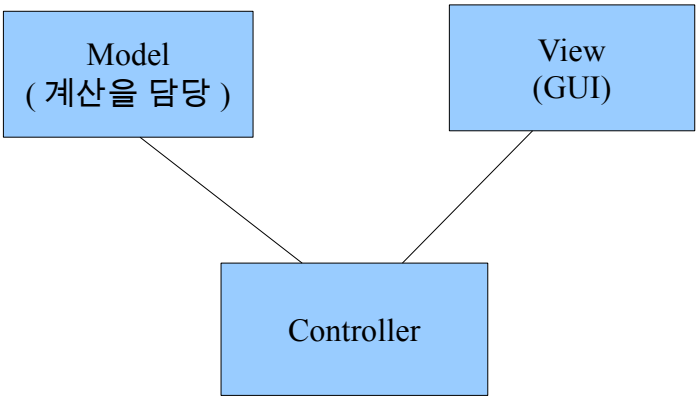
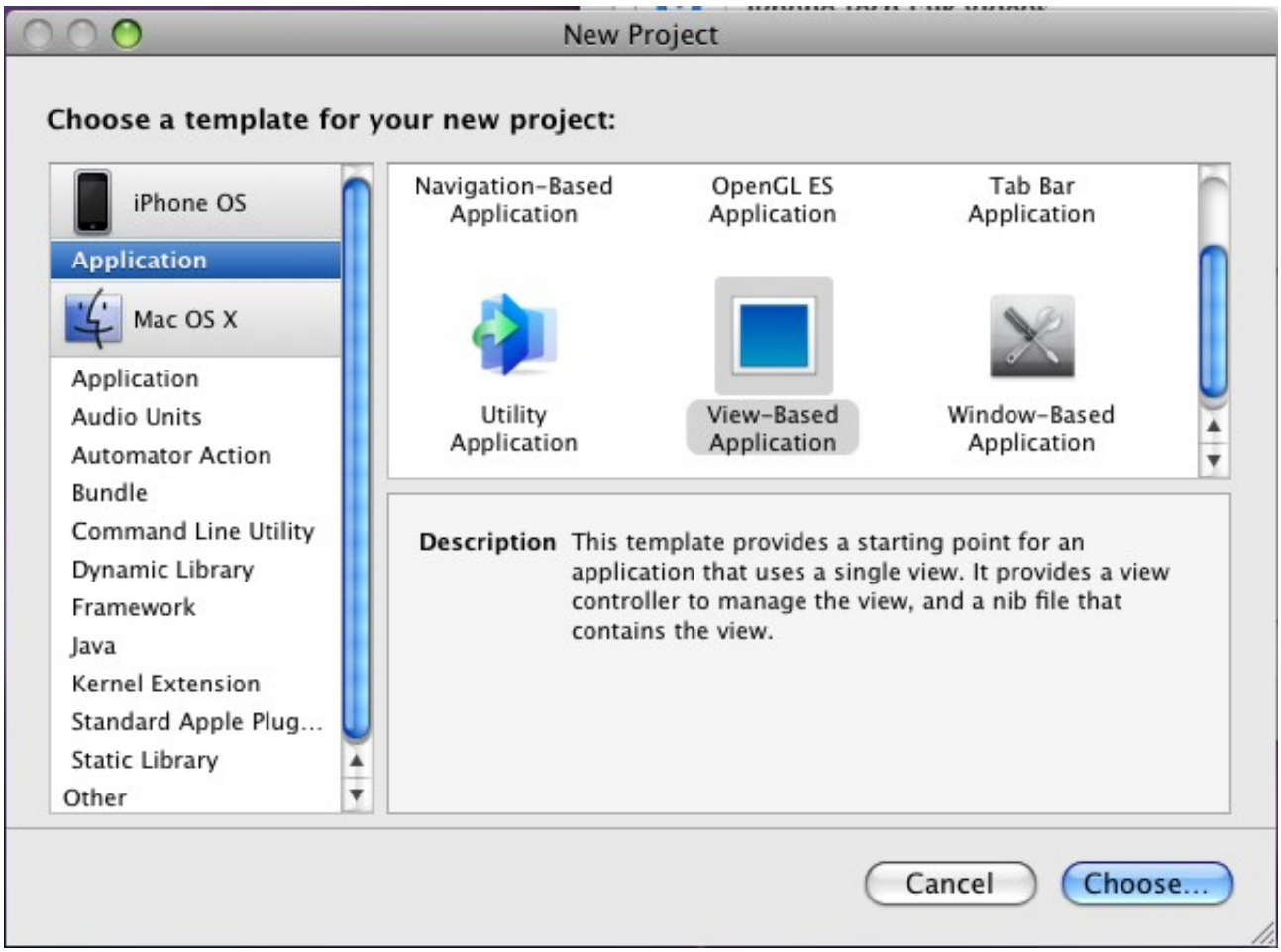
안녕하세요.
약속드린대로 Xcode에서 iPhone SDK를 이용한 간단한 계산기 예제를 통해 프로그램이 어떻게 구성이 되는지 나름대로 최대한 알기 쉽게 설명 드리도록 하겠습니다.
먼저, MVC 모델에 대한 약간의 이해가 필요할 듯 합니다. MVC는 Visual C++ 이나 다수의 많은 tool들에서 적용하고 있는 프로그래밍 모델로, MVC 는 각각 Model, View, Controller 의 머리글자를 딴 모델입니다.
제가 이해하고 있기로는 GUI프로그램을 제작할때, 하나의 모듈 또는 class에서 모든일을 하게 제작하지 말고 각각의 특성에 맞도록 나누어서 프로그래밍 하는 방식을 의미합니다.

