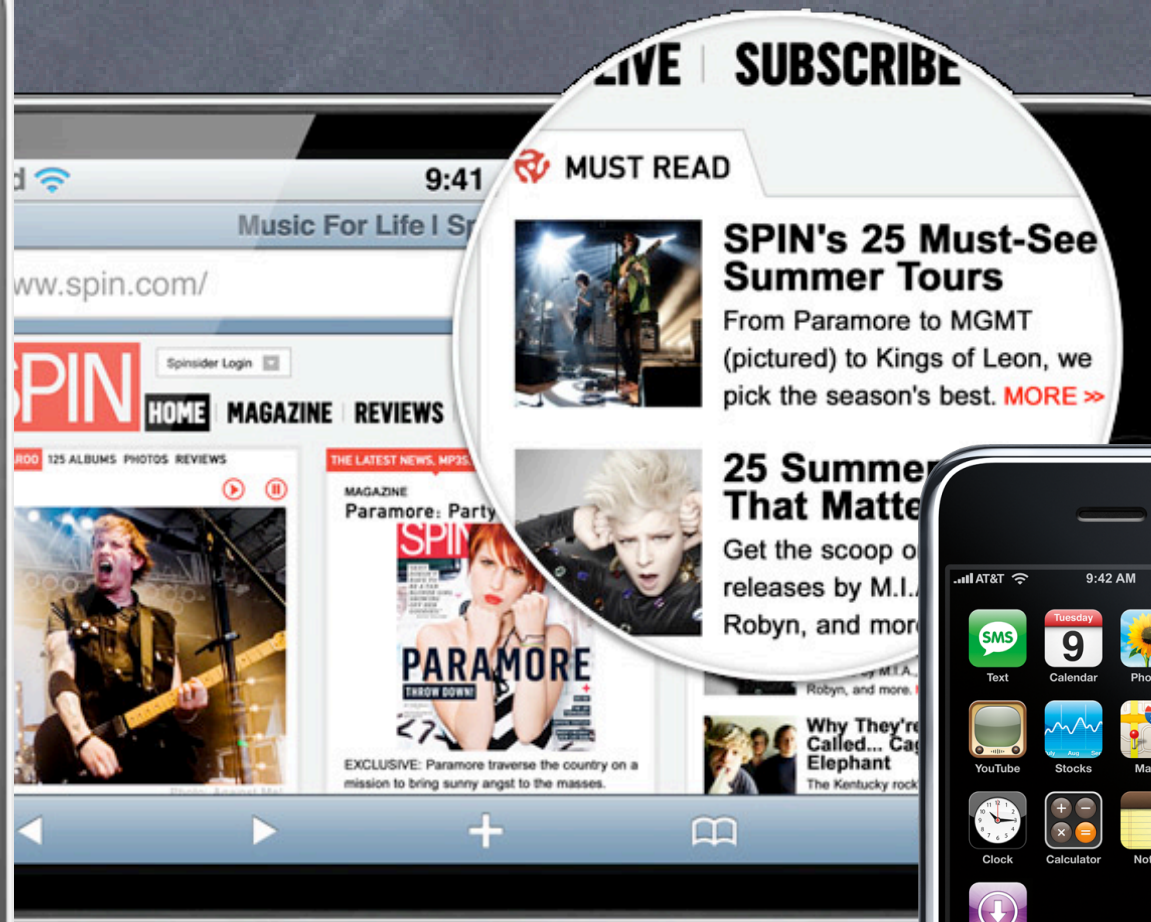


Stanford CS193p

Developing Applications for iPhone 4, iPod Touch, & iPad
Fall 2010



Today

- **MVC**

 - Calculator

- **Objective-C**

 - Declaring and implementing objects

 - Sending messages between objects

- **Interface Builder**

 - Graphically creating your View

 - “Wiring up” objects to send messages to each other

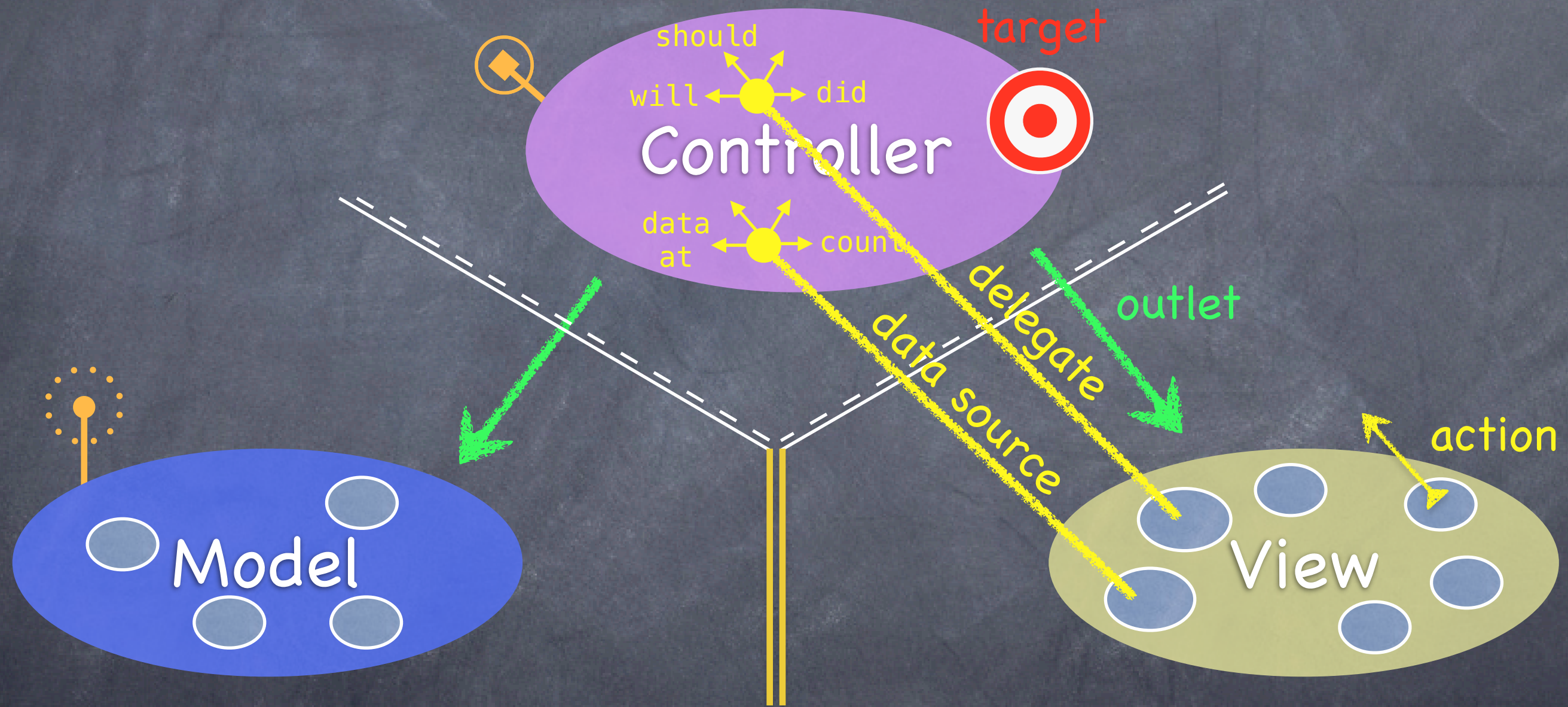
 - Setting the properties of objects

- **Xcode**

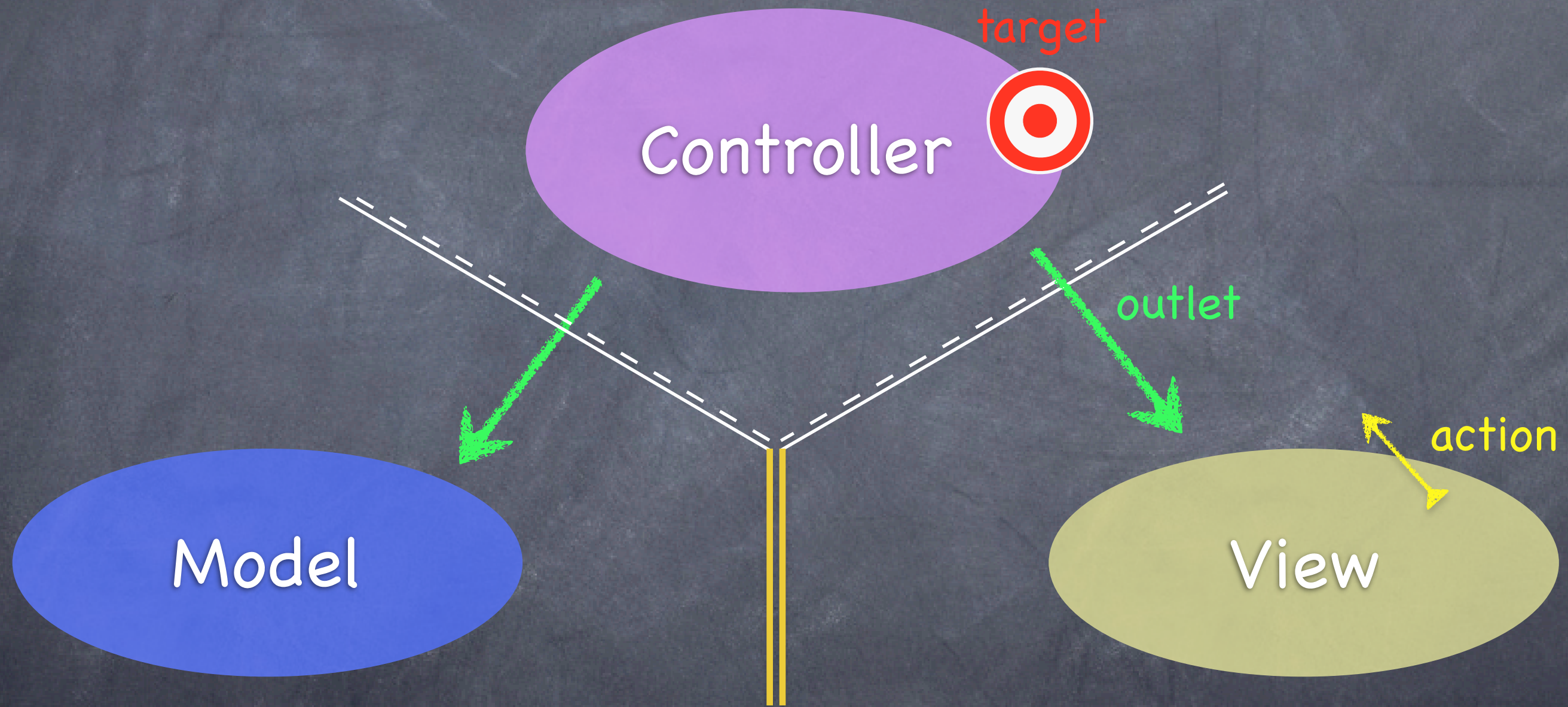
 - Managing and editing your code

 - Running your application in the simulator

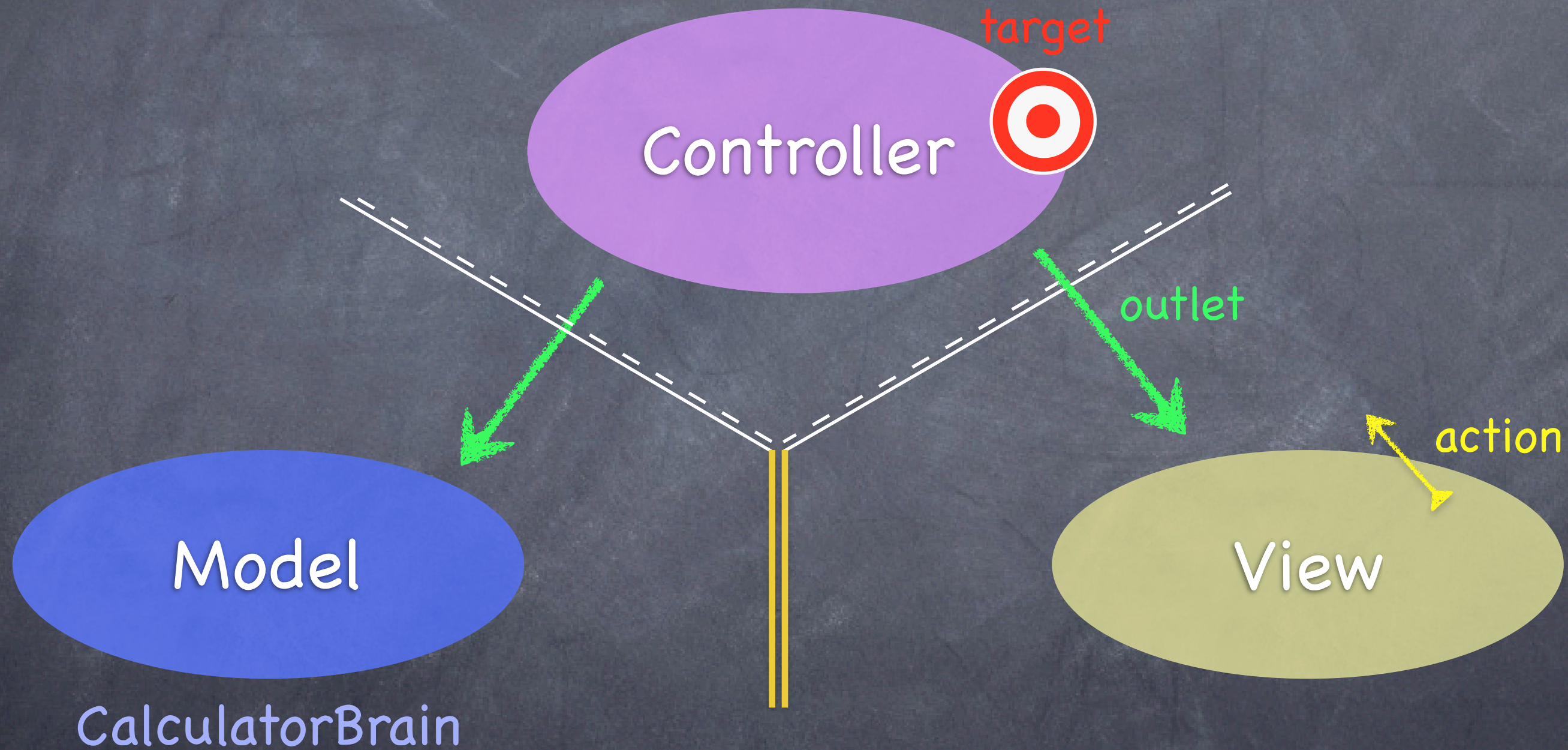
Calculator MVC



Calculator MVC

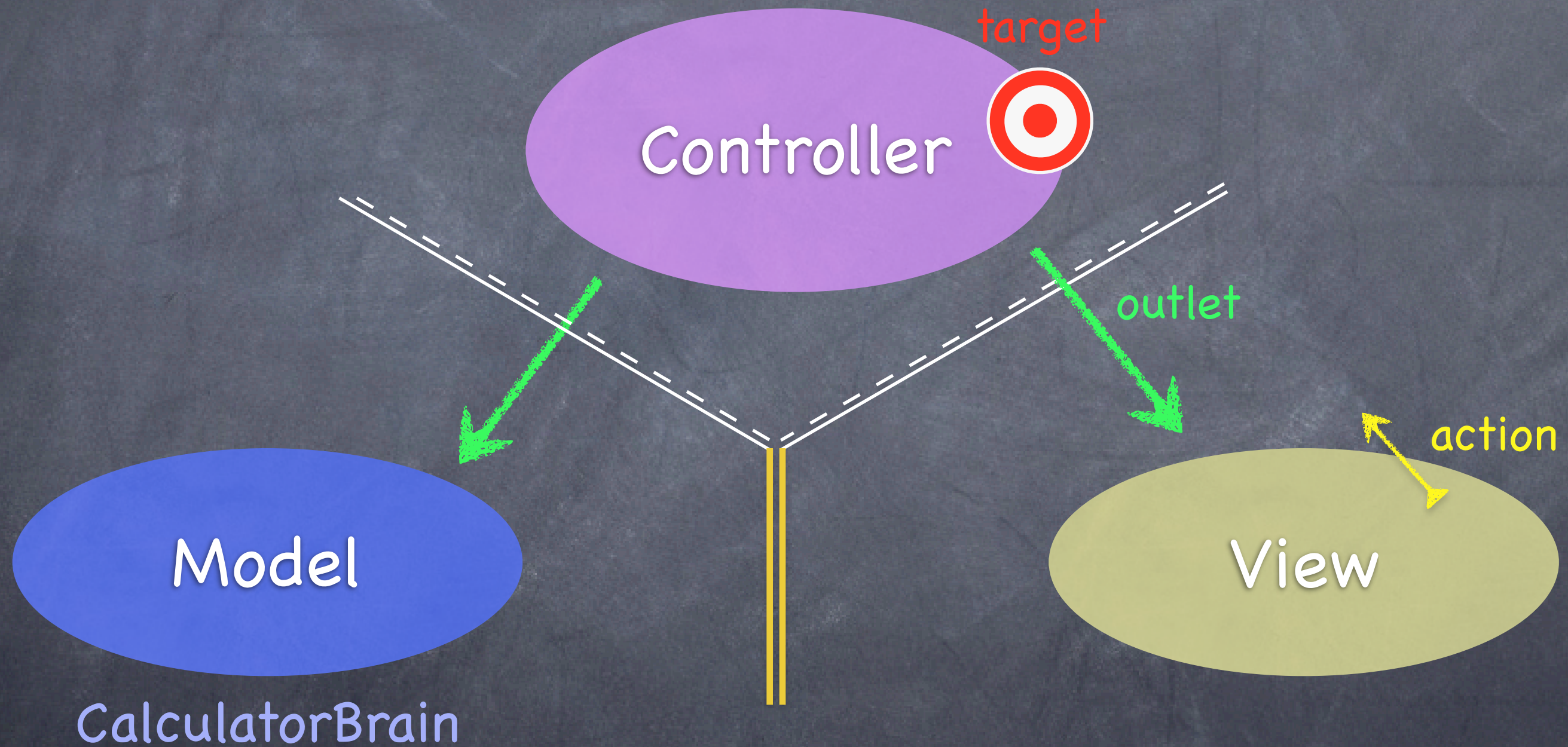


Calculator MVC



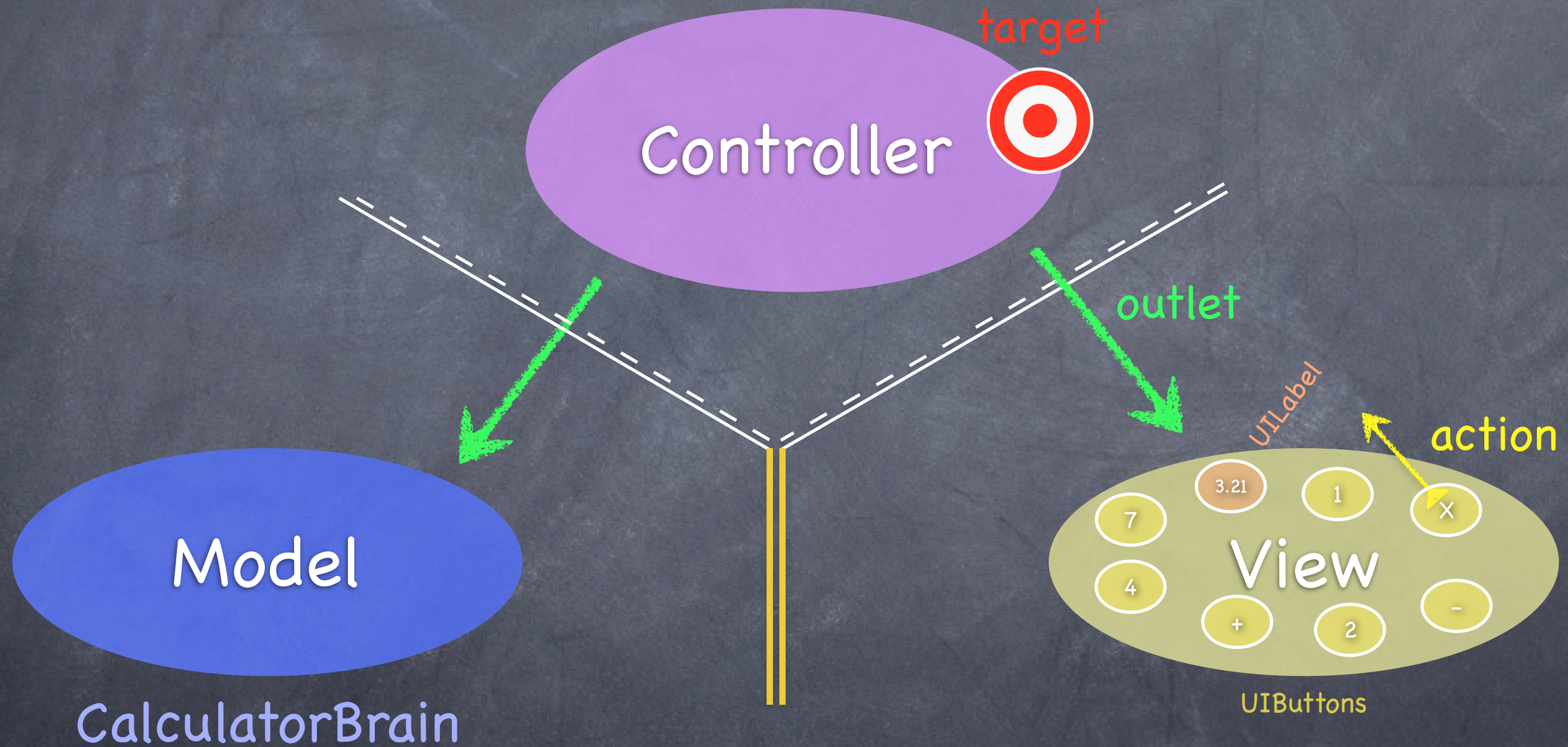
Calculator MVC

CalculatorViewController



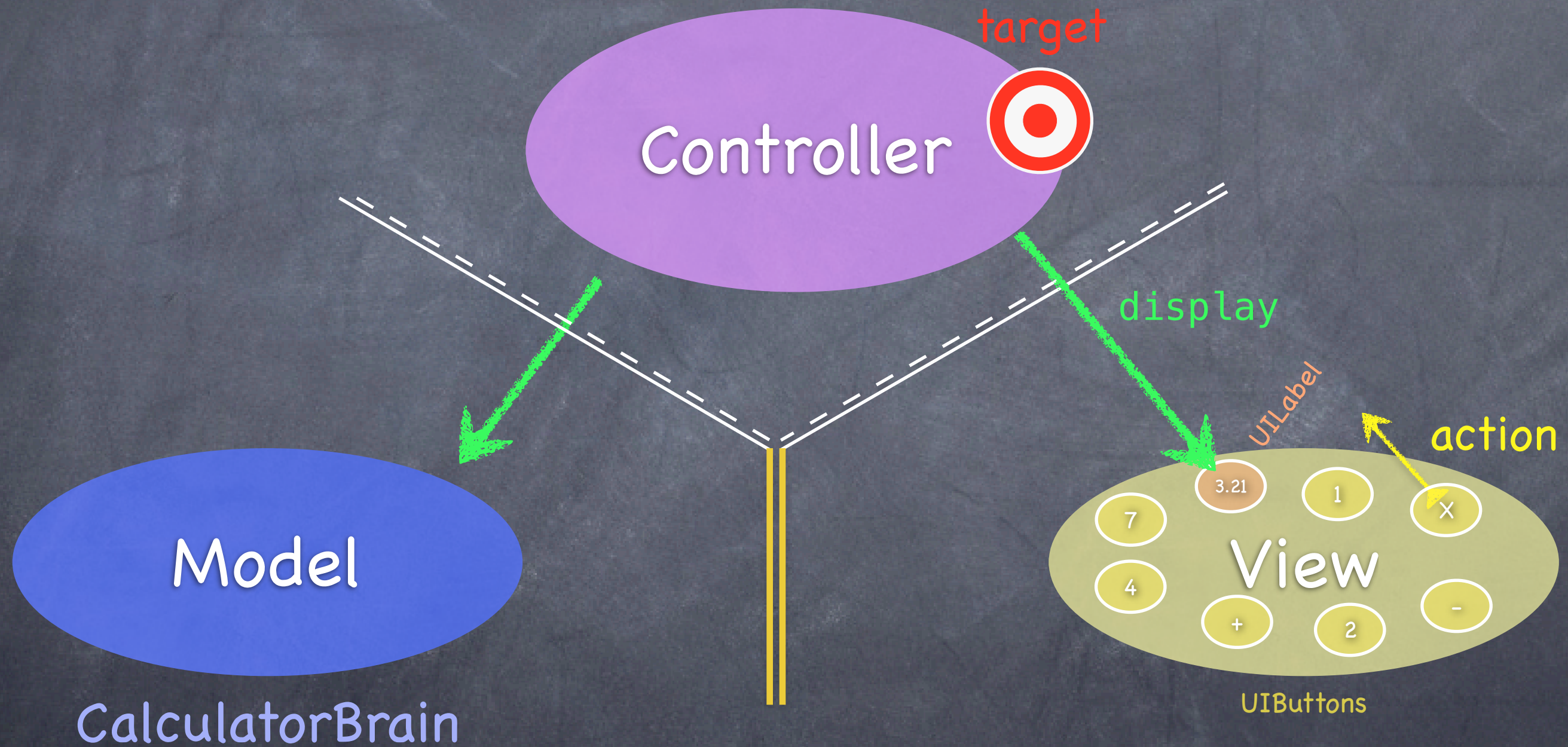
Calculator MVC

CalculatorViewController



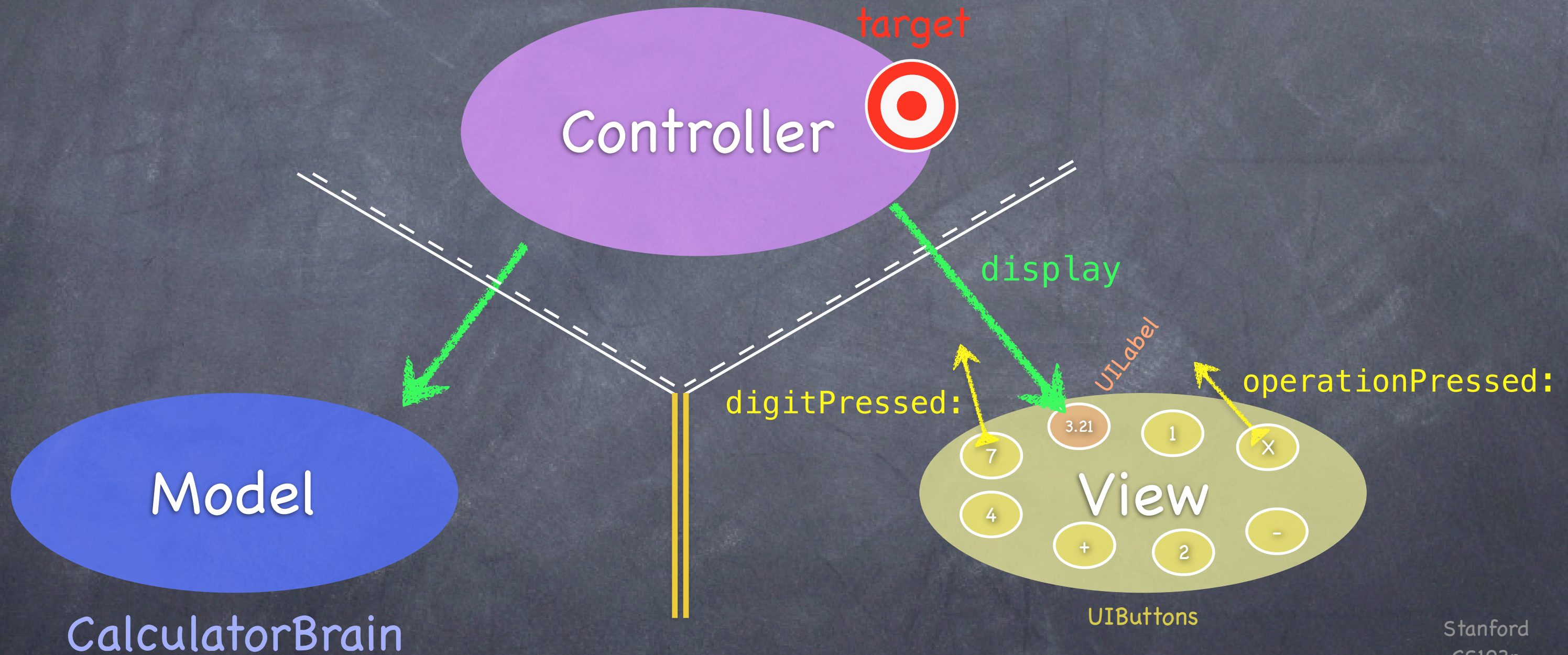
Calculator MVC

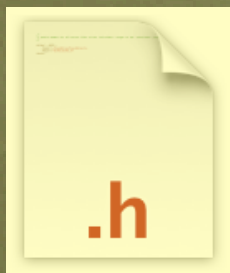
CalculatorViewController



Calculator MVC

CalculatorViewController

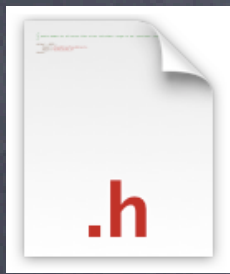




CalculatorBrain.h

This is the header file for this class.
