Bootstrapping iOS Application Development

Chris Zelenak

12/02/2013

Welcome to the 5-Day Bootstrapping iOS Application Development class. Over the course of these next 5 days, we're going to be reviewing Objective C, UIKit, Core Animation, Core Data, and a host of other technologies you may only know by name.

1 Lab 1

Create a new iPhone Project

Create a new Single View based application in XCode called Lab1

File > New > Project

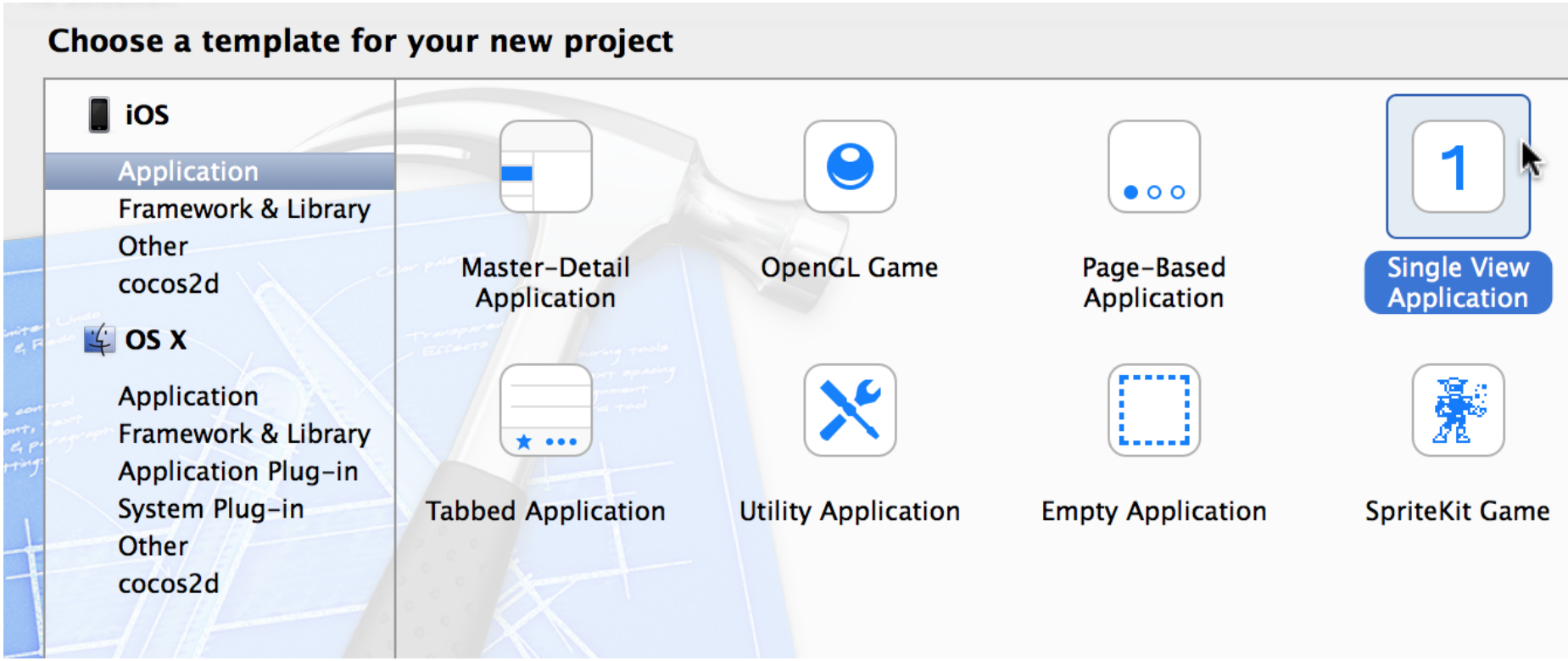


Figure : New Projects

Open Main.storyboard in Interface Builder

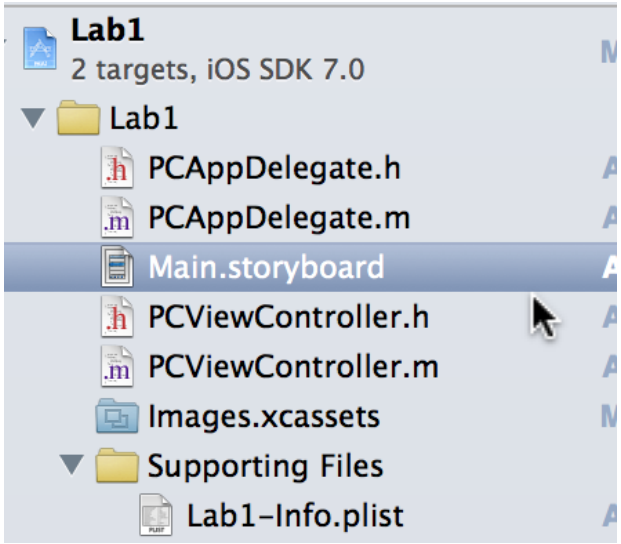


Figure : Main Storyboard

Add a button

Add a button with the text “Hello iPhone”

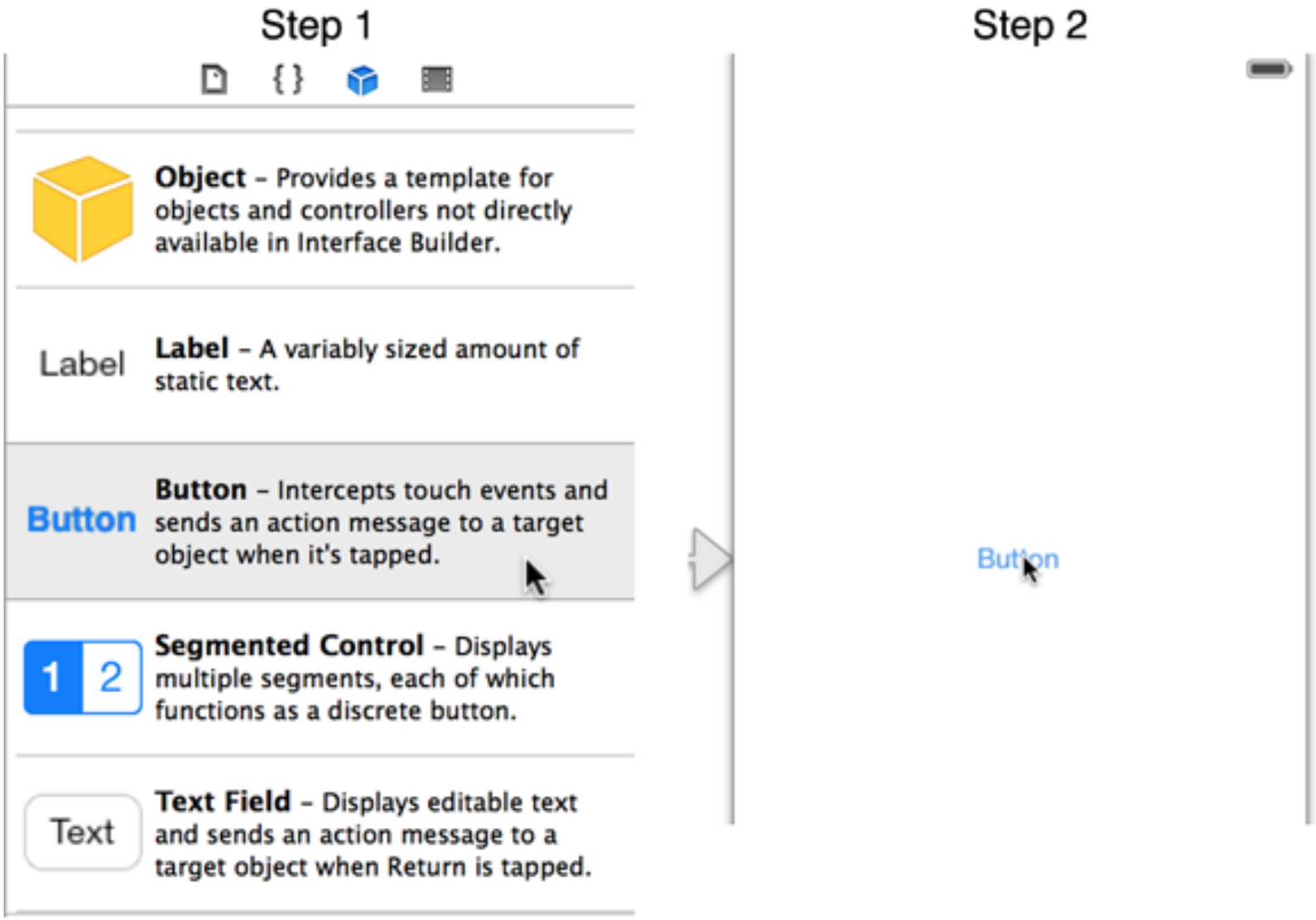


Figure : Drag Button to Storyboard

Connect the button to the view controller

Add an IBAction selector to the view controller

```
1    // In the ViewController.h file
2    -(IBAction) helloWorldTapped:(id)sender;
```

Make it open an alert view

```
1    // In the ViewController.m file
2    -(IBAction) helloWorldTapped:(id)sender {
3        [[[UIAlertView alloc] initWithTitle:@"Yay it worked"
4                                    message:@"Hello World"
5                                   delegate:nil
6                          cancelButtonTitle:@"Dismiss"
7                          otherButtonTitles: nil] show];
8    }
```

Connect the `IBAction` to the Button's `Touch Up Inside` action.

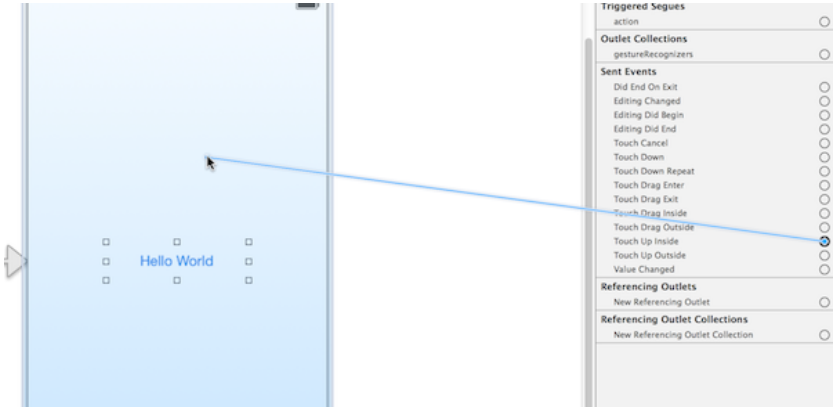


Figure : Connect Touch Up Inside event to IBAction

Build and run in simulator

Run the simulator with ⌘ + R, the Product > Run menu, or the Play button

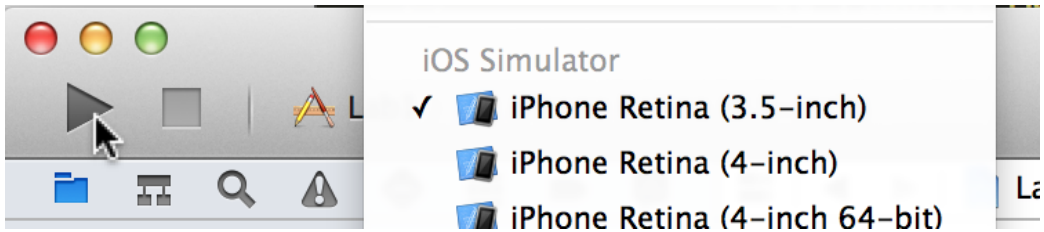


Figure : Launch Simulator

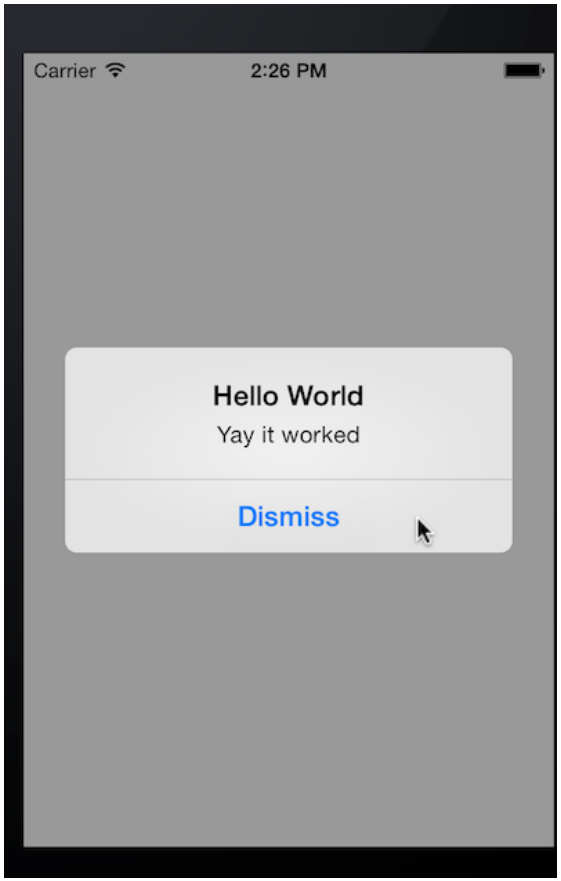


Figure : It Worked!

1 Objective C

Overview

Objective C is a superset of C. It tacks on to C a message-passing object model similar to Smalltalk's (see Ruby), with support for reflection and limited runtime dynamic programming.

Objective C allows you to use weak and dynamicly typed language features along with the static typing of C; it also allows you to take advantage of runtime method resolution, so that your object's implementation of certain messages may be determined at call-time rather than at compile-time.

The language is still physically identical to C in many ways, in that the bulk of your development will be alternating between header files (.h files) and implementation files (.m files). You will also be doing a lot of memory management, which may fill you with either fear or joy, depending on your experience with garbage collected languages.

You will also be dealing a lot with many other C-family familiars, such as enumerated integer constants (enums), structs, pass-by-reference v. pass-by-value semantics and pointers. You don't need to have a very strong grasp on these things immediately, but the more iPhone programming and Objective C programming you do, the more your skills will benefit from a firm grasp of C.

Review of standard C elements

Variable declaration in C follows the form `TYPE_DESCRIPTION NAME`, eg:

```c
int i = -1; // just an integer
unsigned int k = 0; // an unsigned integer
const unsigned int j = 0; // a constant unsigned integer, no reassignment
const char c = 'c'; // a constant char
const char * s = "a constant string"; // a read only pointer to a char array
```

Figure : 001_variables.c

These are items you'll commonly encouter while doing iPhone programming whose roots belong in the C family:

```
1   typedef enum {
2     NONE = 0,
3     SOME,
4     ALL
5   } HowMany;
6
7   HowMany i = SOME; // Use of enumerated constant
```

Figure : 002_enumerations.c

Note the `typedef` in this example, which is creating a type alias to be used later.

```c
// Declaring a point in space type
typedef struct {
  float x;
  float y;
} APointInSpace;

APointInSpace point;
point.x = 20.0;
point.y = 410.0;

// C99 style struct instantiation
APointInSpace point2 = { .x= 21.0, .y=32.0 };
```

Figure : 003_typedef.c

```c
#include <stddef.h>

// a constant byte array
char * constantCharArray = "Or in other words, a string";

// C99 style array instantiation
int aListOfNumbers[5] = { 1, 2, 3, 4, 5 };
int aListOfNumbers2[] = { 1, 2, 3, 4, 5};

// C99 Variable length array declaration
int someLength = 5;
float aListOfFloats[someLength];

// pointer declaration to NULL
int * someIntegerPtr = NULL;
int someInteger = 5;

// address dereference
someIntegerPtr = &someInteger;
```

Figure : 004_arrays_and_pointers.c

```c
float aSimpleAdditionMethod(float a, float b){
  return a + b;
}

float aSimpleSubtractionMethod(float a, float b){
  return a - b;
}

int aFloatAdditionThatReturnsAnInt(float a, float b){
  return (int) (a + b);
}

float (*anFnPtr)(float, float) = aSimpleAdditionMethod;
anFnPtr(1, 1); // returns 2
anFnPtr = aSimpleSubtractionMethod;
anFnPtr(1, 1); // returns 0
```

Figure : 005_functions.c

```c
#include <stdio.h>
#define FOO 1
#define IS_ZERO(n) (n == 0)

int main(){
  printf("FOO is %i\n", FOO);
  if(IS_ZERO(0)) {
    printf("This code is very useful.\n");
  }
}
```

Figure : 006_preprocessor.c

If you never learned C or it's been a long time, a fantastic book for you is "The C Programming Language". It doesn't cover many of the newer items in C99, but it is a thorough reference to the language fundamentals.
If you'd like to read more about the new features introduced by C99, read more at C99Changes.

Syntax

```
1   NSString * anExampleString = @"This is a pointer to an NSString object";
2   int strLen = [anExampleString length];
3
4   NSString * aSecondString = [anExampleString
5     stringByAppendingString:@" that had a second string appended to it"];
6
7   if([anExampleString
8       compare:@"this is a pointer to an nsstring object"
9       options:NSCaseInsensitiveCompare] == NSOrderedSame){
10    NSLog(@"The string %@ was case insensitive equal", anExampleString);
11  }
```

Figure : 007_basic_objc_example.m

Calling functions

Given some Objective C object, like:

```
1    NSString * anObject = @"TheObject";
```

you can invoke functionality on that object via the form:

```
1   [anObject theMethodName];
```

Method names in Objective C follow the form `initialMethodNameAndArgument:theSecondArgument:theThirdArgument:`.

When you see references to “calling a method” or “calling a function” on an object in Objective C, it’s best to consider these synonymous with “dispatching a message”. Objective C objects respond to **messages**, and as you work more with Objective C, it will be more beneficial for you to perceive this as such.

nil, NULL

An object in Objective C may be nil; nil is a “special” object which may have any message at all sent to it, to which it will respond with a nil object.

```
1   NSString * anObject = nil;
2   if([anObject whoaThisProbablyDoesntExist] == nil){
3     NSLog(@"Ah well");
4   }
```

Figure : 008_calling_methods_on_nil.m

nil is the appropriate null object when dealing with pointers to Objective C classes; NULL, on the other hand, is the appropriate null value to use when dealing with pointers to all other types.

Basic Types

There are a host of useful classes in the Cocoa Touch framework that provide special functionality to you. Some of the more common objects you'll use in your programming are NSArray, NSDictionary, NSNumber and NSString. Each of these objects have mutable versions that allow you to modify them after instantiation.

The special type `id` refers to any valid NSObject or its subclass. You can use the `id` type when you're not sure what sort of object you may be given.