이번 예제에서 계산기 프로그램을 제작한다고 하면, 화면상에 나타나는 숫자 버튼, +, -, *, / 등 사칙여난 버튼이나 입력, 계산 결과를 출력하기 위한 TextField 등 사용자가 볼수 있는 GUI 자체를 View 라고 할 수 있겠고, Model은 실제 계산을 하는 함수를 포함하고 있는, 계산을 수행하는 부분 이고, 이 둘을 잘 엮어서 프로그램이 제대로 동작하게 하는 모듈은 controller 라고 할 수 있습니다. 즉 버튼이 눌려지는 이벤트에 반응한다던지, Exe (또는 =) 버튼이 눌렸을때, Model에 계산을 요청하고 결과를 받아 View에 Display 되도록 하는 일들을 controller가 하는 역할입니다. (제가 이해하고 있는 MVC 모델은 그렇습니다. ^^)
그림으로 보면 다음 그림과 같은 구조겠죠.

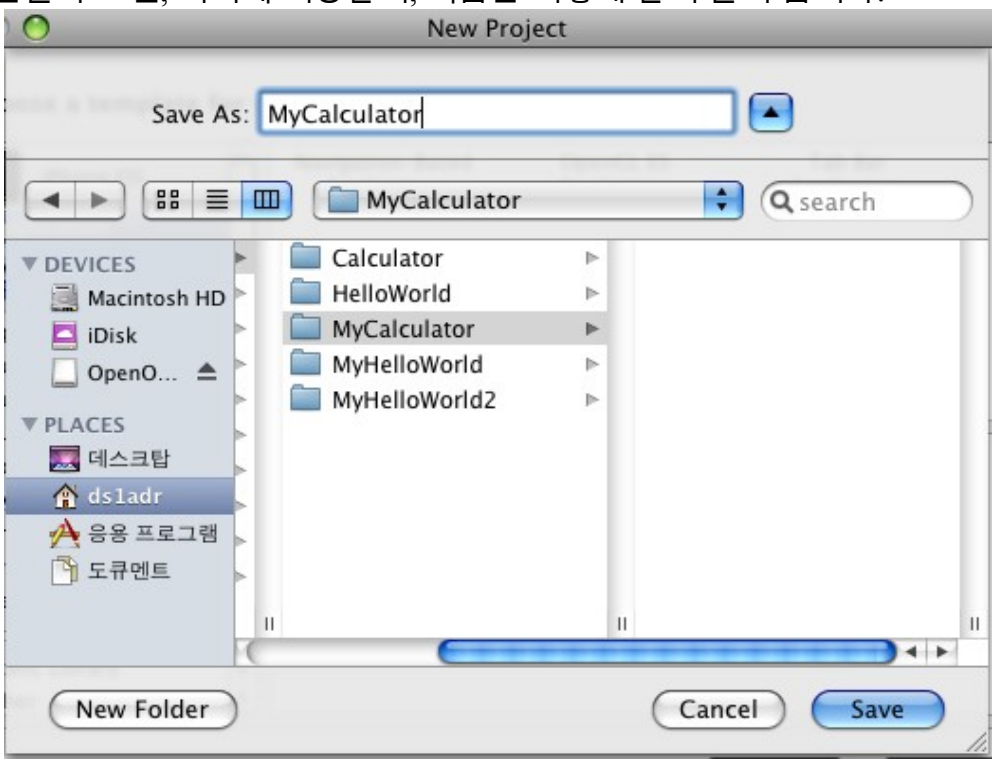


여기서 제가 주로 설명 드릴 부분은 View와 Controller의 생성및 구조, 어떻게 연결 시키는지에 대해서 주로 설명을드릴 겁니다. Model이야 프로그램마다 다 달라지는 부분이고, 제각각일테니까요.

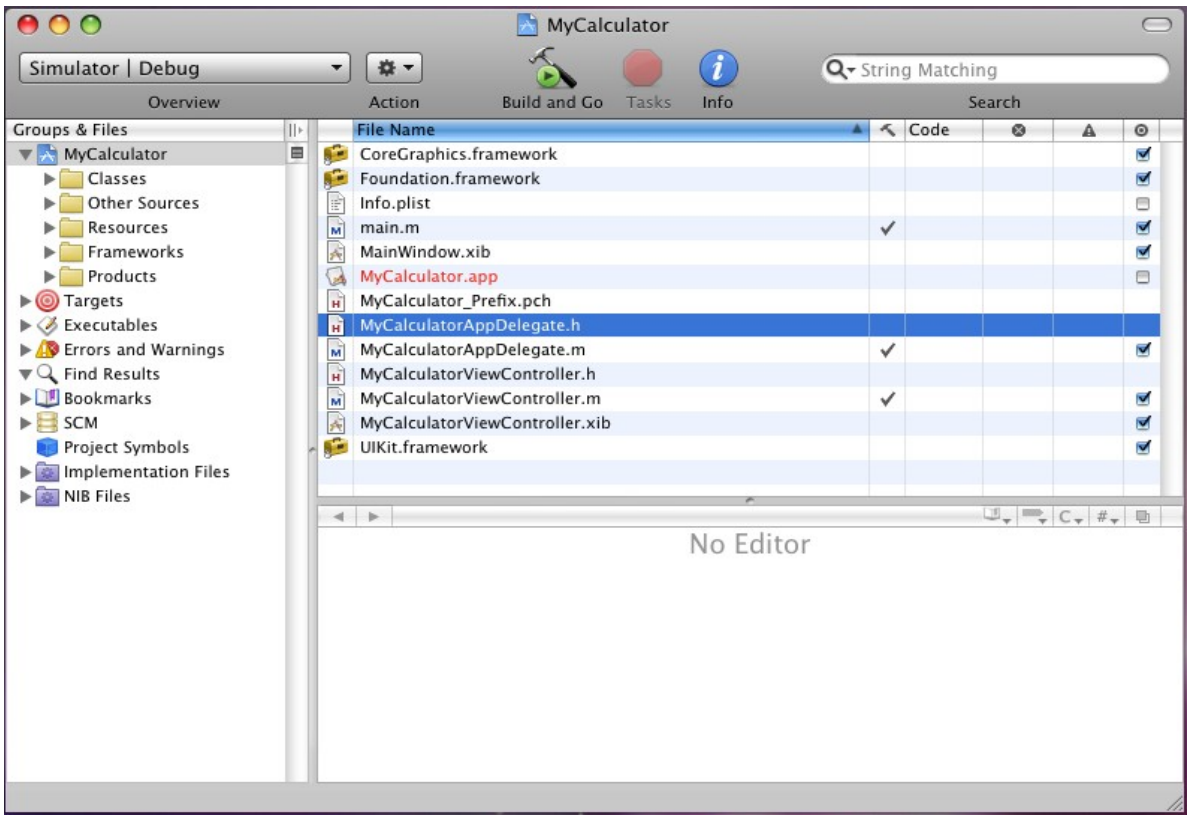
이제 드디어 시작하도록 하겠습니다.
Xcode를 실행시키고 File → New Project … 메뉴를 클릭합니다.
그림에서, 왼쪽 리스트에 iPhone SDK가 설치 되어 있으면 iPhone OS 메뉴에 Application 을 선택하고, 그 오른편 application 종류에는 View-Based Application 을 선택합니다. 참고로 선택할수 있는 Application종류로는 Navigation-Based Application, OpenGL ES Application, Tab Bar Application, Utility Application, View-Based Application, Window-Based Application 이 있습니다. View Based 와 Window-Based의 차이는 프로그램이 윈도우를 통째로 사용하는지 (단일 윈도우) 아니면 다중 윈도우인지 차이라고 어디선가 읽은 기억이 있네요.
두가지 선택을 하고, choose 버튼을 클릭 합니다.