It documents its public API.

Model



CalculatorBrain.h

Model

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

```
@interface CalculatorBrain : NSObject
```

The name of this class.

```
@end
```



CalculatorBrain.h

Model

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

```
@interface CalculatorBrain : NSObject
```

This class's superclass.

```
@end
```



CalculatorBrain.h

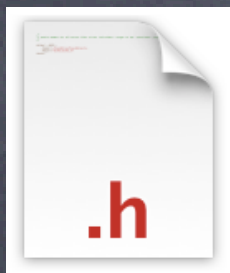
We must import the header for our superclass.

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

```
@interface CalculatorBrain : NSObject
```

```
@end
```

Model



CalculatorBrain.h

Model

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

```
@interface CalculatorBrain : NSObject
```

```
{  
  
}
```

Instance variables go here.

```
@end
```



CalculatorBrain.h

Model

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

```
@interface CalculatorBrain : NSObject  
{  
    double operand;  
}
```

```
@end
```

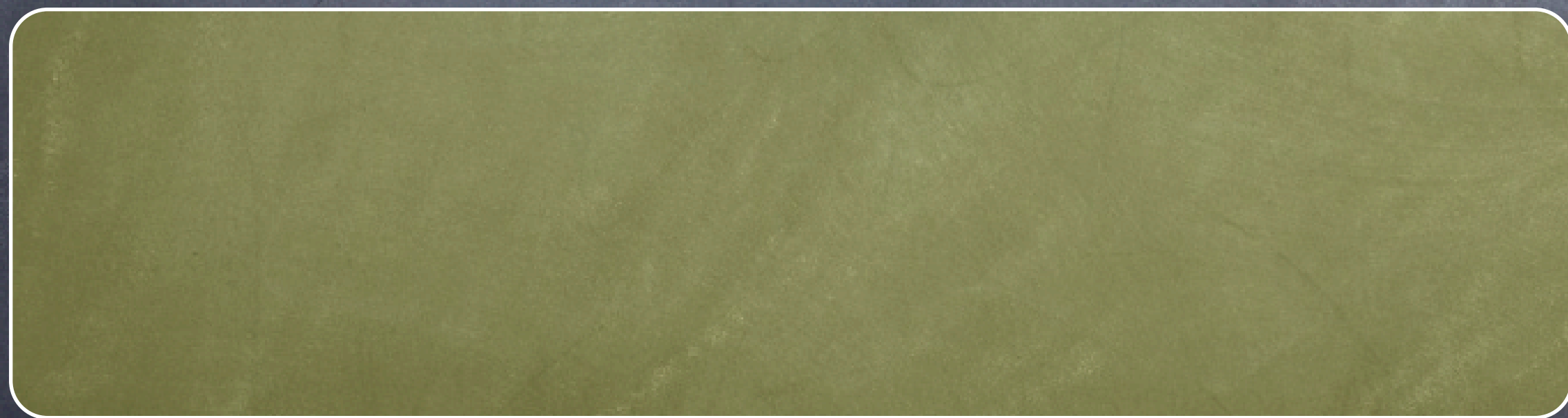


CalculatorBrain.h

Model

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

```
@interface CalculatorBrain : NSObject  
{  
    double operand;  
}
```



Method
declarations
go here.

```
@end
```




CalculatorBrain.h

Model

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

```
@interface CalculatorBrain : NSObject  
{  
    double operand;  
}
```

```
- (void)setOperand:(double)anOperand;
```

```
- (double)performOperation:(NSString *)operation;
```

```
@end
```



CalculatorBrain.h

Model

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

```
@interface CalculatorBrain : NSObject  
{  
    double operand;  
}
```

Specifying void as the return type means
that this method returns no value.

```
- (void)setOperand:(double)anOperand;  
- (double)performOperation:(NSString *)operation;
```

```
@end
```



CalculatorBrain.h

Model

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

```
@interface CalculatorBrain : NSObject  
{  
    double operand;  
}
```

The name of this method is "setOperand:"

```
- (void)setOperand:(double)anOperand;  
- (double)performOperation:(NSString *)operation;
```

```
@end
```



CalculatorBrain.h

Model

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

```
@interface CalculatorBrain : NSObject  
{  
    double operand;  
}
```

