Mysterious Mac Programming: Building a Mac App from Scratch

Among Apple's ever increasing line of innovative products, one could easily argue that books on coding for The Mac OSX platform are in short supply. It's not because programmers don't like coding for The Mac, it's just that the demand isn't always there, and corporate environments are rarely interested in investing the time or money into developing Mac OSX programs. It's a shame because although iOS has captured the world's imagination like no other Apple product, it remains true that the Mac OSX platform has been evolving for over 20 years.

In this short and concise book, I am going to chronicle the difference between coding for iOS versus OSX. I am going to show examples of how I ported a simple iOS app to the Mac OSX platform.

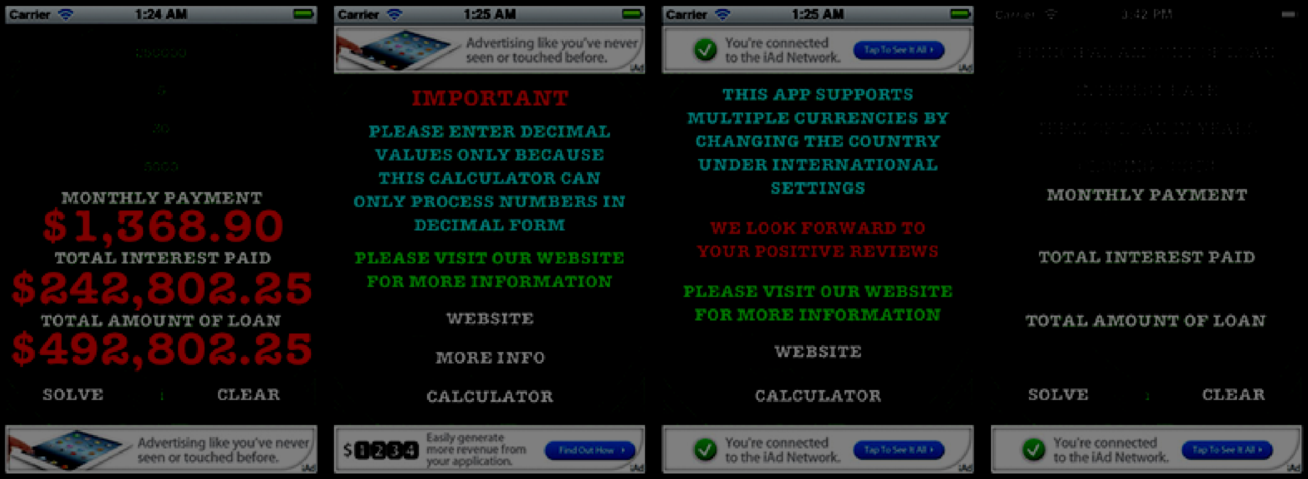
The code base in 100% written in Objective-C but as you will see, there are key differences between iOS and OSX and the workflow is very different. I hope you enjoy this journey and I look forward to your feedback!

Chapter One

A Horse of a Different Color

I am going to be covering the conversion of my iOS application called “Amortize” which is a mortage calculator. It's a great example to show how the classes are set up a bit differently.

Here is a screenshot of my mortgage calculator for iOS.



Carrier 1:24 AM

MONTHLY PAYMENT

$1,368.90

TOTAL INTEREST PAID

$242,802.25

TOTAL AMOUNT OF LOAN

$492,802.25

SOLVE CLEAR

Advertising like you've never seen or touched before.

Carrier 1:25 AM

Advertising like you've never seen or touched before.

