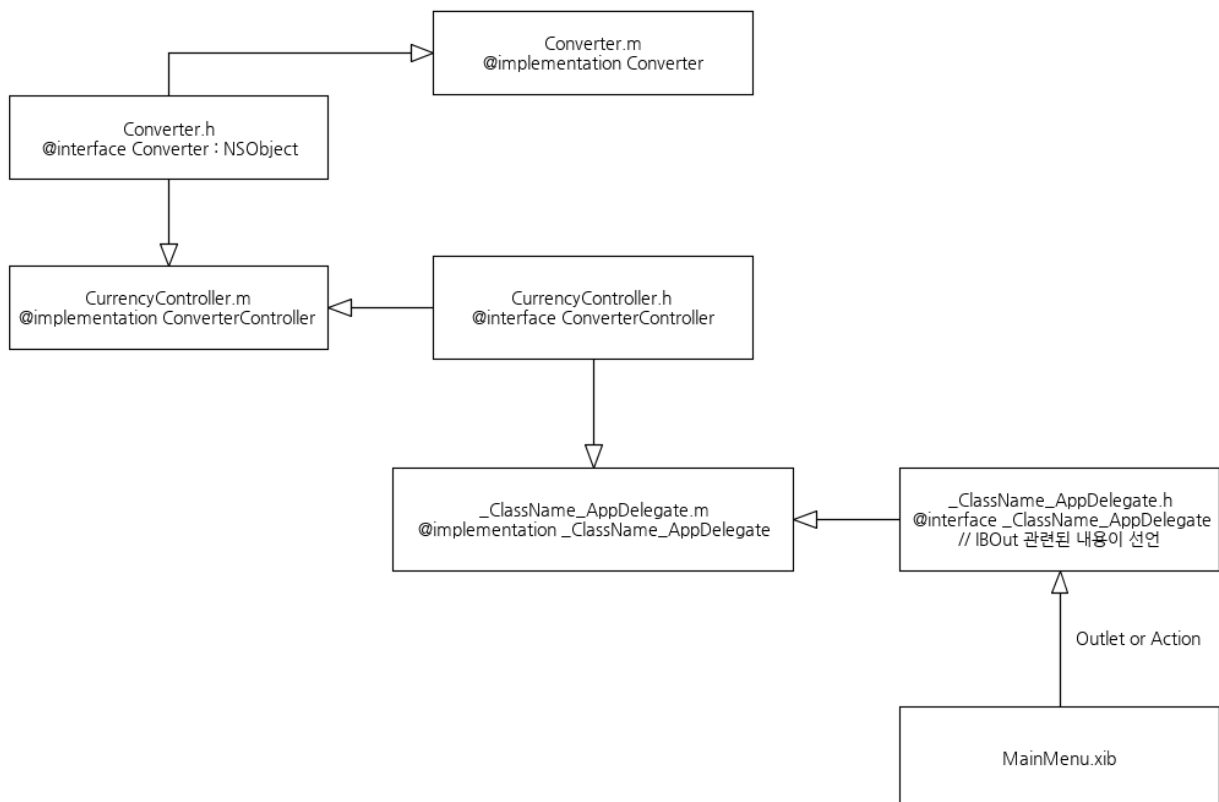
간단하게 살펴보는 Objective-C 의 기호들

Objective-C 기호와 용어 정리 페이지를 참고해 주시기 바랍니다.

Model Class::Converter Class 작성하기

자이제이 프로젝트에서 Model 의 역할을 하게 될 Converter Class 를 작성해보도록 하겠습니다. (이제 스샷이 좀 줄겠군요. 만쵸!)

아까 위 부분에서 Diagram 그렸을 때 여러 개의 파일이 있는 걸 보셨을 거라 생각합니다. 굳이 다시 표시하는 게 어렵지는 않으니 잠시 끌어내려와 볼까요?



이 Diagram 의 바로 위쪽에 아래와 같은 정의를 내려놓은 것도 기억하실지 모르겠습니다.

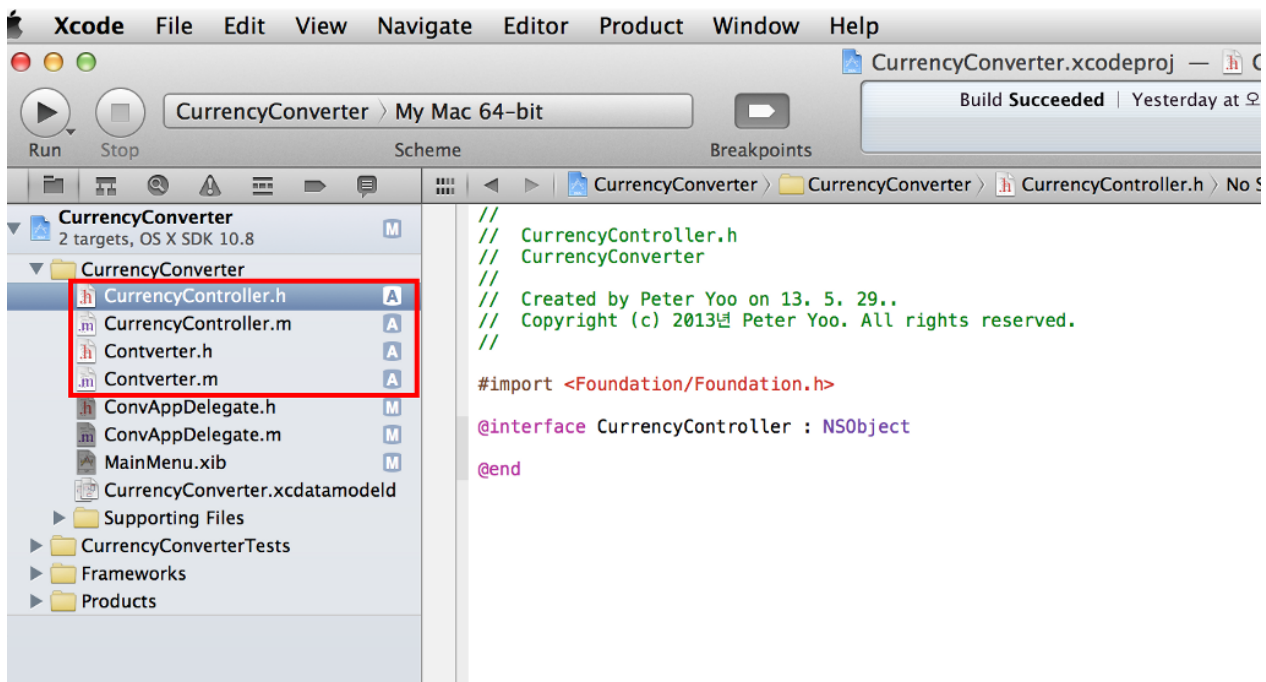
- Class 명은 **Conv** 로정의한다
- Converter :: 통화량을계산해서값을반환한다. Model 에해당됨
- ConverterController :: Converter 객체와 AppDelegate 사이의활동을조정하는클래스. Controller 에해당됨
- ConvAppDelegate :: xib 파일에서정의된 UI 객체들에대한동작을정의하는클래스. View 에 해당됨.
- xib 파일 :: AppDelegate 에 IBOutlet, 및 Action 으로연결되는사용자 UI

자그럼 Converter 를만든다는건데이터를처리하는 Model 을작성한다는의미가되겠습니다. 위쪽에서 MVC 에대한설명을했을때각역할에대한설명을한걸기억하십시오.

Controller 는 View 에서데이터를받아서그걸어딘가의 Model 로전달하는역할만하는거죠. 그렇다면 Model 에서받은데이터에대한처리를다해서결과를돌려주는부분까지의작업을가지고가야하는걸쉽게짐작할수있게됩니다.

이제부터원래문서에있는내용을기초로 Model 에해당하는 Converter Class 를작성하도록하겠습니다. Converter Class 를작업한다면당연히 Converter.* 파일에서작업을진행해야겠죠 :D

파일이있을리가없으니위에서설명한요령대로 Converter Class 파일을추가해서작업을진행할준비를합니다. 작업하는감에 Controller Class 의작업에사용할 Class 까지같이생성하는걸로하겠습니다. 제대로파일이생성이되었다면 **Navigator area** 에다음과같은내용이표시되어야합니다.



작업을시작하도록할까요?

Converter Class 의기본골격

Converter.h 파일을열어서코드를삽입할준비를합니다. 기본적으로다음과같은내용이 Class 생성시이미들어가있습니다.

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

@interface Converter : NSObject

@end
```

Converter Class 에 Method 추가하기

이제기존의코드를다음과같이수정해보겠습니다.

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

@interface Converter : NSObject
{
}

+ (float)convertAmount:(float)amt atRate:(float)rate;
@end
```

수정된코드에서는 2가지가크게변경되었습니다.

1. import 부분에서 Foundation 이 Cocoa 로변경되었습니다
2. convertAmount 인스턴스메서드가추가되었습니다.

-(minus)기호로되어있는부분은 instance method, +(plus) 기호로선언되는부분은 Class method 입니다. 두개간의차이는 여기 ^[2] 링크를보면좀더명확하게알수있습니다. 여기서는간단하게만설명하기로하죠.

Smalltalk 에서유산을받은 Objective-C 는두가지타입의 method 를가지고있습니다. instance 와 object 가대상이되는타입이되는데요..... 이두가지는개념상어떤차이가있을까요?