Choose 버튼을 누르면, 어디에 저장할지, 이름은 어떻게 할지 물어 봅니다.



적당히 아무 곳이나 디렉토리를 만들어서 적당한 이름으로 (저는 여기서 둘다 MyCalculator 라고 했습니다.) 저장 하시면 됩니다. Save 버튼을 누르면 기본적이 file들이 생성이 됩니다.

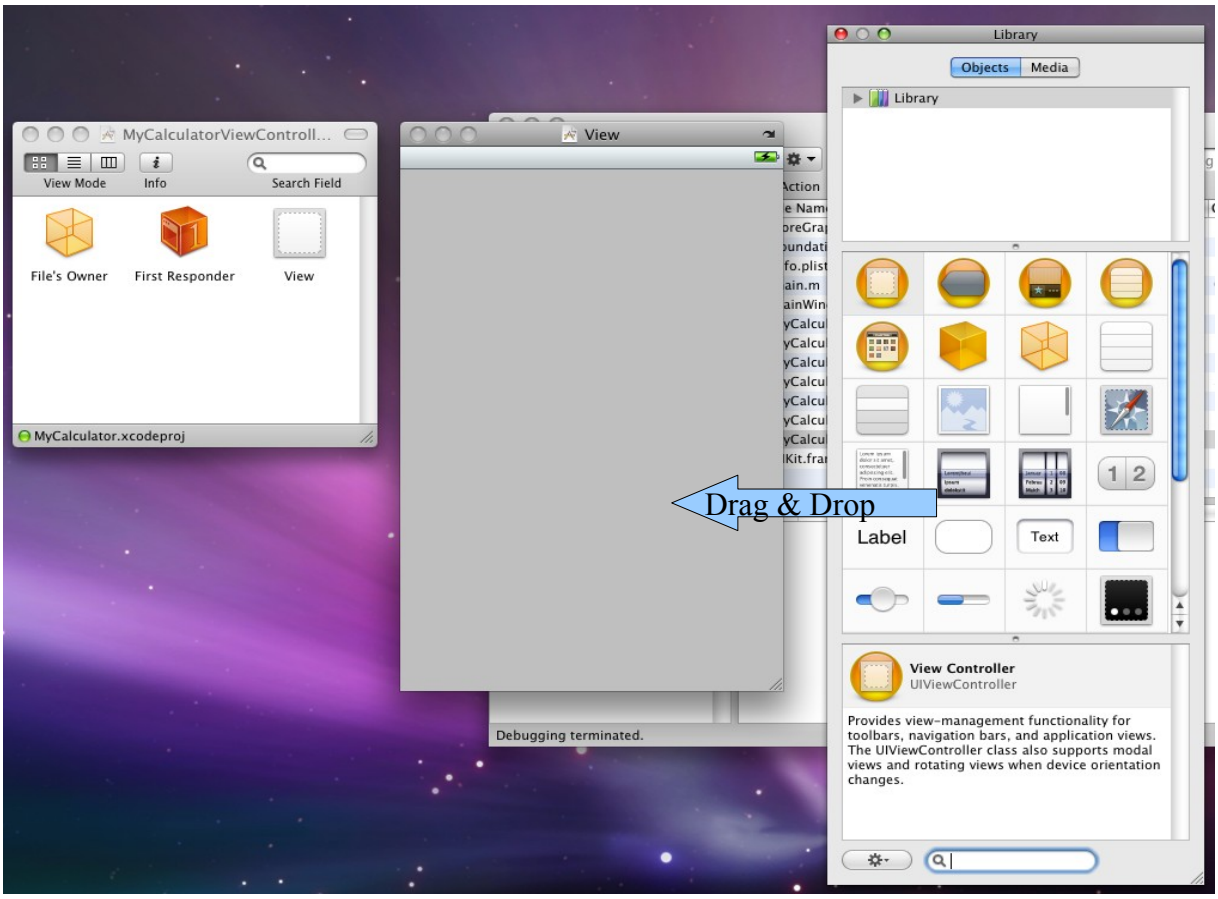


여기서, 솔직히 모든 파일의 세부 사항까지는 잘 모르겠고요. (저도 아직 초보인지라...) 저희가 신경을 써줄 파일은 MyCalculatorViewController.h, MyCalculatorViewController.m, MyCalculatorViewController.xib 이렇게 세 파일입니다. MVC 모델중 Controller 에 해당하고 View 부분은 xib 파일을 Interface Builder 를 이용해서 만들게 됩니다. 계산에 해당하는 Model 부분은 직접 새로 파일을 생성시켜 만들어 줘야 합니다.