IMPORTANT

PLEASE ENTER DECIMAL VALUES ONLY BECAUSE THIS CALCULATOR CAN ONLY PROCESS NUMBERS IN DECIMAL FORM

PLEASE VISIT OUR WEBSITE FOR MORE INFORMATION

WEBSITE

MORE INFO

CALCULATOR

Easily generate more revenue from your application. Find Out How

Carrier 1:25 AM

You're connected to the iAd Network. Tap To See It All

THIS APP SUPPORTS MULTIPLE CURRENCIES BY CHANGING THE COUNTRY UNDER INTERNATIONAL SETTINGS

WE LOOK FORWARD TO YOUR POSITIVE REVIEWS

PLEASE VISIT OUR WEBSITE FOR MORE INFORMATION

WEBSITE

CALCULATOR

You're connected to the iAd Network. Tap To See It All

MONTHLY PAYMENT

TOTAL INTEREST PAID

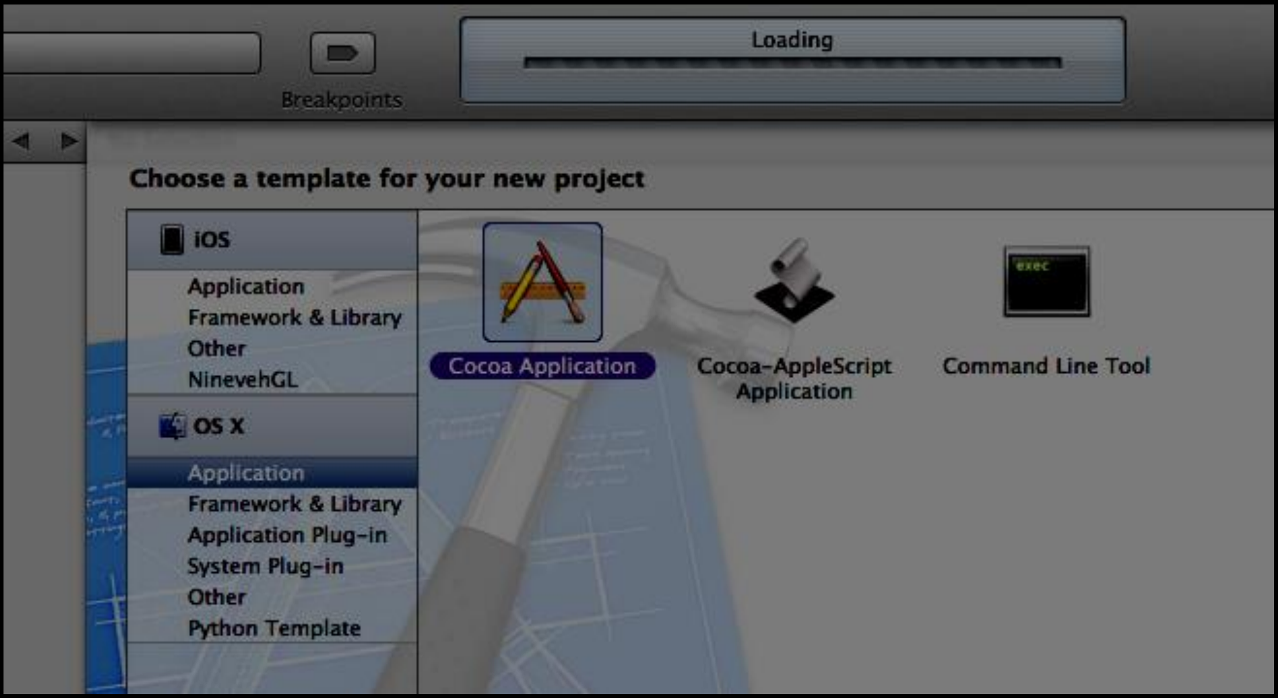
TOTAL AMOUNT OF LOAN

SOLVE CLEAR

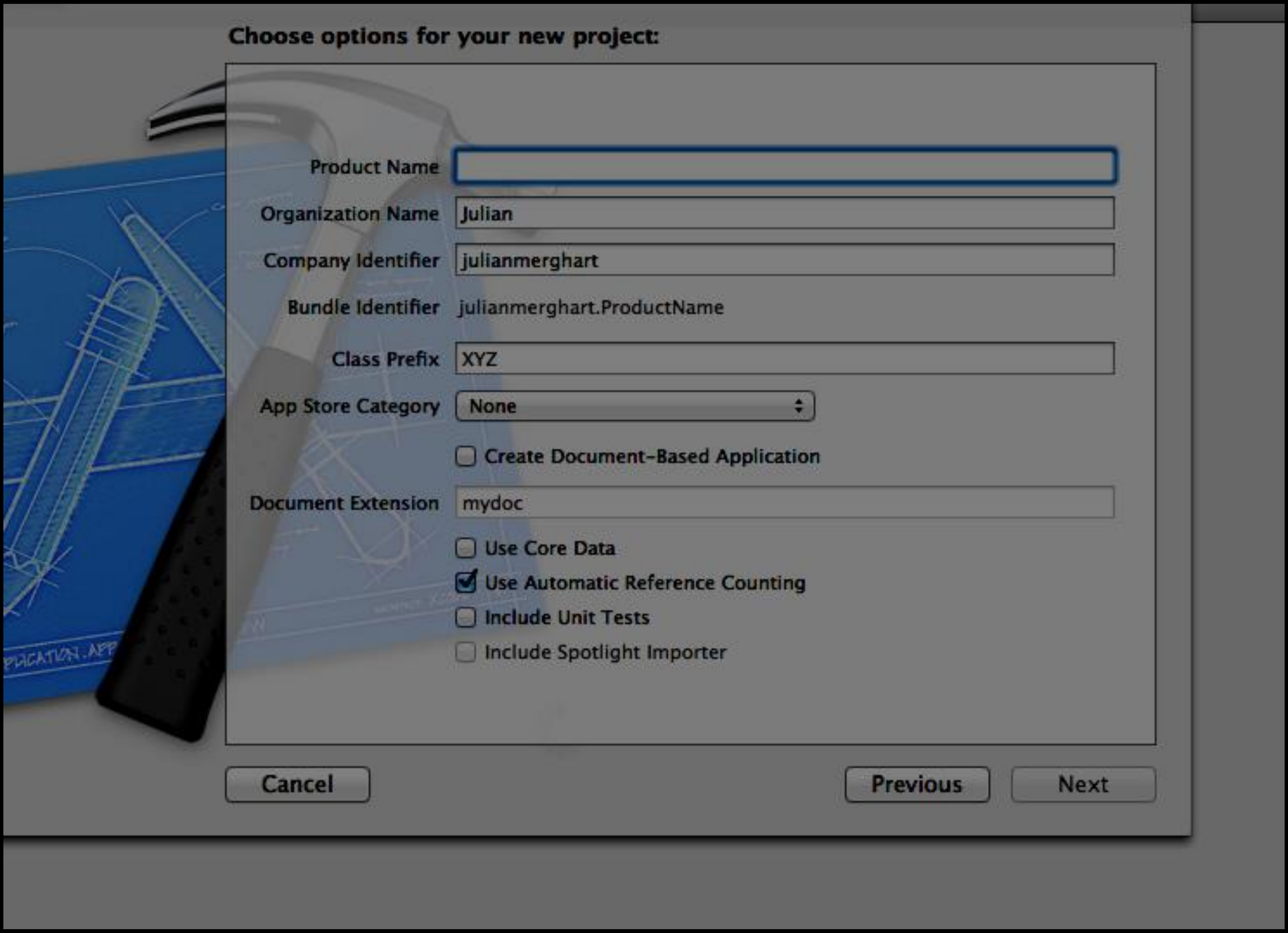
You're connected to the iAd Network. Tap To See It All