It takes one argument, a double called "anOperand"

```
- (void)setOperand:(double)anOperand;  
- (double)performOperation:(NSString *)operation;
```

```
@end
```



CalculatorBrain.h

Model

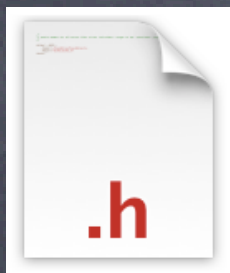
```
#import <Foundation/Foundation.h>
```

```
@interface CalculatorBrain : NSObject  
{  
    double operand;  
}
```

Don't forget a semicolon here!

```
- (void)setOperand:(double)anOperand;  
- (double)performOperation:(NSString *)operation;
```

```
@end
```



CalculatorBrain.h

Model

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

```
@interface CalculatorBrain : NSObject  
{  
    double operand;  
}
```

```
- (void)setOperand:(double)anOperand;
```

```
- (double)performOperation:(NSString *)operation;
```

This method returns a double.

```
@end
```



CalculatorBrain.h

Model

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

```
@interface CalculatorBrain : NSObject  
{  
    double operand;  
}
```

```
- (void)setOperand:(double)anOperand;
```

```
- (double)performOperation:(NSString *)operation;
```

It takes as its argument a pointer to an NSString object.
That's right, we're passing an object to this method.

```
@end
```



CalculatorBrain.h

Model

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

```
@interface CalculatorBrain : NSObject  
{  
    double operand;  
}
```

```
- (void)setOperand:(double)anOperand;
```

```
- (double)performOperation:(NSString *)operation;
```

```
- (NSArray *)foo:(int)zap bar:(id)pow;
```

```
@end
```




CalculatorBrain.h

Model

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

```
@interface CalculatorBrain : NSObject  
{  
    double operand;  
}
```

```
- (void)setOperand:(double)anOperand;
```

```
- (double)performOperation:(NSString *)operation;
```

```
- (NSArray *)foo:(int)zap bar:(id)pow;
```

```
@end
```

This method takes two arguments and is called "foo:bar:"
(pronounced "foo colon bar colon")



CalculatorBrain.h

Model

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

```
@interface CalculatorBrain : NSObject  
{  
    double operand;  
}
```

```
- (void)setOperand:(double)anOperand;
```

```
- (double)performOperation:(NSString *)operation;
```

```
- ((NSArray *)foo:(int)zap bar:(id)pow;
```

```
@end
```

It returns a pointer to an NSArray
(a collection class in Foundation).



CalculatorBrain.h

Model

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

```
@interface CalculatorBrain : NSObject  
{  
    double operand;  
}
```

- (void)setOperand:(double)anOperand;
- (double)performOperation:(NSString *)operation;
- (NSArray *)foo:(int)zap bar:(id)pow;

```
@end
```

The second argument is of type "id".
This means "a pointer to any kind of object!"



CalculatorBrain.h

Model

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

```
@interface CalculatorBrain : NSObject  
{  
    double operand;  
}
```

```
- (void)setOperand:(double)anOperand;
```

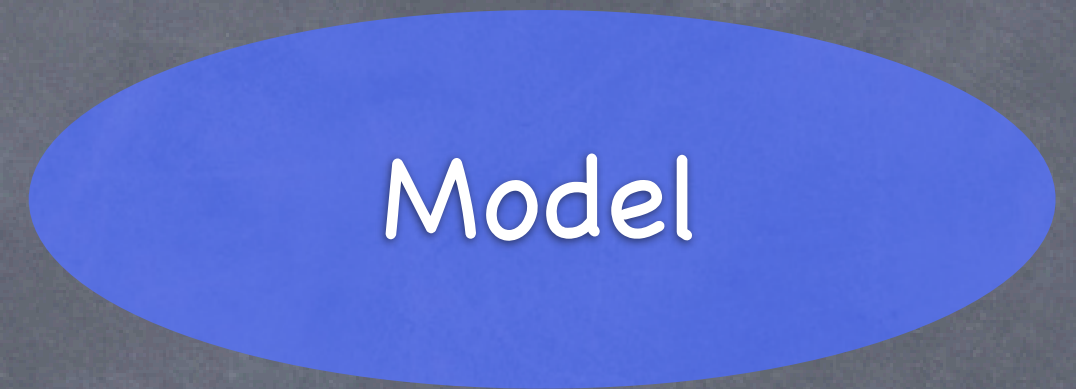
```
- (double)performOperation:(NSString *)operation;
```

```
@end
```



CalculatorBrain.m

This is the implementation file.
Both public and private implementation
goes here.



```
#import "CalculatorBrain.h"
```

```
@implementation CalculatorBrain
```

```
@end
```



CalculatorBrain.m

We must import our own header file.

```
#import "CalculatorBrain.h"
```

```
@implementation CalculatorBrain
```

```
@end
```

Model

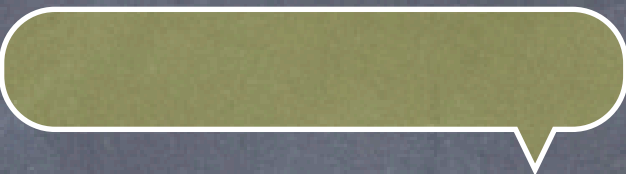


CalculatorBrain.m

Model

```
#import "CalculatorBrain.h"
```

```
@implementation CalculatorBrain
```



Note that we don't specify our superclass in the implementation

```
@end
```



CalculatorBrain.m

Model

```
#import "CalculatorBrain.h"
```

```
@implementation CalculatorBrain
```

No semicolon this time!

```
- (void)setOperand:(double)anOperand
```

```
{
```

```
    <code goes here>
```

```
}
```

```
@end
```




CalculatorBrain.m

Model

```
#import "CalculatorBrain.h"

@implementation CalculatorBrain

- (void)setOperand:(double)anOperand
{
    operand = anOperand;
}

- (double)performOperation:(NSString *)operation
{
    [operation sendMessage:argument];
    return aDouble;
}

@end
```



CalculatorBrain.m

Model

```
#import "CalculatorBrain.h"

@implementation CalculatorBrain

- (void)setOperand:(double)anOperand
{
    operand = anOperand;
}

- (double)performOperation:(NSString *)operation
{
    [operation sendMessage:argument];
    return aDouble;
}

@end
```

Square brackets mean "send a message."



CalculatorBrain.m

Model

```
#import "CalculatorBrain.h"

@implementation CalculatorBrain

- (void)setOperand:(double)anOperand
{
    operand = anOperand;
}

- (double)performOperation:(NSString *)operation
{
    [operation sendMessage:argument];
    return aDouble;
}

@end
```

This is the object to send the message to (in this case, the NSString called "operation" that was passed as an argument to performOperation:).



CalculatorBrain.m

Model

```
#import "CalculatorBrain.h"

@implementation CalculatorBrain

- (void)setOperand:(double)anOperand
{
    operand = anOperand;
}

- (double)performOperation:(NSString *)operation
{
    [operation sendMessage:argument];
    return aDouble;
}

@end
```

sendMessage:argument

This is the message to send.



CalculatorBrain.m

Model

```
#import "CalculatorBrain.h"

@implementation CalculatorBrain

- (void)setOperand:(double)anOperand
{
    operand = anOperand;
}

- (double)performOperation:(NSString *)operation
{
    [operation sendMessage:argument];
    return aDouble;
}

@end
```

And this is its one (in this case) argument.



CalculatorBrain.m

Model

```
#import "CalculatorBrain.h"

@implementation CalculatorBrain

- (void)setOperand:(double)anOperand
{
    operand = anOperand;
}

- (double)performOperation:(NSString *)operation
{
    [operation sendMessage:argument];
    return aDouble;
}

@end
```

Controller

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

```
@interface CalculatorViewController : UIViewController  
{  
    CalculatorBrain * brain;  
    IBOutlet UILabel * display;  
}
```

```
- (IBAction)digitPressed:(UIButton *)sender;  
- (IBAction)operationPressed:(UIButton *)sender;
```

```
@end
```

Controller

Our Controller inherits from
UIViewController. UIKit
supports MVC primarily
through this class.

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

```
@interface CalculatorViewController : UIViewController  
{  
    CalculatorBrain * brain;  
    IBOutlet UILabel * display;  
}
```

```
- (IBAction)digitPressed:(UIButton *)sender;  
- (IBAction)operationPressed:(UIButton *)sender;
```

```
@end
```


Controller

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

```
@interface CalculatorViewController : UIViewController  
{  
    CalculatorBrain * brain;  
    IBOutlet UILabel * display;  
}
```

This is going to point to our
CalculatorBrain

Model

```
- (IBAction)digitPressed:(UIButton *)sender;  
- (IBAction)operationPressed:(UIButton *)sender;
```

```
@end
```

Controller

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

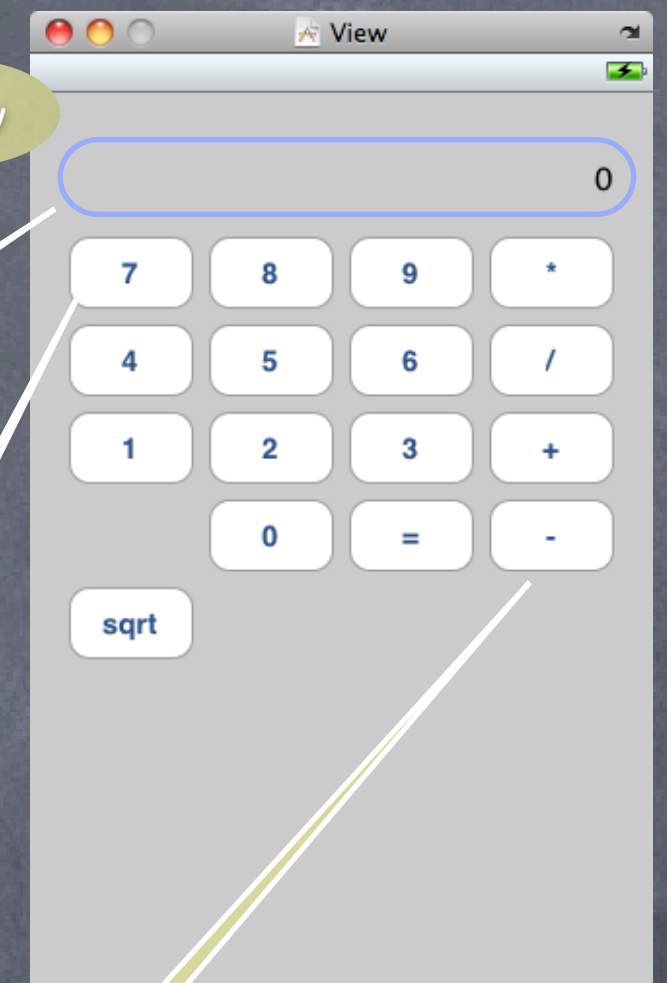
```
@interface CalculatorViewController : UIViewController  
{  
    CalculatorBrain * brain;  
    IBOutlet UILabel * display;  
}
```

```
- (IBAction)digitPressed:(UIButton *)sender;
```

```
- (IBAction)operationPressed:(UIButton *)sender;
```

```
@end
```