NSString is an enhanced, unicode aware string class that goes far beyond the simple byte-array behavior of C's byte-array strings.

```
1   NSString * s = @"This is an NSString";
2   NSMutableString * so = [[NSMutableString alloc]
3                                   initWithString:@"This is a mutable string"];
4
5   [so appendString:s];
6
7   NSLog(@"The mutable string is %@", so);
```

Figure : 008_nsstring_example.m

NSNumber is a simple abstract wrapper around numeric values which allows you to automatically convert its held value to the appropriate form, as well as having an object-like representations of a numeric value.

```
1   NSNumber * n = @(20.0f);
2   double d = [n doubleValue];
3   int i = [n intValue];
4   NSNumber * b = @(YES);
5   NSLog(@"Number is %@, double value is %f, int value is %i", n, d, i);
6   NSLog(@"Boolean number is %@, bool value is %i", b, [b boolValue]);
```

Figure : 009_nsnumber_example.m

NSDictionary is a generic dictionary object that you can use to hold key/value associations. A key can be any NSObject that responds to isEqual: and NSCopying; in most cases, your keys will be either NSString or NSNumber objects.

```
1   NSDictionary * a = @{
2     @"Foo": @"The Foo string",
3     @"Bar": @"The Bar string"
4   };
5
6   NSMutableDictionary * b = [[NSMutableDictionary alloc] init];
7   [b setObject:@"Another string" forKey:@(20)];
8
9   NSLog(@"a's value for Foo is %@, and b's value for 20 is %@",
10                            [a objectForKey:@"Foo"],
11                            [b objectForKey:@(20)]);
12
13  for(id key in a){
14    NSLog(@"The value for %@ is %@", key, [a objectForKey:key]);
15  }
```

Figure : 010_nsdictionary_example.m

NSArray is a simple way to collect NSObject inheriting classes into a sequential list. NSArray automatically retains each object added to it, and releases each object it holds once its own retain count has reached 0.

```
1   NSArray * a = @[@"Foo", @"bar", @"baz"];
2   NSMutableArray * b = [[NSMutableArray alloc] init];
3   [b addObject:@"Not foo"];
4   NSLog(@"The contents of a are %@ and b are %@", a, b);
5
6   for(id obj in a){
7     NSLog(@"The array contains %@", obj);
8   }
```

Figure : 011_nsarray_example.m

NSArray, NSDictionary, NSString and NSNumber may all be represented by the contents of a property list file (Plist). You can easily rebuild an NSArray or NSDictionary full of the property list contents by using the initialization method `initWithContentsOfFile:`

Key	Type	Value
▼ Root	Dictionary	(3 items)
Example Key	Number	2000
Some Other Key	String	I'm a value!
▼ A subarray of values	Array	(2 items)
Item 0	Number	1
Item 1	Boolean	☐

Figure : Sample Property List

```
1    NSDictionary * example = [[NSDictionary alloc]
2                                  initWithContentsOfFile:@"/path/to/sample.plist"]
3    NSLog(@"The subarray contents are %@",
4              [example objectForKey:"A subarray of values"]);
```

Figure : 012_plist_example.m

Apple also provides JSON serialize/deserialize capability in the `NSJSONSerialization` class. This class can decode NSString, NSNumber, NSArray and NSDictionary values much like property lists.

```
1   NSDictionary * dict = @{ @"foo": @"bar" };
2   NSData * jsonData = [NSJSONSerialization dataWithJSONObject:dict
3                                                       options:0
4                                                         error:nil];
5   if([jsonData writeToFile:@"./test.json" atomically:YES]){
6     NSData * inputJsonData = [NSData dataWithContentsOfFile:@"./test.json"];
7     id jsonDict = [NSJSONSerialization JSONObjectWithData:inputJsonData
8                                                   options:0
9                                                     error:&error];
10    NSLog(@"Foo's value is %@", [jsonDict objectForKey:@"foo"]);
11  }
```

Figure : 013_json_example.m

Classes

Classes in Objective C are similar to their C++ cousins in that they come in two parts: interface declaration and implementation. The interface declaration portion of a class follows the form:

```
1   #import <Foundation/Foundation.h>
2   @interface MyNewClass : NSObject {
3     int privateVariable1;
4     NSString * privateVariable2;
5   }
6
7   +(int) someClassMethod;
8   -(void) doSomething;
9   -(NSString *) giveMeAStringOfLength:(int)length
10                    randomizeContents:(BOOL)randomize;
11  @end
```

Figure : 014_objects.m part 1

The implementation of this class follows the form:

```
1   #import "MyNewClass.h"
2
3   @implementation MyNewClass
4
5   -(id) init {
6     self = [super init];
7     if(self){
8       privateVariable1 = 20;
9       privateVariable2 = [[NSString alloc] initWithString:@"Test"];
10    }
11    return self;
12  }
13
14  +(int) someClassMethod {
15    return 0;
16  }
```

Figure : 014_objects.m part 2

```
1   -(void) doSomething {
2     NSLog(@"Something");
3   }
4
5   -(NSString *) giveMeAStringOfLength:(int)length
6                       randomizeContents:(BOOL)randomize {
7     return @"TODO: Make work";
8   }
9
10  @end
```

Figure : 014_objects.m part 3

Properties

Much of the interaction with Objective C objects equates to the usual getter/setter functionality commonly associated with Object Oriented Programming. To make it easier for programmers to declare this functionality, properties were introduced as a way to eschew all the boilerplate code and provide a common framework upon which to enhance Objective C objects.

```
1   @interface MyObject : NSObject {
2
3   }
4   @property(strong) NSString * instanceVariable;
5   @property float numericVariable;
6   @end
7
8   @implementation MyObject
9
10  @end
```

Figure : 015_properties.m part 1

```
1     MyObject * mo = [[MyObject alloc] init];
2
3     // Bracket Syntax
4     [mo setInstanceVariable:@"Test"];
5     [mo setNumericVariable:20.0f];
6     NSLog(@"The values I set were %@ and %f",
7             [mo instanceVariable],
8             [mo numericVariable]);
9
10    // Dot Syntax
11    mo.instanceVariable = @"Test 2";
12    mo.numericVariable = 25.0f;
13    NSLog(@"The values I set were %@ and %f",
14            mo.instanceVariable,
15            mo.numericVariable);
```

Figure : 015_properties.m part 2

You can specify the memory management semantics (strong, weak, copy), atomicity (locking behavior), and access level in the property declaration. You can also specify your own implementation of the getter and setter methods.

```
1    @property(readonly) NSString * privateName;
2    @property(weak, nonatomic, setter=myAgeSetter:) NSNumber * age;
```

Figure : 016_advanced_properties.m

Properties do not merely provide getter/setter functionality. The use of properties also implicitly adds Key-Value-Observing functionality to your class, letting you automatically monitor classes for change events and performing specific code in such cases. Read more in the SDK Documentation on properties and Key Value Observing.

Selectors

Selectors in Objective C are a way to indicate a message as a variable. They allow you to dynamically send a message to an object, as well as to query an object to see if it responds to a given message.

```
1     SEL aSelector = @selector(length);
2     NSString * foo = @"Foo";
3     if([foo respondsToSelector:aSelector]){
4       NSLog(@"The length of foo is %i",  [foo performSelector:aSelector]);
5     }
```

Figure : 017_selectors.m

Selectors can be built from strings, and can refer to any Objective C message that is forwarded to an object.

```
1   NSString * anAllCapsString = @"THIS STRING SHOULD BE LOWERCASE";
2   NSString * theMessageToSend = @"lowercaseString";
3   SEL aSelector = NSSelectorFromString(theMessageToSend);
4   if([anAllCapsString respondsToSelector:aSelector]){
5     NSLog(@"All caps string (%@) converted: %@",
6                 anAllCapsString,
7                 [anAllCapsString performSelector:aSelector]);
8   }
```

Figure : 018_advanced_selectors.m

Protocols

Objective C provides only single-inheritance for its objects; to allow for situations where a class may provide functionality outside of its inheritance chain, the language provides Protocols, which are roughly analogous to interfaces in Java and C#. There are informal protocols (only referred to in documentation) and formal protocols, which are compiler checked. The majority of protocol usage in iOS programming is formal protocols.

```
1   @protocol CameraDevice
2
3   -(NSString *) manufacturerName;
4
5   @optional
6
7   -(int) flashStrength;
8
9   @end
10
11  @interface AnExpensiveCanonCamera : NSObject<CameraDevice>
12  @end
13
14  @interface TheCheapestCameraEver : NSObject<CameraDevice>
15  @end
```

Figure : 019_protocols.m part 1

```
1     NSArray * cameras = @[
2       [[AnExpensiveCanonCamera alloc] init],
3       [[TheCheapestCameraEver alloc] init]
4     ];
5     for(NSObject<CameraDevice> * camera in cameras){
6       if([camera respondsToSelector:@selector(flashStrength)]){
7         NSLog(@"%@ camera has flash strength %i",
8                       [camera manufacturerName],
9                       [camera flashStrength]);
10      } else {
11        NSLog(@"%@ camera has no flash", [camera manufacturerName]);
12      }
13    }
```

Figure : 019_protocols.m part 2

Categories

Categories allow you to mix in new code to existing classes without having to change the original source code for those classes. Examples of categories include automatically adding special serialization rules to NSObject.

You “can’t” add instance variables to a class with categories; they are purely for adding new methods to a class, but cannot change the memory layout of a class after the fact.

```
1   @interface NSDictionary(AsJson)
2
3   -(NSString *) asJson;
4
5   @end
6
7   @implementation NSDictionary(AsJson)
8
9   -(NSString *) asJson {
10      NSData * jsonData = [NSJSONSerialization dataWithJSONObject:self
11                                                          options:0
12                                                            error:nil];
13      return [[NSString alloc] initWithData:jsonData
14                                   encoding:NSUTF8StringEncoding];
15  }
16
17  @end
```

Figure : 020_categories.m part 1

```objc
#import "NSDictionary+AsJson.h"

void main(){
  NSLog(@"Dictionary as json: %@",
                [@{ @"A string": @"String!",
                    @"A number": @(20),
                    @"An Array": @[@(1), @(2), @(3)]} asJson]);
}
```

Figure : 020_categories.m part 2

Memory management

Figure : Memory Management circa 1973

C based languages have traditionally used `malloc` and `free` as the means by which memory was allocated on demand. This code looked something like:

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main(){
5     int * anInteger = malloc(sizeof(int));
6     *anInteger = 10;
7     printf("Integer value is %i", *anInteger);
8     free(anInteger);
9     return 0;
10  }
```

`malloc` and `free` allocated memory on the heap; it was the programmer's responsibility to indicate when a particular item was ready to be released back to the operating system.

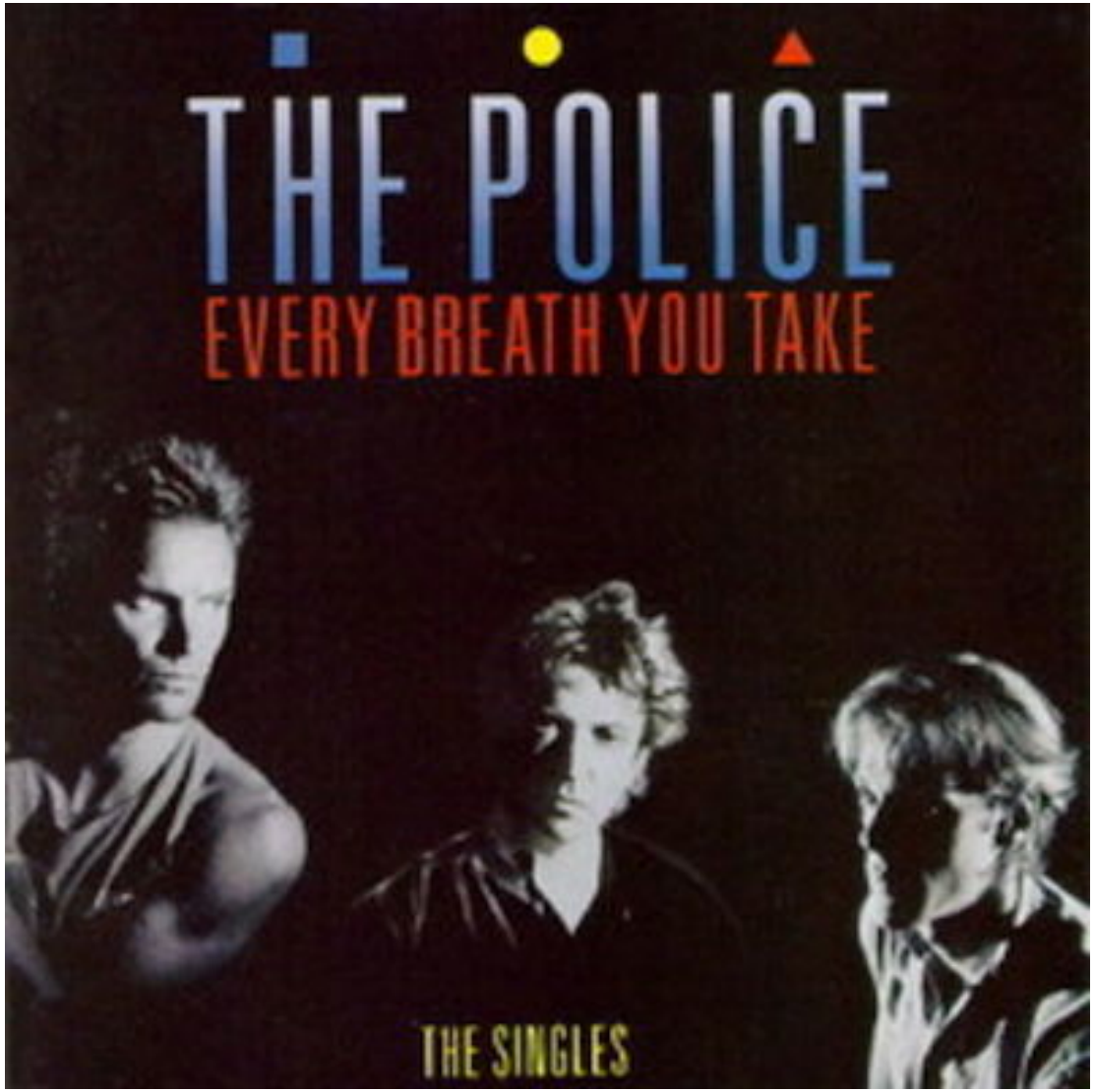


Figure : Memory Management circa 1983

Objective C used an “easier” means of managing the memory for its objects: reference counting.
Reference counting allowed Objective C programmers to write code that looked like this:

```
1    NSString * c = [[NSString alloc] initWithString:@"This is a string"];
2    NSLog(@"String is %@", c);
3    [someOtherObject setString:c];
4    [c release];
```

Each object would respond to two methods: `retain` and `release`. Retain would increment an object's retain count, and release would decrement. When an object's retain count reached 0, it was considered "deallocated", and its memory on the heap would be available for use by other objects.

Objects could also be flagged as `autorelease`, which would send a release to an object at some point in the future. Typically this meant after one main NSRunLoop cycle (one UI loop).

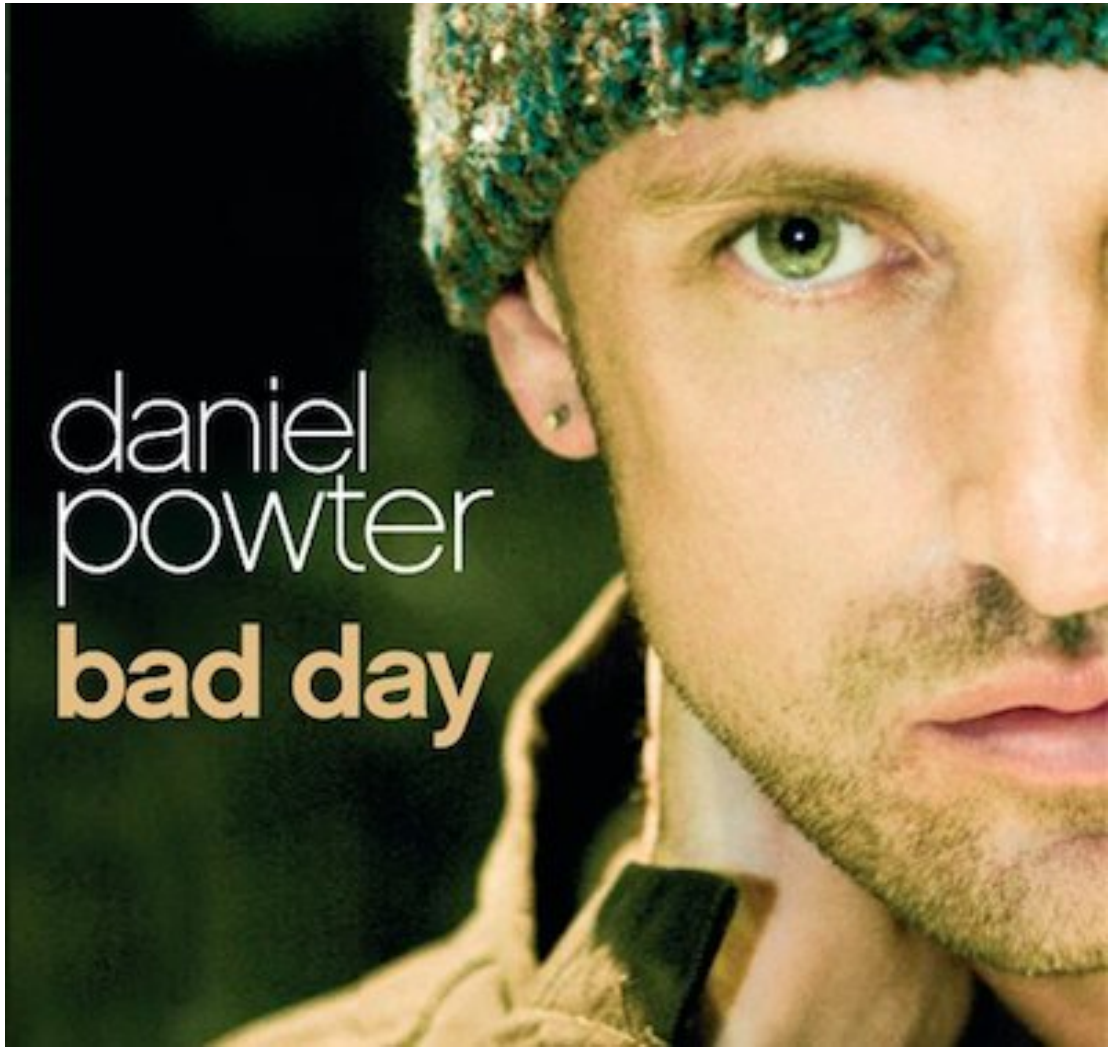


Figure : Memory Management circa 2006 - Apple adds Garbage Collection to Objective C

Figure : Memory Management circa 2008 - The iOS SDK 2.0 and App Store launch. The iPhone does not support Garbage Collection due to performance concerns.

Figure : Memory Management circa 2011 - ARC

Apple introduces a new tool called ARC (*Automatic Reference Counting*). Improvements in static code analysis allow Apple to detect and automatically insert retain/release calls for most code.
In other words, *most of the basic memory management is now done for you*.

What ARC *will do* for you:

- Let you avoid calling `retain` or `release` on Objective C objects

What ARC **will not do** for you:

- Prevent cyclic memory ownership
- Allow you to ignore `malloc` or `free` when using C-based apis
- Allow you to totally ignore memory management

```
1    NSString * s = [[NSString alloc] initWithString:@"Some string"];
2    NSLog(@"My string is %@", s);
3    // That's it.
```

