Praktikum Entwicklung von Mediensystemen mit iOS

SS 2012

Prof. Dr. Michael Rohs
michael.rohs@ifi.lmu.de
MHCI Lab, LMU München

Today

- Schedule
- Organization
- Video watching
- Introduction to iOS
- Exercise 1

Schedule

- Phase 1 – The Tomorrow Talks
  - Concept development
  - Video production
- Phase 2 – Individual Phase
  - Introduction to basics of iOS
  - Exercises (+1 advanced exercise for Master students)
  - Each student works on exercises himself/herself
  - Weekly meetings
- Phase 3 – Concept Phase
  - Concretize concept
- Phase 4 – Implementation
  - Implementation of iOS app
  - Regular milestone meetings
- (optional) Phase 5 – Deployment Phase
  - Deploy iOS app in App Store

Timeline

#	Date	Topic
	19.4.	Introduction & Brainstorming future mobile concepts
1	3.5.	Video watching, Introduction to iOS
	10.5.	no class (CHI Konferenz)
	17.5.	no class (Christi Himmelfahrt)
2	24.5.	More on iOS
3	31.5.	Concept finalization, paper prototyping
	7.6.	no class (Frohnleichnam)
	14.6.	Paper prototyping test, start of software prototype
5	21.6.	
6	28.6.	Think aloud study of software prototype
7	5.7.	
8	12.7.	Completion of software prototype
9	19.7.	Final presentation

Organization

- 6 ECTS-Credits
- 4 SWS
- Weekly meetings
  - Thursday 14:15 – 16:00
  - Room 107, Amalienstraße 17
- Homepage:
  - http://www.medien.ifi.lmu.de/pem
- Submit exercises via UniWorX
- Set up your own version control system
  - Git integrated into Xcode

iOS Developer Account

- University Account
- Send email, we invite you
- Create certificate
- Register as developer
- We send provisioning profile