- instance method :: instance method 는 Class 를이용해서객체(Object)를생성한후에사용할수있는메서드
- class method :: Class method 는 Class 를이용해서객체를생성하지않고도사용할수있는메서드

일단위의코드에서추가한메서드는 class 메서드입니다. 때문에추가한메서드는객체생성없이바로사용할수있습니다.

별도로내부에 instance 변수에접근할필요없이계산결과만반환하기때문에굳이 instance 메서드를사용해야할필요는없습니다.

새로선언된메서드에대한내용

이제선언된부분에대한내용을살펴보도록하겠습니다. 선언부는다음과같이표현할수있습니다.

▶ convertAmount:atRate:

이선언의의미를살펴보면다음과같습니다.

1. convertAmount 에서인자를취합니다. 인자는 Float 형.
2. atRate 에서인자를취합니다. 인자는 Float 형
3. Float 타입으로결과를변환합니다.

이제메서드의 interface를선언했으니실제메서드의동작을정의하도록하겠습니다.

class 메서드의 동작구현을 위한 @implementation 부분확인하기

위에서 선언한 메서드의 interface 에 따라 선언한 메서드의 동작을 구현해보도록 하겠습니다.

일단 **Converter.m** 파일을 편집기로 열도록 하겠습니다. 그러면 아래와 같이 되어있는 기본 틀을 확인할 수 있습니다.

```
#import "Converter.h"

@implementation Converter

@end
```

이제부터 여기에 내용을 채워 넣기로 하겠습니다.

class 메서드의 동작구현

위에서 본 @implementation 코드부분을 다음과 같이 변경합니다.

```
#import "Converter.h"

@implementation Converter

+ (float)convertAmount:(float)amt atRate:(float)rate;

{
    return (amt*rate);
}

@end
```

잘보시면 별로 어려울만한 부분은 없습니다. 입력받은 amt 와 rate 를 곱해서 반환하는게 작업내용의 전부입니다.

View Class::사용자 Interface 에 대한 내용구현

이미 우리는 위쪽에서 xib 파일의 Window 에있는 **Currency** Button 에 대한 Action 을 정의한 것을 기억하실 겁니다. 이제 그 내용에 살을 좀 붙여보도록 하겠습니다.

원문에서는 ConverterController Class 가 View(xib파일)에 바로 연결되는 구조였기 때문에 지금 부터 나오는 코드는 원문과 100% 동일하지는 않게 될 거 같습니다.

기존 코드에 대한 분석

일단 Action 에 대해서 언급해 놓은 원문의 코드를 보도록 하겠습니다.

```
- (IBAction)convert:(id) sender
{
    float rate, amt, total;
    amt = [dollerField floatValue];
    rate = [rateField floatValue];
    total = [converter convertAmount:amt atRate:rate];
    [totalField setFloatValue:total];
    [rateField selectText:self];
}
```

```
}
```

물론위의코드를사용하기위해서는 Converter.h 파일을위의파일의상단부분에서 **#import** 해줘야합니다.

자.. 이제원문에서작성된코드를분석해야필요한부분을바꿔서쓸수가있겠죠? 원문에서위의내용은 **ConverterController.m** 파일의 **@implementation** 부분에선언되어있습니다.

1. 일단 rate, amt, total 이라는 float 형변수를선언합니다. 전형적인 **C Language Style** 임.
2. amt 라는변수에 IBOutlet 으로이미지정한 dollarField 객체의 floatValue 메서드를실행한결과를받아서넣는다.
3. rate 라는변수에 IBOutlet 으로이미지정한 rateField 객체의 floatValue 메서드를실행한결과를받아서넣는다.
4. total 이라는변수에위에서선언한클래스인 converter 클래스의 instance 메서드 **convertAmount:atRate:** 를실행한결과를넣는다. 메서드의인수로는 amt 와 rate 를사용한다.
5. IBOutlet 으로이미선언해놓은 totalField 객체에 **setFloatValue:** 메서드로 total 변수를인수로삼아값을집어넣는다.
6. rateField 객체에 **selectText:** 메서드를 **self** 를이용해서 rateField 객체스스로를인수로삼아실행한다.

대략코드의내용은이와같이나오는거같고.... 이제부터해야할것은이구조를어떻게바꿔야의도한바의 MVC 로변경하고코드를수정해야하는지에대한 plan을세워야하겠죠.

바꿀코드의구조는어떻게되어야할까요?

MVC 에의하면 View 에서 Data를얻고다시 View 로보내는작업을 Controller 에서해야한다고되어있습니다. 일단위쪽의 Converter Class 에서 Model 에대한부분은작업이이미된상태이니지금부터는 Controller(CurrencyController Class) 와 View(ConvAppDelegate Class) 에대해서만살펴보기로하겠습니다.

Class Struct(Controller)

1. View 에서 Action이호출되었을때연산을진행하고값을처리하는메서드

Class Struct(View)

1. 다른곳에서호출이있을때각객체의값을반환하는메서드
2. 다른곳에서값이넘어왔을때해당되는객체의값을변환하는메서드
3. Action 이일어나는경우 Controller Class 를호출하는메서드

View Class 의구현

위에서다룬대로구조를바꾸려면일단 View Class 부터바꿔야합니다. 새로언급한 3가지요소중 Action 관련된것을제외한나머지메서드들을먼저구현해보기로하겠습니다. 일단 interface 부터구현을하는게맞겠죠? **ConvAppDelegate.h** 파일에새로운코드를선언해보기로하겠습니다.

TextField 의값을얻어와서반환하는메서드구현

각파일에아래와같은내용을작성해넣도록하겠습니다. (위치는눈치껏.. 아시죠?)

```
//ConvAppDelegate.h file
- (float)getDollerField;
- (float)getRateField;

//ConvAppDelegate.m file
- (float)getDollerField
{
    return [_dollerField floatValue];
}

- (float)getRateField
{
    return [_rateField floatValue];
}
```

위의 2개의메서드는요청에따라해당되는 TextField 의값을가져와서반환하는일만진행합니다.

Convert 버튼을눌렀을때에동작을하는내용에대한메서드구현

- **(IBAction)Convert:(id)sender;** 메서드(이미위에서이메서드를추가했었죠) 에대한내용을구현해보겠습니다

```
//ConvAppDelegate.m file

- (IBAction)Convert:(id) sender
{
    float total;
    CurrencyController * CalcObject = [CurrencyController new]
    total = [CurrencyController calculate:self];
    [_totalField setFloatValue:total];
    [_rateField selectText:self];
}
```

위의 Action 메서드에대한구현의내용은 CurrencyController 클래스의 calculate 라는메서드를실행해서그결과값을받은다음결과값을출력해야할곳에출력하는일을진행합니다.

위에서사용된 new 라는메서드(메시지)는 alloc 과 init 를한번에동작시키는걸로이해하시면되겠습니다. 원래부터 Objective-C 에있던기능은아닙니다만.. 추후에추가된것으로보입니다.

이내용을동작시키기위해서는 ConvAppDelegate.m 파일의상단에 **#import "CurrencyConverter.h"** 를넣어줘야하는걸잊지말아주세요.

위의부분에서는정확하게한곳이중요합니다.

```
total = [CurrencyController calculate:self];
```

일단저부분이왜중요하냐면 calculate 메서드에서연산을진행하기위해값을얻어오려면생성되어있는 instance 에서값을얻어와야합니다. 그런데 c 나 c++ 의경우는생성된 instance 의 Pointer 값을넘기게되는데여기서는현재 UI windows 를나타내는 self 를인자로 던져줍니다. 그럼이후 calculate 메서드에서는이결전달받아서객체(instance)의 instance 메서드를실행할수있게됩니다.

자그럼이제실제로계산작업의프로세스를진행할 CurrencyController(Controller Class) 클래스의메서드를구현해보기로하겠습니다. 현재 Convert Action 은 calculate 라는메서드를메시지로사용하기때문에관련된메서드를새로구현해주어야합니다.

Controller Class::CurrencyController 의내용구현

CurrencyController 클래스의내부에서사용될 calculate 메서드의내부동작에서의순서는다음과같습니다.

1. getDollerField 를이용해서 dollerField 의내용을가져온다
2. getRateField 를이용해서 rateField 의내용을가져온다
3. 가져온내용을 Converter Class 의 convertAmount:atRate: 메서드에대입해서결과를받아낸다
4. 받아낸결과값을반환한다.

위의내용을기초로이제각파일에메서드를작성해보겠습니다.

```
//CurrencyController.h
- (float)calculate: (ConvAppDelegate*)MainWindow;

//CurrencyController.m
- (float)calculate: (ConvAppDelegate*)MainWindow
{
    float amount, rate, result;
    amount = [MainWindow getDollerField];
    rate = [MainWindow getRateField];
    result = [Converter convertAmount:amt atRate:rate];
    return result;
}
```

CurrencyController.h 파일의상단에는다음의한줄이추가되어야합니다. Class 에대한정의가필요하기때문이지요.

- **#import ConvAppDelegate.h**

CurrencyController.m 파일의상단에는다음의한줄이추가되어야합니다.