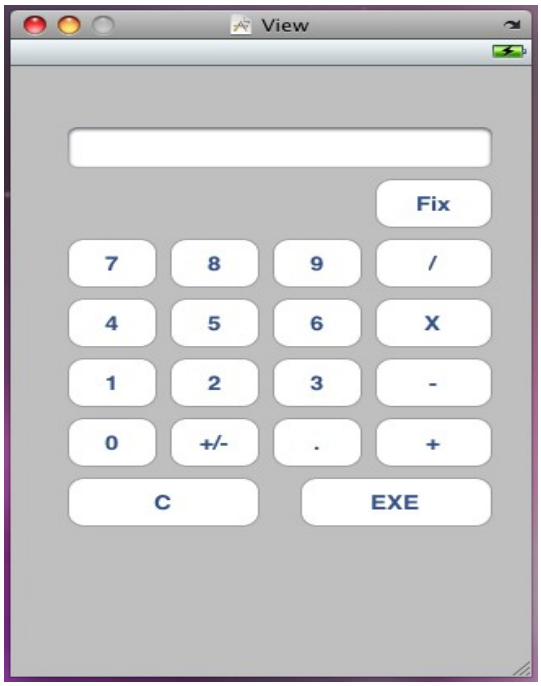
이상태도 완전하게 컴파일및 실행이 됩니다. 물론 프로그램은 아무 view component도 없고 아무일도 하지 않는 프로그램이 되겠죠. 위 그림에서 중앙 상단에 망치 아이콘이 있는 Build and Go 를 클릭하면 컴파일되면서 iPhone Simulator 가 수행 됩니다.
실행시켜 본다고 손해 보는건 아니니 한번 클릭해 보면 다음과 같이 수행 됩니다.
왼쪽에 iPhone 시뮬레이터가 수행 되었고 회색 화면에 아무것도 없죠... 또 아무일도 하지 않는 프로그램입니다. (제 Mac이 사양이 제일 낮은 거라 그런지 처음 실행 시킬때는 시간이 조금 걸리네요. 그래봐야 5~7초 정도 입니다만... 워낙 우리나라 사람들은 바로 안뜨면 '왜이리 늦어' 이런 습성이 있기 때문에.... ^^)

이제 본격적으로 프로그램에 살을 붙이는 작업을 하겠습니다.

일단 GUI 구성하는것 부터 해보면... 위의 그림에서 MyCalculatorViewController.xib 를 더블 클릭하면 Interface Builder 가 수행됩니다. 다음 그림과 같습니다.



여기서 library는 사용 가능한 GUI component들의 모임입니다. 이중 계산기 프로그램에서 쓰게될 component는 button 과 text field가 되겠지요. Button 은 위 그림 라이브러리에서 위에서 5번째, 왼쪽 2번째 항목입니다. (Label 과 Text 라고 씌여진 항목 사이에 끼어 있는 component) 드래그 & 드롭 으로 view 의 적당한 위치에 놓습니다. 크기를 조절하고 버튼을 더블 클릭하면 글자를 넣을 수도 있습니다. UIButton reference를 보면 setImage 함수를 이용해서 image도 넣을 수 있는것 같지만 여기서는 일단 Text만 사용하겠습니다.
Text 까지 입힌 최종 GUI 형태입니다.



이제 어쩌면 이글에서 가장 중요한, controller와 위의 GUI 를 연결 시켜 주는 작업이 남았네요. 어차피 계산기 프로그램 본체는 큰 의미가 있는 것은 아니기 때문에, 이부분이 가장 중요한 작업이 되겠군요. 연결 시켜줄 항목은 두가지 종류가 있습니다. 첫번째로 controller상 변수와 GUI component를 연결 시켜 줄수 있고요, 두번째는 GUI 에서 발생되는 이벤트에 대한 이벤트 핸들러와 연결 시켜주는 작업입니다. 변수는 꼭 연결 시켜야 하는건 아니고 필요한 경우만 연결 시키면 됩니다.

고민을 해야할 시간이군요. 어떤 component를 변수와 연결시켜 다른 곳에서 변수를 access 하도록 하는게 좋을까요... 우선 숫자키는 꼭 그렇게 하지 않아도 상관 없을 것 같습니다. 사칙 연산자도 마찬가지인것 같고요. Fix 라는 버튼은 숫자를 부동 소숫점 형태로 보이게 할건지, 지수 형태로 보이게 할 건지 선택을 하기 위해 만들었습니다. 버튼을 누르면 title이 Exp 로 바뀌면서 지수형태로 바뀌게 할 부분입니다. 그렇다면 계산을 하고 최종 값을 보여줄때 title을 읽어 어떤 title인지에 따라 출력 형태가 달라지니까, 이 버튼은 controller의 변수와 연결 시켜 놓는게 좋을듯 하네요. 그리고 값을 보여주는 TextField도 값을 읽어 와야 하니까 있는게 나을 듯 합니다. (사실 꼭 이렇게 해야 하는건 아니고 클릭할때마다 별도의 BOOL 형 변수의 값이 바뀌게 하는 방법도 생각해 볼 수 있을 듯 합니다만... 여기서는 변수와 연결 시키는게 목적이므로... ^^)

그 다음은 이벤트 핸들러입니다.
일단 가장 쉬운 숫자키는 각각 버튼마다 함수를 다르게 지정할 수 도 있겠지만 편의상 pressNum이라는 함수를 버튼 터치 라는 이벤트에 반응하도록 연결 시키고, 사칙 연산은

pressOperator... 이런식으로 부호, 실행, 리셋(C), 그리고 소숫점 버튼들도 적당한 함수를 만들고 연결시키는 작업을 해 주면 됩니다.