These hook up to our View



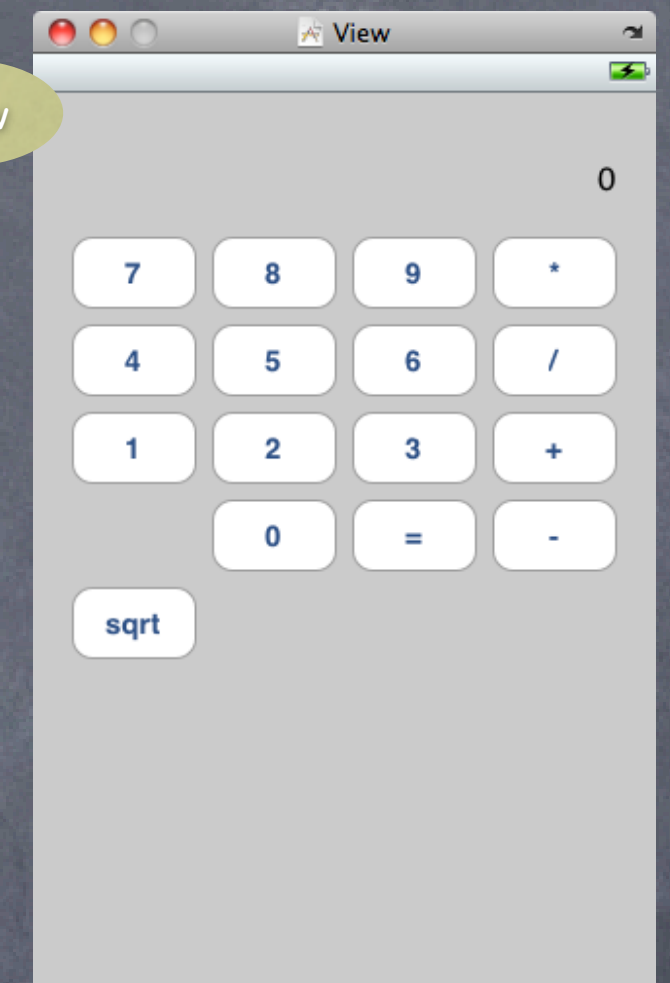
Controller

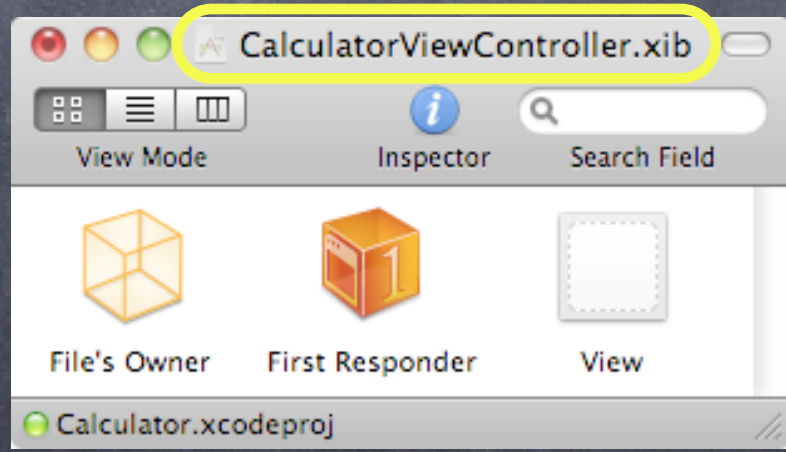
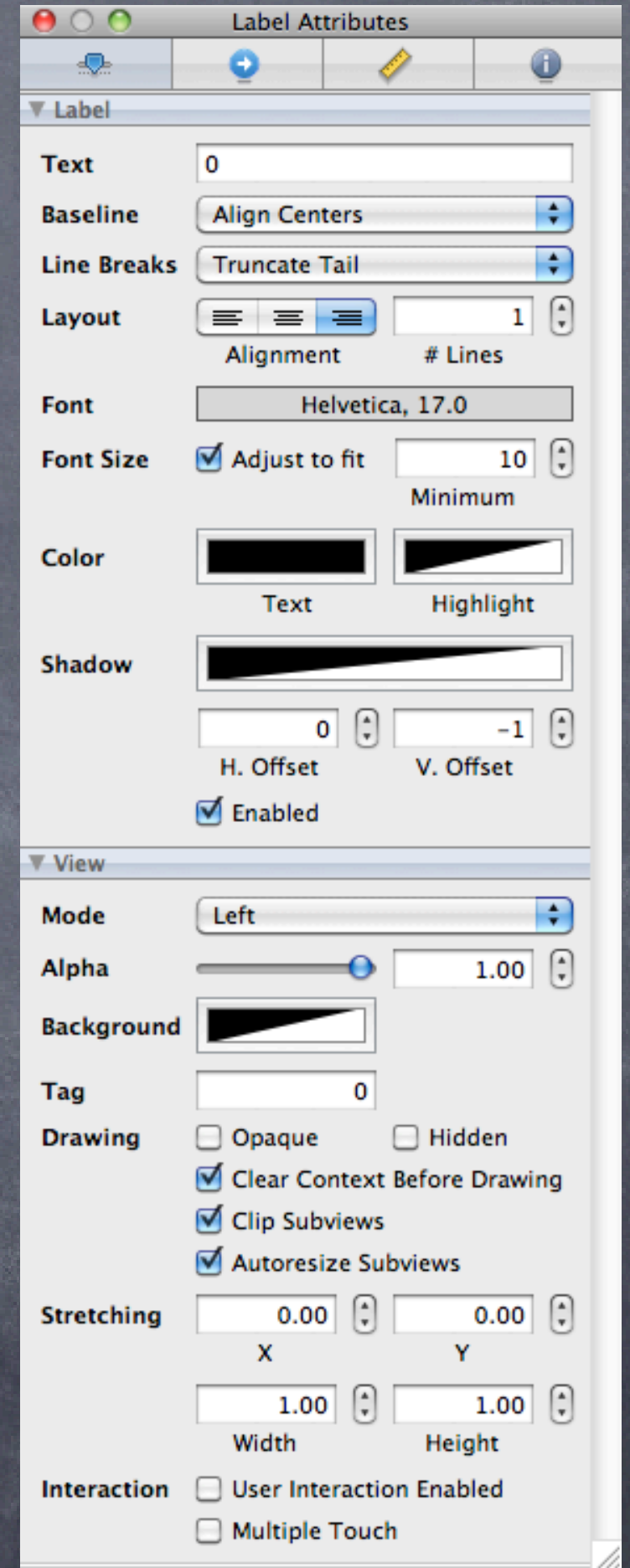
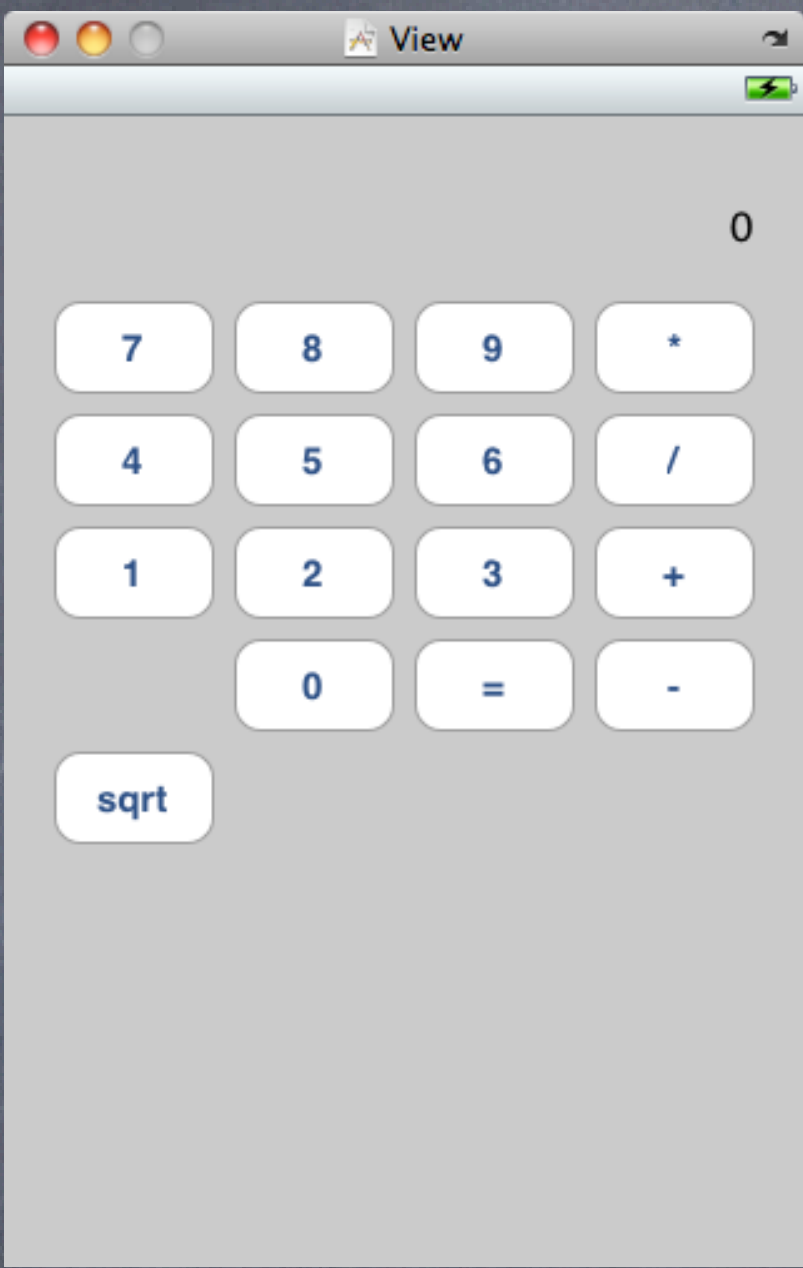
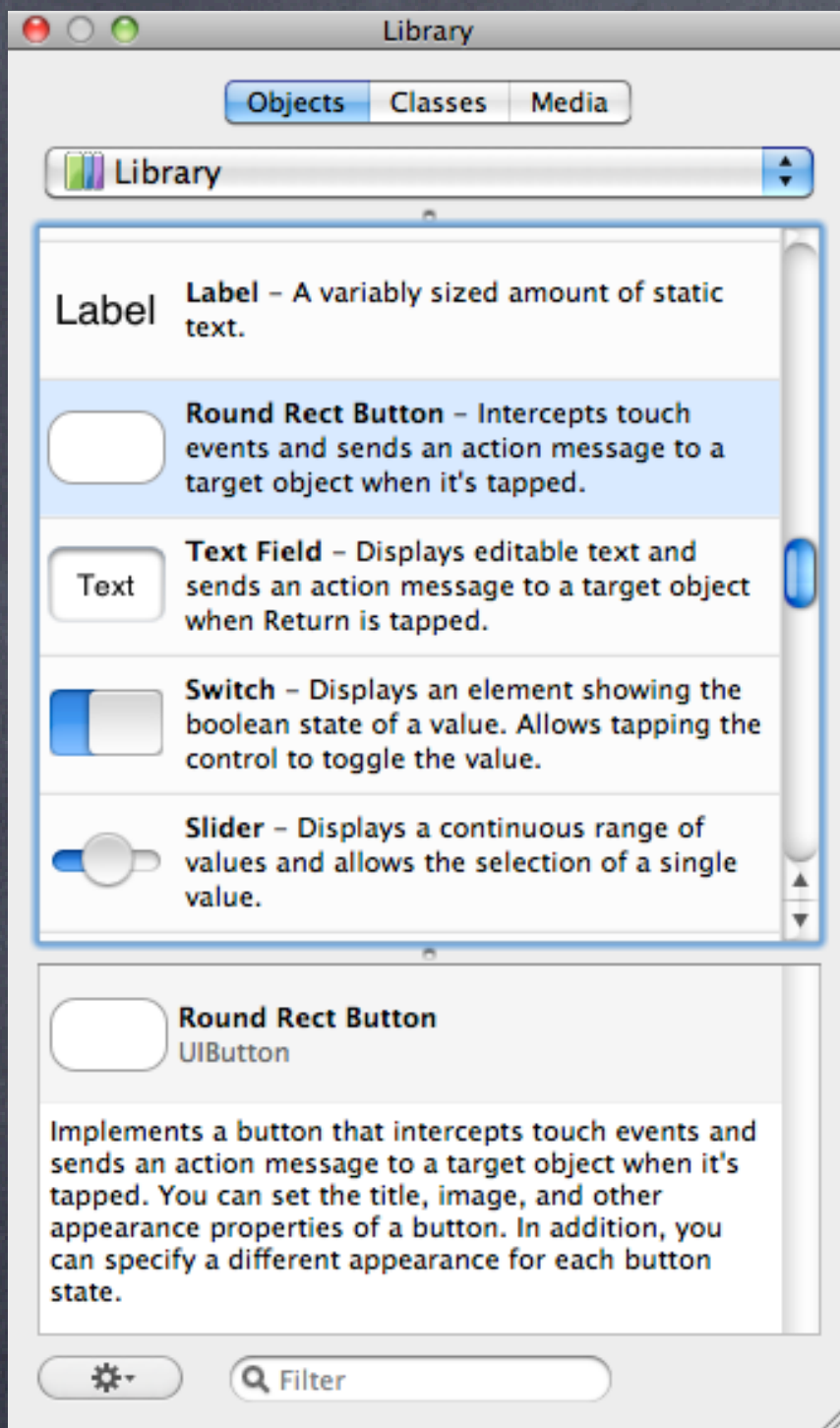
```
#import <UIKit/UIKit.h>
```

```
@interface CalculatorViewController : UIViewController  
{  
    CalculatorBrain * brain; Model  
    IBOutlet UILabel * display;  
}
```