- **#import Converter.h**

CurrencyController 클래스의 calculate 메서드내용분석

내용은그리복잡하지않습니다. View class 에서값을가져온다음그값을 Converter 클래스를이용해서계산하고계산된결과값을 calculate 메서드를호출한객체에게반환합니다. 조금주의할것은위쪽에서설명을이미한번했습니다만.. 아래의부분입니다.

```
//CurrencyController.h
#import ConvAppDelegate.h

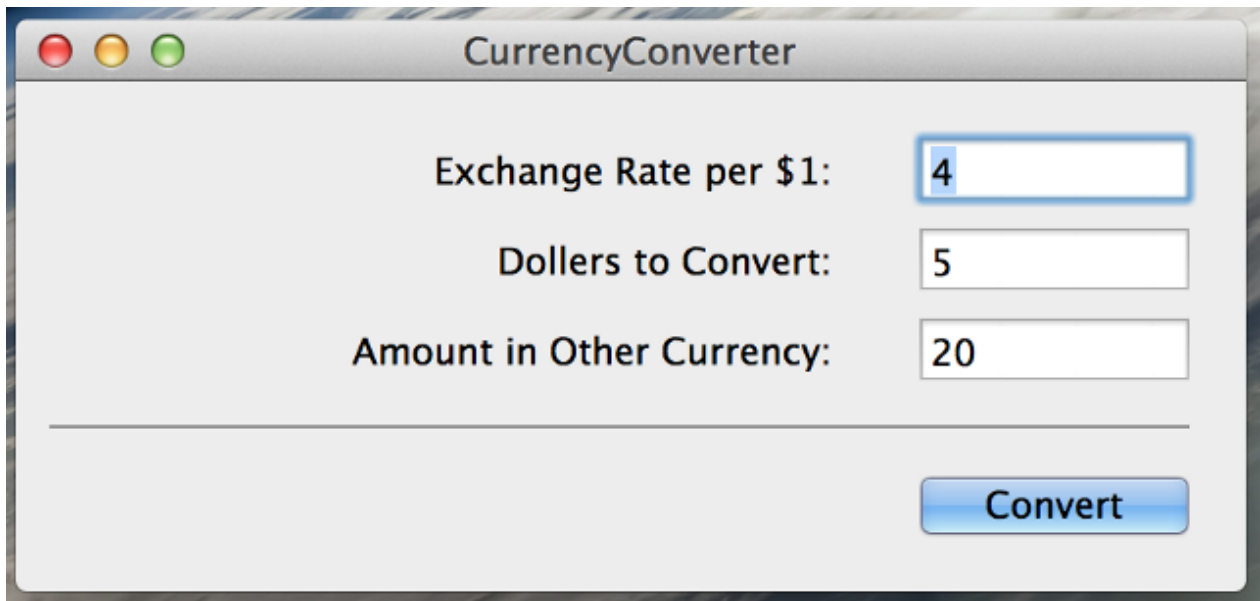
- (float)calculate: (ConvAppDelegate*)MainWindow;
```

위처럼 UI의 Delegate 를담당하는 Class를 import 한다음 receive 되는객체의 Type Casting을정확하게해주면됩니다.

객체의타입을정확하게모르겠는경우는 Action 부분에서 `NSLog([self className]);` 정도로 xcode의 log 에서확인해주시면되겠습니다.

자이제만들어진프로그램을실행해볼까요?

완성된프로그램의실행



이렇게간단한예제프로그램의제작이끝났습니다. 물론이것말고도할수있는작업이 야 많 겠 습 니 다 만.... 나 름 xcode의기본에대해서충실하게배울수있었던기회였네요. (사실이런방식으로설명한책이따로없기도하고...)

모쪼록읽으시는분들께도좋은내용이되었으면 좋 겠 습 니 다.

첨부파일

참고자료

한글자료

- Objective-C 2.0 관련강의 ^[3]
- Objective C Design Pattern 관련한글자료 ^[4]
- Xcode4.2에서스토리보드를사용해보자!! ^[5]
- Xcode 4 PR2 개선점 ^[6]
- Objective-C 2.0 개요 from ADC ^[7]
- Protocol(델리게이트) ^[8]
- Xcode4에서기존프로젝트에 ARC(Automatic Reference Counting) 적용시키는방법 ^[9]
- "Objective-C" NSArray 사용법예제 ^[10]
- NSMutableArray, NSArray, 배열의생성과요소접근, 배열 ^[11]
- Objective-C 기호와용어정리
- 책의내용을참고로한 Objective C for linux 관련문서

영문자료

- XCode 4 Tutorial And Guide ^[12]
- An Introduction to Cocoa(Inside NeXTSTEP Development Environment) ^[13]

Notes

- [1] 제가헛갈렸던부분인데이전의자료에서는 interface builder 에서그냥 UI 를설계한다음이과정이필요했기 때문에별도설명이있었는데 **Xcode 4** 에서는 보시다시피이미프로젝트생성시 xib 파일을 Delegate 해줄 instance object 를미리만들어줍니다. 사실지금하는과정이딱히필요가없기는하죠. 다만공부하는셈치고나머지과정을진행해보면이런구조구나.. 라는걸아실수있을겁니다
- [2] <http://theeye.pe.kr/entry/iPhone-Object-C-Class-Method-VS-Instance-Method>
- [3] <http://seungngil.tistory.com/category/C/Objective%20C>
- [4] <http://tory45.egloos.com/m/5179484>
- [5] <http://red-submarine.tistory.com/m/post/view/id/25>
- [6] <http://ebcbn.tistory.com/96>
- [7] <http://www.apple.com/kr/developer/leopard/overview/objectivec2.html>
- [8] <http://blog.naver.com/PostView.nhn?blogId=lcblue&logNo=100172517925>
- [9] <http://blesseddeveloper.tistory.com/12>
- [10] <http://nolran.tistory.com/6>
- [11] <http://blog.naver.com/PostView.nhn?blogId=pooh4880&logNo=100127603084&categoryNo=23&viewDate=¤tPage=1&listtype=0>
- [12] <http://maniacdev.com/xcode-4-tutorial-and-guide>
- [13] <http://www.mactech.com/articles/mactech/Vol.16/16.05/HotChocolate/index.html>

ObjectiveC Linux

책의내용을참고로한 Objective C for linux 관련문서

개요

이문서는프로그래밍오브젝티브 C 2.0^[1] 책을보면서공부하는내용을나름대로정리하며만드는문서입니다. Linux 기반에서내용을진행하는것을전제로하고있으니문서를읽으시는분들은참고해주시기바랍니다.

기본조건

- Gentoo linux (정말간단하군요 :D)

기초환경준비

gentoo linux 에서 Objective-C (이하 objc) 를공부하기위해서는다음의조건이만족되어야합니다.

- 현재설치되어있는 gcc 에서 objc 를지원하는가.
- Foundation 을사용하기위한 gnustep-base 가설치되어있는가.

이제부터위의 2개항목을확인하는법을간단하게알아보도록하겠습니다.

gentoo 에서 gcc 의 objc 지원여부확인

아시는분들은아시겠습니까만.. gentoo 는 portage 로관리되고있죠. 그럼제경우를한번에로들어보겠습니다.

```
localhost ~ # emerge -pv sys-devel/gcc
```

These are the packages that would be merged, in order:

Calculating dependencies... **done!**

```
[ebuild R ] sys-devel/gcc-4.6.3:4.6 USE="cxx fortran gcj gtk mudflap (multilib) nls nptl objc objc++ objc-gc openmp (-altivec) -doc (-fixed-point) -graphite (-hardened) (-libssp) -multislot -nopie -noss { -test } -vanilla (-bootstrap%) (-build%) (-go%)" 24 kB
```

이야.. gentoo 를스시는분들이라면익숙한화면입니다. 제경우는 USE Flag 에 "**objc objc++ objc-gc**" 이 3개가포함되어있군요. 이중 **objc-gc** 는조금중요한데이 FLAG 를사용하지않으면나중제가비지컬렉터^[2]를사용할수없다고합니다. 고로꼭넣어줘야겠죠? **objc++**FLAG 를사용하지않으면 Objective-C 내부에서 c++ 라이브러리를사용할수가없다고합니다. 일단저 FLAG가들어가있으면상관이없구요.... FLAG 가포함되어있지않다면포함해서 compile을진행해주시기바랍니다.(gentoo의 portage 사용법에대해서는따로설명은하지않겠습니다.)

gnustep-base 의 설치

간단합니다.

```
localhost ~ # emerge gnustep-base
```

아마도저렇게하면아래와같이 2개정도의패키지가설치될겁니다.

- gnustep-make
- gnustep-base

gnustep-base 는이후다루게될 Foundation Framework 을가지고있는패키지로서이게없으면대부분의예제는사용불능상태라고보시면되겠습니다. 가볍게설치해줍니다.

gnustep-base 의 설치후작업

gnustep-base 를설치하고나면터미널을새로열때마다다음과같은메세지를보게됩니다.

```
2013-05-31 06:09:45.685 gdnc[9632] File NSBundle.m: 2600. In -[NSBundle
localizedStringForKey:value:table:] Localisation file
/usr/lib64/GNUstep/Libraries/gnustep-base/Versions/1.24/Resources/Korean.lproj/Localizable.strings
not in portable encoding so I'm using the default encoding for the
current system, which may not display messages correctly.
The file should be ASCII (using \U escapes for unicode characters) or
Unicode (UTF16 or UTF8) with a leading byte-order-marker.

2013-05-31 15:09:45.685 gdnc[9632] File NSDictionary.m: 676. In
-[NSDictionary initWithContentsOfFile:] Contents of file
'/usr/lib64/GNUstep/Libraries/gnustep-base/Versions/1.24/Resources/Languages/Korean'
does not contain a dictionary
localhost ~ #
```

어라뭐가없다네요?

```
/usr/lib64/GNUstep/Libraries/gnustep-base/Versions/1.24/Resources/Languages/Korean
```

이디렉토리가없다는데간단하게해결을보도록하겠습니다.

```
localhost ~ # cd
/usr/lib64/GNUstep/Libraries/gnustep-base/Versions/1.24/Resources/Languages
localhost Languages # cp -R English Korean
```

이이후로는경고따위는나오지않습니다 :D

First Application : Programming is fun!