```
1     NSString * s = [[NSString alloc] initWithString:@"Some string"];
2     [someOtherObject setString:s];
3
4     // and in the someOtherObject's implementation
5
6     @interface SomeOtherObject
7
8     @property(strong,nonatomic) NSString * string;
9
10    @end
```

Usage of ARC is by default on all new iOS projects. You can choose to disable it and manually manage memory. You can also still manually manage memory created through `malloc` and `free` if necessary.

More learning

Objective C is a deep language. There are lots of resources on the web and your computer where you can learn more. The best is easily the XCode documentation, but there are blogs out there that can provide invaluable help:

- Mike Ash's NSBlog
- objc.io
- NSHipster
- Apple Developer Forums

1 iPhone App Basics: UIViewControllers

MVC, as applied to an app

Apple strongly encourages you to adhere to Model View Controller pattern

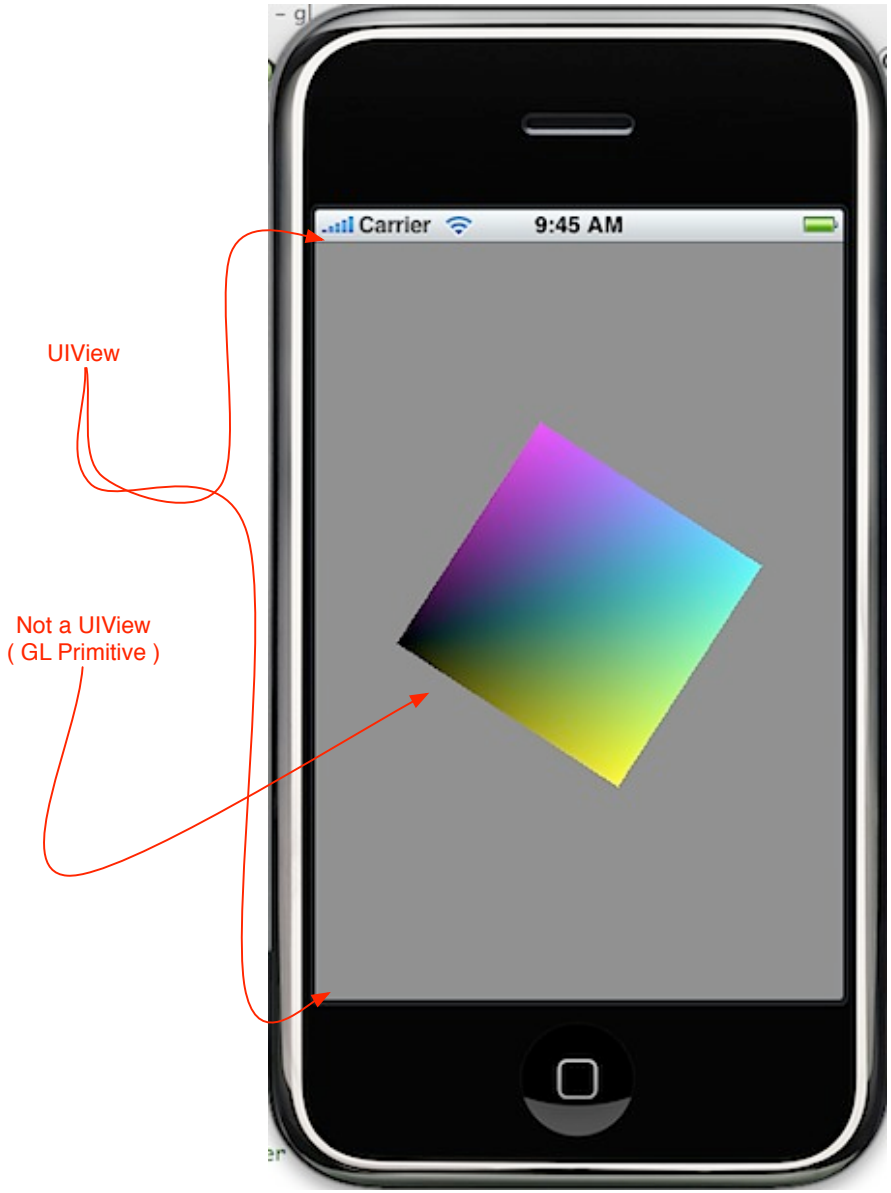
Figure : Model, View and Controller

Application
$\Rightarrow$ *Window* $\Rightarrow$ *ViewController*(*s*) $\Rightarrow$ *View*(*s*) + *Models*

On iOS, a UIApplication (your app) typically has one UIWindow, which is the primary UIView upon which everything else displays. A UIView is “just” a rectangle upon which you draw things.

Almost everything visual is a UIView (except the things that aren't).


UIView
1:09 PM
CNN
Top Stories
World
U.S.
Politics
Crime
Breaking news
Flight 1549; Airplane crash-lands into Hudson River
How a 'perfect storm' led to the economic crisis
How long does Obama have to fix the economy?
My CNN
Video
iReport


UIView
Not a UIView
(GL Primitive)
Carrier
9:45 AM

UIViewControllers are objects that manage what the views on screen are currently doing; typically where you write event handling code, user interaction code, and interact with your models.

There is no “official” model object in this pattern. Your business objects are the models, however you decide to implement them.

Storyboards

Storyboards let you visualize the interaction between UIViewControllers. You can design both the UIView layout and the interaction between UIViewControllers using the Storyboard designer.

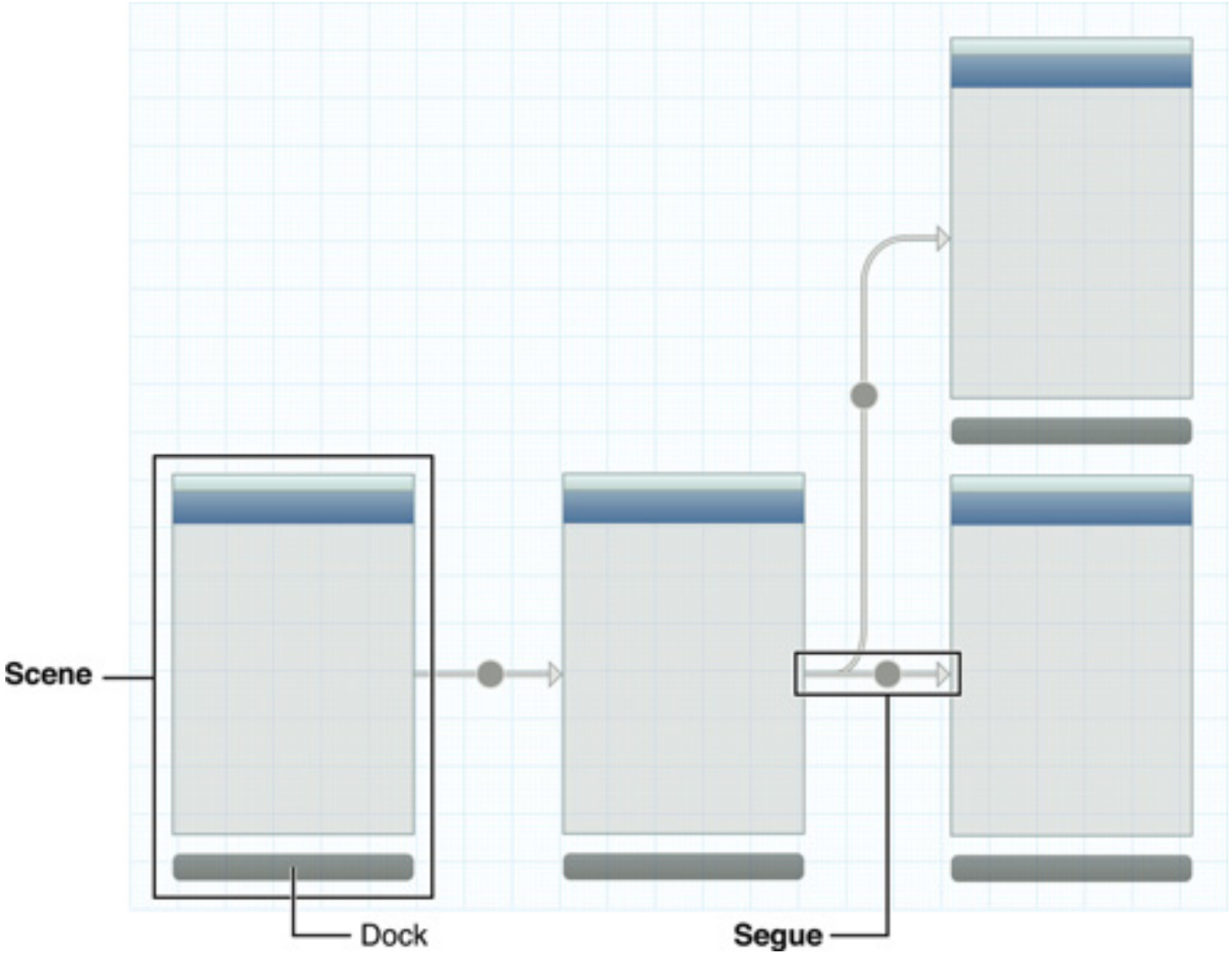


Figure : The Storyboard designer

Storyboards are comprised of **scenes** and **segues**. A **scene** represents a UIViewController's presentation. A **segue** is the transition between **scenes**, and carry information about the transition origin and destination, as well as the nature of the transition.

Lab 2

Create a new “Single View” project

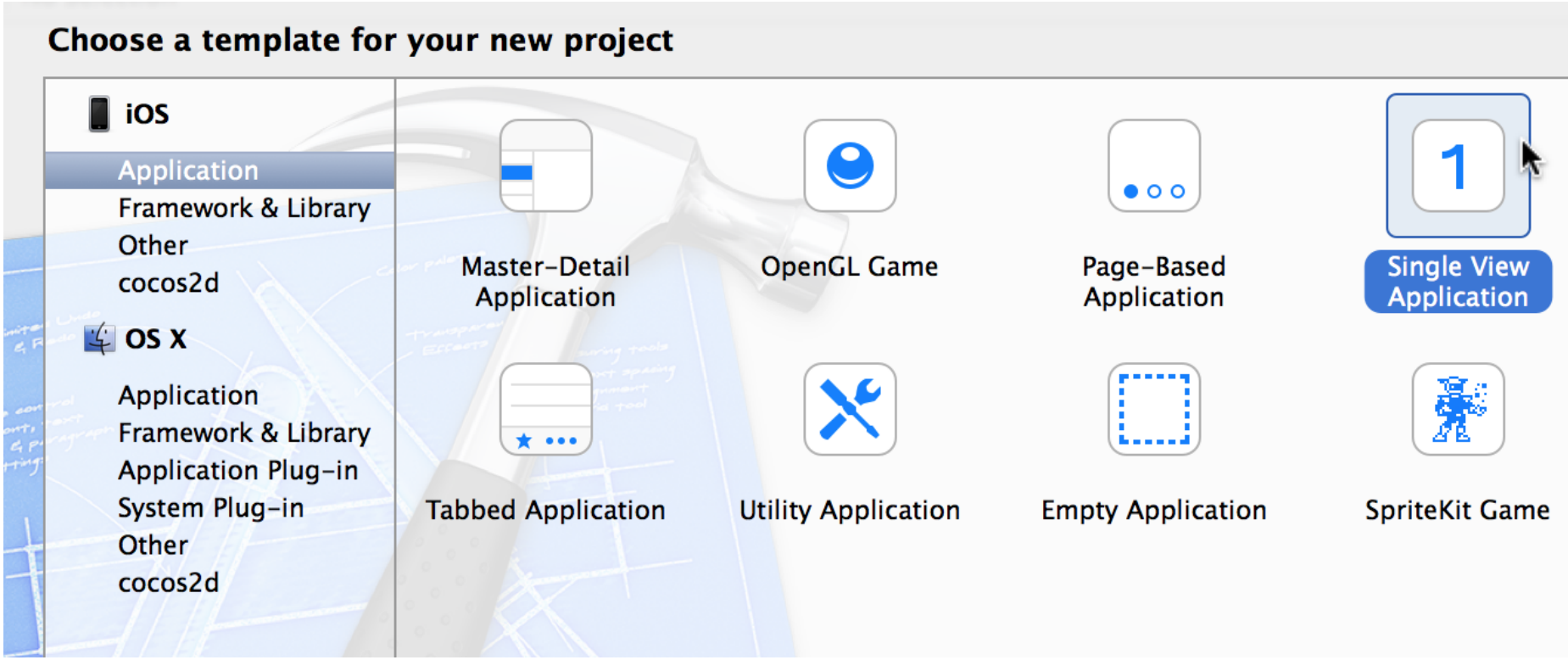


Figure : Create a new project

Add a new NSObject subclass of “UIVIewController” to the project named “DetailViewController”

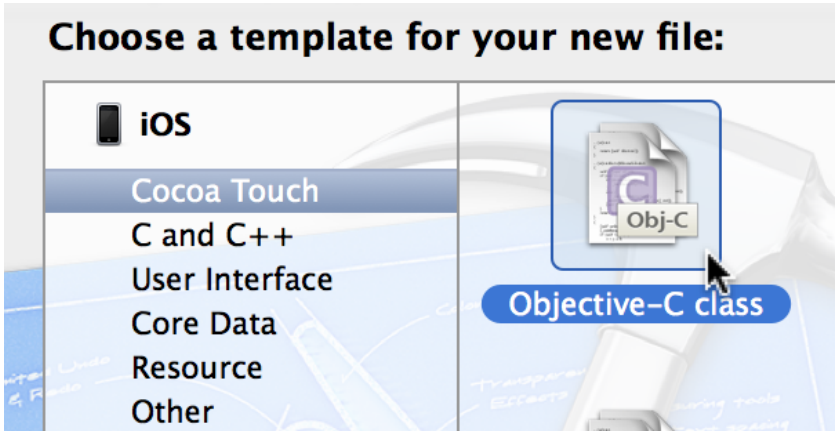


Figure : New Objective C class

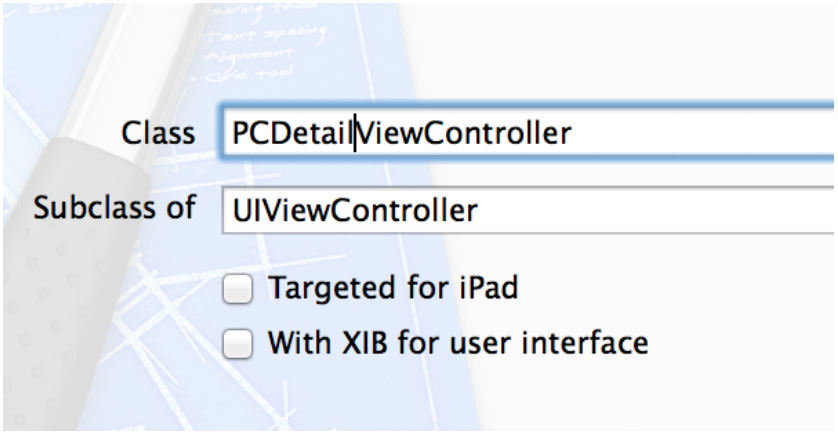


Figure : Subclassing UIViewController

Add a “UIViewController” object from the Object library on to your storyboard.

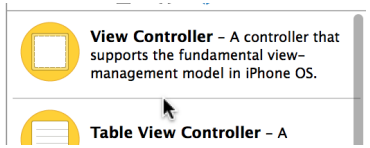


Figure : The UIViewController object in the Library

Figure : Drop UIViewController on Storyboard

Set the subclass of the UIViewController to “DetailViewController”.

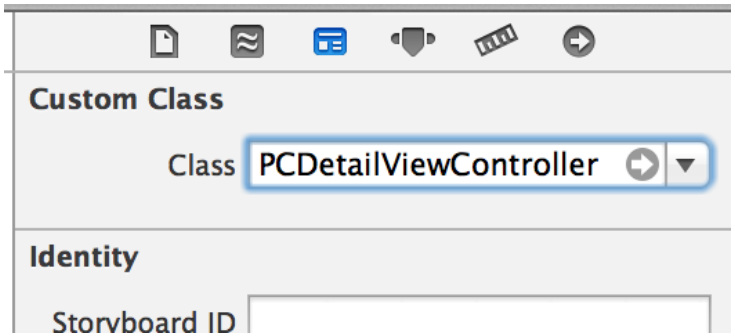


Figure : Setting the UIViewController subclass

Double click on the first UIViewController in the stoyrboard and add a button with the text “Show Detail” to its scene.

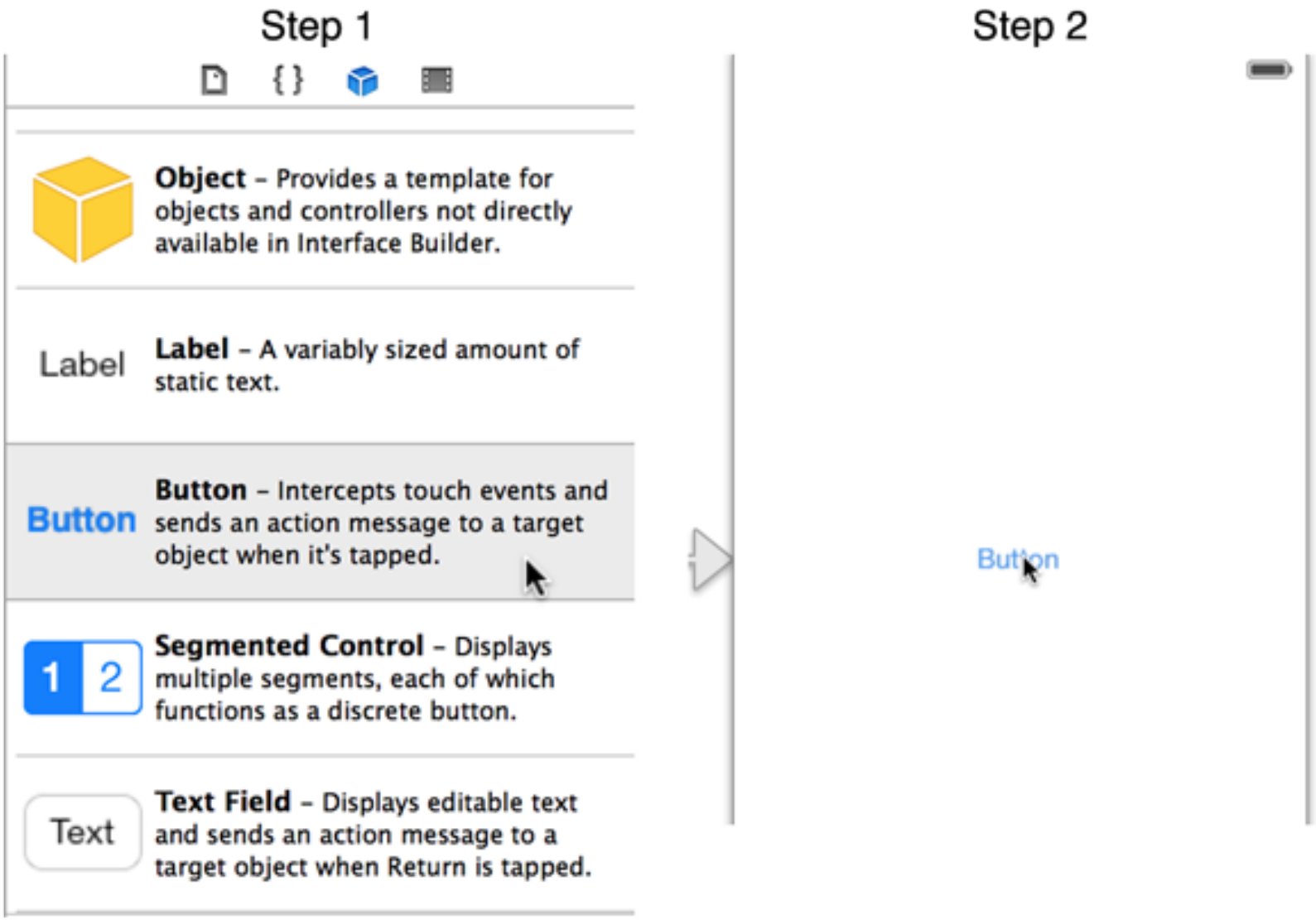


Figure : Drag a button to the scene

Select the button, and view its outlets in the inspector pane. Drag from the `action` outlet under `Triggered Segues` to the “Detail View Controller” scene in your storyboard. Choose “Modal” when asked what sort of segue style will be used.

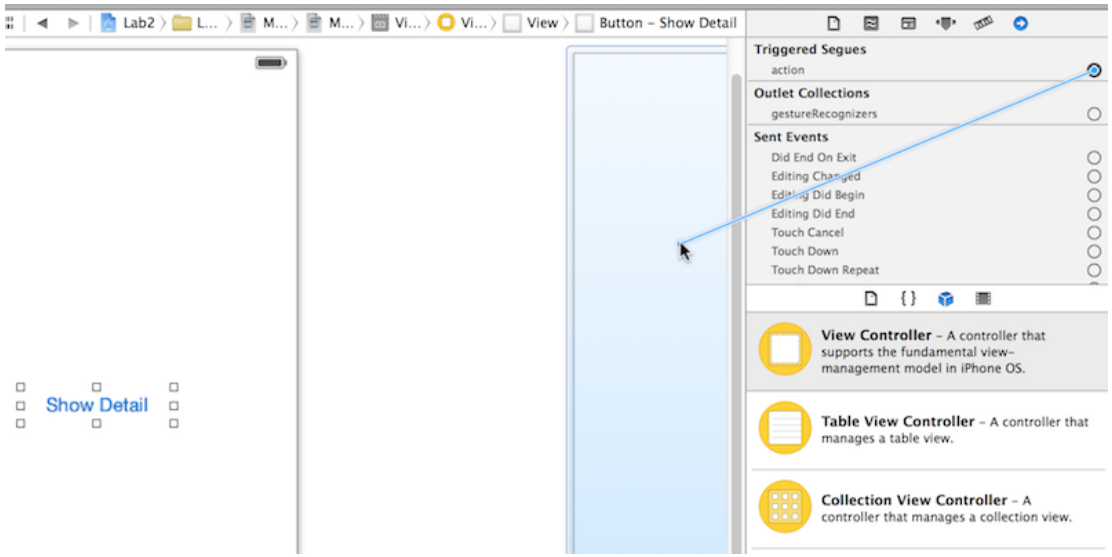


Figure : Connect the action to the Detail View Controller

Double click on the Detail View Controller and add a button to its scene with the text “Close”.

Add the following code to `DetailViewController.h`

```
1     -(IBAction) dismissDetailView:(id) sender;
```

Add the following code to `DetailViewController.m`