Xcode의 MyCalculator 작업 창에서 MyCalculatorViewController.h 와 MyCalculatorViewController.m 을 수정할 차례입니다.
먼저 헤더 파일을 더블 클릭하면 에디터가 뜹니다. 최초에 자동으로 입력 되어 있는 부분은 다음과 같이 내용이 비어 있는 클래스입니다.

```
//
//  MyCalculatorViewController.h
//  MyCalculator
//
//  Created by wontai ki on 08. 11. 06.
//  Copyright S 2008. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface MyCalculatorViewController : UIViewController {

}

@end
```

View의 component와 연결시킬 변수는 앞에 IBOutlet 이라고 붙여줘야 나중에 Interface Builder 에서 인식을 하게 됩니다. 출력 형식을 지정하기 위한 버튼과 textField를 변수와 연결시키기로 했으니 @interface 의 {} 괄호 사이에 다음줄을 추가 합니다.

```
    IBOutlet UIButton *_outputTypeButton;
    IBOutlet UITextField *_outputField;
```

이줄은 다음과 같이 바꿀 수 도 있습니다.

```
    IBOutlet id _outputTypeButton;
    IBOutlet id _outputField;
```

id 라는 변수 형은 클래스의 인스턴스 형입니다. 객체의 포인터 라고 이해 하셔도 될것 같고, 아니면 C 언어 식이면 void pointer 와 비슷한 개념이 될것 같습니다.
id는 임의의 객체를 가리킬수 있기 때문에 어떤 함수가 오더라도 상관 없습니다. 이는 Obj-C가 기본적으로 동적 바인딩 방식이라 가능한것 같습니다.
하지만 개인적으로는 혹시라도 있을 실수에 대비하기 위해서는 확실하게 버튼의 포인터라라는 식으로 위의 방법을 사용하는것이 더 낫지 않나 싶습니다.

다음은 getter, setter함수를 자동으로 설정해 주는, property 를 { } 괄호 다음에 추가 합니다. 다음과 같습니다.

```
@property (nonatomic, retain) UIButton *_outputTypeButton;
@property (nonatomic, retain) UITextField *_outputField;
```

property만 지정했다고 무조건 만들어 주는것은 아니고 나중에 m file에 클래스의 구현부에서 @synthesize 문으로 지정해 줘야 합니다. 해당 부분 설명할때 다시 설명하겠습니다. nonatomic은 thread safe하지 않아도 된다는 의미고요. 그런데 솔직히 retain은 의미를 잘 모르겠네요. ^^ 죄송 저도 아직은 초보인지라...

이제 이벤트에 대한 핸들러를 만들 차례입니다. 이벤트 핸들러는 리턴 타입을 IBAction이라고 지정해야 마찬가지로 Interface Builder에서 인식 가능합니다.
Property 아래 줄에 위에서 설명했던 pressNum 과 pressOperator 등을 선언합니다.

```
-(IBAction) pressNum:(id)sender;
-(IBAction) pressOperator:(id)sender;
-(IBAction) pressDot:(id)sender;
-(IBAction) pressExe:(id)sender;
-(IBAction) pressOutputType:(id)sender;
-(IBAction) pressSign:(id)sender;
-(IBAction) pressClear:(id)sender;
```

생각보단 많네요. 사실 프로그래밍에 정답은 없고, 꼭 이렇게 해야 하는건 아닙니다. 그냥 pressButton 하나만 선언하고 그 함수 안에서 어떤 키가 눌려졌는지를 체크해서 일일이 처리 해 줄 수도 있지만... 그래도 type별로 구분해 놓는게 더 깔끔하지 않나 싶네요.
그리고 매개변수에 (id)sender라는게 조금 생소하실지 모르겠는데, 이는 메세지를 보내는 객체의 instance입니다.

이 외에도 필요한 함수가 있으면 추가로 더 선언해서 사용해도 됩니다.
여기까지 전체 source입니다.

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

@interface MyCalculatorViewController : UIViewController {
    IBOutlet UIButton *_outputTypeButton;
    IBOutlet UITextField *_outputField;

}
@property (nonatomic, retain) UIButton *_outputTypeButton;
@property (nonatomic, retain) UITextField *_outputField;

-(IBAction) pressNum:(id)sender;
-(IBAction) pressOperator:(id)sender;
-(IBAction) pressDot:(id)sender;
-(IBAction) pressExe:(id)sender;
-(IBAction) pressOutputType:(id)sender;
-(IBAction) pressSign:(id)sender;
-(IBAction) pressClear:(id)sender;
@end
```

다음은 실제 구현부인 m file을 수정하겠습니다. 작업창에서 MyCalculatorViewController.m 을 더블클릭 하시면 다음과 같이 기본으로 만들어져 있는 소스가 뜹니다.