이제부터 linux상에서첫번째프로그램을 compile 해보도록하겠습니다.

프로그램의실행조건은다음과같습니다.

 **Programming is fun!** 이라는문자열을화면에출력하기

입력해야하는예제코드는다음과같습니다.

```
/*
 *
 * 첫번째 예제 프로그램
 * Programming is fun! 이라는 문자열을 화면에 출력하기
```

```

*
*/

#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    NSLog (@@"Programming is fun!");

    [pool drain];
    return 0;
}

```

모르는게좀보이기는? 합니다만... 대략구조는그리복잡하지않은거같습니다.

위의파일을 **example_01.m** 이라는이름으로저장하겠습니다.

이제이파일을 compile 하는과정을자동으로진행하기위한 **Makefile** 을만들어보도록하겠습니다.

Makefile 이야워낙에유명하니별로설명을할필요는없겠죠?

```

# example_01.m 을 위한 makefile 예제

.SUFFIXES : .c .o .m

CC = gcc
EXAMPLE_NUMBER = 01

LIBS = `gnustep-config --base-libs`
OBJCFLAGS = `gnustep-config --objc-flags`

OBJS = example_$(EXAMPLE_NUMBER).o
SRCS = example_$(EXAMPLE_NUMBER).m

TARGET = example_$(EXAMPLE_NUMBER)

all : $(TARGET)

$(TARGET) : $(OBJS)
    $(CC) -o $@ $(OBJS) $(LIBS)

clean :
    rm -rf $(OBJS) $(TARGET) core

distclean :
    rm -rf $(OBJS) $(TARGET) core *.d

new :

```

```
$(MAKE) clean
$(MAKE)
```

Makefile 이쪼오금복잡해보이기는합니다. 일반적인 c 언어또는 c++ 언어등에서사용하는 Makefile 과는약간틀린데요.. 뭐가틀린지잠시만짚어보도록하겠습니다.

- INC 가없다 :: LIBS 에서관련된내용을한꺼번에다루게됩니다.
- dep section 이없다 :: 원래그런지는모르겠습니다만.... *.d 파일이 compile 과정에서알아서생성되는거같습니다.
- OBJCFLAGS 를사용한다 :: 기존의 Makefile 에서는 CFLAGS 를사용합니다. OBJCFLAGS 로해야에러없이 compile이되더군요.

프로그램의 compile 및 실행

다음과같이입력하면 compile 및 compile 된프로그램의결과를볼수있습니다.

```
localhost project_objcivec_study # make
cc `gnustep-config --objc-flags` -c -o example_01.o example_01.m
gcc -o example_01 example_01.o `gnustep-config --base-libs`

localhost project_objcivec_study # ./example_01
2013-05-31 07:02:14.441 example_01[10130] File NSBundle.m: 2600. In
-[NSBundle localizedStringForKey:value:table:] Localisation file
/usr/lib64/GNUstep/Libraries/gnustep-base/Versions/1.24/Resources/Korean.lproj/Localizable.strings
not in portable encoding so I'm using the default encoding for the
current system, which may not display messages correctly.
The file should be ASCII (using \u escapes for unicode characters) or
Unicode (UTF16 or UTF8) with a leading byte-order-marker.

2013-05-31 16:02:14.441 example_01[10130] Programming is fun!
```

흠.. 뭔가많이출력되는군요.. 내가구경하고싶은건한줄이었는데... 아마도인코딩관련부분에서뭔가문제가있는거같습니다만.... 별일아니니무시하도록하겠습니다. (사실저 NSBuldle 의 class method 에뭔가들어가는지예제를만들어서 NSLog 로찍으면될거같은데.. 귀찮으니 pass)

중요한건이한줄입니다.

2013-05-31 16:02:14.441 example_01[10130] Programming is fun!

이야... 프로그램이제대로실행이됐군요! 이제 Linux 에서 Foundation Framework 을사용할수있는준비는되었다고봐도되겠습니다! :D

이제부터 Makefile 과프로그램소스에대해아주쪼금더자세히알아보겠습니다.

프로그램소스코드살펴보기

```
/*
 *
 * 첫번째 예제 프로그램
 * Programming is fun! 이라는 문자열을 화면에 출력하기
 *
 */
```

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

int main (int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    NSLog (@\"Programming is fun!\");

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

위매.. 겁나게 간단한 코드지라~

자.. 하나하나 살펴 보도록 하겠습니다.

- 주석 스타일은 C 언어 스타일이다.
- **NS.....** 로 시작하는 것들은 다 Foundation Framework 에서 나온 것들이다.
- 때문에 위의 **#import <Foundation/Foundation.h>** 부분이 없다면 NS 로 시작하는 것들은 올바른 동작을 하지 못한다.

[] 부분에 대한 설명

objective-c 는 Smalltalk 에서 컨셉을 가져온 것 덕분에 객체와 메서드로 구분하게 되는데, [] 로 둘러싸인 부분은 이렇게 객체를 이용한 작업에 사용되는 구문이라고 생각하면 됩니다. 기본 구조는 다음과 같습니다.

```
[Object instanceMethod]
```

그럼 위의 구조를 참고로 **NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];** 구문의 진행 순서를 한번 짚어 보겠습니다.

1. **NSAutoreleasePool * pool :: NSAutoreleasePool** 클래스 타입으로 pool 이라는 객체를 생성한다.
2. **NSAutoreleasePool alloc :: pool** 객체에 메모리를 할당한다.
3. **NSAutoreleasePool init :: pool** 객체를 초기화한다.

쉽죠? 그럼 다음의 구문에서 참고해야 될 부분을 살펴 보겠습니다.

@로 시작하는 String에 대한 설명

NSLog 안에 들어가는 내용에서 사실 중요한 부분은 (**@\"Programming is fun!\"**) 입니다.

이 구문이 중요한 이유는 **@\"String\"** 형태가 NSString 이기 때문입니다.

일반적인 C 스타일의 문자열은 **\"string\"** 으로 묶입니다만.... Foundation Framework 에서는 **@\"String\"** 으로 문자열을 다루게 됩니다.

이후에 다루게 될 예제에서는 꾸준히 봐야 하는 것들이니 기억해 두면 좋은 부분이라 생각합니다 :D

Makefile 살펴보기

```
# example_01.m 을 위한 makefile 예제

.SUFFIXES : .c .o .m

CC = gcc
EXAMPLE_NUMBER = 01

LIBS = `gnustep-config --base-libs`
OBJCFLAGS = `gnustep-config --objc-flags`

OBJS = example_$(EXAMPLE_NUMBER).o
SRCS = example_$(EXAMPLE_NUMBER).m

TARGET = example_$(EXAMPLE_NUMBER)

all : $(TARGET)

$(TARGET) : $(OBJS)
    $(CC) -o $@ $(OBJS) $(LIBS)

clean :
    rm -rf $(OBJS) $(TARGET) core

distclean :
    rm -rf $(OBJS) $(TARGET) core *.d

new :
    $(MAKE) clean
    $(MAKE)
```

Makefile 은 사실 한번 익혀놓으면 앞으로 도도움이 꽤나 많이 되는 부분입니다. 상대적으로 소스 코드에 비해 살펴볼 부분은 적기 때문에 한 가지만 살펴보도록 하겠습니다.

```
LIBS = `gnustep-config --base-libs`
OBJCFLAGS = `gnustep-config --objc-flags`
```

위의 2줄을 보면 **gnustep-config** 이라는 부분이 있습니다. 저 프로그램은 **gnustep-make** 패키지에 포함된 프로그램으로서 GNUStep 을 사용한 프로그램을 compile 하는 경우 그에 따른 compile 시에 사용할 수 있는 옵션들을 얻을 수 있게 해줍니다. 조금 독특한 방법이지만 합니 다만.... 이후에 linux에서 Objective-c 를 사용하려는 경우에 기억해야 할 유용한 방법입니다. (맨 첫 줄의 # 이 붙은 부분은 주석인 건 다들 잠작하실 수 있겠죠? :D)

Second Application : Objective-C 의 소스코드 구조

이번에 다루게 될 예제는 숫자를 2개를 입력받아서 그걸 분수(fraction) 스타일로 출력하는 예제입니다. 원래 책에서 나오는 설명은 한개의 파일에 모든 내용이 다 있습니다만 이 문서에서는 어떤 식으로 구조가 이루어지는지를 같이 살펴보기 위해서 특별히 3개 정도의 파일로 분리했습니다. 이번에는 살펴볼 게 몇 가지가 있겠군요. 일단 어떤 내용의 파일인지부터 살펴보겠습니다.

```
/*
 *
 * example_02.h
 * @interface 에 대한 구현예제
 *
 */
```

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

```
@interface Fraction: NSObject
```

```
{
    int numerator;
    int denominator;
}
```

```
-(void) print;
```

```
-(void) setNumerator: (int) n;
```

```
-(void) setDenominator: (int) d;
```

```
@end
```

```
/*
 *
 * example_02.m
 * @implementation 에 대한 구현예제
 *
 */
```

```
#import "example_02.h"
```

```
@implementation Fraction
```

```
-(void) print
```

```
{
    NSLog(@"%i/%i", numerator, denominator);
}
```

```
-(void) setNumerator: (int) n
```

```
{
    numerator = n;
}
```

```
-(void) setDenominator: (int) d
{
    denominator = d;
}
```

```
@end
```

```
/*
 *
 *  main.m
 *  프로그램의 메인파트
 *
 */
```

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

```
@class Fraction;
```

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

    Fraction *myFraction;

    myFraction = [[Fraction alloc] init];

    [myFraction setNumerator: 1];
    [myFraction setDenominator: 3];

    NSLog (@"The Value of myFraction is:");
    [myFraction print];
    [myFraction release];

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

```
# Makefile Example
```

```
.SUFFIXES : .c .o .m
```

```
CC = gcc
```

```
EXAMPLE_NUMBER = 02
```

```
LIBS = `gnustep-config --base-libs`
```

```
OBJCFLAGS = `gnustep-config --objc-flags`
```

```

OBJS = example_$(EXAMPLE_NUMBER).o main.o
SRCS = example_$(EXAMPLE_NUMBER).m main.m

TARGET = example_$(EXAMPLE_NUMBER)

all : $(TARGET)

$(TARGET) : $(OBJS)
    $(CC) -o $@ $(OBJS) $(LIBS)

clean :
    rm -rf $(OBJS) $(TARGET) core

distclean :
    rm -rf $(OBJS) $(TARGET) core *.d

new :
    $(MAKE) clean
    $(MAKE)

```

각파일의주석에도이미적혀있습니다만... 각파일의역할을다시한번보겠습니다.