```
1   -(IBAction) dismissDetailView:(id) sender {
2     [self dismissViewControllerAnimated:YES completion:nil];
3   }
```

In your Storyboard, select the “Close” button you added to the Detail View Controller, and connect its “Touch Up Inside” outlet to the Detail View Controller. Select the “dismissDetailView:” selector when prompted which selector to connect to.

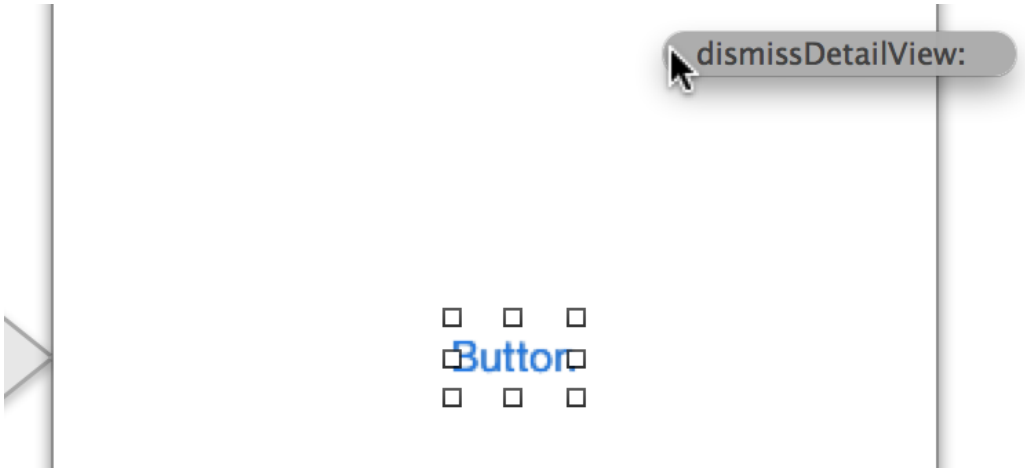


Figure : Connect the action to the dismiss selector

Run the project!

Figure : Yay, it works!

One more thing: Add another button on the Detail View controller that opens another View Controller as a modal, and this new view controller has a button on it that when tapped opens an alert box that says “Hello World”.

UINavigationController

UINavigationController is a stack based manager of view controllers that the user can navigate through. The tiny left-facing back arrow present in most iOS apps is its most obvious characteristic.

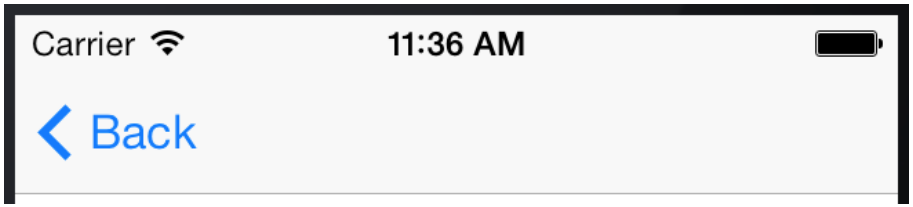
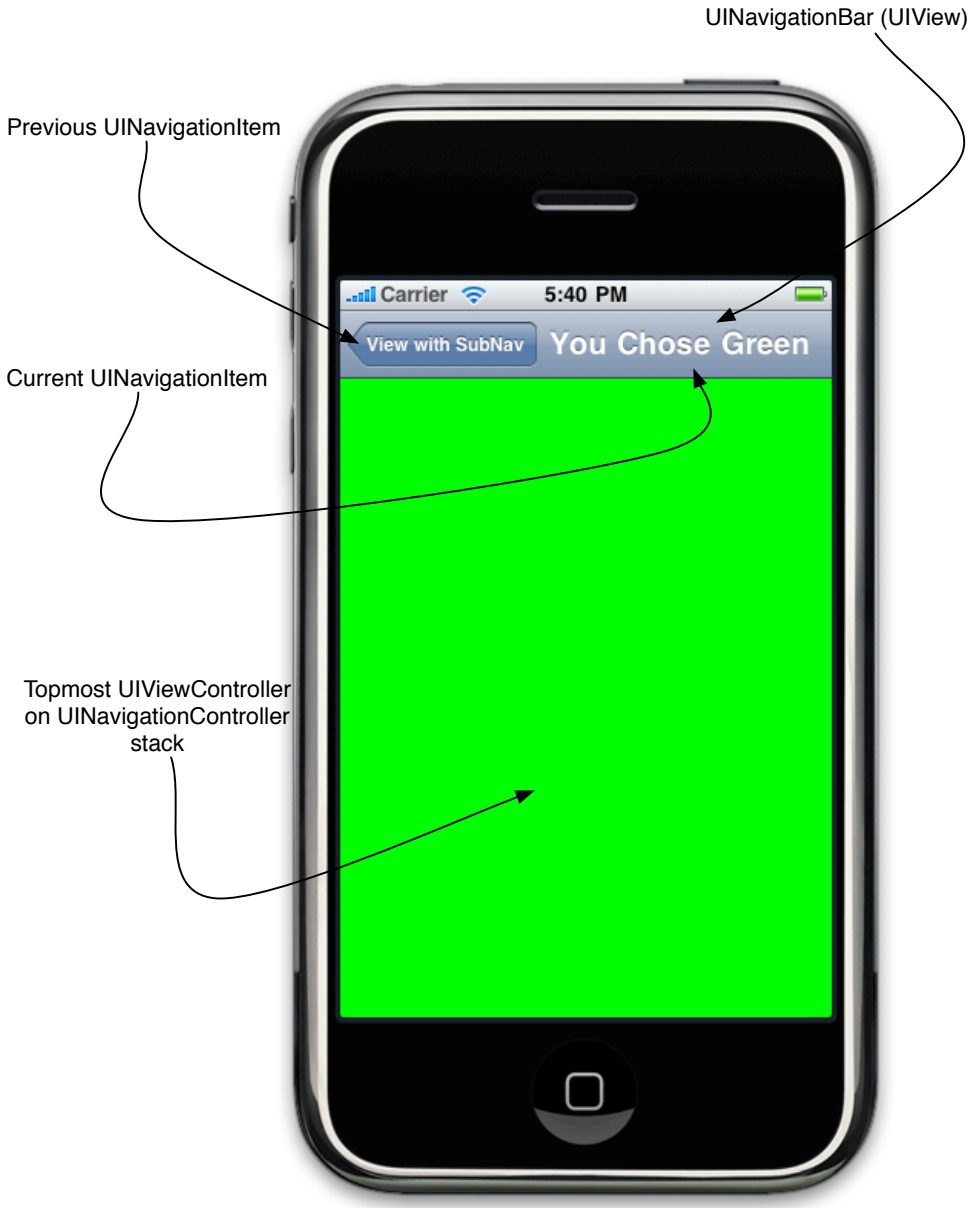


Figure : UINavigationController at work


UINavigationBar (UIView)
Previous UINavigationItem
Current UINavigationItem
Topmost UIViewController on UINavigationController stack
Carrier
5:40 PM
View with SubNav
You Chose Green

Every UIViewController managed by a UINavigationController has a reference to that UINavigationController in `[self navigationController]`.

```
1   -(void) viewDidLoad {
2     [super viewDidLoad];
3     NSLog(@"There are currently %i controllers in the navigation stack",
4           self.navigationController.childViewControllers.count);
5   }
```

You can add another UIViewController to the navigation stack by using the `push` segue when connecting view controllers in Storyboard.

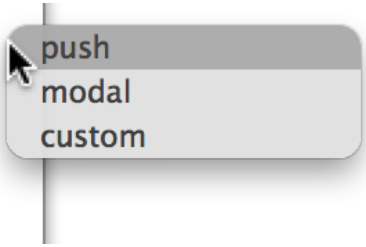


Figure : Selecting the push segue

Note: This segue will only work if you've already set up a UINavigationController, it will not automatically create a UINavigationController for you.

You can also push UIViewControllers on the stack manually.