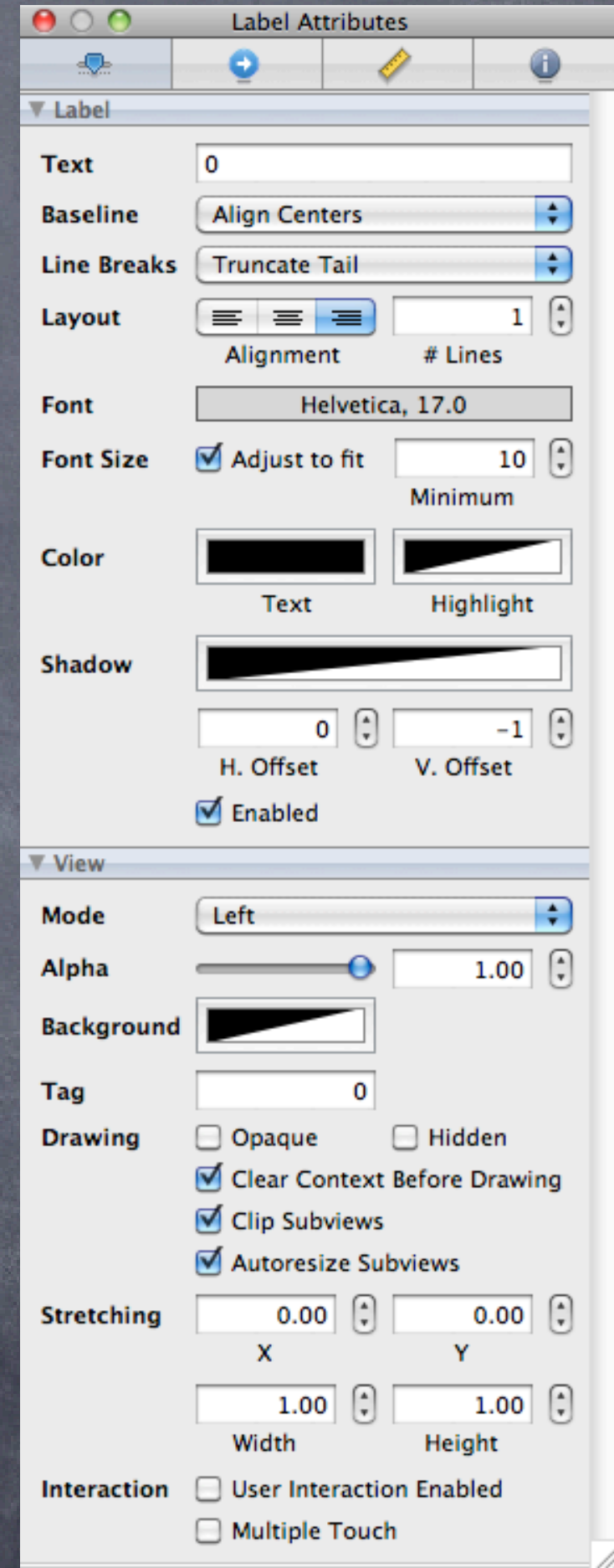
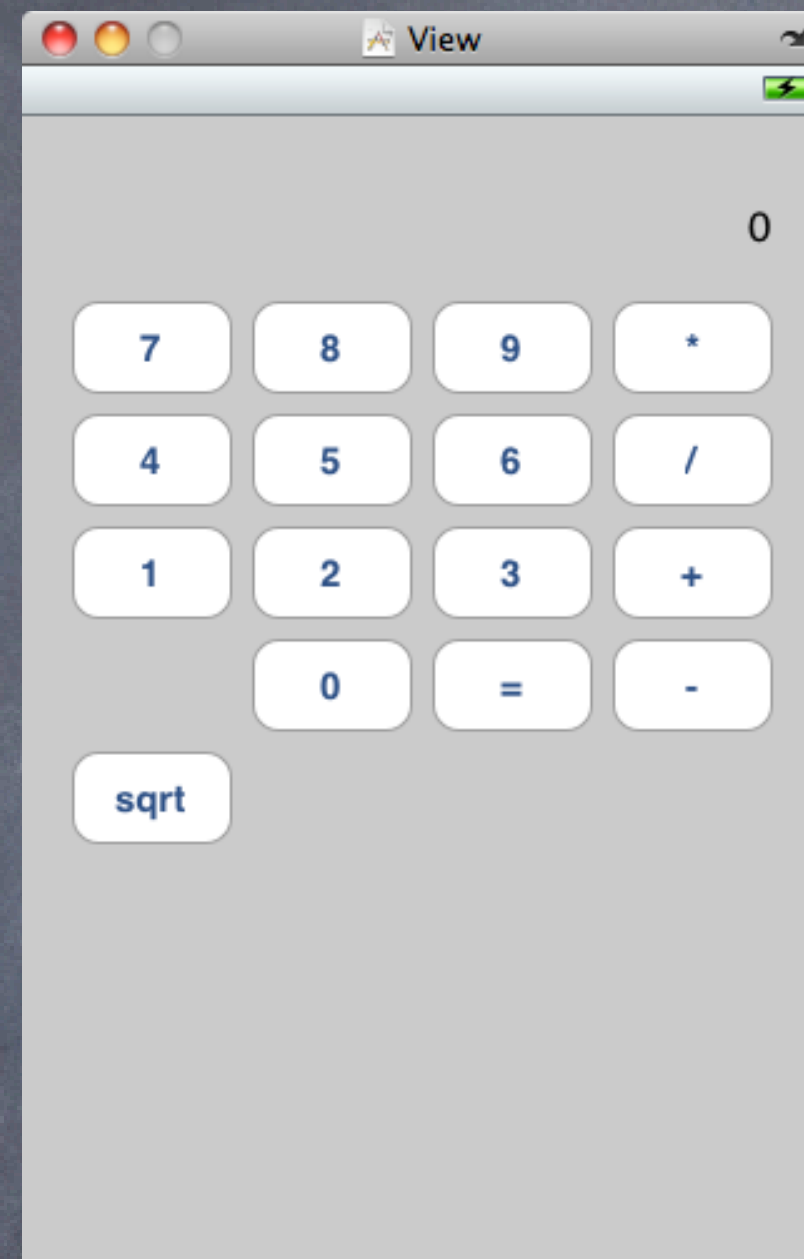
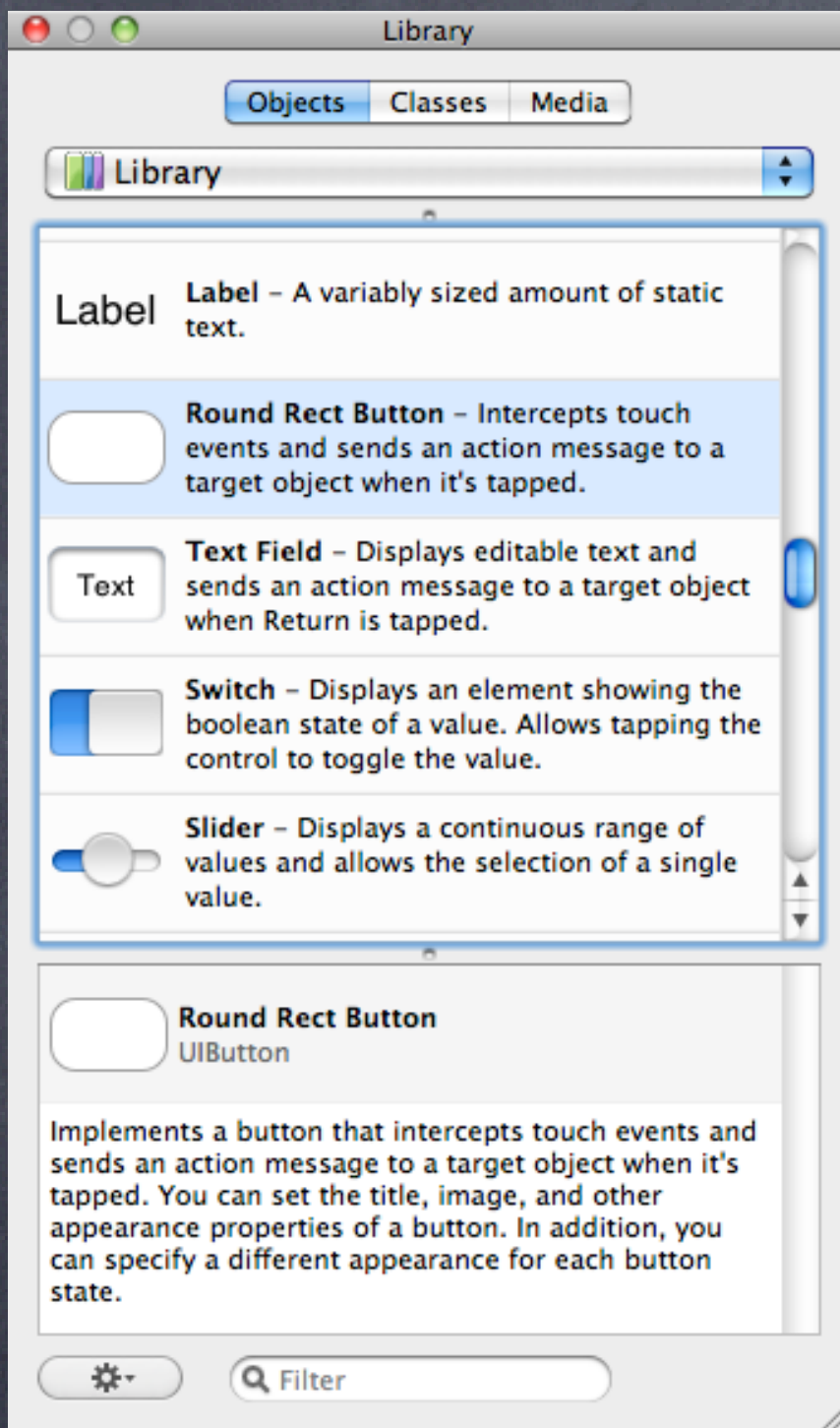
```
- (IBAction)digitPressed:(UIButton *)sender;  
- (IBAction)operationPressed:(UIButton *)sender;
```

```
@end
```