```
//
//  MyCalculatorViewController.m
//  MyCalculator
//
//  Created by wontai ki on 08. 11. 06.
//  Copyright S 2008. All rights reserved.
//

#import "MyCalculatorViewController.h"

@implementation MyCalculatorViewController



/*
// Override initWithNibName:bundle: to load the view using a nib file then
perform additional customization that is not appropriate for viewDidLoad.
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    if (self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil]) {
        // Custom initialization
    }
    return self;
}
*/

/*
// Implement loadView to create a view hierarchy programmatically.
- (void)loadView {
}
*/



/*
// Implement viewDidLoad to do additional setup after loading the view.
- (void)viewDidLoad {
    [super viewDidLoad];
}
*/


- (BOOL)shouldAutorotateToInterfaceOrientation:
(UIInterfaceOrientation)interfaceOrientation {
    // Return YES for supported orientations
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}


- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning]; // Releases the view if it doesn't have a
superview
    // Release anything that's not essential, such as cached data
}
```

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

@end
```

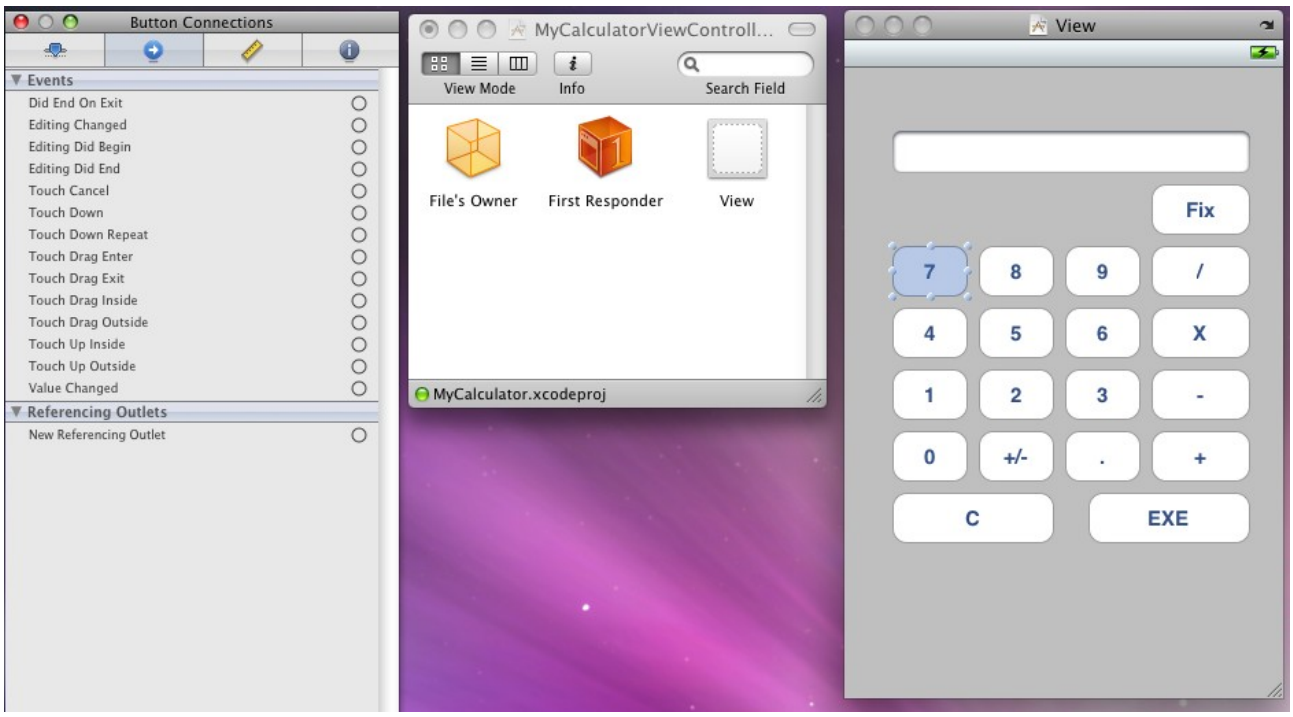
일단 편의상 헤더에서 선언했던 함수의 내용은 빼고 빈 파일만 빨리 만들어 연결 시키는 방법을 빨리 보여 드리겠습니다. Getter, setter를 만들기 위해 @implementation 다음에 @synthesize 를 추가하고 각 함수의 본체를 만들겠습니다.
최종 소스 입니다.

```
#import "MyCalculatorViewController.h"

@implementation MyCalculatorViewController
@synthesize _outputTypeButton, _outputField;

-(IBAction) pressNum:(id)sender
{
}

-(IBAction) pressOperator:(id)sender
{
}

-(IBAction) pressDot:(id)sender
{
}

-(IBAction) pressExe:(id)sender
{
}

-(IBAction) pressOutputType:(id)sender
{
}

-(IBAction) pressSign:(id)sender
{
}

-(IBAction) pressClear:(id)sender
{
}


/*
// Override initWithNibName:bundle: to load the view using a nib file then
perform additional customization that is not appropriate for viewDidLoad.
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    if (self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil]) {
        // Custom initialization
```