```
1    MyViewController * controller = [[MyViewController alloc] initWithNibName:nil
2                                                                    bundle:nil];
3    [[self navigationController] pushViewController:controller animated:YES];
```

Lab 3

Create a new “Single View” project

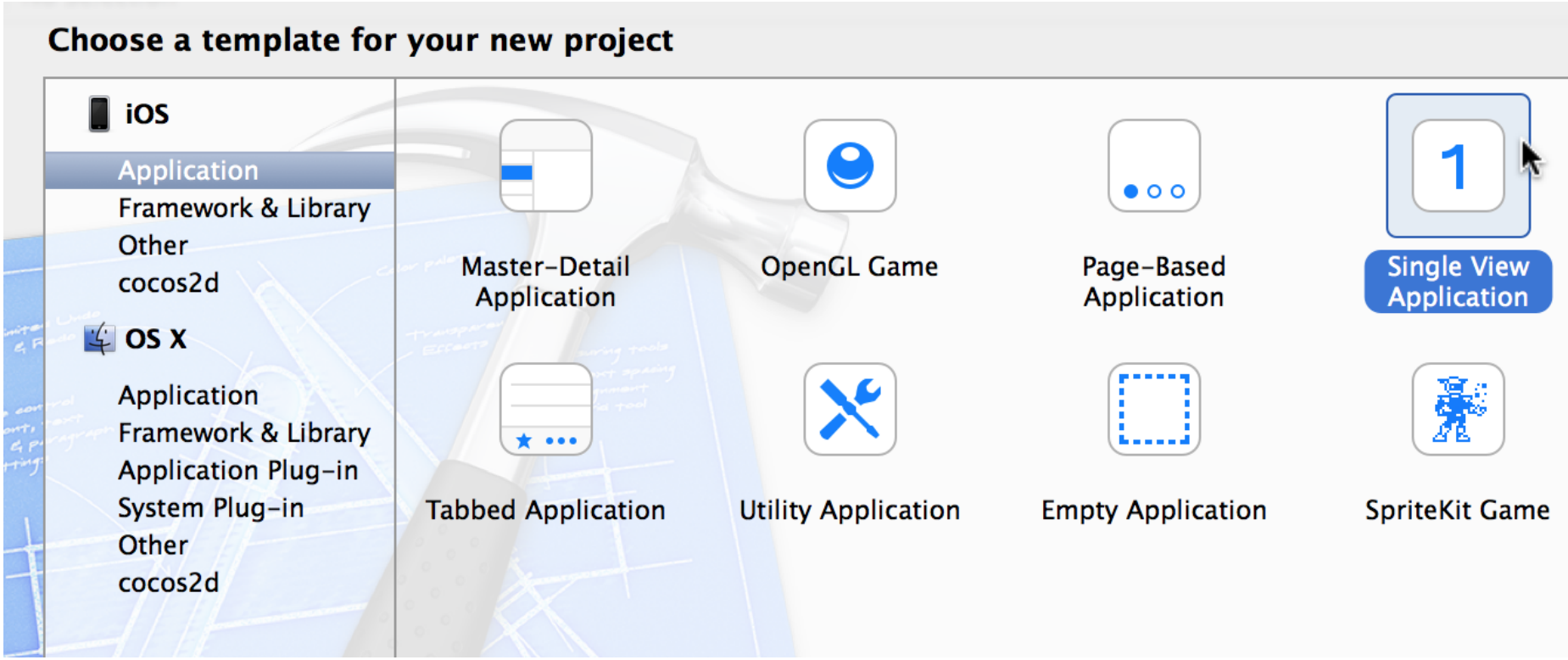


Figure : Create a new project

Open the project's storyboard and add a new "UINavigationController" to the storyboard.

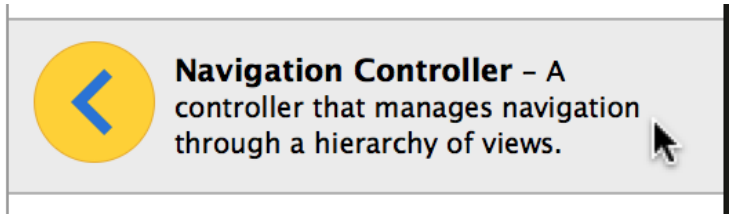


Figure : Select UINavigationController in the Object library

Drag the storyboard's "initial scene" arrow from the default view controller scene to the UINavigationController you placed.

Figure : Drag the initial scene arrow to the UINavigationController

Drag the “root view controller” outlet from the UINavigationController to the default view controller in the storyboard.

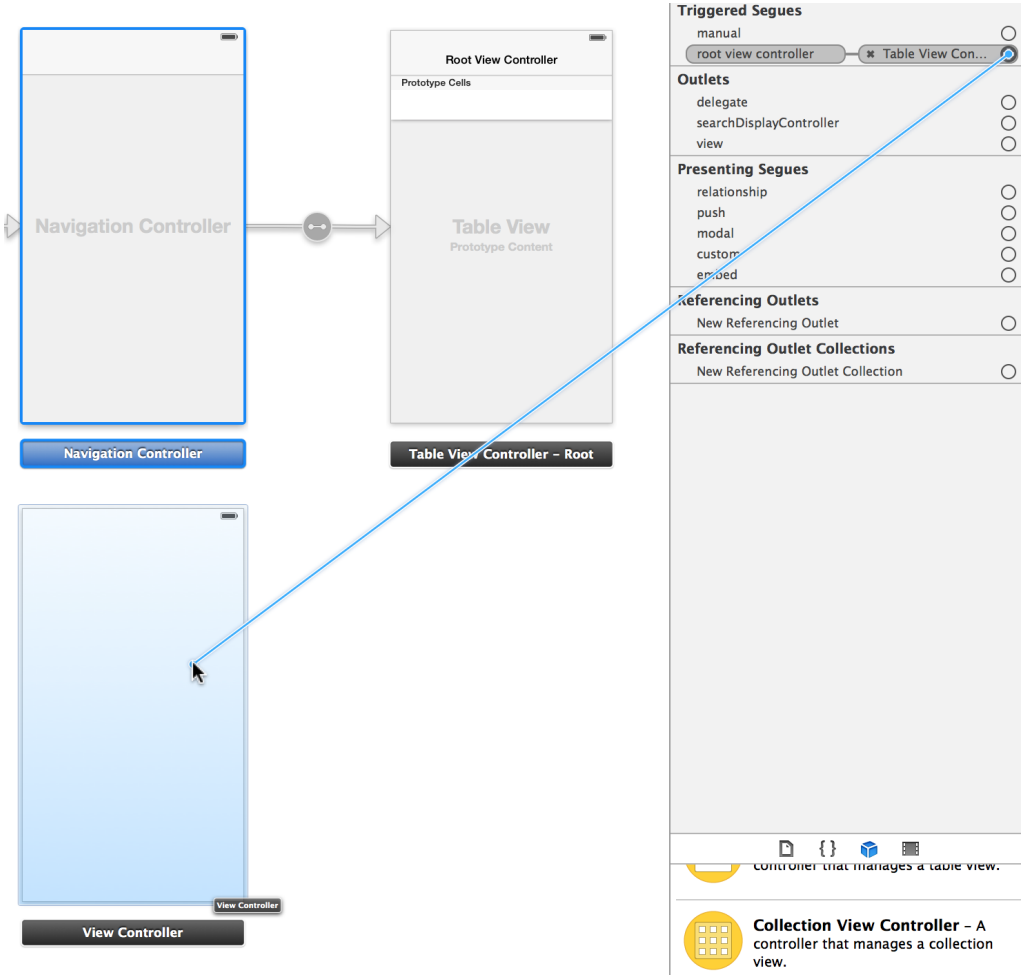


Figure : Change the root view controller outlet

Add a new NSObject subclass of “UIVIewController” to the project named “DetailViewController”

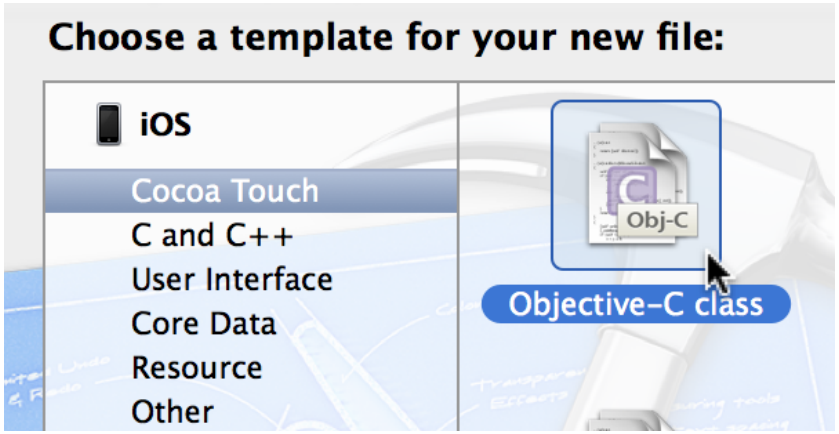


Figure : New Objective C class

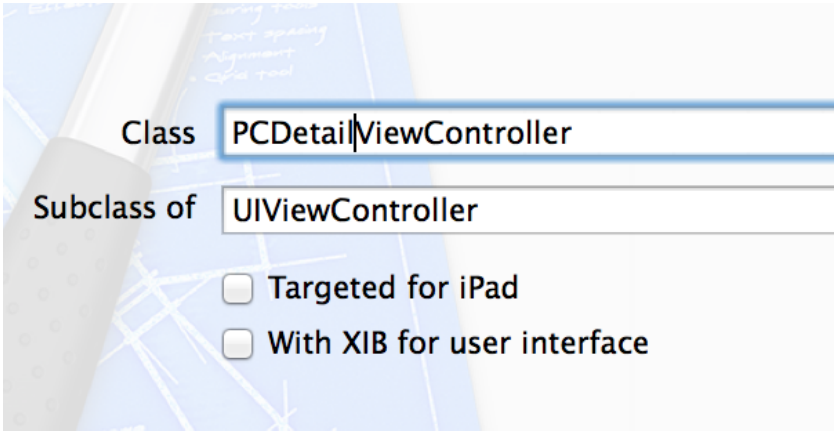


Figure : Subclassing UIViewController

Add a “UIViewController” object from the Object library on to your storyboard.

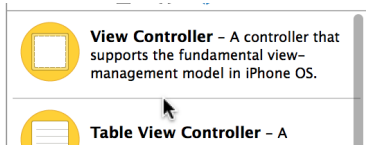


Figure : The UIViewController object in the Library

Figure : Drop UIViewController on Storyboard

Set the subclass of the UIViewController to “DetailViewController”.

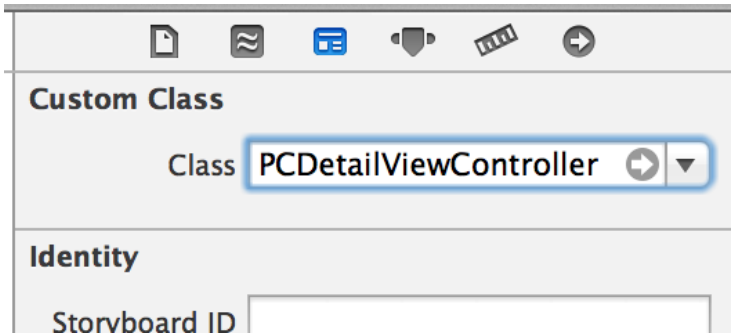


Figure : Setting the UIViewController subclass

Double click on the first UIViewController in the stoyrboard and add a button with the text “Show Detail” to its scene.

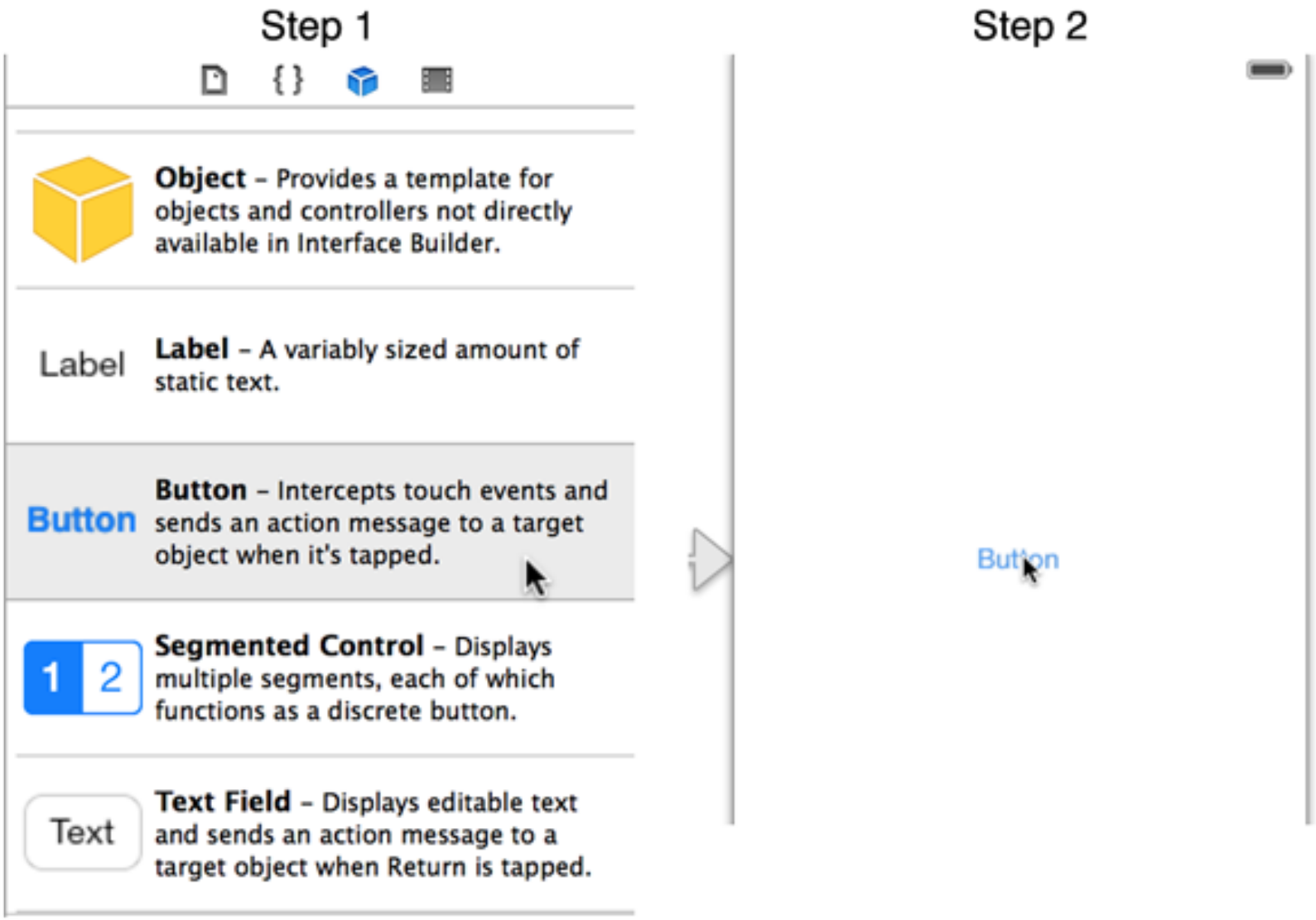


Figure : Drag a button to the scene

Select the button, and view its outlets in the inspector pane. Drag from the `action` outlet under `Triggered Segues` to the “Detail View Controller” scene in your storyboard. Choose “Push” when asked what sort of segue style will be used.

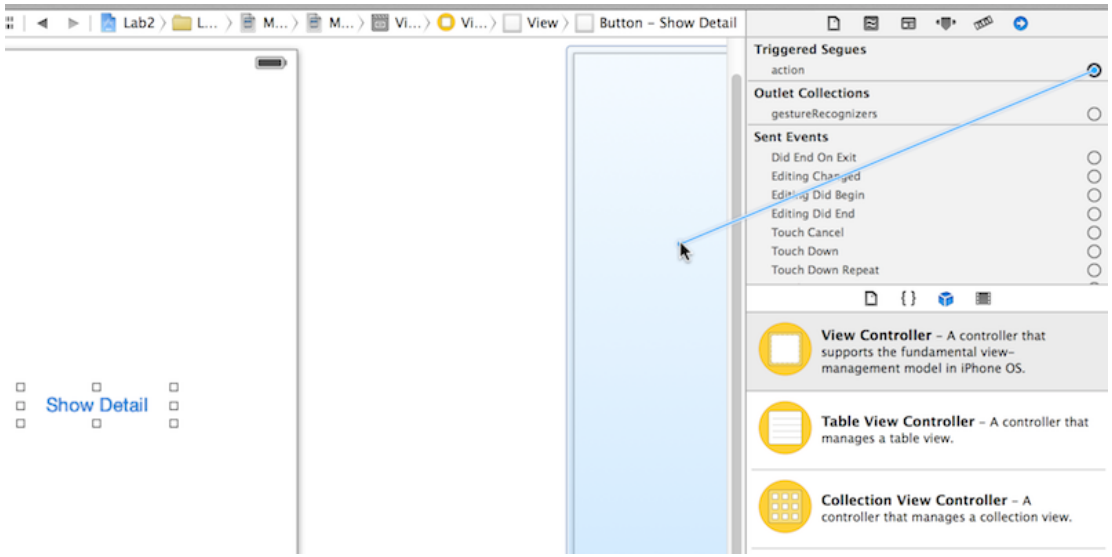


Figure : Connect the action to the Detail View Controller

Double click on the Detail View Controller and add a label to its scene with the text “Valuable Detail”.

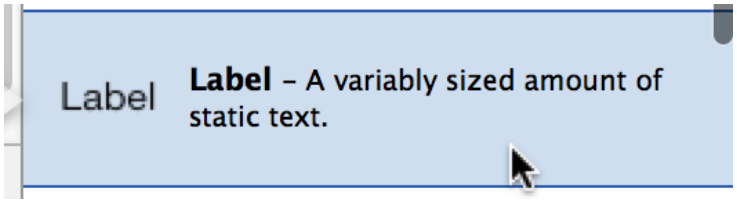


Figure : Select and place the Label control

Select the navigation bar in the detail view controller, and change its title attribute to “My Detail View”.

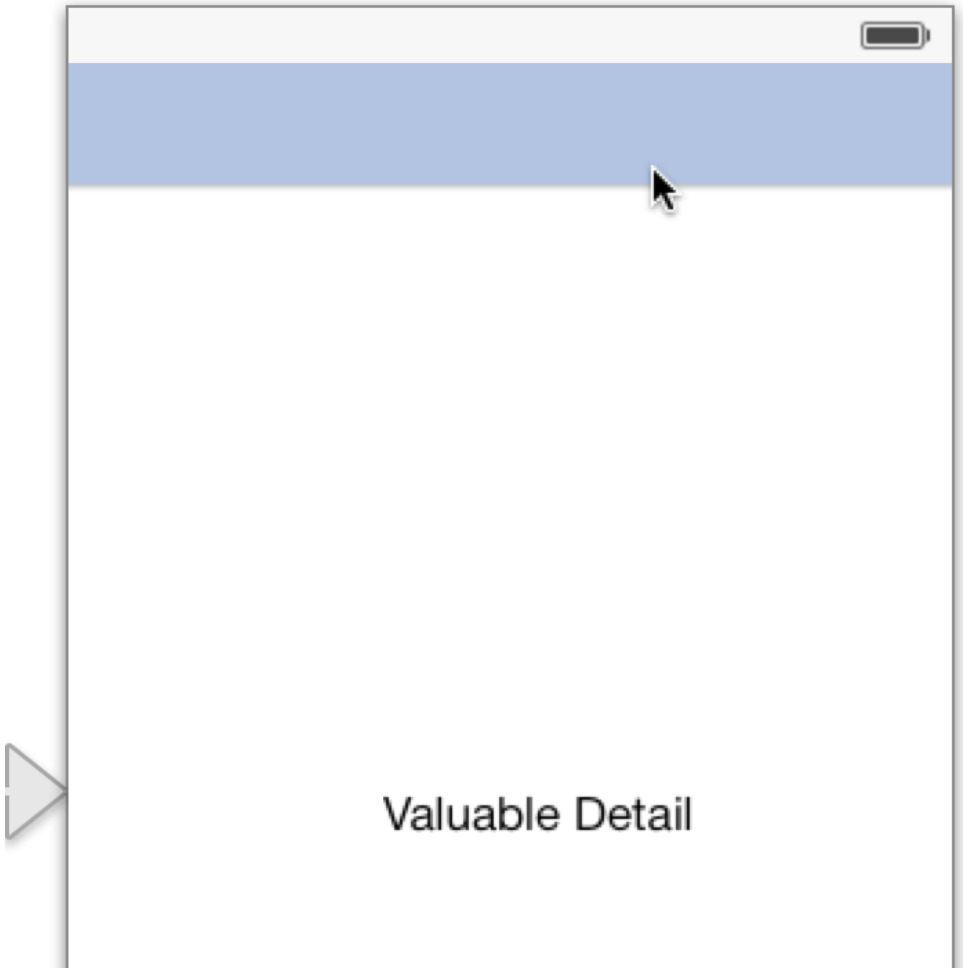


Figure : Select UINavigationItem in controller

Run the app!

Figure : Put it on the App Store!

One more thing: Add a button on your Detail View Controller that opens up a new View Controller that shows an image.

UITabBarController

A UITabBar is the control that sits at the bottom of a screen, and allows you to switch between different views based on the button you click.

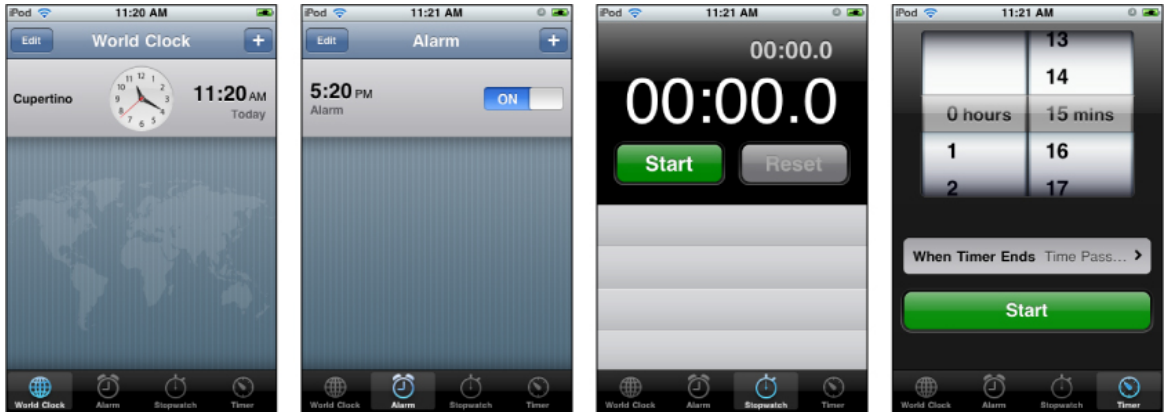


Figure : UITabBar examples

A UITabBarController swaps in UIViewControllers assigned to it when users click on the associated button

Each UIViewController has a tabBarItem property that UITabBarController uses to populate its UITabBar

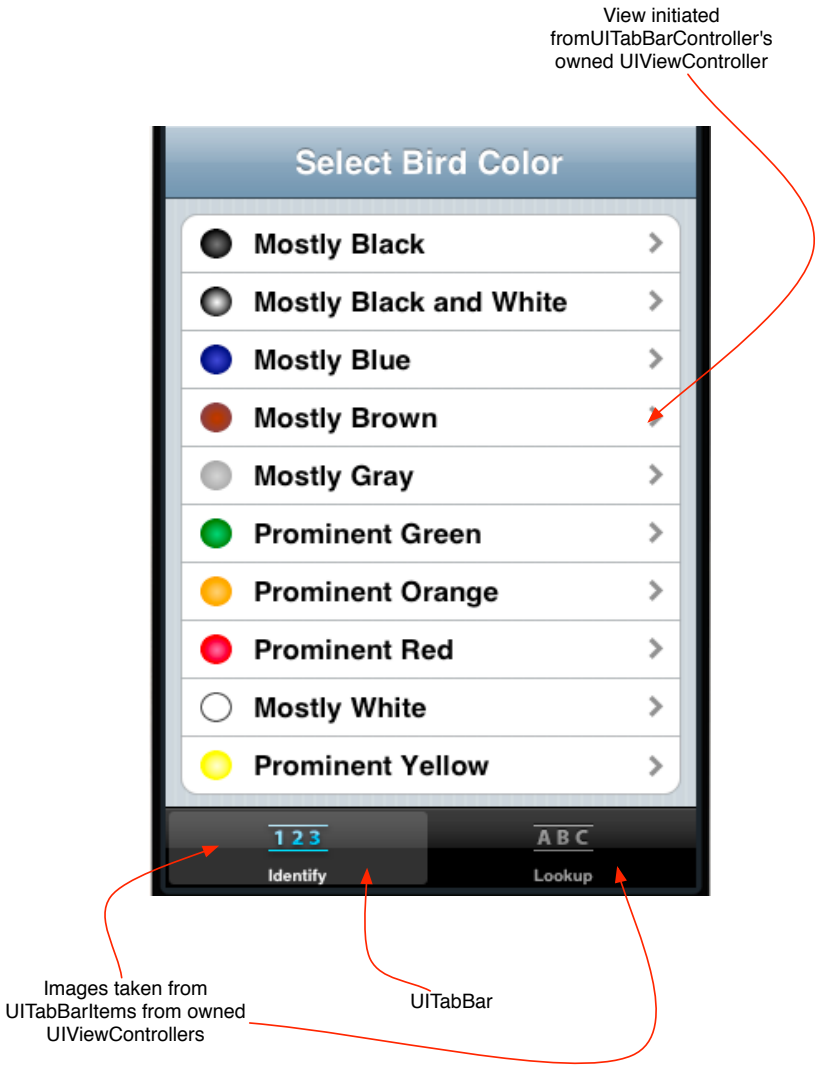


Figure : UITabBarController example

Figure : Sir Not Appearing In This Class

UITableViewController

UITableViews make up the majority of navigational aids in most iOS apps. If you see a vertical list of selectable items, it is almost certainly a UITableView.

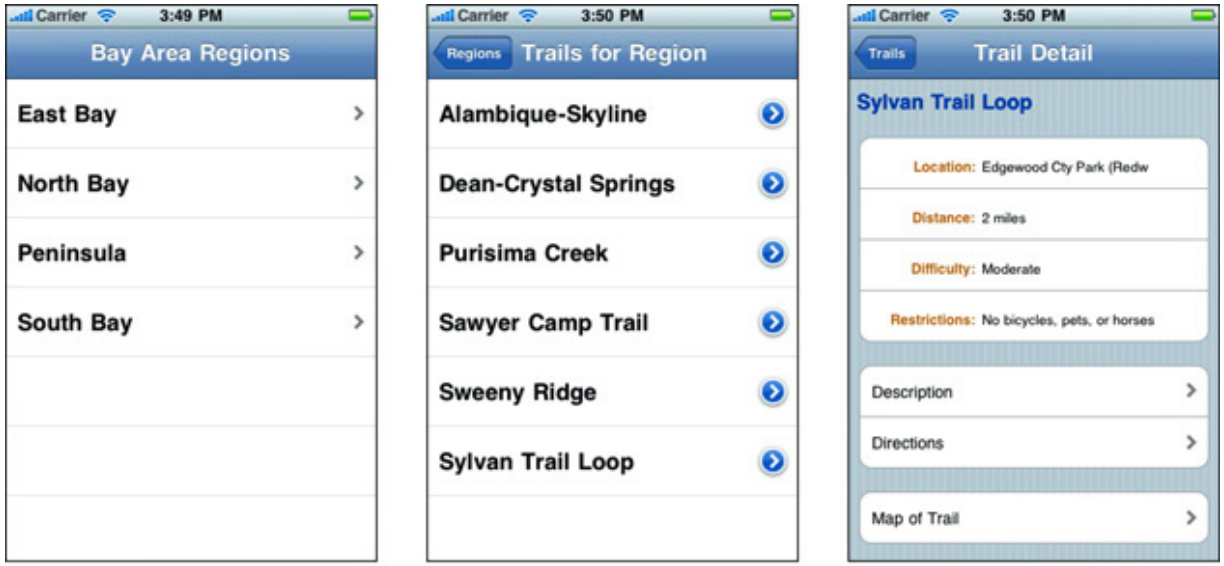


Figure : UITableView examples

A UITableView presents data in two dimensions: **sections** and **rows**.
For any section **N**, there may be **M** rows of information to present.

Each cell in a UITableView is represented by a UITableViewCell. A UITableViewCell in its default implementation allows for a title label, a left aligned image view, and an optional accessory view that is right aligned.

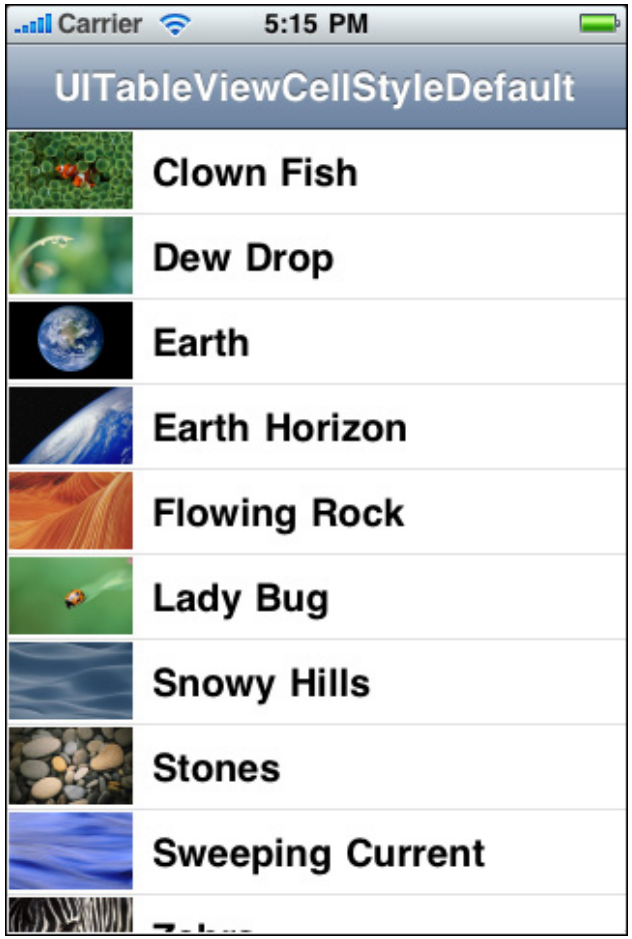
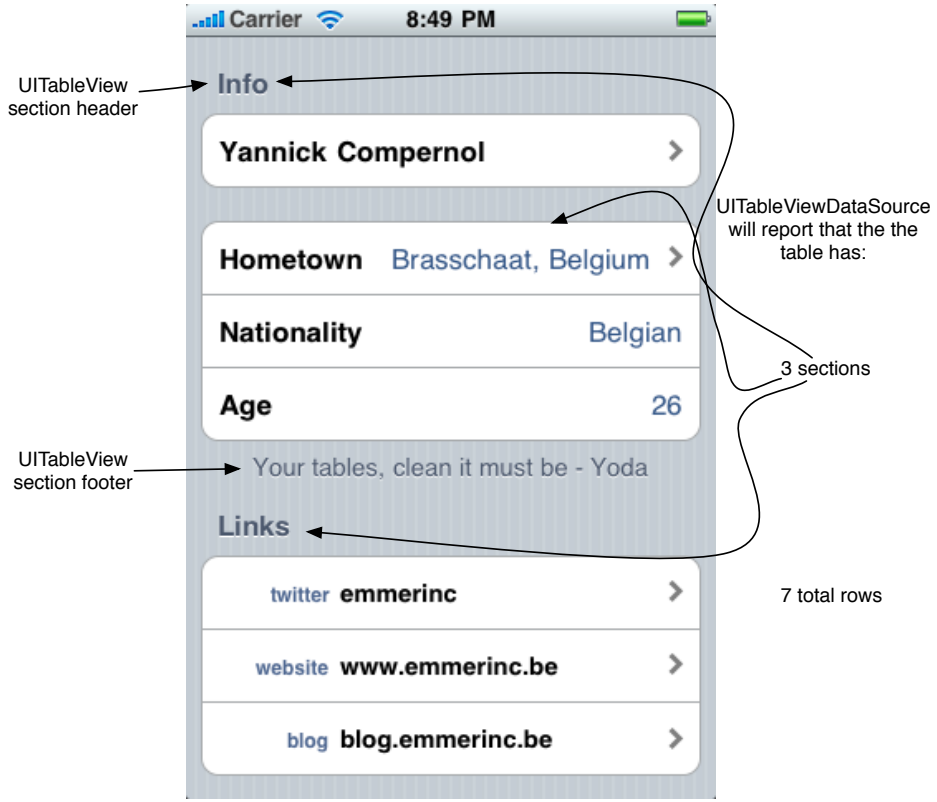


Figure : UITableViewCell default style

A UITableView has special code that allows it to only create as many UITableViewCells as are currently on the screen, as opposed to the number of rows in your source data. This allows you to easily support paging through many thousands of items w/ negligible performance impact.

A UITableViewController has many pre-filled methods to make interacting with UITableViews easier. It is a subclass of UIViewController, and is merely a convenience to the programmer.



Lab 4

Create a new “Single View” project

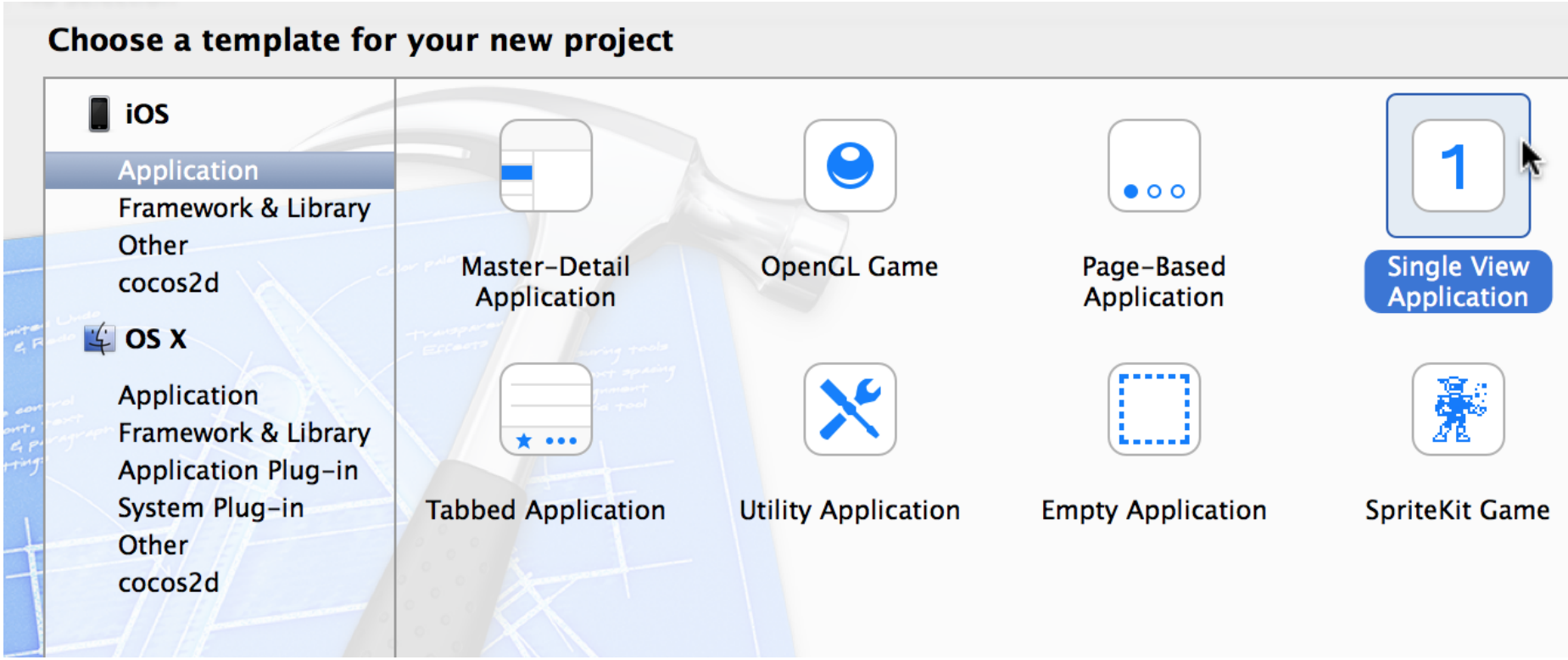


Figure : Create a new project

Open the project's storyboard and add a new "UINavigationController" to the storyboard.

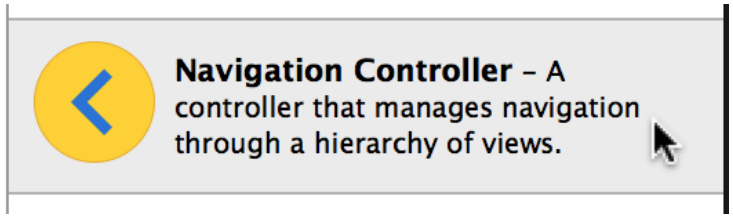


Figure : Select UINavigationController in the Object library

Drag the storyboard's "initial scene" arrow from the default view controller scene to the UINavigationController you placed.

Figure : Drag the initial scene arrow to the UINavigationController

Add a new subclass of NSObject to your project that is a subclass of “UITableViewController”.

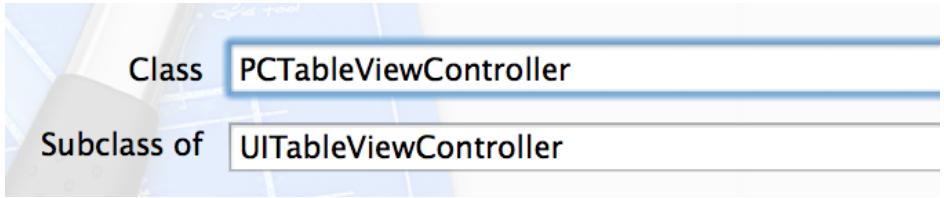


Figure : Add a new subclass of UITableViewController

Change the subclass of the UITableViewController connected to your UINavigationController in Storyboard to the UITableViewController you created.

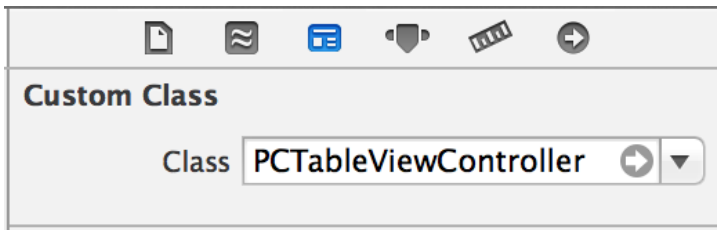


Figure : Change subclass to new UITableViewController

Add the following code to PCTableViewController.h:

```
@property NSArray * items;
```

Add the following code to PCTableViewController.m:

```
-(void) viewDidLoad {
  [super viewDidLoad];
  self.items = @[@"One", @"Two", @"Three"];
}
```

Change the method `numberOfSectionsInTableView:(UITableView *) tableView` in PCTableViewController.m to return 1.

Change the method `tableView:(UITableView *) tableView cellForRowAtIndexPath:(NSIndexPath *) path` to return `self.items.count`.

Change the method `tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath` to:

```
1   UITableViewCell *cell = [tableView
2      dequeueReusableCellWithIdentifier:@"DefaultCell"
3                           forIndexPath:indexPath];
4
5   cell.textLabel.text = [self.items objectAtIndex:indexPath.row];
6
7   return cell;
```

Double click on the table view controller in Storyboard and select the prototype UITableViewCell. Change its reuse identifier to “DefaultCell”.

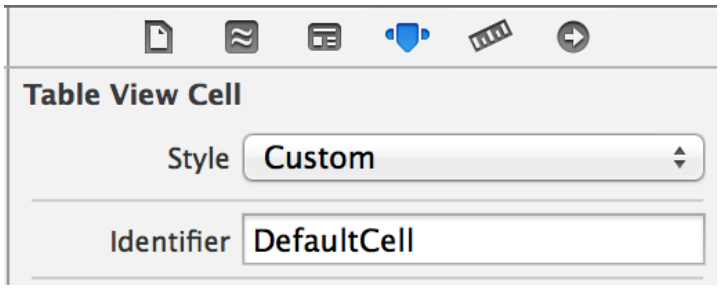


Figure : Change Reuse Identifier

In the `PCViewController.h`, add the following code.