This app is simple in nature. It has 4 input fields, 3 result fields, and a solve button and a button that clears everything out. They way these things are set up in OSX are a bit different though, and as we will see, there is much to learn.

From here, we are going to come out of the gate swinging. Open Xcode and create a new Mac OSX Project.



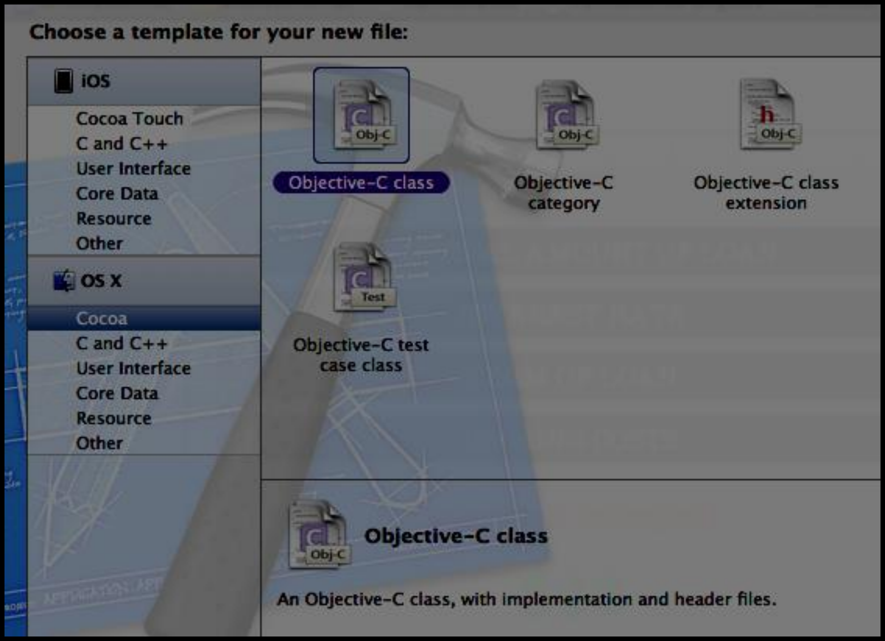
After that, name and Save the project.


Choose options for your new project:

Product Name
Organization Name: Julian
Company Identifier: julianmerghart
Bundle Identifier: julianmerghart.ProductName
Class Prefix: XYZ
App Store Category: None
☐ Create Document-Based Application
Document Extension: mydoc
☐ Use Core Data
☑ Use Automatic Reference Counting
☐ Include Unit Tests
☐ Include Spotlight Importer