- example_02.h :: 프로그램에서사용할클래스의 interface 가선언되어있다.
- example_02.m :: 프로그램에서사용할클래스의 implementation(구현) 이선언되어있다.
- main.m :: 프로그램이실행되는시점에서처음실행되는 main() 함수가선언되어있다.

이프로그램의 compile 후실행결과는다음과같습니다.(사실똑바로 compile 이되지않습니다. 이유는아래부분에서따로알아보도록하겠습니다)

```

2013-05-31 22:09:04.552 example_02[14311] The Value of myFraction is:
2013-05-31 22:09:04.554 example_02[14311] 1/3

```

이제두번째예제에대한부분을진행하기전에 Objective-C 의지시어(지시자, Detective) 를최소한으로알아보고이후분석을시작해보도록하겠습니다.

Objective-C 의지시어::@interface, @implementation

이번에살펴볼코드에서나오는 Objective-C 의지시어는두가지입니다. 다른언어들의예를같이보면서어떤차이가있는지를살펴보겠습니다.

Class Example::Object Pascal

살짝~ 설명치고는긴느낌도있습니다만.... Object Pascal 의 Class 관련예제입니다.

```

// example from here ::
http://www.tutorialspoint.com/pascal/pascal_classes.htm
{$mode objfpc} // directive to be used for defining classes
{$m+}           // directive to be used for using constructor

program exClass;
type
    Rectangle = class
    private

```

```
    length, width: integer;
public
    constructor create(l, w: integer);
    procedure setlength(l: integer);
    function getlength(): integer;
    procedure setwidth(w: integer);
    function getwidth(): integer;
    procedure draw;
end;
var
    r1: Rectangle;
constructor Rectangle.create(l, w: integer);
begin
    length := l;
    width := w;
end;

procedure Rectangle.setlength(l: integer);
begin
    length := l;
end;

procedure Rectangle.setwidth(w: integer);
begin
    width := w;
end;

function Rectangle.getlength(): integer;
begin
    getlength := length;
end;

function Rectangle.getwidth(): integer;
begin
    getwidth := width;
end;

procedure Rectangle.draw;
var
    i, j: integer;
begin
    for i:= 1 to length do
    begin
        for j:= 1 to width do
            write(' * ');
        writeln;
    end;
end;
```

```

end;
begin
    r1:= Rectangle.create(3, 7);
    writeln(' Darw Rectangle: ', r1.getlength(), ' by ' ,
r1.getwidth());
    r1.draw;
    r1.setlength(4);
    r1.setwidth(6);
    writeln(' Darw Rectangle: ', r1.getlength(), ' by ' ,
r1.getwidth());
    r1.draw;
end.

```

간단하게 보자면 크게 두 가지 부분으로 나눌 수 있습니다.

- 선언부 :: type 부분
- 구현부 :: function or procedure

Object Pascal 은 C언어의 **prototype** 선언(**declearlation**) 처럼, 상단부분에서 현재 파일 내에서 class 의 구현에서 사용할 function 또는 procedure, 그리고 instance variables 등을 선언하는 선언부가 상단에 있습니다. 그리고 아래쪽에서 선언된 내용에 맞는 구현을 작성하는 구조로 되어 있습니다.

C언어의 prototype 은 사실이 이후에 설명할 Objective-C 의 내용과 비슷한 부분이 있는데요 내용은 여기를 참고하시면 좀 더 명확한 내용을 확인할 수 있습니다.

Class Example::PHP

아래는 php 의 예제입니다.

```

class Vegetable {

    var $edible;
    var $color;

    function Vegetable($edible, $color="green")
    {
        $this->edible = $edible;
        $this->color = $color;
    }

    function is_edible()
    {
        return $this->edible;
    }

    function what_color()
    {
        return $this->color;
    }

} // Vegetable 클래스 끝

```

php의 경우는 Object Pascal 보다는 좀 덜 복잡한편이죠. 별도로 선언부는 존재하지 않으며 그냥 편하게 막 쓰면 됩니다. Class를 선언하면서 구현의 작성까지 같이 가지고 있는 경우 라고 보시면 되겠습니다.

How do you define a Class?: Objective C

Objective-C에서는 class에서 사용하게 될 element의 interface를 가지고 있는 @interface 부분과 선언된 메서드 등을 구현하는 @implementation 부분으로 나눌 수 있습니다.

@interface

@interface에서는 다음과 같은 것들을 선언할 수 있습니다.

- instance (객체) 상태에서 사용할 변수를 선언
- instance method 선언 :: (-) 기호를 사용합니다. 생성된 객체 상태에서 사용할 수 있습니다. @class 지시어에 반응하지 않습니다.
- class method 선언 :: (+) 기호를 사용합니다. Class에서 바로 사용할 수 있습니다. instance 변수에 접근할 수 없으며 @class 지시어로 사용할 수 있습니다.

이부분을 참고해서 예제 코드를 다시 한번 읽으면 보이는 것이 좀 틀려지겠죠? @interface에서는 Class의 흐름을 정의한다고 생각하시면 되겠습니다.

```
@interface Fraction: NSObject
{
    int numerator;
    int denominator;
}

-(void) print;
-(void) setNumerator: (int) n;
-(void) setDenominator: (int) d;

@end
```

이렇게 @interface에서 선언되는 instance 변수들은 Class가 실제로 객체(Object)로 생성되는 경우 메모리에 존재할 수 있기 때문에 Class method에서는 접근할 수 있는 방법이 없다는 것을 참고해주시기 바랍니다.

@implementation

@implementation에서는 다음과 같은 것들을 작성할 수 있습니다.

- instance method 구현 :: (-) 기호를 사용합니다.
- class method 구현 :: (+) 기호를 사용합니다.

```
@implementation Fraction

-(void) print
{
    NSLog ("%i/%i", numerator, denominator);
}

-(void) setNumerator: (int) n
{

```

```

        numerator = n;
    }

- (void) setDenominator: (int) d
{
    denominator = d;
}

@end

```

@implementation 의 경우는 method 에 대한 내부구현등을 작성하게 됩니다 사실 별내용이 아닙니다만...

위의 내용에서 조금 신경써서 봐야 할 부분이 있습니다. **setNumerator:** 메서드의 내부구현을 보면 **numerator = n;** 이라는 부분이 있습니다. 위에서도 설명했습니다만 instance method 는 Class 내부의 instance variables(변수) 에 접근이 가능하지만 Class method 는 접근이 불가능하다는 걸 기억해주세요.

@class 지시어와 **#import**

이 부분에 대해서 알아보기 전에 윗부분에서 소스코드를 나열할 때 나왔던 **main.m** 파일의 내용을 한번 더 보도록 하겠습니다.

```

#import <Foundation/Foundation.h>

@class Fraction;

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

    Fraction *myFraction;

    myFraction = [[Fraction alloc] init];

    [myFraction setNumerator: 1];
    [myFraction setDenominator: 3];

    NSLog (@\"The Value of myFraction is:\");
    [myFraction print];
    [myFraction release];

    [pool drain];
    return 0;
}

```

이 파일의 내용을 적으면서 올바르게 compile 이 되지 않을 거라고 한 얘기혹시 기억이나 실지 모르겠습니다. 사실 compile 이 안된다고 보기는 싫은 경고가 나오는 정도입니다만.... 일단 현재의 코드를 그대로 compile 하면 어떤 결과가 나오는지 보도록 하겠습니다.

```
localhost example02 # make
cc `gnustep-config --objc-flags` -c -o main.o main.m
main.m: In function 'main':
main.m:21:2: warning: no '-setNumerator:' method found [enabled by
default]
main.m:21:2: warning: (Messages without a matching method signature
[enabled by default]
main.m:21:2: warning: will be assumed to return 'id' and accept
[enabled by default]
main.m:21:2: warning: '...' as arguments.) [enabled by default]
main.m:22:2: warning: no '-setDenominator:' method found [enabled by
default]
main.m:25:2: warning: no '-print' method found [enabled by default]
gcc -o example_02 example_02.o main.o `gnustep-config --base-libs`
```

역시 경고작렬입니다.. 내용인즉슨말입니다.... 사실메서드못찾겠습니다...라는거죠.