```
1     @property NSString * name;
2     @property IBOutlet UILabel * nameLabel;
```

In Storyboard, drag a UILabel to PCViewController and connect it to the nameLabel outlet.

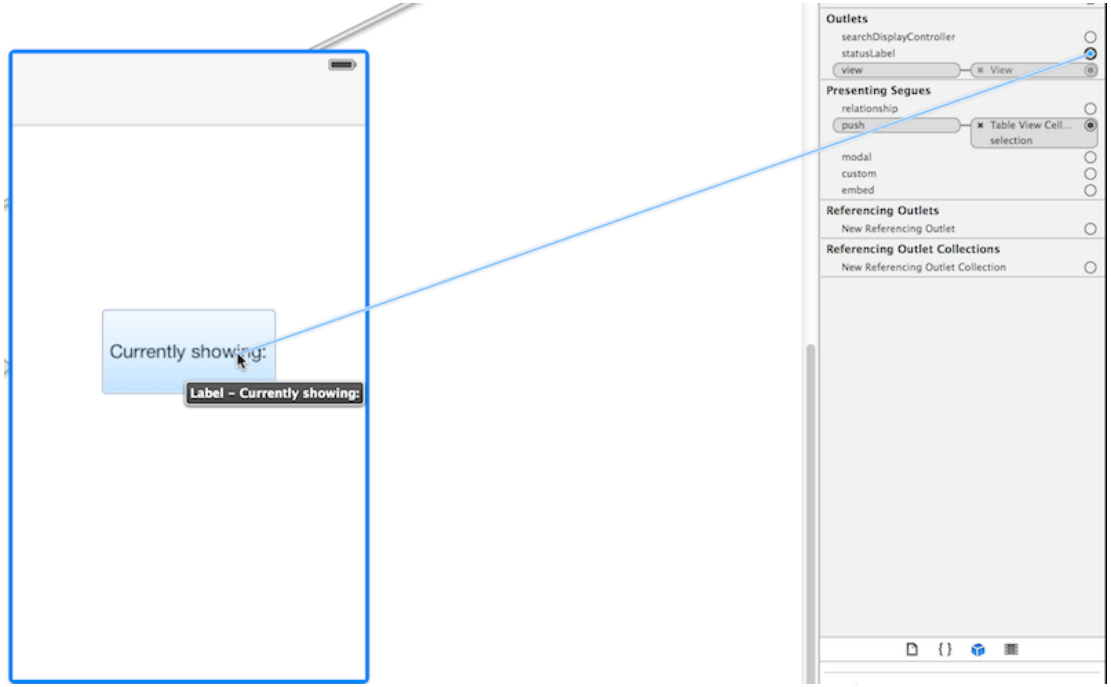


Figure : Connect the UILabel to the nameLabel outlet

In PCViewController.m, add the following code:

```
1    -(void) viewDidLoad {
2      [super viewDidLoad];
3      self.nameLabel.text = [NSString stringWithFormat:@"Name: %@", self.name];
4    }
```

In Storyboard, select the prototype cell in the table view controller and connect its “selection” triggered segue to the PCViewController and choose “Push”.

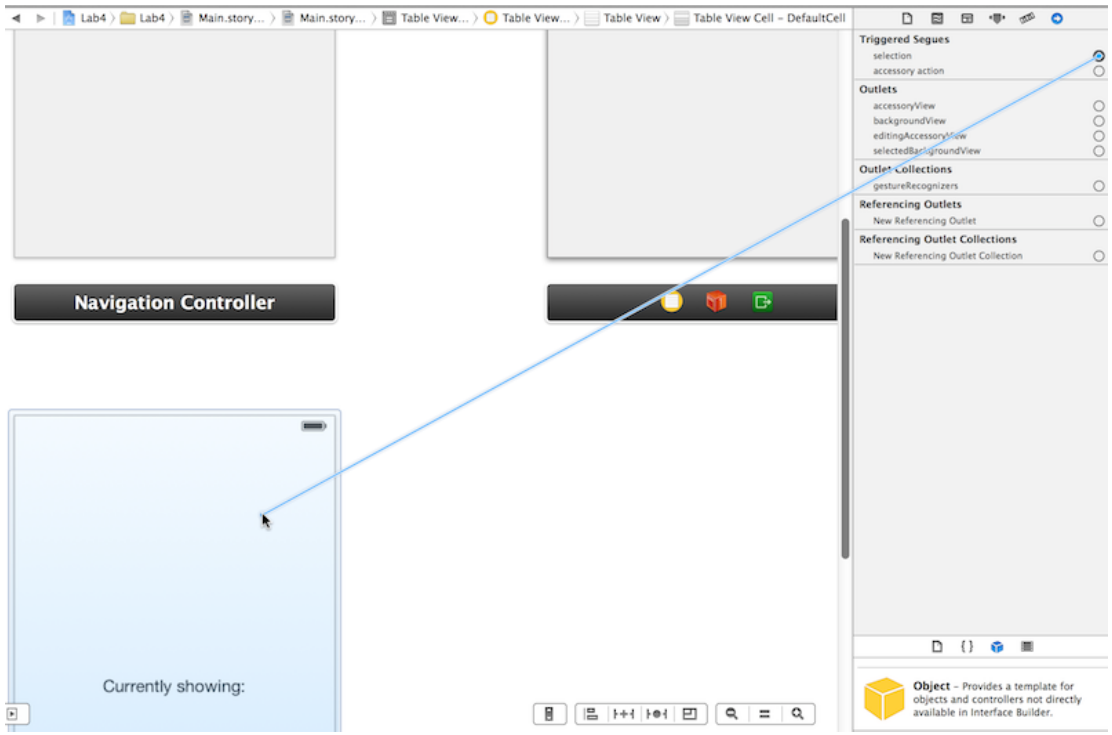


Figure : Connect the selection segue of the UITableViewCell

In PCTableViewController.m, uncomment the method `prepareForSegue:sender:` and add the following code:

```
1   PCViewController * c = (PCViewController *)[segue destinationViewController];
2   c.name = [self.items objectAtIndex:self.tableView.indexPathForSelectedRow.row
```

Add the following line to the top of the file:

```
1   #import "PCViewController.h"
```

Run it!

Figure : Time to get pumped

One more thing: Add a JSON file to your project that is an array of items. Deserialize it in your UITableViewController, and use the information from the JSON file to populate your UITableView and UIViewController.

An Aside: The Delegate Pattern

You may have noticed that UITableViewController had odd methods like `- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section`. Why does the UITableViewController need to have a UITableView passed to its numberOfRowsInsection method?

The delegate pattern allows for a weak binding between two objects, such that a “delegate” can be queried for configuration information and special behavior from an acting object.

Objective C formal protocols are typically how a delegate can indicate at compile time that it provides certain functionality. See UITableViewDelegate and UITableViewDataSource.

The delegate pattern is used widely throughout Apple frameworks.

```
netshade@shade [11:50:31] [/Applications/Xcode.app/Contents/Developer/Documentation
-> % grep -ri 'delegate' ./* | wc -l
    2446
```

Figure : Delegate usage throughout the documentation

Review Concepts: UIViewController Basics

Build an iOS app that:

- Has a UINavigationController
- Loads a UITableViewController as its first scene
- Populates the UITableViewController rows names taken from a JSON file in the local file system
- Populates the UITableViewController rows with images specified from the JSON file
- When a table row is selected, loads a detail view controller that presents the name and image that the user selected

NSURL, NSURLRequest, NSMutableURLRequest AND NSURLConnection

NSURL is meant to only represent a single resource location

```
1    NSURL * theUrl = [NSURL URLWithString:@"http://bootstrapping-ios.com/"];
```

NSURL can be allocated to represent either a filesystem location, or a web resource

```
1    NSURL * aWebUrl = [NSURL URLWithString:@"http://news.ycombinator.org/"];
2    NSURL * aFileUrl = [NSURL fileURLWithPath:[[NSBundle mainBundle] pathForResou
```

NSURLRequest and NSMutableURLRequest represent specific web resources that you'd like to initiate a connection to; NSURLRequest should be used for simple GET HTTP requests,while NSMutableURLRequests can be used for more complex HTTP requests (POST, PUT, file uploads).

NSMutableURLRequest allows you to set HTTP headers, the HTTP method used, and the request body.

```
1     NSURLRequest * request = [NSURLRequest
2       requestWithURL:[NSURL URLWithString:@"http://bootstrapping-ios.com/"]];
3
4     NSMutableURLRequest *  customRequest = [NSMutableURLRequest
5       requestWithURL:[NSURL URLWithString:@"http://bootstrapping-ios.com/"]];
6     [customRequest setHTTPMethod:@"POST"];
7     [customRequest setValue:@"Some-Token"
8          forHTTPHeaderField:@"X-Request-Token"];
9     [customRequest setValue:@"application/x-www-form-encoded"
10         forHTTPHeaderField:@"Content-Type"];
11    [customRequest setHTTPBody:[@"some-form=values"
12            dataUsingEncoding:NSUTF8StringEncoding]];
```

NSURLConnection can send data both synchronously or asynchronously. Asynchronous response information is passed back to the connection delegate via the NSURLConnectionDelegate, NSURLConnectionDataDelegate and NSURLConnectionDownloadDelegate protocols.

NSURLConnection initiates the download and returns the NSHTTPURLResponse which contains the body of the response, as well as http status code and response headers.

```
1   NSURLRequest * request = [NSURLRequest
2     requestWithURL:[NSURL URLWithString:@"http://bootstrapping-ios.com/"]];
3
4   NSError * error;
5   NSHTTPURLResponse * response;
6   NSData * syncResponse = [NSURLConnection sendSynchronousRequest:request
7     returningResponse:&response
8     error:&error];
9
10  NSURLConnection * asyncConnection = [[NSURLConnection alloc]
11    initWithRequest:request
12    delegate:self
13    startImmediately:YES];
```

NSURLSession adds conveniences for downloading content in the background.

Lab 5

Create a new “Single View” project

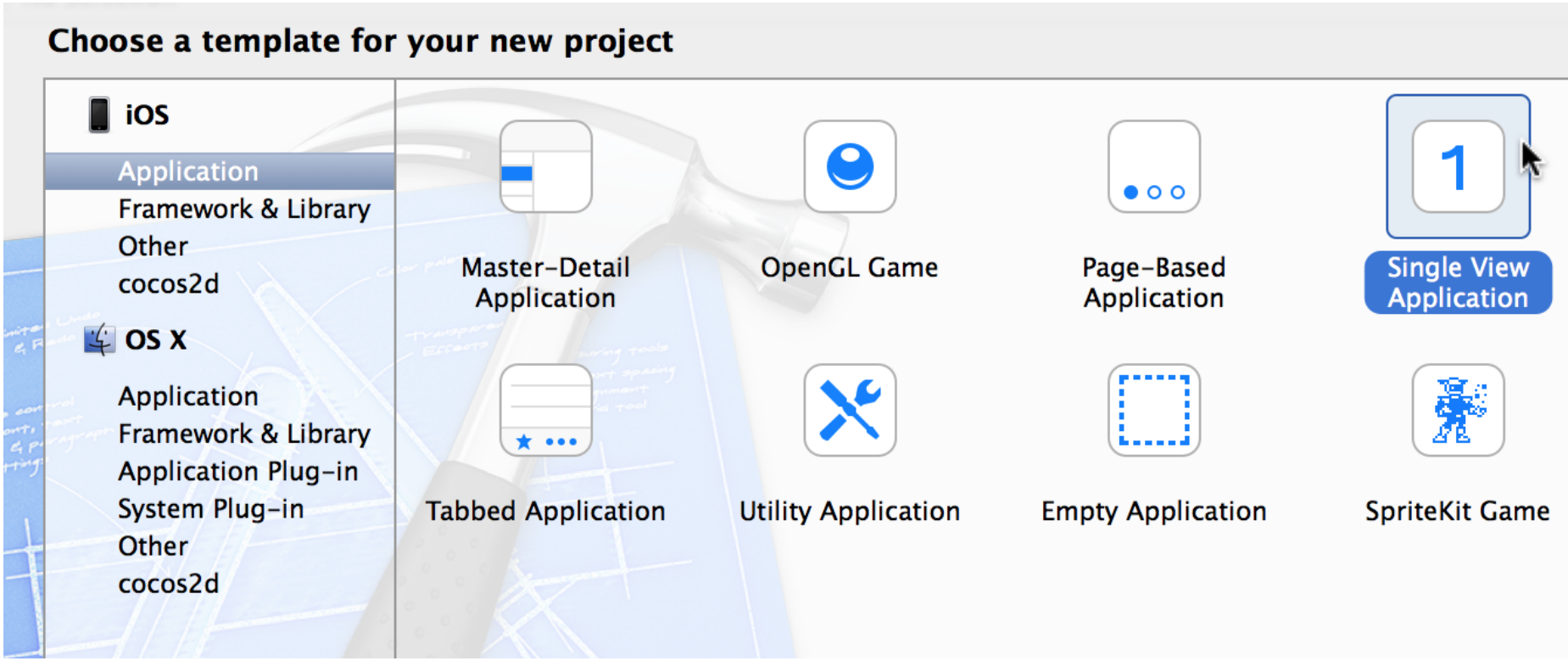


Figure : Create a new project

Open the project's storyboard and add a new "UINavigationController" to the storyboard.

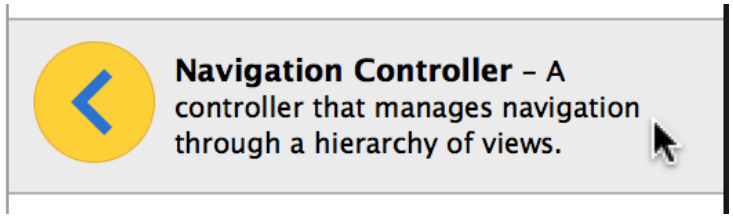


Figure : Select UINavigationController in the Object library

Drag the storyboard's "initial scene" arrow from the default view controller scene to the UINavigationController you placed.

Figure : Drag the initial scene arrow to the UINavigationController

Add a new subclass of NSObject to your project that is a subclass of “UITableViewController”.

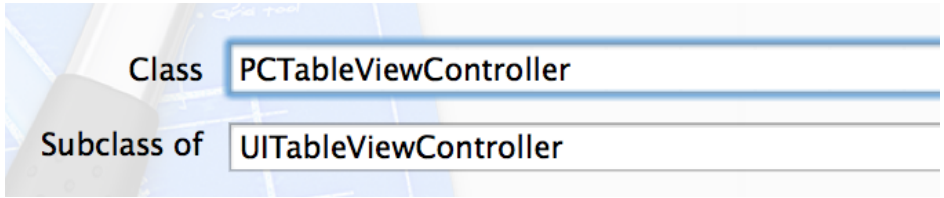


Figure : Add a new subclass of UITableViewController

Change the subclass of the UITableViewController connected to your UINavigationController in Storyboard to the UITableViewController you created.

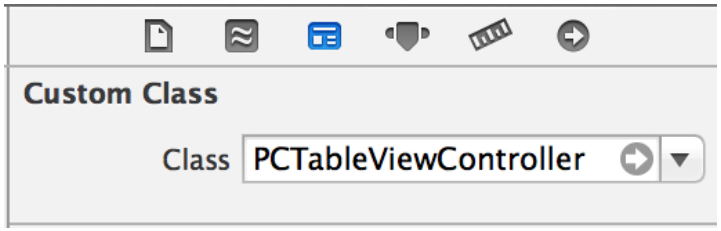


Figure : Change subclass to new UITableViewController

In the `viewDidLoad` method of your `UITableViewController`, initialize a request to the URL:

```
http://bootstrapping-ios.com/echo?names=one&\
                                  names=two&\
                                  names=three
```

Using NSURLConnection, asynchronously request the above JSON, deserialize it into an NSArray, and then reload your tableview.

You will need to have your view controller adopt specific methods from the NSURLConnectionDelegate and NSURLConnectionDataDelegate protocols. You will also need to store the request data in something like NSMutableData.

Run the app!

Figure : Basically an expert

One more thing: Change your request to a POST request with `application/x-www-form-urlencoded` content type, and instead of sending a querystring in the URL, send the key value pairs in the HTTP body.

And another thing: Select your TableViewController and enable "Refreshing" in its attributes. Connect the UIRefreshControl's Value Changed outlet to an IBAction method on your TableViewController that triggers a new download. Tell the refresh control to change its appearance through `[self.refreshControl beginRefreshing]` and `[self.refreshControl endRefreshing]`.

NSTimer, NSRunLoop, NSThread and the Event Loop

Cocoa represents application threads using the `NSThread` object. Each `NSThread` has an `NSRunLoop` associated with it, which is responsible for managing IO and timer events.

The main application thread (the UI thread) is one of a set of default threads created for you. Any operations that occur on this main thread can cause the user interface to become non-responsive; it is important that you defer long running processes to outside this thread.

To perform an action at some later point, or on an interval, you can register an NSTimer with the current NSRunLoop. An NSTimer will invoke a selector you specify on a schedule you control.