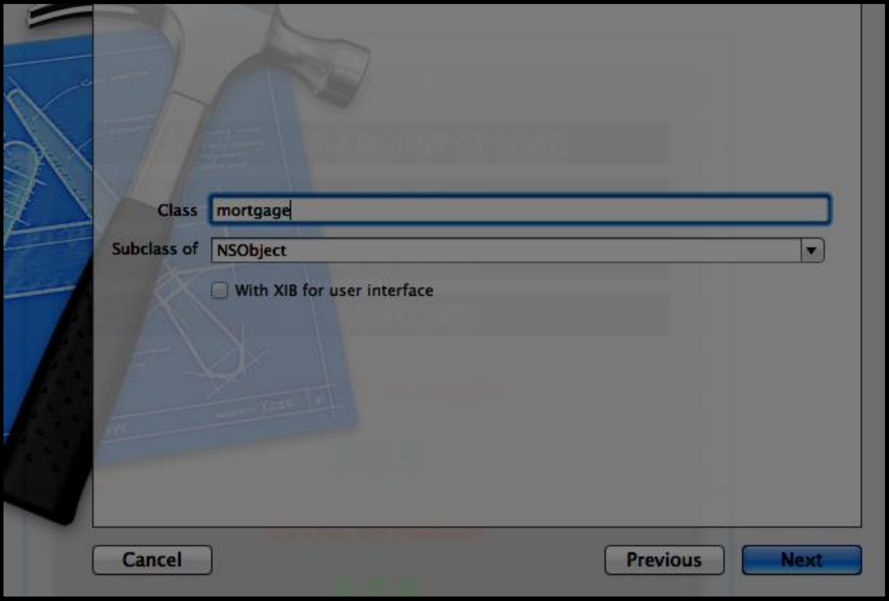
Next, grab a Textured Window from the right side menu and drag it into the XIB container.

Before you do anything, create a new Objecive-C class called mortgage.



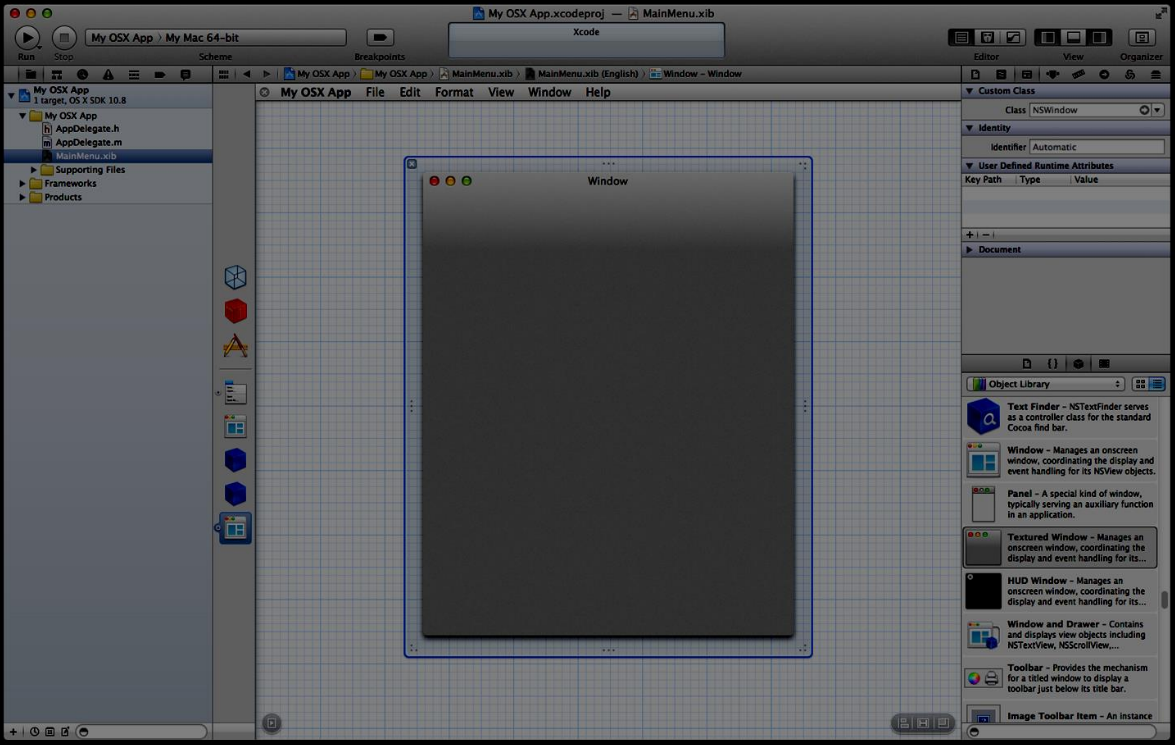
An Objective-C class, with implementation and header files.

Name the class mortgage and save it.



Class: mortgage
Subclass of: NSObject
[ ] With XIB for user interface

Notice how this class is a subclass of NSObject. That will be important later.



Text Finder - NSTextFinder serves as a controller class for the standard Cocoa find bar.

Window - Manages an onscreen window, coordinating the display and event handling for its NSView objects.

Panel - A special kind of window, typically serving an auxiliary function in an application.

Textured Window - Manages an onscreen window, coordinating the display and event handling for its...