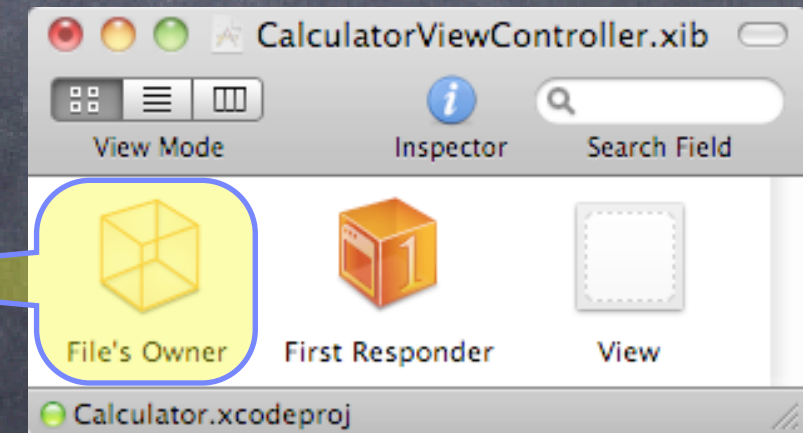


CalculatorViewController.xib



"File's Owner" is our
Controller

CalculatorViewController.xib



Library

Objects Classes Media

Library

Label Label - A variably sized amount of static text.

Round Rect Button - Intercepts touch events and sends an action message to a target object when it's tapped.

Text Field - Displays editable text and sends an action message to a target object when Return is tapped.

Switch - Displays an element showing the boolean state of a value. Allows tapping the control to toggle the value.

Slider - Displays a continuous range of values and allows the selection of a single value.

Round Rect Button
UIButton

Implements a button that intercepts touch events and sends an action message to a target object when it's tapped. You can set the title, image, and other appearance properties of a button. In addition, you can specify a different appearance for each button state.

Filter

View

0

7 8 9 *

4 5 6 /

1 2 3 +

0 = -

sqrt

Events

- digitPressed:
- operationPressed:

CalculatorViewController.xib

View Mode Inspector Search Field

File's Owner First Responder View

Calculator.xcodeproj

Label Attributes

Label

Text 0

Baseline Align Centers

Line Breaks Truncate Tail

Layout Alignment # Lines 1

Font Helvetica, 17.0

Font Size Adjust to fit 10 Minimum

Color Text Highlight

Shadow H. Offset 0 V. Offset -1 Enabled

View

Mode Left

Alpha 1.00

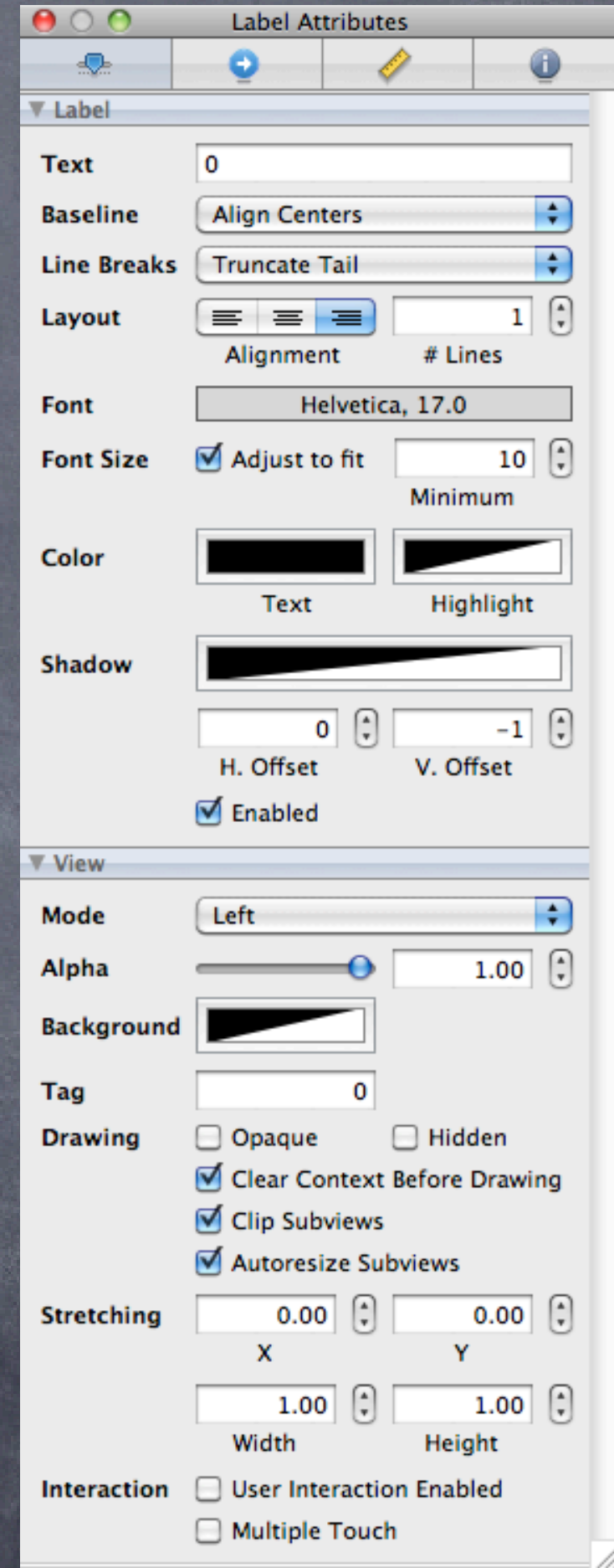
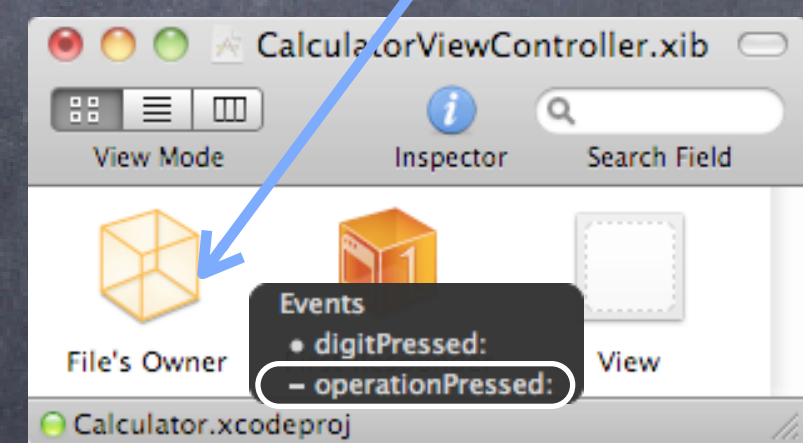
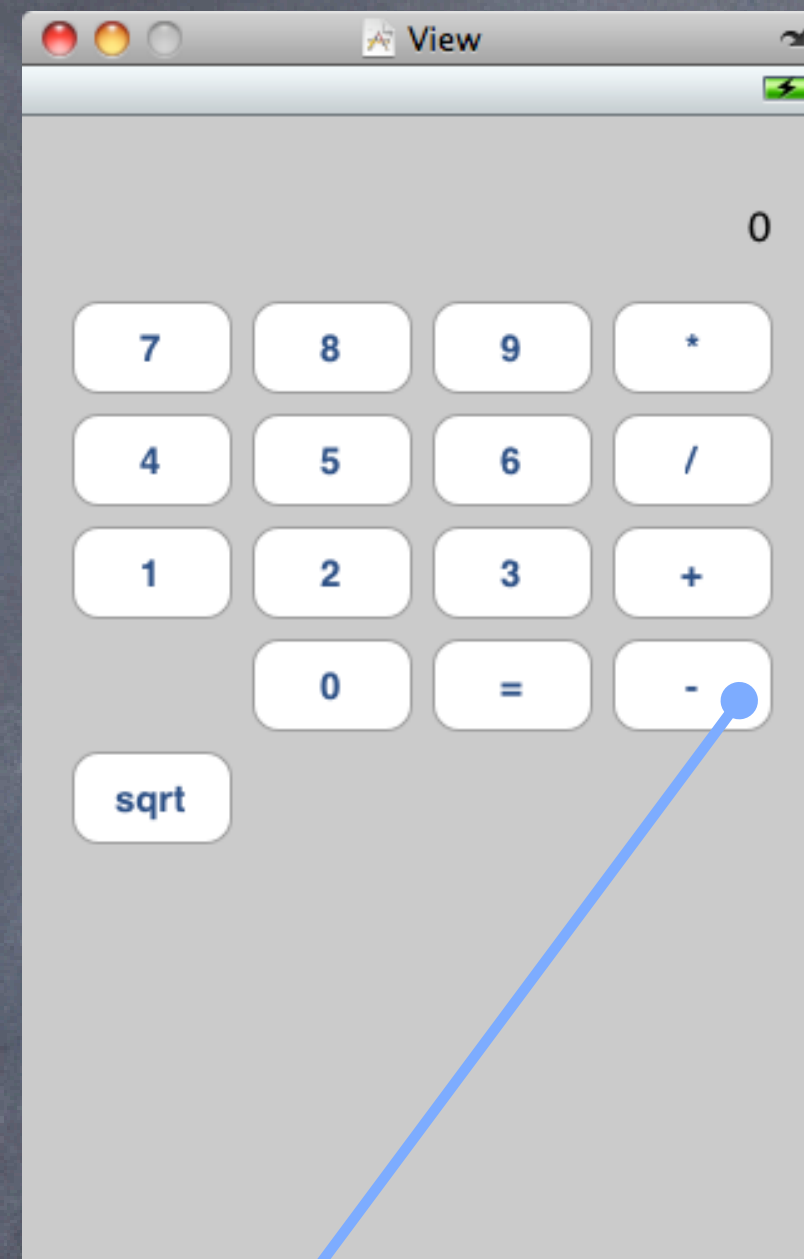
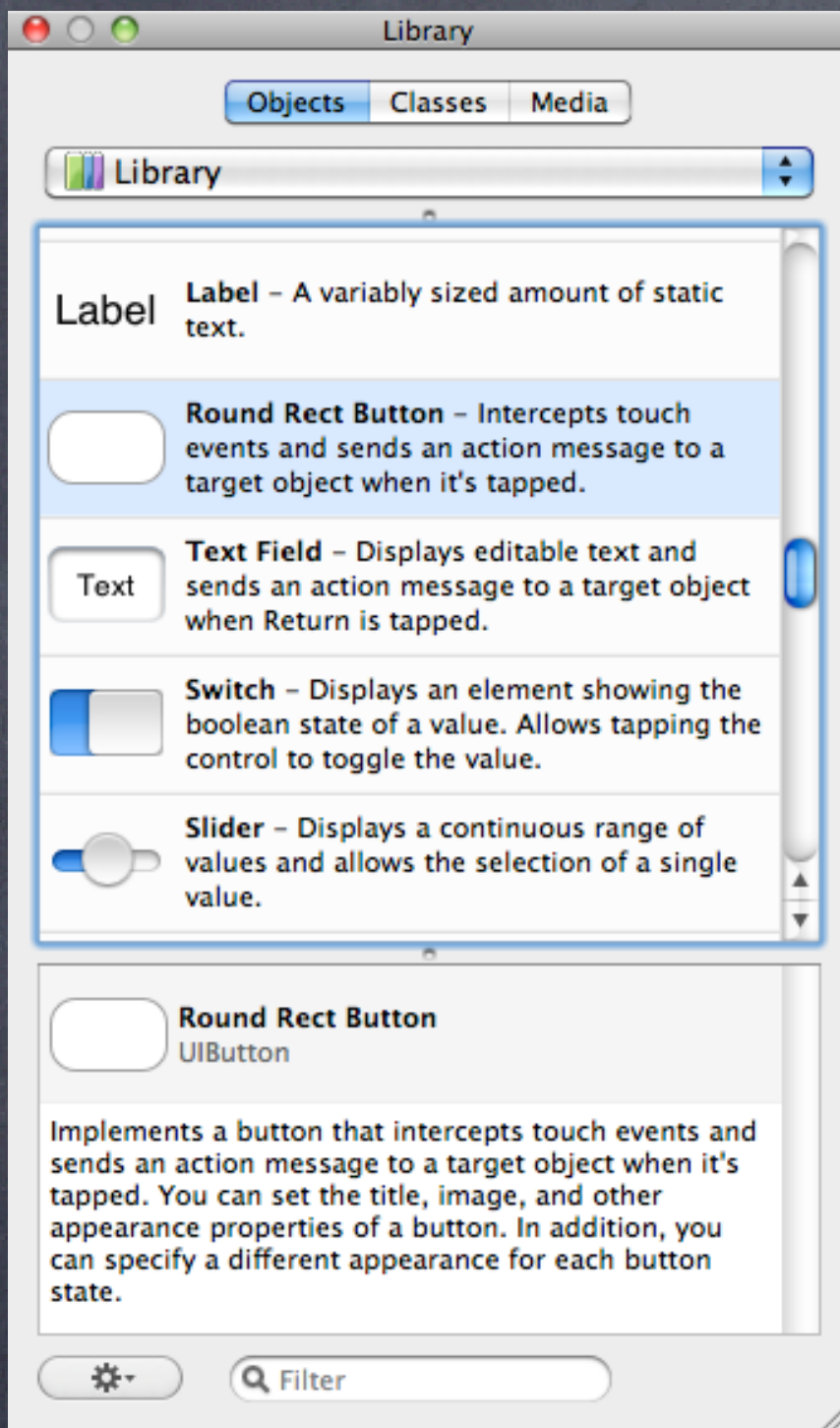
Background