문제는 @class 라는지시어덕분입니다. 위의내용중저부분을보도록하겠습니다.

```
@class Fraction;
```

오호.. 저건될테고저내용밖에없는걸까요? 이부분을이해하기위해서는잠시 C 언어를 살펴보는게좋을거 같습니다.

C 언어에서의 extern

자세한내용은 여기 를참고해주세요.

사실 C에서 extern 이문제가되는시점은 compile time 이아니라 link time 입니다. extern 은 compile time 에서굳이문제가안될거다.... 라고 compiler 에게문제가없다...라고보장을해주는게되 는거죠. 이함수는있는거니깐걱정마 라는의미가되겠습니다.

Objective-C 에서의 @class

@class 는 c 에서 extern 과비슷하게사용됩니다. 하지만사용할수있는것은 class 메서드에 국한됩니다.

instance 메서드를사용하려한다면반드시해당되는 class 의 @interface 가있는파일이 #import 로선언되어야합니다^{[3][4]}.

간단하게말하면 (+) 로선언된 method 는사용가능하지만 (-) 로선언한메서드를사용하 려면 #import 를써야한다는의미가됩니다.

올바른 main.m 파일의코드

경고없는 compile 을원한다면아래처럼코드를수정하면됩니다.

```
#import <Foundation/Foundation.h>
#import "example_02.h"

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

    Fraction *myFraction;
```

```

myFraction = [[Fraction alloc] init];

[myFraction setNumerator: 1];
[myFraction setDenominator: 3];

NSLog(@"The Value of myFraction is:");
[myFraction print];
[myFraction release];

[pool drain];
return 0;
}

```

위처럼 코드를 수정하면 경고 없는 깔끔한 compile 결과를 얻을 수 있습니다. :D

추가 학습 :: Objective-C 에서의 다른 지시어들

@protocol

출처

<http://soooprmx.com/wp/archives/2432>

프로토콜의 개념 잡기

프로토콜은 “선언만 되고 구현되지 않은” 메소드를 말한다. 이 프로토콜은 역시나 “너무 단순해서 쉽게 감이 잡히지 않는” Objective-C의 기능이다. 프로토콜은 크게 다음 3가지 경우에 유용하게 사용된다.

1. 다른 객체가 구현해주면 되는 메소드를 선언
2. 클래스를 숨기고 인터페이스만 선언하고자 할 때
3. 상속 관계가 아니지만 비슷한 인터페이스를 만들고자 할 때

등이다. 1의 경우는 델리게이트 의 형태를 생각하면 된다. 즉 어떤 객체가 그 자신은 구현하지 않지만 델리게이트에게 위임하여 실행할 어떤 기능이 필요하다고 하면 프로토콜로 선언하고, 델리게이트가 그 프로토콜을 따르도록 하여 실제 기능의 실행을 델리게이트에서 처리하도록 할 때 프로토콜을 사용한다.

여기서 실제 메소드의 선언을 해당 객체가 해주고, 다른 객체는 구현만 한다는 측면에서 2, 3의 경우는 이해할 수 있을 것이다.

다른 객체가 구현해 줄 인터페이스 선언하기

클래스 인터페이스나 카테고리라는 특정한 클래스가 구현해야 하는 기능을 선언해 준다. 프로토콜은 이와 반대로 (알 수 없을지도 모르는) 다른 객체가 구현해 줄 기능을 선언하는 것이다. 따라서 프로토콜은 단순히 메소드 선언의 묶음으로 구성된다. 예를 들어 어떤 이벤트가 발생하면 이를 처리해야 할 때, 대충 다음과 같은 모양의 메서드들이 필요할 수 있다.

```

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

```

이 메소드를 해당 객체에서 직접 구현해도 되지만, 다른 객체가 그 작업을 대신 받아서 하도록 하려면 이를 프로토콜로 선언하고, 다른 객체가 그 프로토콜을 따르도록 하면 되는

것이다.

이방식은 메소드의선언부를클래스상속관계에서완전히분리시키게된다. 프로토콜에서선언된메소드들은어딘가에서구현되겠지만, 이를구현할메소드의클래스는무엇이어도상관없다. 따라서전혀다른클래스들이라도같은프로토콜을따른다면같은메소드를가지고있다고생각할수있다. (프로토콜은어떤메소드들의선언이므로) 같은프로토콜을따르는클래스들은상속관계에서서로관련이없다고하더라도기능적인측면으로분류할때유사한그룹으로묶을수있다.

다른객체가구현할메소드

앞서언급했지만다른객체에게작업을위임하거나 (델리게이트) 다른객체로부터정보를받아와야하는경우 (데이터소스)에는보통해당객체에게메시지를보내어이를처리하면된다. 일련의메시징은양방향이므로, 이때메시지를보내는객체는이메시지를받을객체의헤더파일을임포트하여수신객체에게보낼메시지의셀렉터를알수있다.

하지만이를수신할객체가아직만들어지지않은상황이라면, 인터페이스파일을임포트할수가없다. 따라서이경우에는아직만들어지지않은객체가받을메시지를정할수없게된다. 이때프로토콜을사용하면 수신객체의헤더를임포트하지않고서도수신객체에게메시지를보낼수있게된다. 즉어떤객체가특정한프로토콜을따른다는것은그프로토콜에서선언한메소드를구현하고있다는의미이므로그객체의클래스가뭔지몰라도 (따라서내부에구현된메소드가뭔지몰라도) 프로토콜에선언된메소드는가지고있다고간주하게되는것이다.

만약어떤객체에게 `helpOut:` 이라는메시지를보내어처리할일이있다고한다면다음과같이처리하게될것이다.

```
-setAssistant:anObject
{
    assistant = anObject;
}
```

만약이객체를 `assistant`라는프로퍼티라생각한다면,

```
-setAssistant:anObject
{
    assistant = anObject;
}

- (BOOL) doWork
{
    ...
    if ( [self.assistant respondsToSelector:@selector(helpOut:)] ) {
        [self.assistant helpOut:self];
        return YES;
    }
    return NO;
}
```

위와같이해당객체가 `helpOut:` 이라는메시지에응답할수있는지확인한다음메시지를보내게된다. 하지만이객체가특정한프로토콜을따른다면이러한검사절차없이바로호출할수있다.

이것의단적인예가델리게이트이다. 보통기본적으로우리가작성하는코드에서는델리게이트입장에서코드를쓴다. 대표적으로테이블뷰컨트롤러가그예이다. 데이터소스나델리게이트프로토콜을따르는객체를만들어그프로토콜의필수메소드를구현해주고있는것이다. 반대로델리게이트에게일을시키는객체에서는다음과같은코드를볼수있다. 먼저헤더파일에서는프로토콜을선언하고, 그프로토콜을따를델리게이트를선언한다.

```
// SomeClassNeedsDelegate.h
import <Foundation/Foundation.h>

@protocol MyClassDelegateProtocol
-(void)helpOut:(id) sender
@end

@interface SomeClassNeedsDelegate : NSObject
@property (nonatomic, weak) id<MyClassDelegateProtocol> delegate;
@end
```

구현부분에서는델리게이트에게메시지를보낼수있다. 프로퍼티로설정한델리게이트는만약이객체를생성하거나참조하는쪽에서델리게이트를지정할것이고, (하지않았다면 nil) 따라서바로메시지를보내도문제는없다.

```
[self.delegate helpOut:self]
```

델리게이트가되는객체는인터페이스를선언할때 (내부든외부든무관하다) 이프로토콜이선언된헤더를임포트하고인터페이스시작지점에다음과같이쓴다.

```
// another object that will be the delegate of 'SomeClassNeedsDelegate'
import "SomeClassNeedsDelegate.h"

@interface MyAnotherClass : NSObject <MyClassDelegateProtocol>
<# 클래스 인터페이스 선언 #>
```

이와관련한자세한내용은다음절에서설명하겠다.

프로토콜정의방법

프로토콜은헤더파일의인터페이스영역과별개의영역을만들어메소드의선언을하게된다.

```
@protocol 프로토콜이름 : 상속받을 프로토콜
- 메소드 이름
@end
```

다음은프로토콜선언의예시이다.

```
@protocol MyProtocol : NSObject // 프로토콜은 다른 프로토콜을 상속할 수 있다.
-(void)requiredMethod;
@optional
-(void)anOptionalMethod;
-(void)anotherOptionalMethod;
@required
-(void)anotherRequiredMethod;
```

```
@end;
```

@optional 키워드다음에나오는메소드들은모두선택적으로구현하면되는메소드들이며, 이런키워드가없는경우에는모두필수적으로구현해야하는메소드로인식한다.

@optional 키워드로선언한메소드들뒤에다시필수메소드를추가로선언하려면

@required 키워드를다시써주면된다.

프로토콜따르기

객체가어떤프로토콜을따른다고할때에는다음과같은절차를따른다.

1. 프로토콜이선언된인터페이스파일을임포트한다. 이파일은프로토콜에선언된메소드를필요로하는객체의헤더파일인경우가많은데, 별도의인터페이스파일에프로토콜을선언할수도있다. 많은프로토콜이혼재하는프로젝트의경우라면, 별도의프로토콜인터페이스파일을따로설정하는것이편리할수있다.
2. 인터페이스블럭시작줄에 <프로토콜이름> 이라고꺾은괄호로둘러싸고프로토콜이름을명시해준다.
3. 해당프로토콜에서필수로선언한메소드들을구현한다.