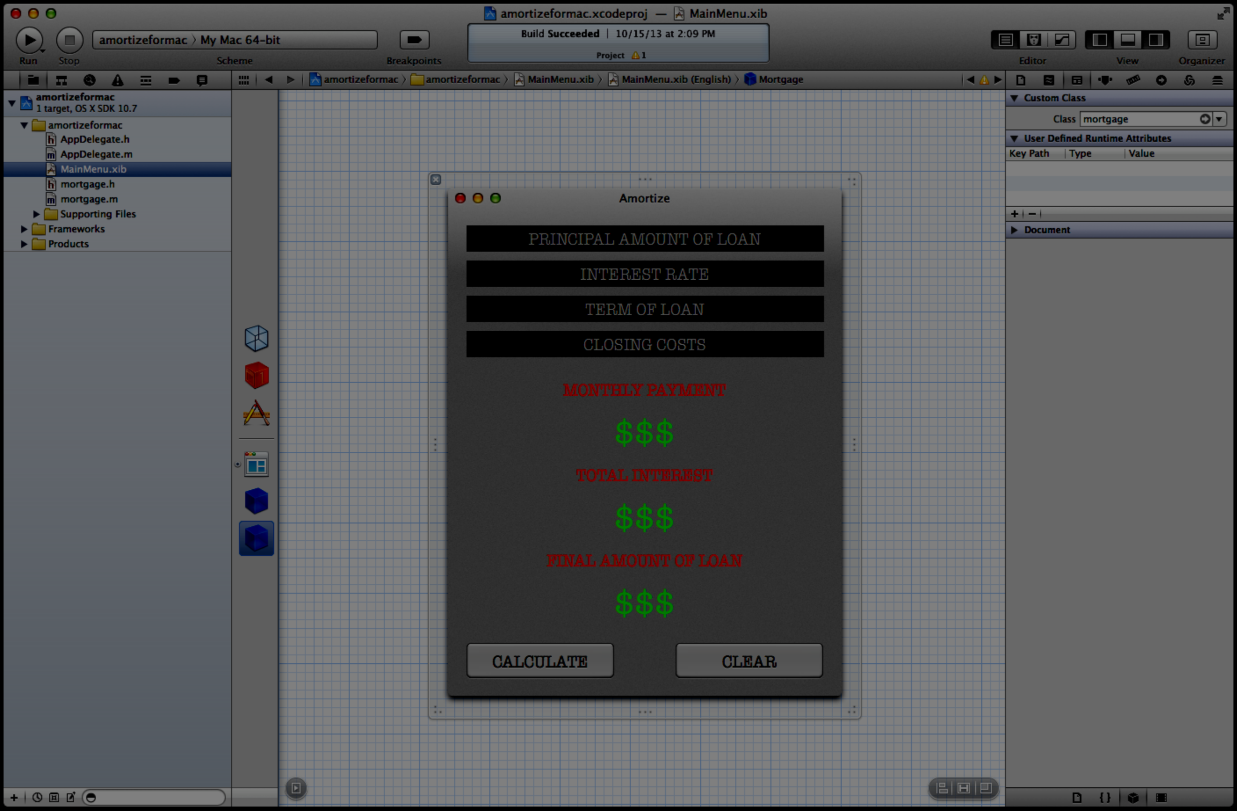
HUD Window - Manages an onscreen window, coordinating the display and event handling for its...

Window and Drawer - Contains and displays view objects including NSTextView, NSScrollView,...

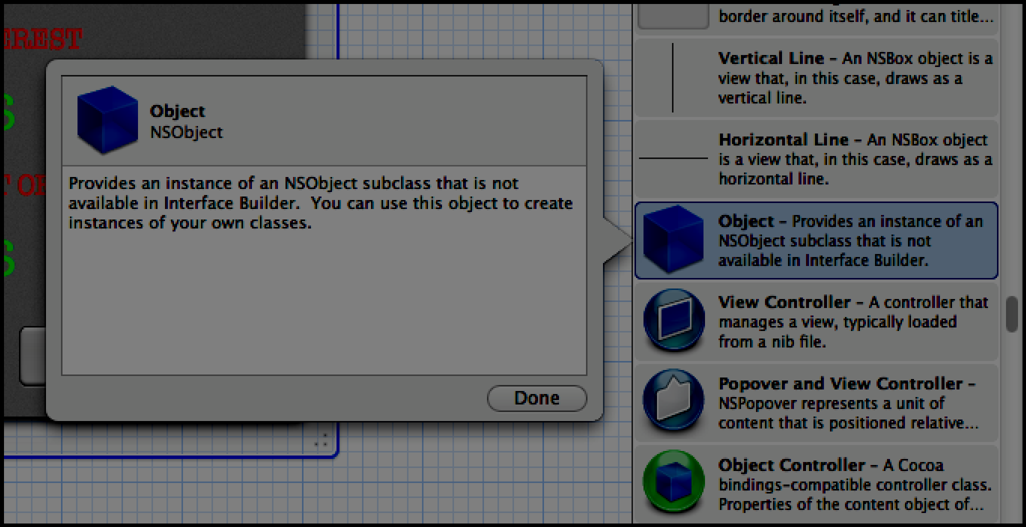
Toolbar - Provides the mechanism for a titled window to display a toolbar just below its title bar.