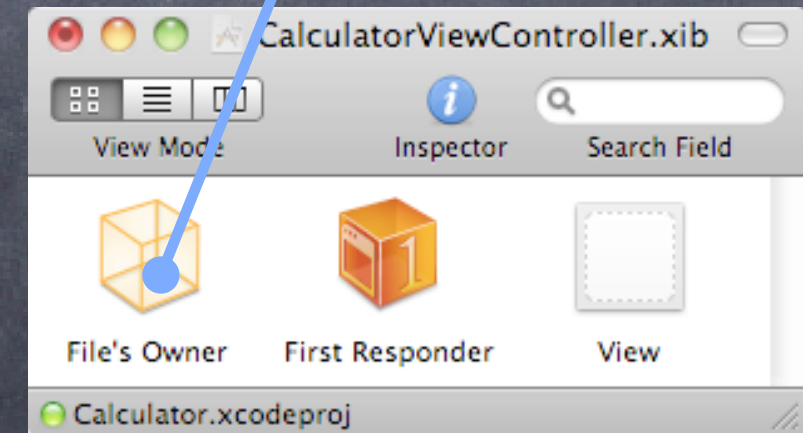
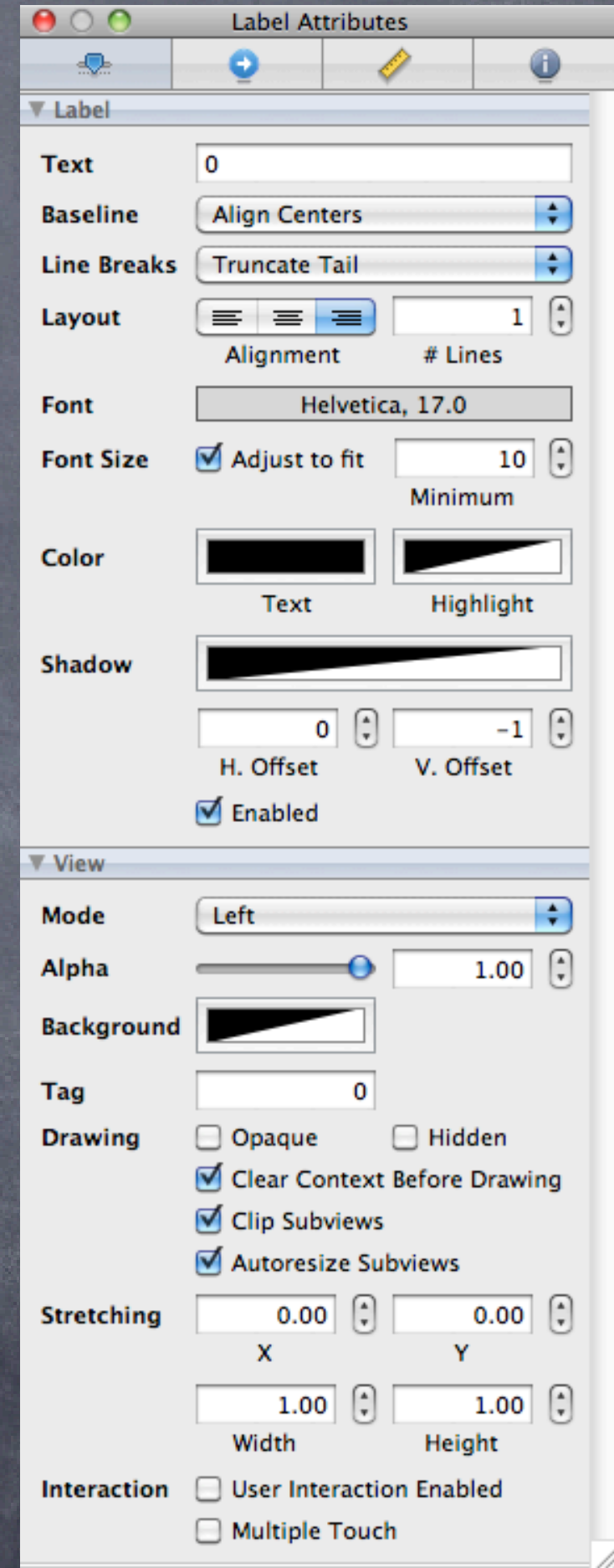
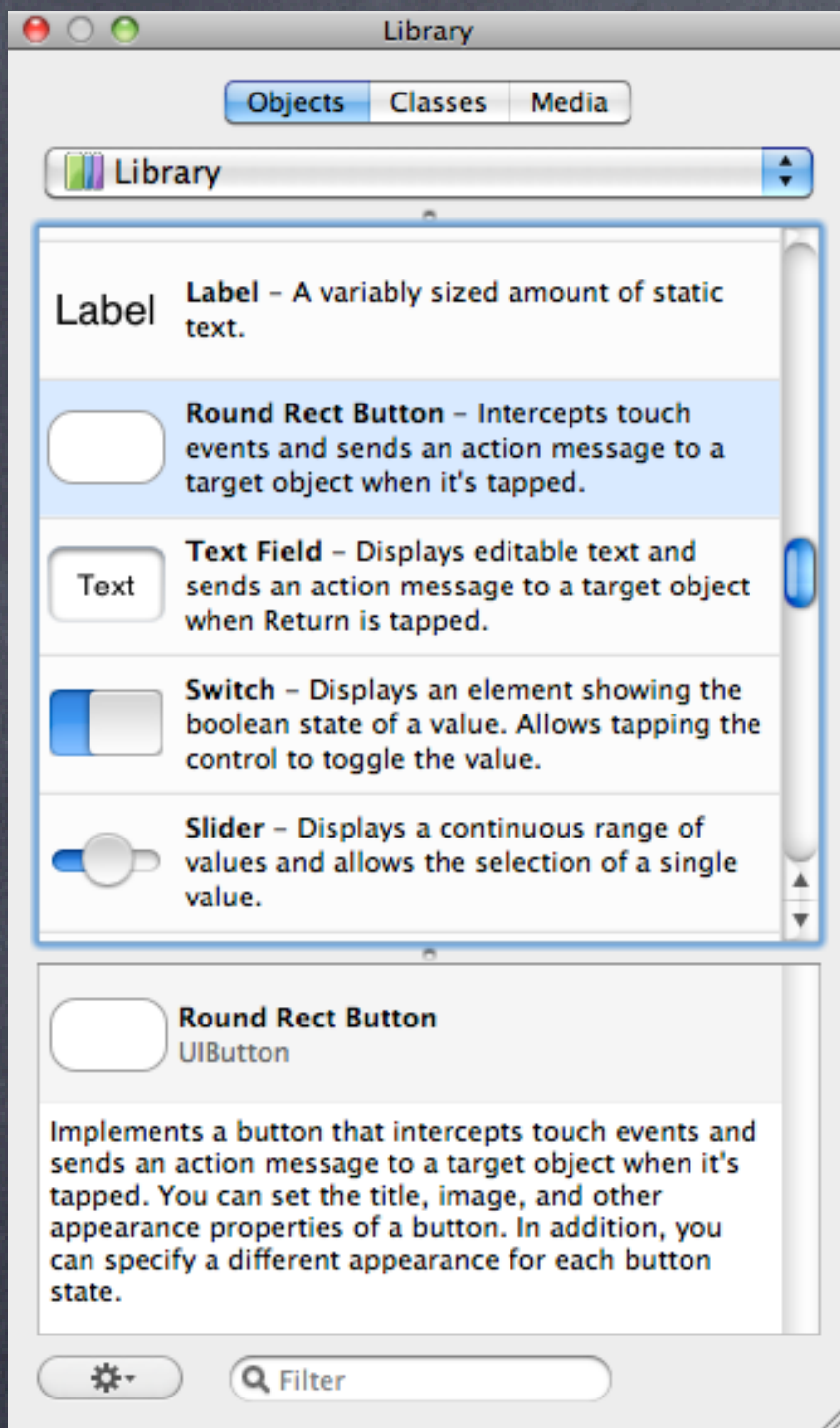
Tag 0

Drawing Opaque Hidden Clear Context Before Drawing Clip Subviews Autoresize Subviews

Stretching X 0.00 Y 0.00 Width 1.00 Height 1.00

Interaction User Interaction Enabled Multiple Touch





My First Project

A picture (or demo) is worth 1,000 words.