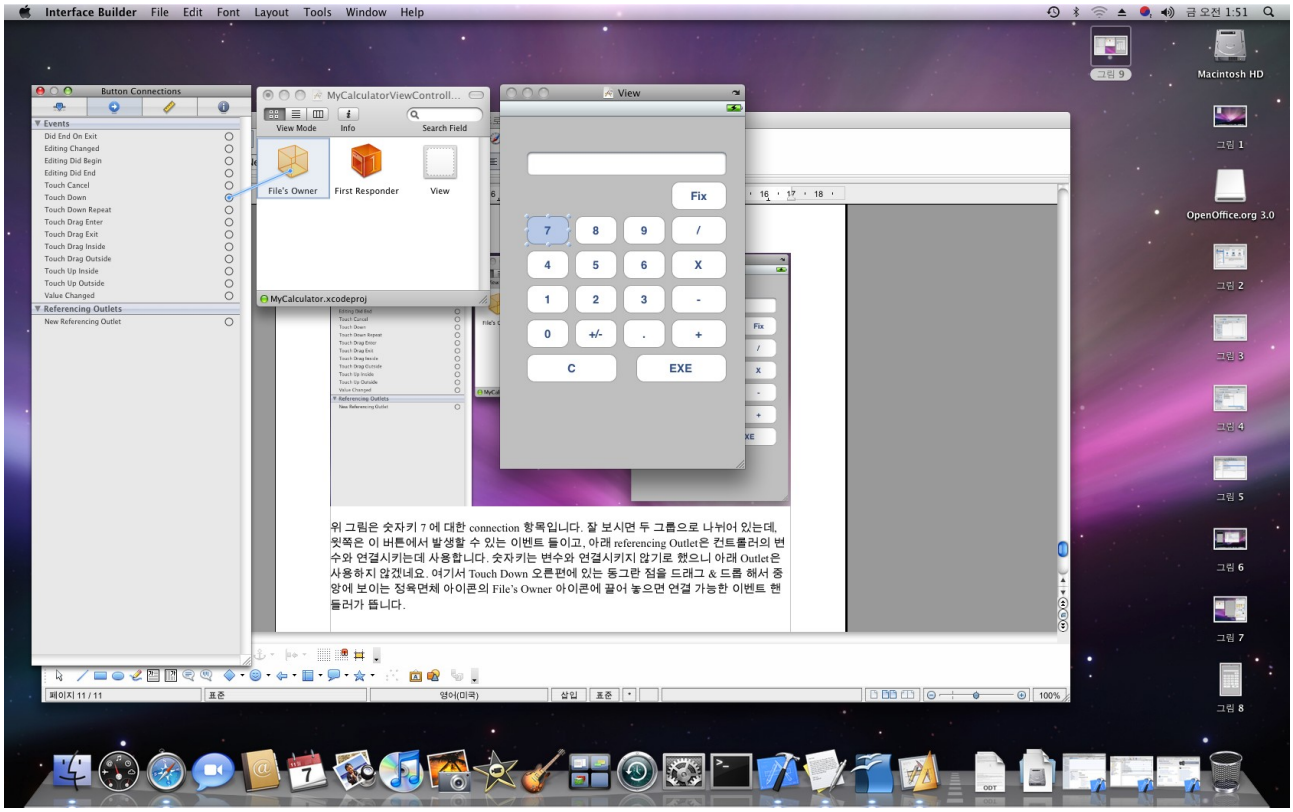
```
    }
    return self;
}
*/

/*
// Implement loadView to create a view hierarchy programmatically.
- (void)loadView {
}
*/


/*
// Implement viewDidLoad to do additional setup after loading the view.
- (void)viewDidLoad {
    [super viewDidLoad];
}
*/


- (BOOL)shouldAutorotateToInterfaceOrientation:
(UIInterfaceOrientation)interfaceOrientation {
    // Return YES for supported orientations
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}


- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning]; // Releases the view if it doesn't have a
superview
    // Release anything that's not essential, such as cached data
}


- (void)dealloc {
    [super dealloc];
}

@end
```

이제 기본 구성은 다 끝나고 연결 시켜주고 실제 함수의 내용을 짜면 되겠네요.
일단 이번 글에서는 연결 시키는 것까지만 하겠습니다. (글을 쓰면서 만들다 보니 생각보다 시간이 많이 드네요... 헥헥...)

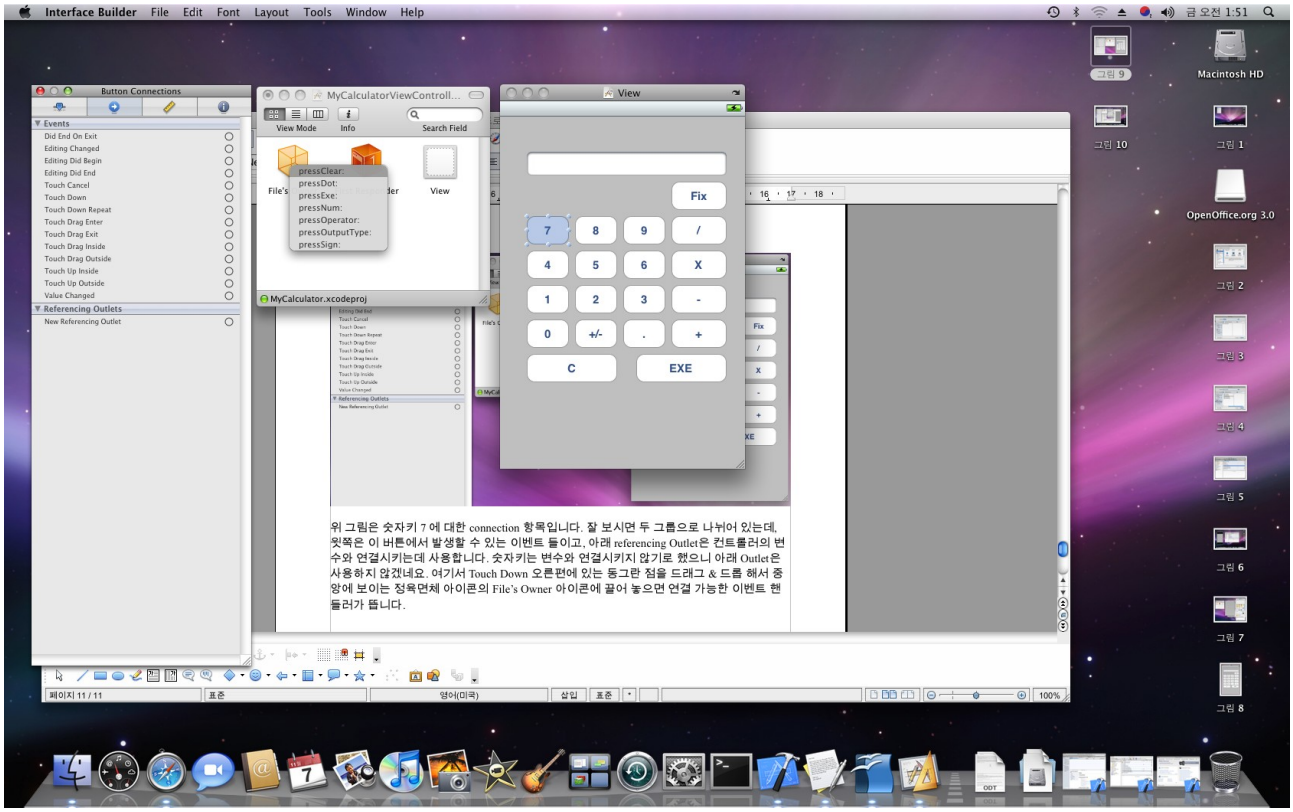
이번엔 다시 Interface Builder를 활성화 시키시고, Tools → connection inspector 메뉴를 수행해서 connection 창을 엽니다. 여기까지의 상태를 캡쳐했습니다.