Image Toolbar Item - An instance

After that drag 4 Textfields, 3 labels, and 2 Buttons.



After you have set up the visual elements and you have to drag in a NSObject Controller.



Object
NSObject

Provides an instance of an NSObject subclass that is not available in Interface Builder. You can use this object to create instances of your own classes.

Done

border around itself, and it can title...

Vertical Line - An NSBox object is a view that, in this case, draws as a vertical line.

Horizontal Line - An NSBox object is a view that, in this case, draws as a horizontal line.

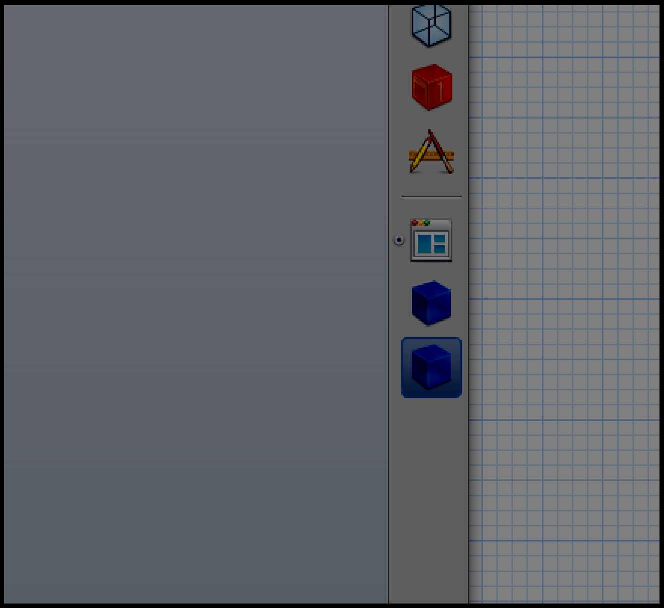
Object - Provides an instance of an NSObject subclass that is not available in Interface Builder.

View Controller - A controller that manages a view, typically loaded from a nib file.

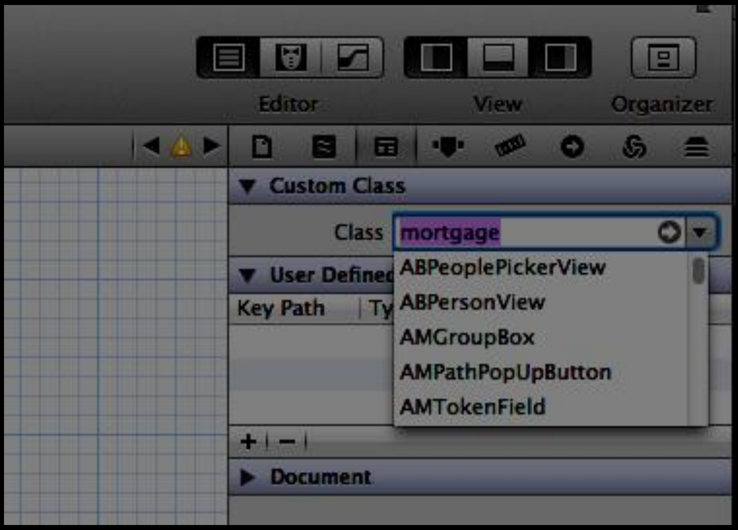
Popover and View Controller - NSPopover represents a unit of content that is positioned relative...

Object Controller - A Cocoa bindings-compatible controller class. Properties of the content object of...

Drag it over to the bar on the left.



Now assign that NSObject to the mortgage class by clicking in the upper tab bar above like below and selecting the mortgage class.



Once that is all set up, we are ready to write some code. As you may have guessed, those little blue cubes are generic objects that you use to assign classes and elements.

Chapter Two:
The Code

Our mortage.h file looks like this:

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

@interface mortgage : NSObject

@property(nonatomic, strong) IBOutlet NSTextField * variable1;
@property(nonatomic, strong) IBOutlet NSTextField * variable2;
@property(nonatomic, strong) IBOutlet NSTextField * variable3;
@property(nonatomic, strong) IBOutlet NSTextField * variable4;
@property(nonatomic, strong) IBOutlet NSTextField * solve1;
@property(nonatomic, strong) IBOutlet NSTextField * solve2;
@property(nonatomic, strong) IBOutlet NSTextField * solve3;

-(IBAction)calculate: (id)sender;
-(IBAction)clear: (id) sender;

@end
```

Our mortage.m file looks like this:

```
#import "mortgage.h"

@implementation mortgage

@synthesize variable1;
@synthesize variable2;
@synthesize variable3;
@synthesize variable4;
@synthesize solve1;
@synthesize solve2;
@synthesize solve3;


-(IBAction)clear: (id)sender

{
   variable1.stringValue  = @"";
   variable2.stringValue  = @"";
   variable3.stringValue  = @"";
   variable4.stringValue  = @"";
   solve1.stringValue        = @"";
   solve2.stringValue        = @"";
   solve3.stringValue        = @"";
}