Formatter라는클래스가 Formatting, Prettifying 이라는두개의각각다른프로토콜을따른다고할때다음과같이인터페이스를작성하면된다.

```
@interface Formatter : NSObject <Formatting, Prettifying>
<# 인터페이스 선언 #>
@end
```

또한, 객체를선언할때그객체가특정프로토콜을따르고있음을명시할때 (프로토콜을따른객체를아직작성하지않아서모를때사용함) 다음과같이타입명뒤에프로토콜명을사용할수있다.

```
Formatter <Formatting> *aFormatterObject;
id<Prettifying> *aPrettifier;
```

비정규프로토콜

비정규프로토콜은 카테고리의형식으로프로토콜을대체하는방법이다. 코코아의거의모든객체들은 NSObject를상속받으므로, NSObject의카테고리로메소드들을선언하면 별도의프로토콜을따른다는명시없이도해당메소드를선언한것과같은효과를낼수있다. 이방법은모든메소드가선택적일때사용하면명시적인코드를조금더줄일수있는효과가있다.

아직선언되지않은프로토콜

만약어떤프로토콜 A에서사용하는타입이 B라는프로토콜을따라야하는데, 이 B에대한프로토콜이아직작성되지않았다면프로토콜이선언된인터페이스파일조차임포트하지못하는경우가있다.

이럴때는 @class ClassName; 과마찬가지로 @protocol someProtocolName; 과같이이런프로토콜이있을것이라고컴파일러에게알려줄수있다. 이는두개의프로토콜이서로를동시에참조하는경우에사용하여컴파일러가에러를내지않도록할수있다.

@property / @synthesize / @dynamic

출처

<http://blog.outsider.ne.kr/604>

다른언어와마찬가지로 Objective-C에서도객체의멤버변수를외부에서접근하도록하려면접근자메서드(accessor method)인 getter와 setter를만들어주어야합니다. Objective-C에서 클래스는 @interface와 @implementation의 2부분으로나누어지는데 @property는 @interface부분과연결되고 @synthesize와 @dynamic은 @implementation부분과연결됩니다. @property와 @synthesize는 preprocessor macro이라서 Xcode에서 Build and Run을실행했을때 prewritten/preformatted 코드블락으로교체됩니다.

@property

```
//header file
@interface Example : NSObject
{
    float value;
}

- (float) value;
- (void) setValue: (float) newValue;
```

위와같이프로퍼티에대한 getter와 setter 메서드를선언해줄수있습니다. 하지만이접근자메서드를만들어주는것은항상같은형태로반복되는지루한작업이기때문에 Objective-C 2.0에서부터는 @property 지시어를사용해서 getter와 setter에대한코드를자동으로생성할수있도록해줍니다.

```
//header file
@interface Example : NSObject
{
    float value;
}

@property float value;
```

정확한문법은 @property(attributes [, attribute2, ...]) type name;의형태이고앞의코드에서 6,7라인의코드대신에위처럼 @property 지시어를사용하면자동으로앞의코드로변환해줍니다. 위에서보는것처럼 getter의이름은 "프로퍼티이름"이되고 setter의이름은 "set프로퍼티이름"이됩니다. 만약 getter의이름을바꾸려면 @property(getter=getValue) float value;와같이써주면 getter를 getValue 함수를써서사용할수있습니다. 프로퍼티를 IB아웃렛으로지정하려면 @property (nonatomic, retain) IBOutlet NSButton *myButton;와같이 IBOutlet 식별자를사용할수있습니다.

(attributes [, attribute2, ...])부분에서사용이가능한속성들은아래와같습니다.

- **getter=getterName** - getter의이름을 getterName로지정합니다.
- **setter=setterName** - setter의이름을 setterName로지정합니다.
- **readwrite** - 기본동작으로 getter와 setter를모두만듭니다. Mutually exclusive로 readwrite합니다.
- **readonly** - getter만만듭니다. Mutually exclusive로 readwrite합니다. 값을할당하려고하면컴파일오류가발생합니다.
- **assign** - 기본동작이며 setter가간단한할당을사용합니다.(예 location = where;) 객체를소유할필요가없을때사용합니다.

- **retain** - assign과 비슷하지만 레퍼런스 카운트를 증가시킵니다. Mutually exclusive로 assign과 copy합니다. 포인터 객체를 할당할 경우에는 외부에서 객체가 릴리즈되어 파괴된 객체를 참조하는 문제를 막기 위해서 클래스가 멤버 객체를 소유하도록 레퍼런스 카운트를 증가시킵니다.(이전 값을 release 합니다.)
- **copy** - 할당하는데 객체의 복사본을 사용합니다. Mutually exclusive로 assign과 retain합니다. 포인터 객체의 경우 레퍼런스의 값이 바뀌어 프로퍼티의 값이 바뀌는 걸 막기 위해 setter에서 복사본을 만들어서 할당하며 copy를 사용하려면 NSCopying 프로토콜을 구현한 객체에서만 유효합니다.
- **nonatomic** - 엑세서들을 nonatomic으로 지정합니다. 멀티프로세서 환경에서는 지정해 줘야 합니다. 이는 Mutually exclusive락으로 접근자 메서드를 보호하지 말라고 지시하는 것입니다. atomic이 기본 동작입니다.

@synthesize

@property를 사용해서 프로퍼티들에 대한 getter와 setter를 선언했으면 @implementation에서 실제 코드를 추가해 주어야 합니다. @property를 사용한 것은 단지 컴파일러가

@implementation에서 getter와 setter 메서드가 작성되었다는 것을 기대하도록 하는 것입니다.

```
//module file
@implementation Example
-(float) value
{
    return value;
}

-(void) setValue: (float) newValue
{
    value = newValue;
}
@end
```

위와 같이 실제 구현 코드를 작성해 주어야 합니다. 이는 아래 코드처럼 @synthesize 지시어를 사용하면 다음과 같이 작성해 줄 수 있습니다.

```
//module file
@implementation Example
@synthesize value;
@end
```

@dynamic

@dynamic 지시어는 @synthesize 대신에 사용할 수 있으며 getter와 setter 메서드가 클래스 자신에 의해서 구현되지 않고 (슈퍼클래스 같은) 다른 어딘가에 구현되어 있다고 알려주어 getter/setter가 구현되어 있지 않아도 컴파일러 경고 받지 않도록 해줍니다.

Super class :

```
//header file
@property (nonatomic, retain) NSButton *someButton;

//module file
```

```
@synthesize someButton;
```

Sub class :

```
//header file
```

```
@property (nonatomic, retain) IBOutlet NSButton *someButton;
```

```
//module file. @synthesize 를 대신한다
```

```
@dynamic someButton;
```

위와같이작성함으로써 someButton에대한 구현책임이텔리게이트되었음을의미합니다. 혹은 CoreData의 NSManagedObject 클래스처럼접근자메서드들이컴파일타임이아닌런타임시의제공되는경우에도컴파일타임에오류가나지않도록하기위해서 @dynamic 지시어를사용할수있습니다.

@selector

출처

<http://blog.daum.net/iamuzoooin/33>

이강좌는, C 언어(ISO/IEC 9899:1990)를이해하고있는것이전제입니다.

출처 :: <http://wisdom.sakura.ne.jp/programming/objc/index.html>

메소드의내부표현

Objective-C 컴파일러는메소드에특별한이름을컴파일시에내부표현으로변환합니다. 이, 메소드의내부표현을 셀렉터라고부르고, 메세지의송수신의뒤에서는, 이셀렉터가교환되고있습니다. 메소드를특정짓기위한내부표현에대해서는컴파일러에의존하는문제이며, 개발자가알아야할범위가아닙니다. 개발자에게있어서중요한것은, 이셀렉터를 **SEL** 형태로서취급할수있다고하는사실입니다.

메소드가어떠한데이터에변환되어어떻게식별되고있을까는문제가아닙니다. 그러나, Objective-C는이내부표현을 SEL 형태의변수로서취급하고있습니다. 즉,SEL 형태의변수에는, 메소드명을식별하기위해서컴파일러가할당한특수한코드를저장합니다.

메소드를특정하는실렉터는 **@selector** 컴파일러지시문을이용해취득할수있습니다.

```
@selector ( method )
```

method 에는, 실렉터를취득하고싶은메소드의이름을지정합니다. 지정한메소드의이름이존재할지어떨지는, 메소드를호출할때, 실행시에판정되기때문에, 컴파일시에는알수없을것입니다.

그럼, 취득한셀렉터의값을 SEL 형태의변수에저장해서어떻게이용할수있는것입니까. 실렉터가메소드를별도로알고있으므로함수에서사용하는포인터처럼, 동적으로메소드를지정하는방법이지요. 셀렉터로부터메소드를호출하는기능을제공하는것은루트클래스입니다.

Object 클래스에는,SEL 형태의값을받는 **perform** 메소드가선언되고있습니다. 이메소드는인수로받은셀렉터는특정한메소드를실행합니다.

```
- perform:(SEL) aSel;
- perform:(SEL) aSel with:anObject;
- perform:(SEL) aSel with:anObject1 with:anObject2;
```

aSel 에호출하는메소드의셀렉터를지정합니다. anObject,anObject1,anObject2 에는메소드에 건네주는인수를지정할수있습니다.

```
#import <stdio.h>
#import <objc/Object.h>

@interface Test : Object
- (void)Write;
@end

@implementation Test
- (void)Write {
    printf("I am the bone of my sword.\n");
}
@end

int main() {
    id obj;
    SEL method;

    obj = [Test new];
    method = @selector(Write);
    [obj perform:method];

    return 0;
}
```

이프로그램에서는,Write 메소드를나타내는셀렉터를 SEL 형태의변수 method 에보존하고있습니다. 그리고, Test 클래스의인스턴스 obj 에 perform 메세지를 method를인수로서송신합니다. perform 메소드는주어진셀렉터로부터실행해야할메소드를산출해, 실행합니다. 이것을잘이용하면, 실행시에호출해야할메소드를상황에따라바꾸는프로그램을구현할수있습니다.