```
1   [NSTimer scheduledTimerWithTimeInterval:10.0
2       target:self
3       selector:@selector(runEveryTenSeconds:)
4       userInfo:nil
5       repeats:YES];
6
7   -(void) runEveryTenSeconds:(NSTimer *) {
8     NSLog(@"Timer fired");
9   }
```

These timer actions occur in the thread they were scheduled from, so be cautious about scheduling expensive operations.

Lab 6

Create a new “Single View” project

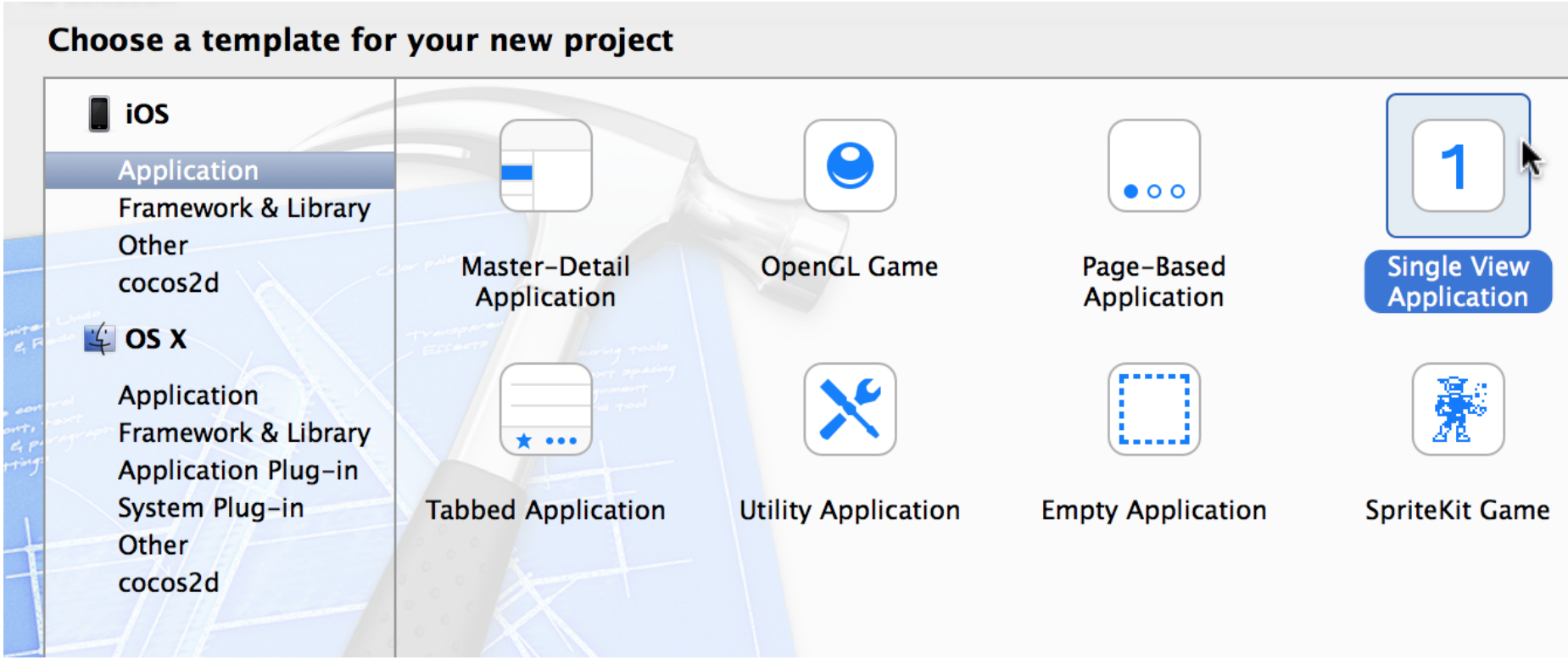


Figure : Create a new project

Open the project's storyboard and add a new "UINavigationController" to the storyboard.

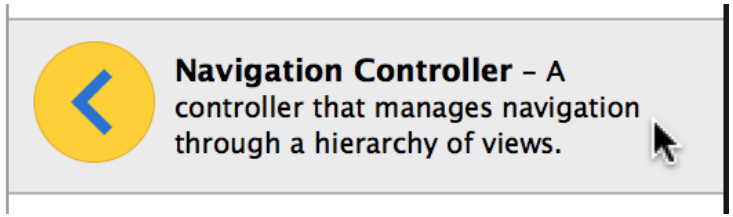


Figure : Select UINavigationController in the Object library

Drag the storyboard's "initial scene" arrow from the default view controller scene to the UINavigationController you placed.

Figure : Drag the initial scene arrow to the UINavigationController

Add a new subclass of NSObject to your project that is a subclass of “UITableViewController”.

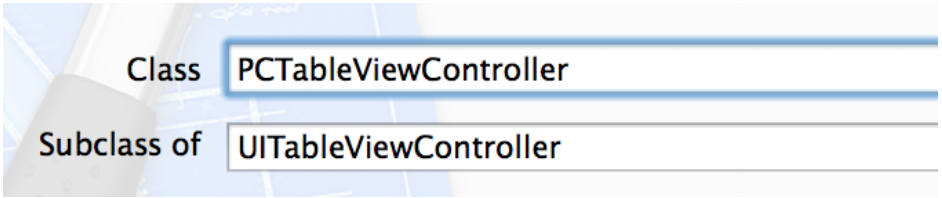


Figure : Add a new subclass of UITableViewController

Change the subclass of the UITableViewController connected to your UINavigationController in Storyboard to the UITableViewController you created.

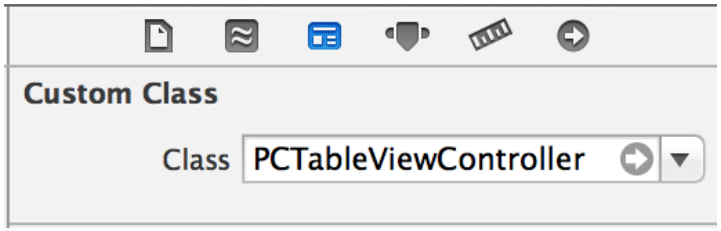


Figure : Change subclass to new UITableViewController

In the `viewDidLoad` method of your `UITableViewController`, initialize a request to the URL:

```
1    NSTimeInterval since1970 = [[NSDate date] timeIntervalSince1970];
2    NSURL * theUrl = [NSURL URLWithString:[NSString stringWithFormat:@"http://boo
```

Using NSURLConnection, asynchronously request the above JSON, deserialize it into an NSArray, and then reload your tableview.

Create an NSTimer and schedule refreshes of the data every 10 seconds.

```
1    timer = [NSTimer scheduledTimerWithTimeInterval:5.0 target:self selector:@sel
```

Add a method called dealloc that cancels the timer:

```
1    -(void) dealloc {
2      [timer invalidate];
3      timer = nil;
4    }
```

Run the app!

Figure : Yay!

One more thing: Add a call to sleep() (`man 3 sleep`) in your timer that intentionally delays response time of the refresh. Observe how UI suffers as the timer action takes longer to execute.

Installing on the iOS Device

So far we've been running everything on the simulator. In order to run code on our device, we need to do two things:

- Have a valid code signing certificate
- Have a valid provisioning profile

A code signing certificate allows us to prove we are the trusted origin of our code. A provisioning profile creates the association between the certificate and the devices that it's allowed to run on.

You will need a separate certificate and profile for:

- Development deployment
- AdHoc deployment
- AppStore deployment
- Enterprise deployment

XCode will generally create the files you need, but you may manually create (and debug) the requisite files through:

- The XCode Organizer Window > Organizer

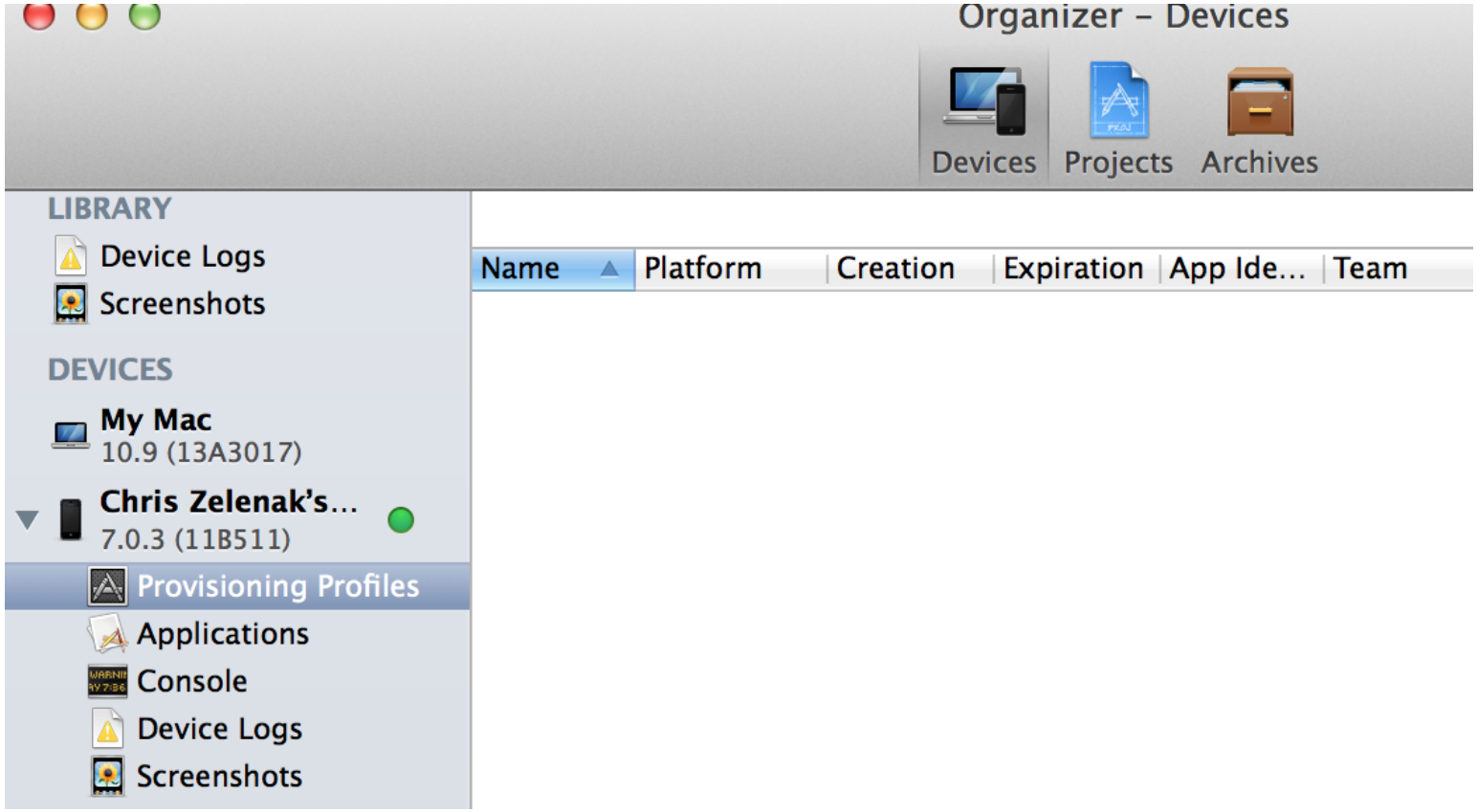


Figure : XCode Organizer

- iTunes Connect `http://itunesconnect.apple.com/`

Lab 7

Open up “Lab 6” in XCode.

Connect your iPhone to your machine.

Select the iOS device in the “Destination” dropdown.

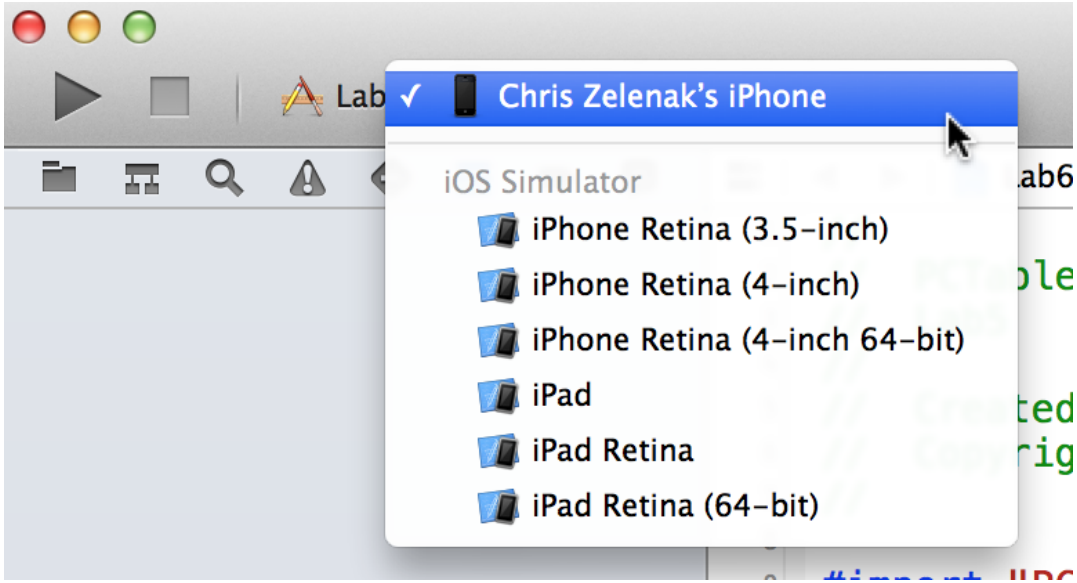


Figure : Select your iOS device

Run the app. If you've never actually ran a developer app on your device before, XCode should automatically create the files you need to sign code for the device.
Alternatively, you can manually set the device to be a developer device on your account by adding it to the member center in XCode Organizer.

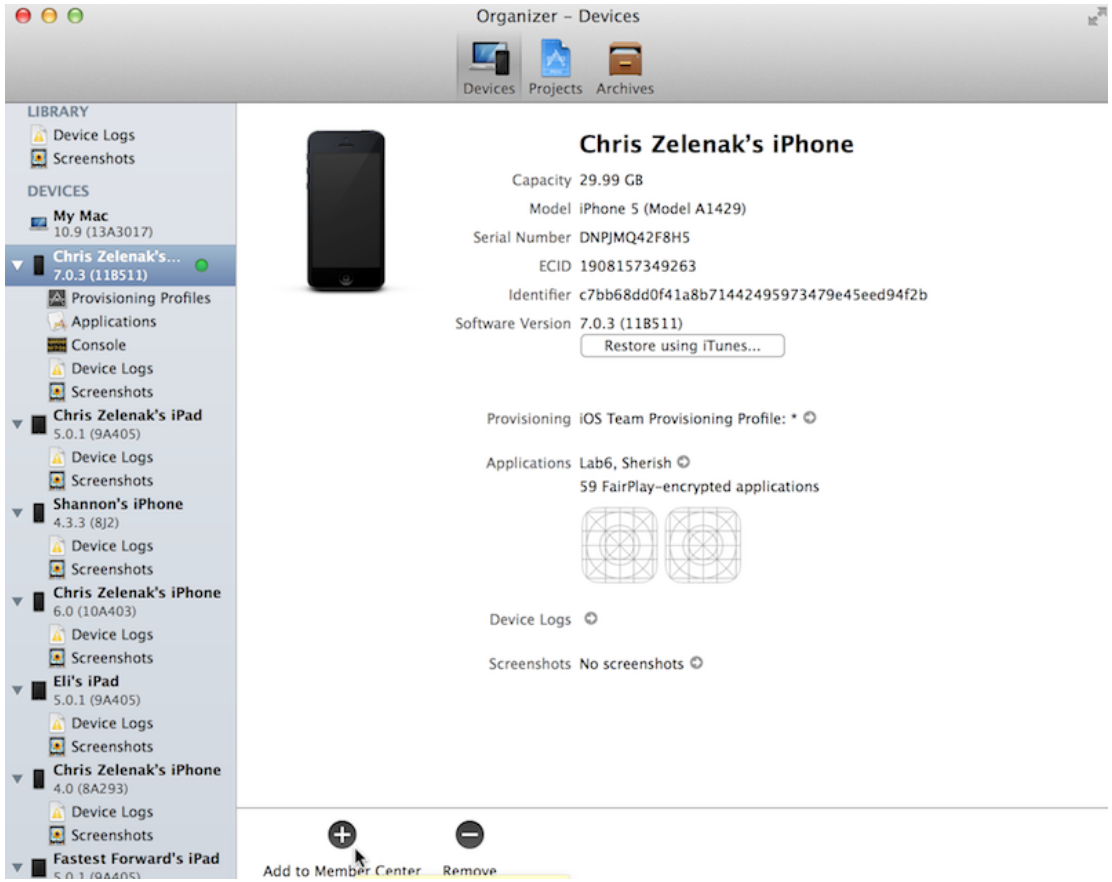


Figure : Adding to the Member Center manually

While the app is running on your device, you can:

- Access Console output in Organizer

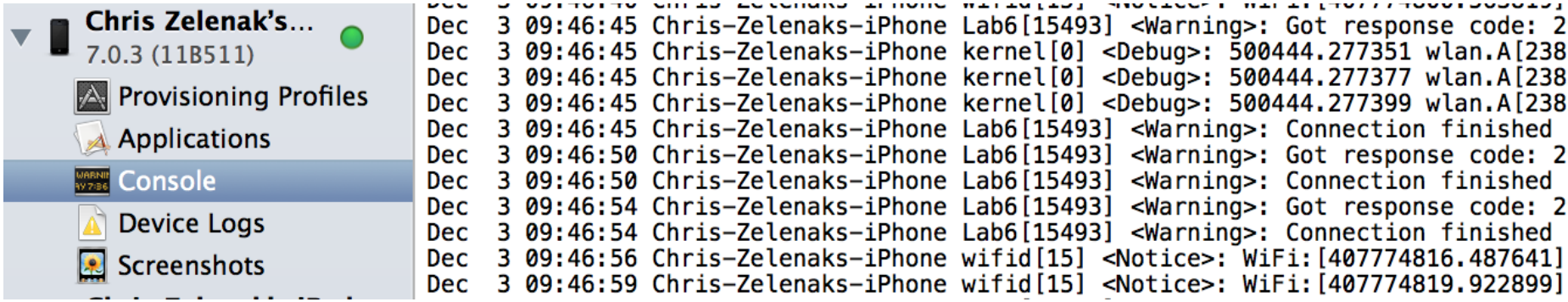


Figure : Watching console output

- Set breakpoints and inspect memory in XCode

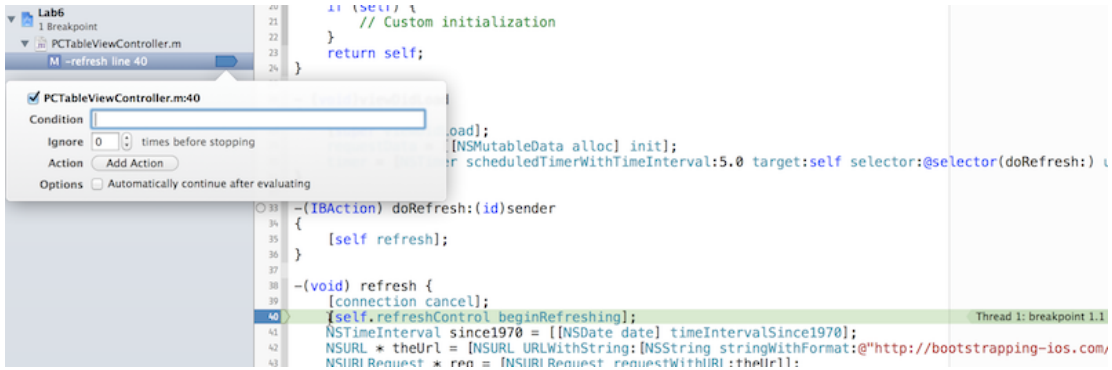


Figure : Inspecting breakpoints

- Interactively query memory at breakpoints with LLDB

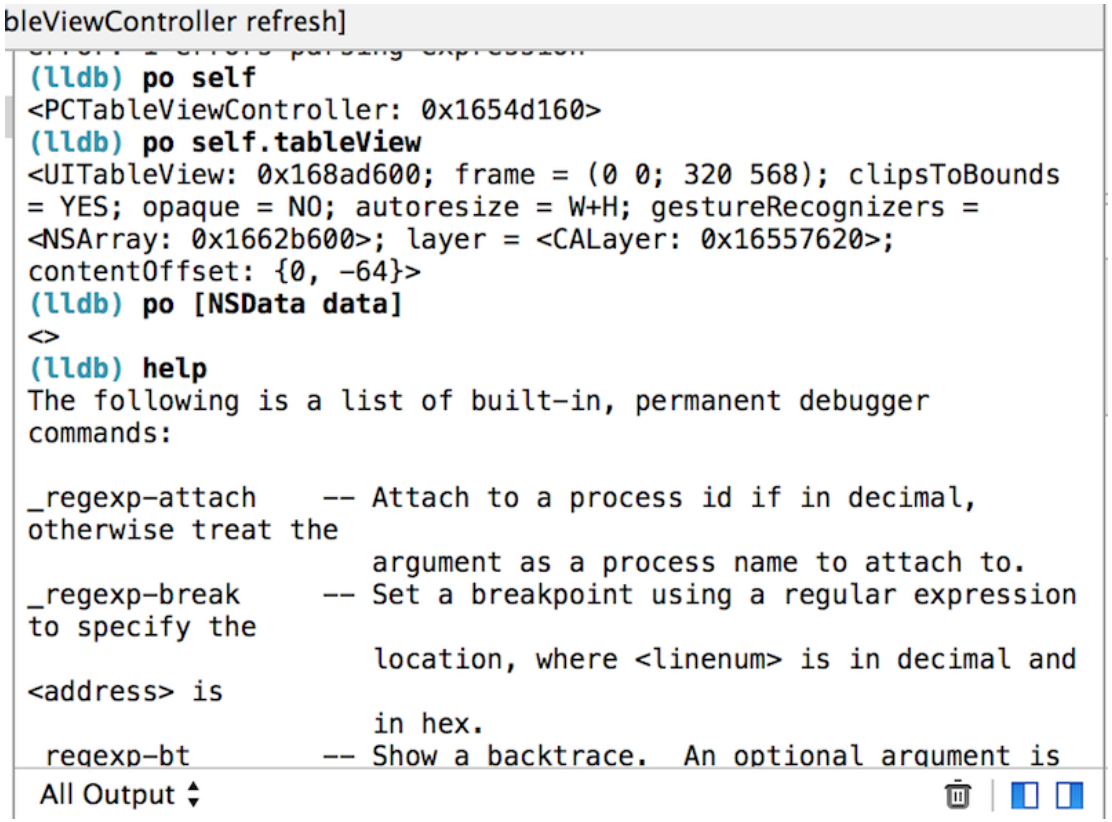
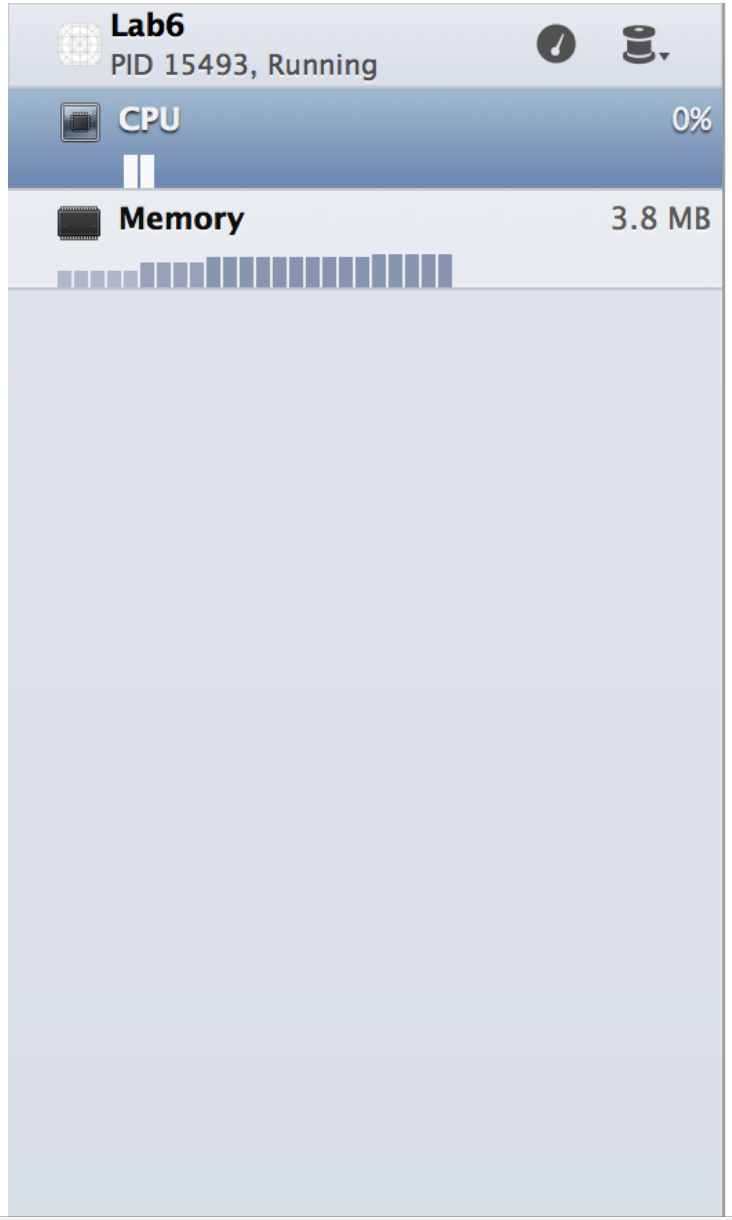
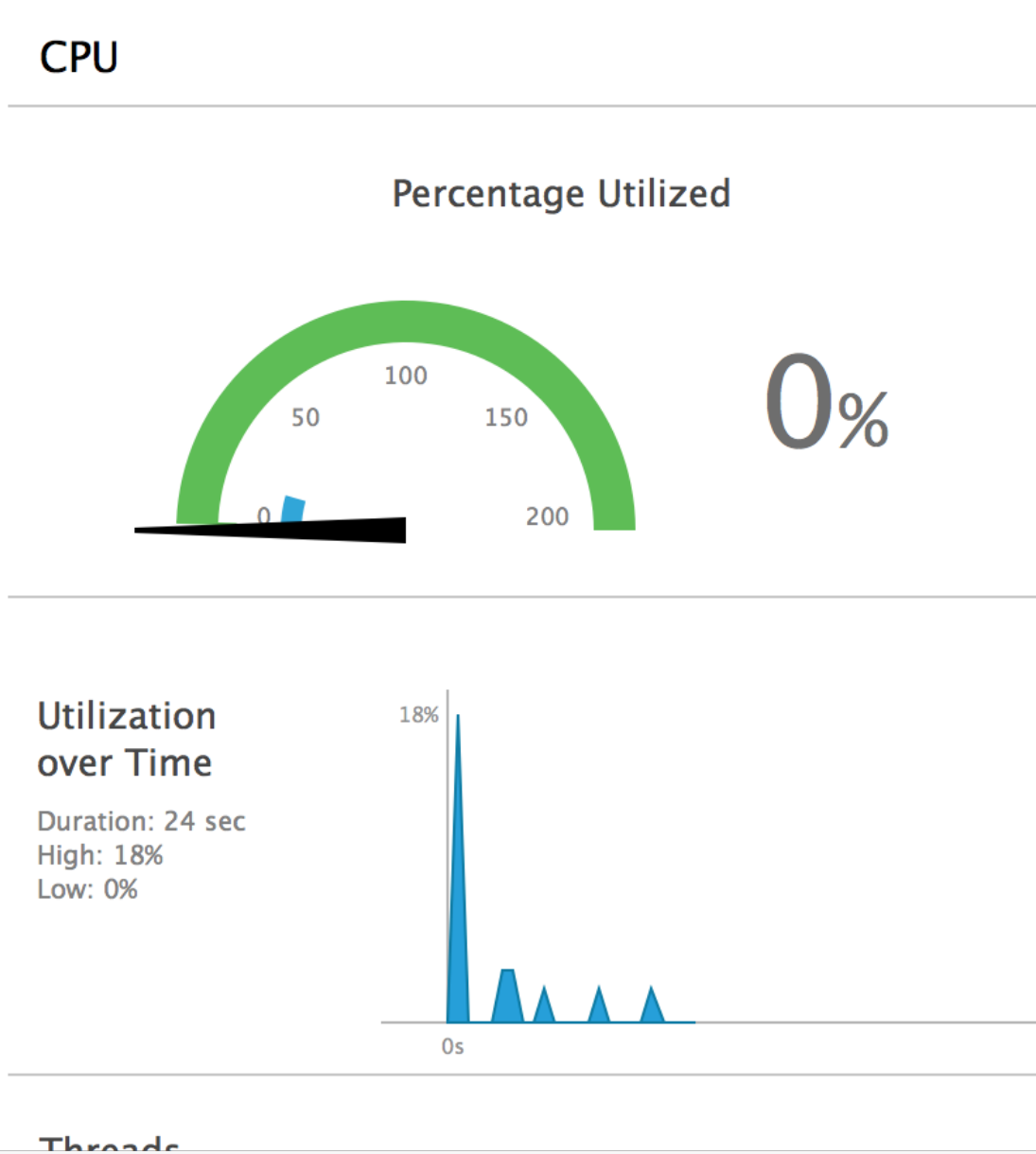


Figure : Querying objects and running code via LLDB

- Observe basic system metrics





Custom Views

Views are descendants of `UIView`, and have several properties that allow for some basic customization. Some commonly used attributes are:

- `.frame`, of type `CGRect`
- `.backgroundColor` of type `UIColor`
- `.alpha` of type `CGFloat`
- `.transform` of type `CGAffineTransform`
- `.hidden` of type `BOOL`

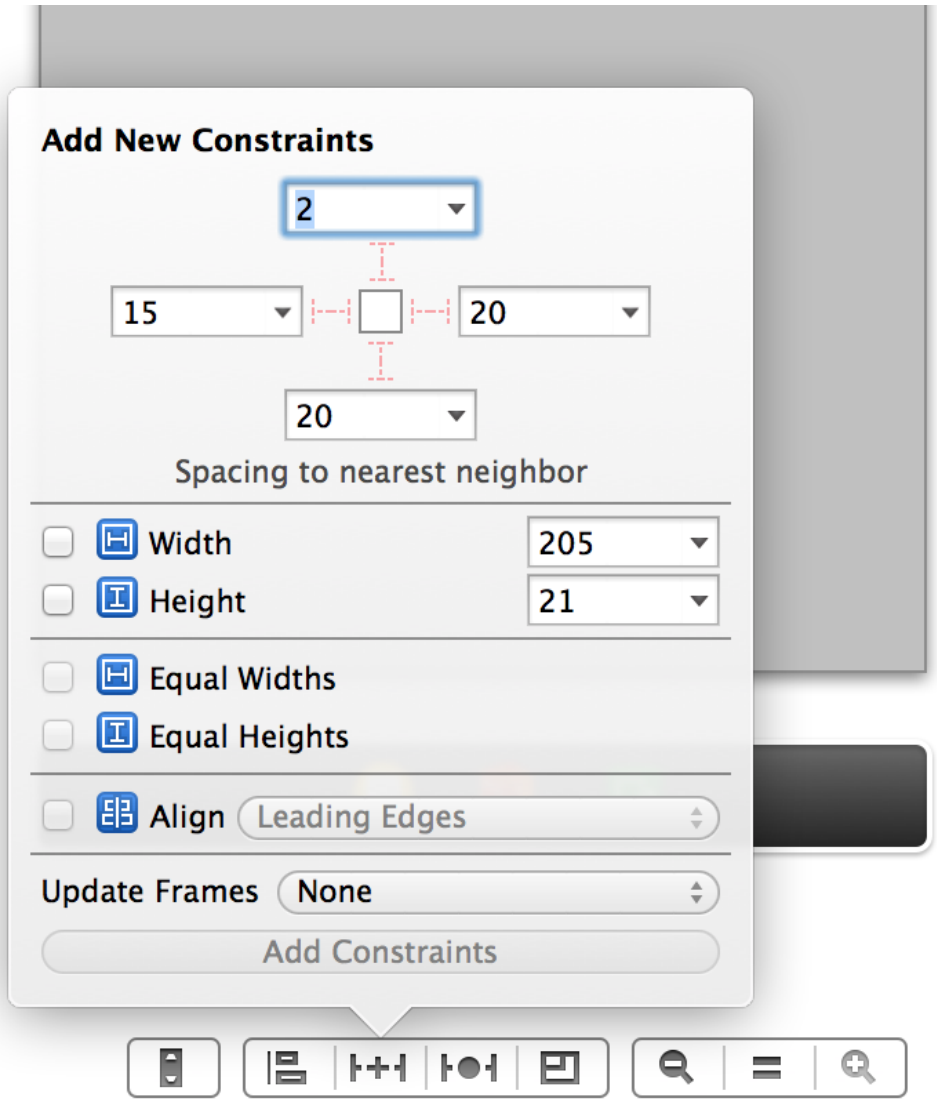
UIView can be subclassed and customized, just like any other view. A UIView may contain other UIViews to **compose** a higher order control. They can also draw directly to the screen if you need to manually draw certain types of graphics using an API called `Core Graphics`.

Any UIViewController has a `view` attribute which represents the top level view it is currently displaying. You can programmatically add other views to this using methods like `addSubview:`, or you can modify the `view` in Storyboard as we have been doing.

Views may have a `tag` associated with them, an integer identifier that allows specific views to be requested with the `viewWithTag:` method of UIView.