-(IBAction)calculate: (id)sender
```

```
{
    double principle =
([variable1 floatValue]);
    double rateinwholenumber =                ([variable2
floatValue]);
    double termofloan =
([variable3 floatValue]);
    double closingcosts =
([variable4 floatValue]);

    double I =      (rateinwholenumber/100)/12;
    double N =      termofloan * 12;
    double X =      1 + I;
    double Y =      pow (X,N);

    double monthlypayment = (principle + closingcosts) * (Y * I)
/ (Y - 1);
    double totalloanamount = monthlypayment * N;
    double totalinterest = totalloanamount - principle;

    //change from double floats to currency

    NSNumberFormatter *formatter = [[NSNumberFormatter alloc]
init];
    [formatter setNumberStyle:NSNumberFormatterCurrencyStyle];
    [formatter setAllowsFloats:YES];
    [formatter setMaximumFractionDigits:2];
    [formatter setAlwaysShowsDecimalSeparator:YES];
    [formatter setGeneratesDecimalNumbers:YES];

    NSString *$monthlypayment$ = [formatter stringFromNumber:

                                  [NSNumber numberWithDouble:
monthlypayment]];

    NSString *$totalloanamount$ = [formatter stringFromNumber:

                                   [NSNumber numberWithDouble:
totalloanamount]];

    NSString *$totalinterest$ = [formatter
                                 stringFromNumber:

                                 [NSNumber numberWithDouble:
totalinterest]];

    NSLog(@"Output as Currency: %@", $monthlypayment$);
    NSLog(@"Output as Currency: %@", $totalinterest$);

    NSLog(@"Output as Currency: %@", $totalloanamount$);

    //change from number objects to strings

    solve1.stringValue = $monthlypayment$;

    solve2.stringValue = $totalinterest$;

    solve3.stringValue = $totalloanamount$;

}

@end
```

As you can imagine, this code requires some explanation. The differences here between iOS and OSX are extreme.

Obviously, the foundation of any Mac or iOS application is Objective-C, but that is where the similarities end. Mac and iOS are different operating systems, and they have very different requirements.

Take this block for example:

```
variable1.stringValue  = @"";
variable2.stringValue  = @"";
variable3.stringValue  = @"";
variable4.stringValue  = @"";
solve1.stringValue        = @"";
solve2.stringValue        = @"";
solve3.stringValue        = @"";
```

In an the iOS version of this app, that block looks like:

```
variable1.text = @"";
variable2.text = @"";
variable3.text = @"";
variable4.text = @"";
solve1.text = @"";
solve2.text = @"";
solve3.text = @"";
```

Basically, what this shows is that there are slight syntax changes between iOS and OSX and they must be maintained.

In the code above, there should also be mention given to @synthesized variables. There cool kids refer to this as setters and getters. What it means in reality is to activate or give access to the related methods for a class. For example, NSNumberFormatter has a method called setGeneratesDecimalNumbers.

You only have access to these methods if you allocate and initialize the NSNumberFormatter class.

It should also be noted that if you wanted to, you could set up a class with the @property declaration in your .h file, and then use @synthesize in your .m, but then you would avoid allocating and initilizing.

You would notice that the methods would automatically be recognized in your .m when the time comes to call them. This is often the most confusing part of Apple programming.

Ultimately, this means that:

```
NSNumberFormatter *formatter = [[NSNumberFormatter alloc] init];
[formatter setNumberStyle:NSNumberFormatterCurrencyStyle];
[formatter setAllowsFloats:YES];
[formatter setMaximumFractionDigits:2];
[formatter setAlwaysShowsDecimalSeparator:YES];
[formatter setGeneratesDecimalNumbers:YES];
```

is the same as this in your .h:

```
@property(nonatomic, retain) NSNumberFormatter * formatter;
```

and this in your .m:

```
@synthesize formatter;

[formatter setNumberStyle:NSNumberFormatterCurrencyStyle];
[formatter setAllowsFloats:YES];
[formatter setMaximumFractionDigits: 2];
[formatter setAlwaysShowsDecimalSeparator:YES];
[formatter setGeneratesDecimalNumbers:YES];
```

You will notice that your compiler will recognize that you are calling these methods, and it will turn those method calls into another color.

In Objective-C, the use of [brackets] means a message is being sent to a class.

Really, the important idea here is that the generic NSObject is how you control your classes, and it's how you assign different classes to elements of your user interface.

Chapter 3

Major Differences Between iOS and OSX

Take this block of iOS code for example:

```
-(IBAction)clear

{
variable1.text = @"";
variable2.text = @"";
variable3.text = @"";
variable4.text = @"";
solve1.text = @"";
solve2.text = @"";
solve3.text = @"";
}
```

Later, in the next block, when we get to this piece of code in iOS, this:

```
-(IBAction)calculate

{

//...

solve1.text = [NSString stringWithFormat:   @"%@",
$monthlypayment$];

solve2.text = [NSString stringWithFormat:   @"%@",
$totalinterest$];

solve3.text = [NSString stringWithFormat:   @"%@",
$totalloanamount$];

}
```

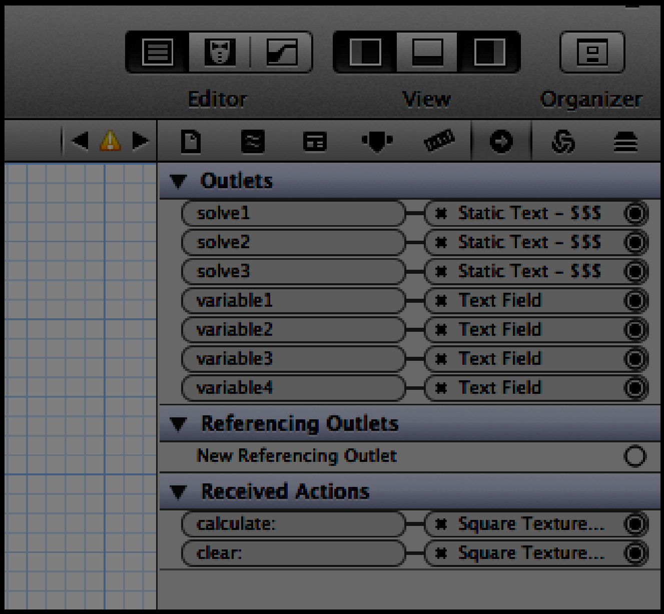
becomes this is OSX:

```
-(IBAction)calculate: (id)sender

{

//...

solve1.stringValue = $monthlypayment$;

solve2.stringValue = $totalinterest$;

solve3.stringValue = $totalloanamount$;

}
```

As you can see, in OSX (id)sender is part of the required syntax.

Also, stringValue is a property that eliminates the need to use the NSString class and it's associated methods when converting C floating point values to string values that can be displayed.

As far as making connections, it's the same process as with iOS, but you have to click on the blue NSObject boxes to drag and make connections.



▼ Outlets

solve1	Static Text - $$$
solve2	Static Text - $$$
solve3	Static Text - $$$
variable1	Text Field
variable2	Text Field
variable3	Text Field
variable4	Text Field

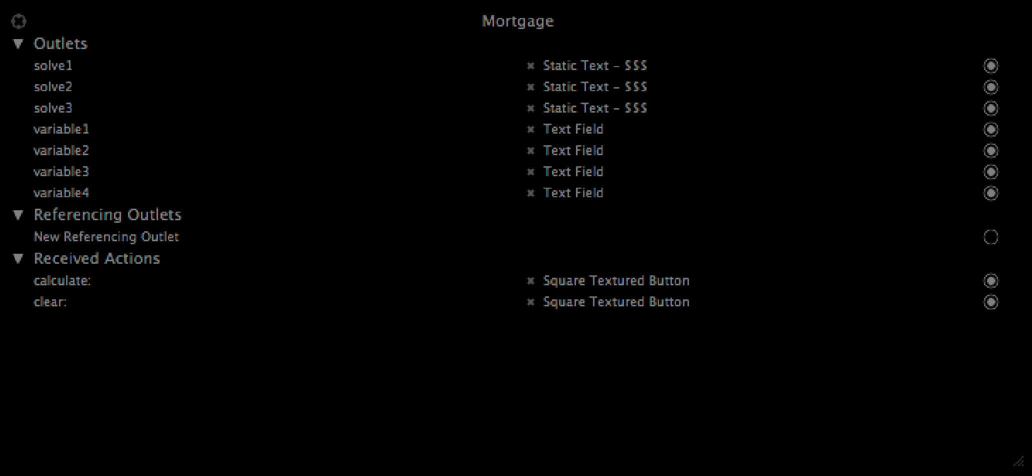
▼ Referencing Outlets

New Referencing Outlet

▼ Received Actions

calculate:	Square Texture...
clear:	Square Texture...

Or by right clicking the blue object box:



Mortgage		
▼ Outlets		
solve1	✖ Static Text - $$$	◉
solve2	✖ Static Text - $$$	◉
solve3	✖ Static Text - $$$	◉
variable1	✖ Text Field	◉
variable2	✖ Text Field	◉
variable3	✖ Text Field	◉
variable4	✖ Text Field	◉
▼ Referencing Outlets		
New Referencing Outlet		○
▼ Received Actions		
calculate:	✖ Square Textured Button	◉
clear:	✖ Square Textured Button	◉

Conclusion

Mac programming is a horse of a different color. The syntax is not the same as iPhone code, even though Objective-C is still used. The two codebases, though similar, are not an exact translation, as illustrated above. I hope this singular example helps to demonstrate this and will spark your interest in developing apps for the Mac.