@private / @public

출처

<http://blog.daum.net/iamuzoooin/30>

Objective-C 접근제어

이강좌는, C 언어(ISO/IEC 9899:1990)를이해하고있는것이전제입니다.

출처 :: <http://wisdom.sakura.ne.jp/programming/objc/index.html>

인스턴스변수의공개범위

클래스는구조체와는다른역할이있기때문에, 데이터형의제공으로서의클래스라는 것은존재해야하는것이아닙니다. 그러나 RGB 형식의색을표현하기위한클래스나좌표나사이즈를나타내는클래스이면, 메시지에서부터값을얻는등의처리는약간장황해서, 직접인스턴스변수와교환할수있는편이좋다고생각하는개발자도있겠지요.

원칙으로서는, 클래스의인스턴스변수에외로부터액세스가능해야만한것은아니고, 디폴트에서는밖으로부터인스턴스변수에액세스할수없습니다. 기본적으로인스턴스변수를선언하는클래스의메소드이외로부터인스턴스변수가조작되면, 변수에보존되고있는값의정합성을유지할수없게되기때문입니다. 외부로부터인스턴스변수에접근하려면, 우선, 인스턴스변수의접근제어를명확하게하지않으면안됩니다.

인스턴스변수의접근제어은, 공개를나타내는 `public`, 비공개를나타내는 `private`, 서브클래스로부터의액세스를허가한다 `protected`로나누어집니다. `public`은클래스의외부나서브클래스를포함해모든장소로부터인스턴스를통해액세스되는변수를나타냅니다. `private`는, 반대로인스턴스변수를선언한클래스의메소드로부터접근가능한변수입니다. 그리고,`protected`는변수를선언한클래스와그클래스의서브클래스로부터접근할수있는변수를나타냅니다.

선언하는인스턴스변수에이러한접근제어를지정하려면컴파일러지시문을이용해 **@public**, **@private**, **@protected** 를인스턴스변수선언부분의사이에삽입합니다. 이러한컴파일러지시문이쓰게되면그이후에선언된인스턴스변수는모두, 지정된접근제어로설정됩니다. 접근제어가지정되어있지않은경우에인스턴스변수는 `protected` 로설정됩니다.

```
@interface A : Object
{
    int z;          //protected
    @public
    int a;          //public
    @protected
    int b;          //protected
    @private
    int c;          //private
    int d;          //private
}
@end
```

이와같이, 접근제어를지정한후에선언된인스턴스변수는, 그접근제어를가집니다. 접근제어를지정하는컴파일러지시문의출현순서, 출현회수에제한은없습니다.

`public` 으로공개되고있는변수에, 클래스의외로부터인스턴스를통해액세스하려면, 클래스형의포인터변수로부터 -> 연산자를사용해인스턴스변수를지정합니다. 이것은, 구조체에의포인터로부터멤버변수에액세스하는방법과같습니다. 인스턴스변수에의액세스는컴파일시에형태체크가필요하기위해 `id` 형태의변수로부터액세스할수없습니다.

```
#import <stdio.h>
#import <objc/Object.h>

@interface A : Object
{
```

```

@public
    int a;
@protected
    int b;
@private
    int c;
}
- (id)initWithA:(int)a int:(int)b int:(int)c;
- (void)WriteA;
@end

@interface B : A
- (void)WriteB;
@end

@implementation A
- (id)initWithA:(int)a int:(int)b int:(int)c {
    self->a = a;
    self->b = b;
    self->c = c;
    return self;
}
- (void)WriteA {
    printf("[A Write Method, a=%d, b=%d, c=%d]\n", a , b , c);
}
@end

@implementation B
- (void)WriteB {
    printf("[B Write Method, a=%d, b=%d]\n", a , b);
}
@end

int main() {
    B * objb = [[B new] initWithA:1000 int:100 int:10];
    printf("[main() scope, a=%d]\n" , objb->a);
    [objb WriteB];
    [objb WriteA];
    [objb free];
    return 0;
}

```

이 프로그램으로 선언하고 있는 A 클래스는, public 인스턴스 변수 a, protected 인스턴스 변수 b, private 인스턴스 변수 c 를 선언하고 있습니다. 그리고, 이것들에 액세스할 수 있는 것을 증명하기 위해서, main() 함수로부터 B 형태의 포인터 변수 objb를 통하고, 공개되고 있는 변수 a 에 objb->a 그렇다고 하는 형태로 액세스하고 있습니다.

A 클래스를 상속받는 서브클래스 B에서는 WriteB라는 메소드를 선언하고, 이 메소드의 구현으로 공개되고 있는 변수 a 라고 서브클래스로부터의 액세스가 허가되고 있는 변수 b를 표시합니다. 다만, 서브클래스로부터도 private 변수에 액세스할 수 없기 때문에 c를 표시할 수 없습니다. 모든 변수에 액세스할 수 있는 것은, 인스턴스 변수를 선언한 A 클래스 뿐입니다. A 클래스의 메소드 WriteA 에서는, 모든 인스턴스 변수를 표시하는 것이 가능합니다.

참고문서

- http://trans.onionmixer.net/mediawiki/index.php?title=PHPUnit_Manual
- <http://www.roccoangeloni.it/wp/2008/08/07/objectivec-on-ubuntu-linux/>
- 우분투리눅스 (Ubuntu Linux) 에서 objective c 환경구성하기
 - <http://khamirage.tistory.com/303>
 - <http://khamirage.tistory.com/348>
- Objective-C 에 대한 좋은 글 모음 : <http://seungngil.tistory.com/category/C/Objective%20C>
- 간편하지만 독이 되어 버리는 NSAutoreleasePool

Notes

- [1] ISBN 9788991268654 or ISBN 899126865X 프로그래밍 오브젝티브 C 2.0/Programming in Objective C 2.0. 인사이트출판.
- [2] garbage collector. gentoo 에서는 Boehm-gc 라는 패키지입니다.
- [3] 사실 c에서 extern 이라고 하면 function 을 사용하기 위한 거지 class 문법도 없는 상태에서 존재하는 거라 개념이 맞기는 하다.
- [4] 미묘하지만 gcc 에서는 그냥 동작은 합니다. 일단은 말이죠... 이게 어떻게 된 건지 확인은 해 봐야 할 거 같기는 합니다.

ObjectiveC Symbole etc

Objective-C 기호와 용어 정리

출처

http://blog.daum.net/_blog/BlogTypeView.do?blogid=0N45F&artid=7924267&categoryId=829975®dt=20111029110024#ajax_history_home

[] 사용법

-> 메서드를 사용하기 위해 중괄호를 사용한다.

방법은

```
[object method]; -> [객체 메소드]
[object methodWithInput:input]; -> [객체 메소드:인자]
```

로 쓰이며,

```
output = [object methodWithOutput];
output = [object methodWithInputAndOutput:input];
```

처럼 값을 돌려준다.

중복된 [] 사용법

1번에서봤듯이객체의메소드를호출하기위해 []을사용한다.

그렇다면함수호출결과를다시호출하는형태도가능할까?

예를들어 C의경우 func1(func2()); 이런형태말이다.

당연히가능하다.

```
[NSString stringWithFormat:[prefs format]];
```

다만 2개이상호출하게되면복잡해서추천하지는않는다.

id(아이디)

```
id myObject = [NSString string];
```

[NSString string] 가어떤타입을리턴해도해당타입으로전환하여저장해준다.

여기서는당연히 myObject에 NSString 타입으로저장될것이다.

원래는아래와같이써야할것이다.

```
NSString* myString = [NSString string];
```

즉 id 는 NSString* 을나타내는것이다. id가포인터를내장하고있음을짐작할수있다.

:(콜론)

[]에서메소드를호출할때메소드와인자값을구분해준다.

```
[myData writeToFile:@"tmp/log.txt" atomically:NO]; -> [객체 메소드 인자  
메소드:인자];
```

.(dot,닷)

Objective-C 2.0 에서 getter와 setter를위해 dot 을사용한다.

```
photo.caption = @"오늘은좋은날!!";  
output = photo.caption;
```

로값을 get할수있다.

@(골뱅이또는 at 문자)

문자열앞이나 Objective-C만의독특한지시어를나타내기위하여사용된다.

다시말해 C와는다른것임을구분하기위해쓰이고있다.

+, -

연산자로도사용하지만여기에서는메소드를구분하기위해썼다.

클래스내에메소드를기술할때 +는클래스메소드, -는인스턴스메소드를가리킨다.

Article Sources and Contributors

Xcode4 CurrencyConverter *Source:* <http://workspace.onionmixer.net/mediawiki/index.php?oldid=573> *Contributors:* Onionmixer

ObjectiveC Linux *Source:* <http://workspace.onionmixer.net/mediawiki/index.php?oldid=567> *Contributors:* Onionmixer

ObjectiveC Symbole etc *Source:* <http://workspace.onionmixer.net/mediawiki/index.php?oldid=538> *Contributors:* Onionmixer

[illegible]

License

크리에이티브 커먼즈 저작자표시
<http://creativecommons.org/licenses/by/3.0/>