```
1  UIView * v = [[UIView alloc] initWithFrame:CGRectMake(0., 0., 20., 20.)];
2  v.tag = 100;
3
4  [self.view addSubview:v];
5
6  UIView * theView = [self.view viewWithTag:100];
```

UIViews may also specify certain constraints with respect to their sibling views. This is called `Auto Layout`, and allows for UIViews to change their appearance w/ respect to other views.



Additionally, UIViews can have application wide settings applied to them using `UIApperance` objects, which set defaults for object appearance:

```
1   [[UINavigationBar appearance] setBarTintColor:[UIColor blackColor]];
2   [[UITableViewCell appearanceWhenContainedIn:[UITableView class], nil]
3     setBackgroundColor:[UIColor purpleColor]];
```

Lab 8

Create a new “Single View” project

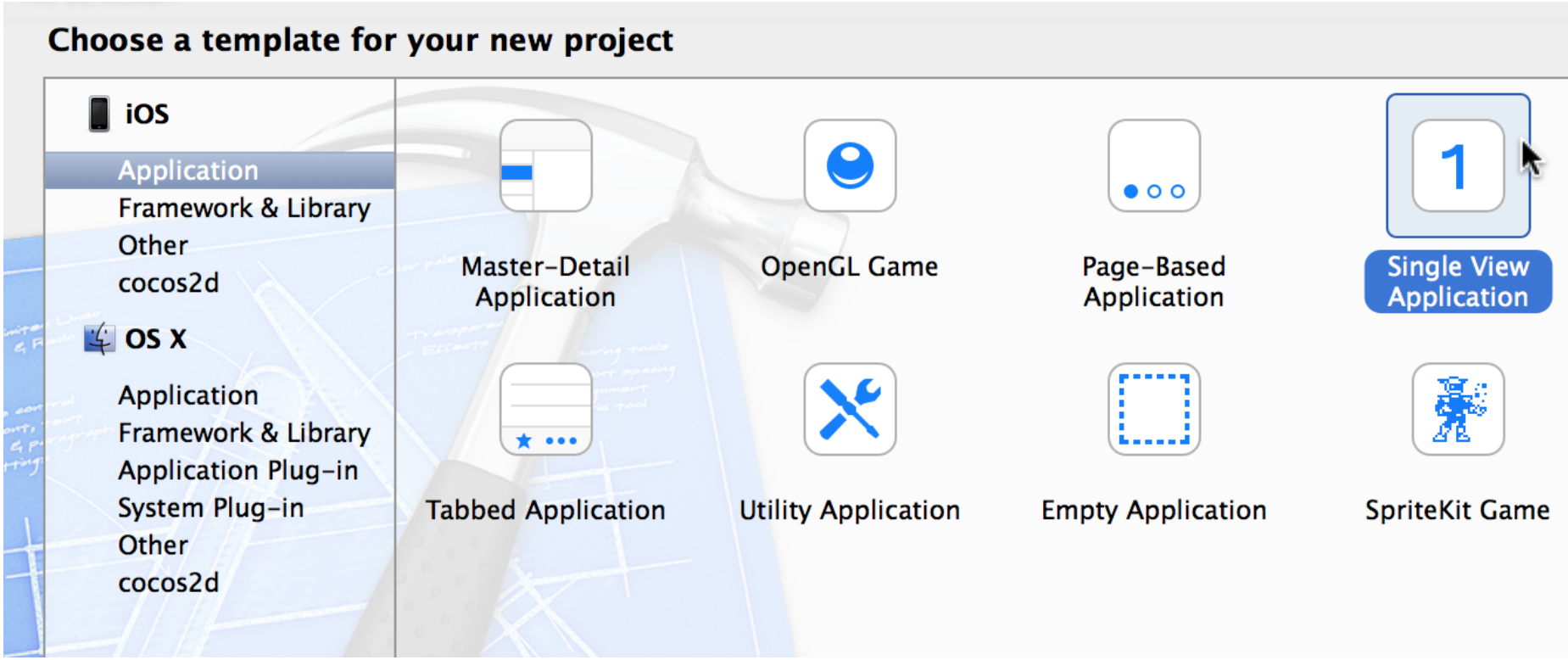


Figure : Create a new project

Open the project's storyboard and add a new "UINavigationController" to the storyboard.

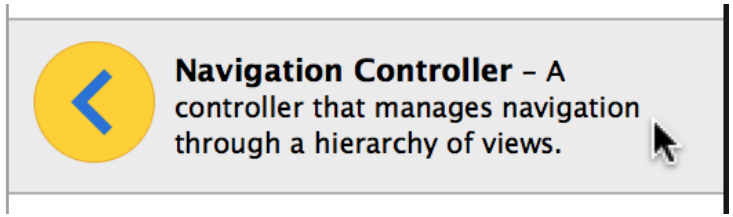


Figure : Select UINavigationController in the Object library

Drag the storyboard's "initial scene" arrow from the default view controller scene to the UINavigationController you placed.

Figure : Drag the initial scene arrow to the UINavigationController

Add a new subclass of NSObject to your project that is a subclass of “UITableViewController”.

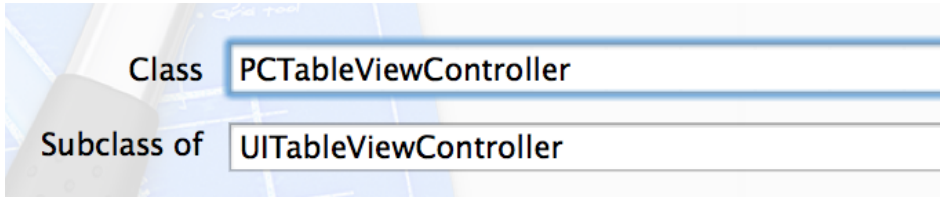


Figure : Add a new subclass of UITableViewController

Change the subclass of the UITableViewController connected to your UINavigationController in Storyboard to the UITableViewController you created.

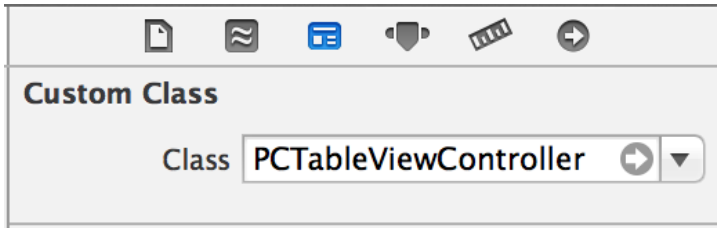


Figure : Change subclass to new UITableViewController

Populate your UITableView with data. :)

In Storyboard, add the following items to the prototype cell in your UITableView:

- A UIImageView (tag 100)
- A UILabel that is the “Title” label (tag 101)
- A UILabel that is the “Subtitle” label (tag 102)

Add an alignment constraint to the “Title” and “Subtitle” such that they keep a strict right margin with respect to their superview.

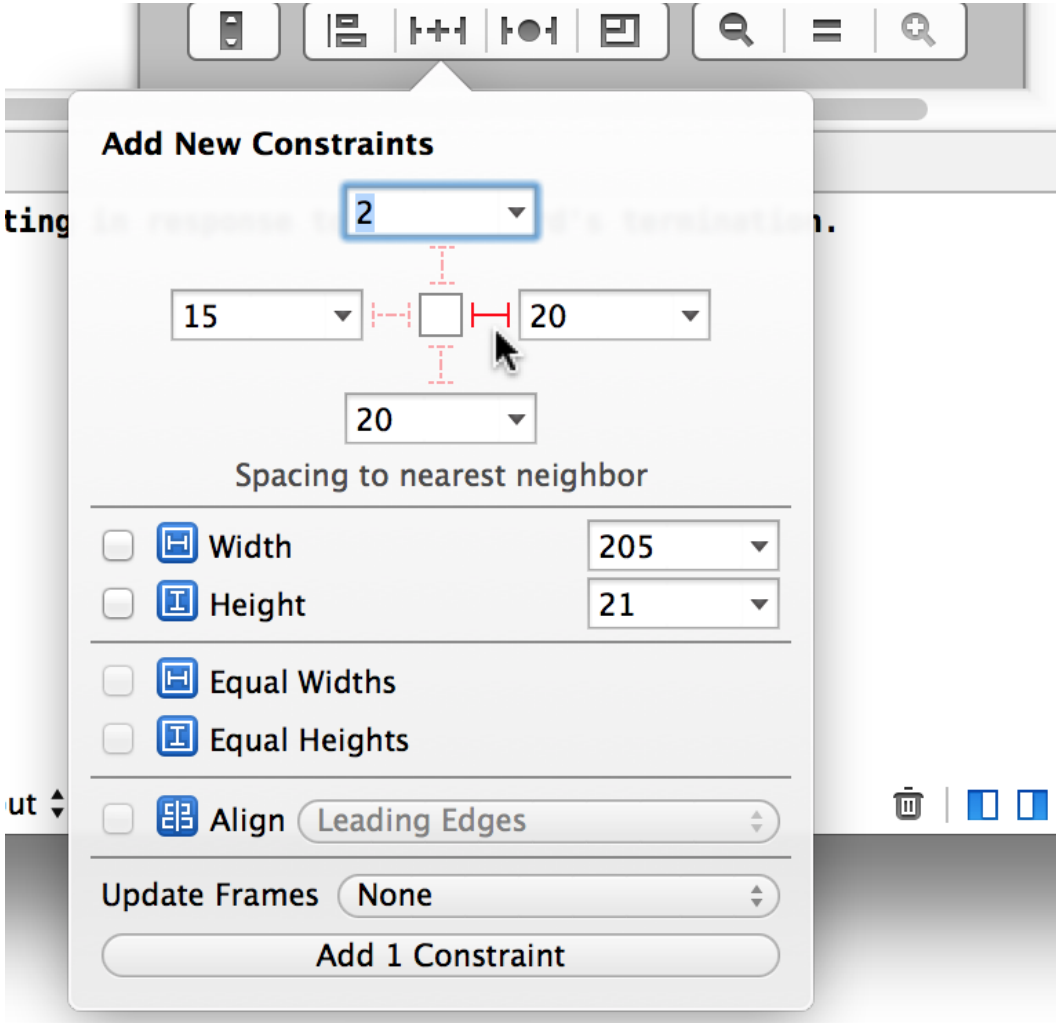


Figure : Set right margin

In the `tableView:cellForRowAtIndexPath:` method of your table view controller, retrieve the 3 views based on their tag, and set their values respectively.

```
1    UIImageView * imageView = [cell viewWithTag:100];
2    UILabel * title = [cell viewWithTag:101];
3    UILabel * subtitle = [cell viewWithTag:102];
4    title.text = @"Title"
5    subtitle.text = @"image"
6    imageView.image = anImage;
```

Run the app!

Figure : Put it on the App Store!

One more thing: Using the UIAppearance protocol, set the default font and text color of any UILabel object when your app starts up.