위 그림은 숫자키 7 에 대한 connection 항목입니다. 잘 보시면 두 그룹으로 나뉘어 있는데, 윗쪽은 이 버튼에서 발생할 수 있는 이벤트 들이고, 아래 referencing Outlet은 컨트롤러의 변수와 연결시키는데 사용합니다. 숫자키는 변수와 연결시키지 않기로 했으니 아래 Outlet은 사용하지 않겠네요. 여기서 Touch Down 오른편에 있는 동그란 점을 드래그 & 드롭 해서 중앙에 보이는 정육면체 아이콘의 File's Owner 아이콘에 끌어 놓으면 연결 가능한 이벤트 핸들러가 뜹니다.
다음의 두 그림에 과정이 잘 나타나 있습니다.

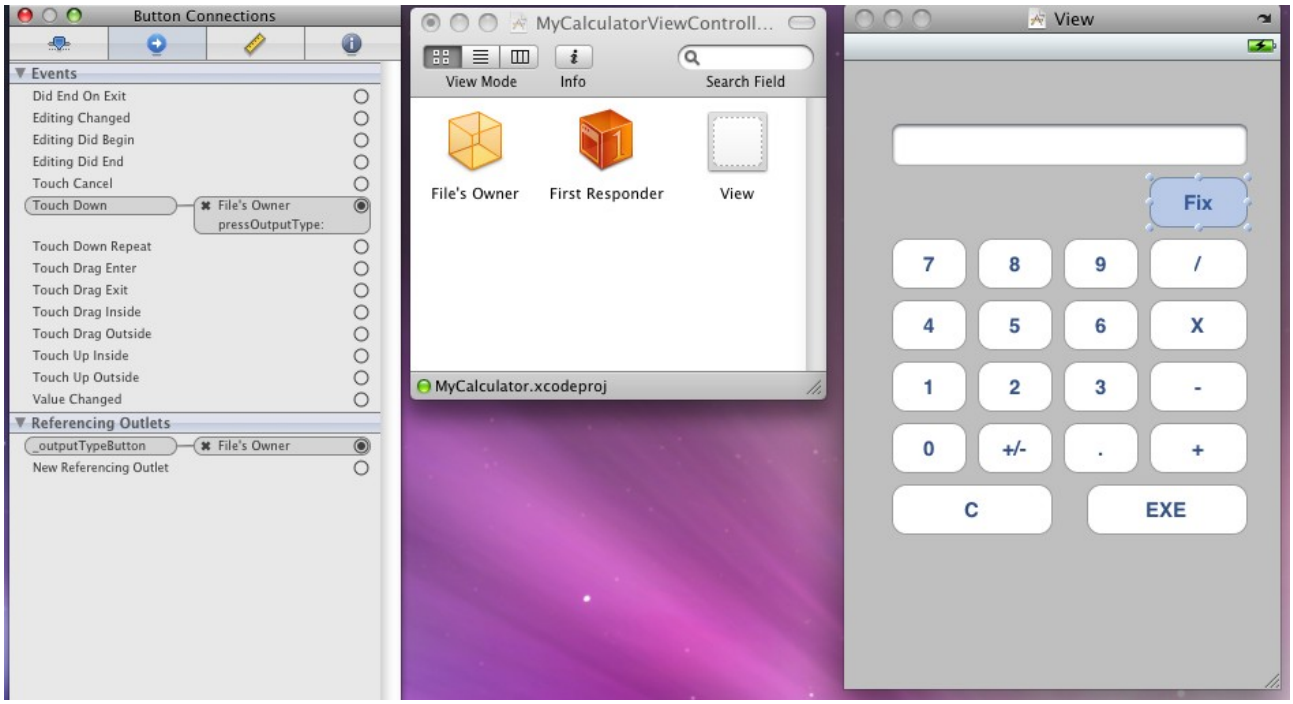


드래그를 하면 파란선으로 연결이 되고 이상태에서 drop을 하면...



연결 가능한 함수 (이벤트 핸들러) 목록이 보입니다.
여기서는 pressNum을 선택합니다. (숫자 키 이므로...) 마찬가지 방법으로 모든 버튼의 touch down 이벤트를 맞는 핸들러와 연결 시킵니다. 변수와 연결하는 것도 똑같은 방법입니다. Fix button을 클릭해서 선택하신후 Referencing Outlet 의 New Referencing Outlet 오른편 동그란 점을 드래그 & 드롭 하면 연결 가능한 변수 목록이 뜨고, 선택하면 됩니다.

다음 그림은 Fix 버튼에대해 이벤트 핸들러와 변수 연결 후에 나타나는 모습입니다.



연결을 해제할때는 Button connections 의 File's owner 라고 씌여진 부분 바로 앞의 조그만 X

표를 클릭하면 됩니다.

여기까지 기본적으로 프로젝트를 생성하고 view와 controller 간 변수와 이벤트 핸들러와 연결 시켜 주는 부분까지 설명을 드렸습니다. 쉽게 느껴 지셨는지 모르겠습니다.
이제 각각의 핸들러 함수를 완전히 마무리 하고, 실제 계산을 담당하는 클래스를 만들어 controller에서 객체를 만들어 Exe 버튼이 눌렸을때 계산 class가 계산을 수행해서 return 한 결과를 제일 상단의 TextField에 뿌려 주면 프로그램이 완성 됩니다.