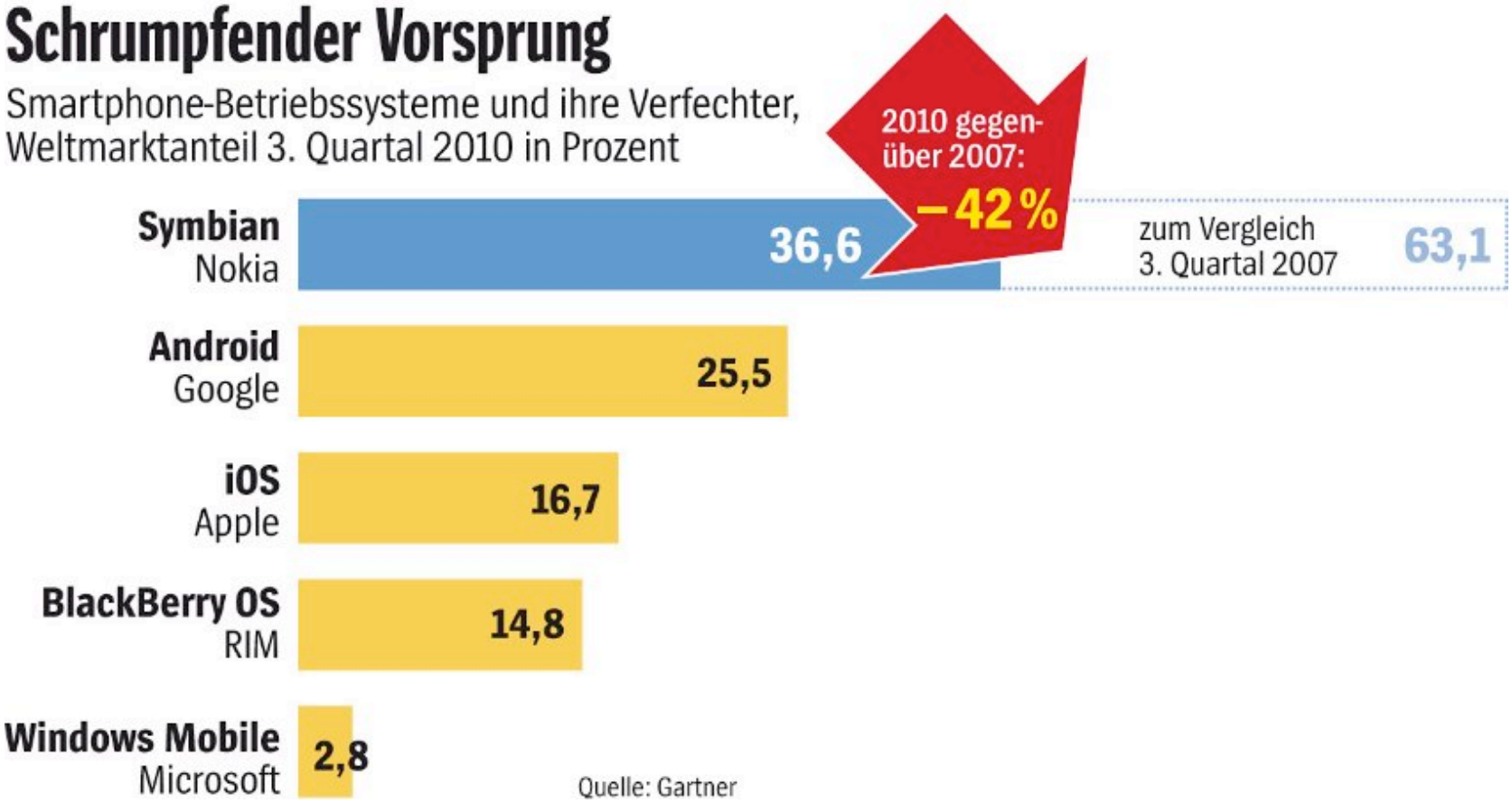
Video Watching

- What? Core functionality
- Who? Target group
- Where? When? With whom?
  Scenario, situation / context of use
- Why? Why should the intended person(s) be motivated to use the app in the intended context?

- Ideas? Additional ideas, suggestions, improvements
- Concerns? Obstacles, problems, show stoppers
- Novelty? Is the concept new? Existing apps?

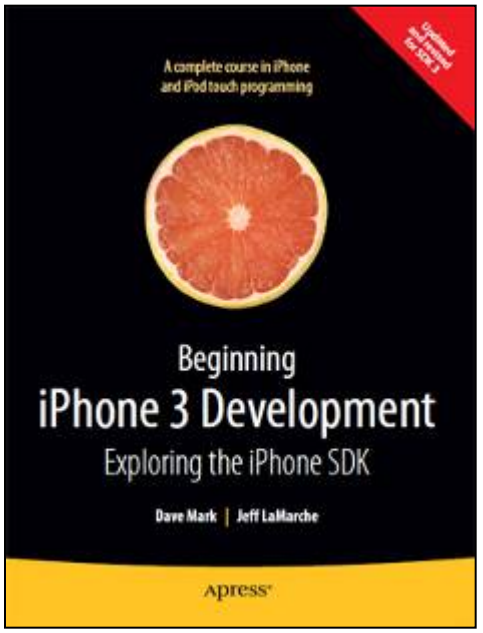
APPLE IOS OVERVIEW

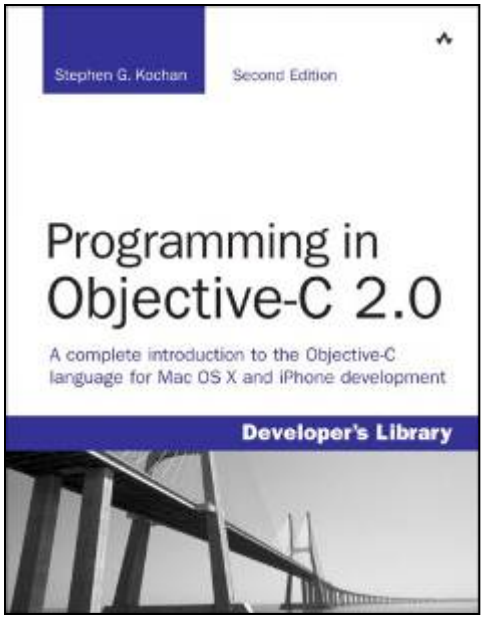
Smartphone Operating Systems



Books

- iPhone development
  - Dave Mark, Jeff LaMarche: Beginning iPhone 3 Development: Exploring the iPhone SDK. Apress, 2009.
  - http://www.amazon.com/Beginning-iPhone-Development-Exploring-SDK/dp/1430224592/
- Objective C
  - Stephen G. Kochan: Programming in Objective-C 2.0. Addison-Wesley, 2nd edition, 2009.
  - http://www.amazon.com/Programming-Objective-C-2-0-Stephen-Kochan/dp/0321566157/





User Interface Guidelines

- Concrete guidelines for look-and-feel and behavior
  - Visual appearance, e.g., icon design
  - Purpose of user interface elements
  - Layout of user interface elements
  - Behavior, conventions of system features
- iOS Human Interface Guidelines
  - http://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/MobileHIG.pdf
  - Aesthetic integrity, consistency, direct manipulation, feedback, metaphors, user control, …

Apple iOS

- Optimized version of Mac OS X
  - New components for handling touch
  - Memory optimized
- Hardware
  - 620 MHz ARM 1176 – 1GHz Apple A5
  - 128-512 MB DRAM
  - 4/8/16/32 GB flash RAM
  - Graphics: PowerVR OpenGL ES chip
  - Camera: 2.0-8.0 megapixels
  - Screen: 320x480 pixels, 163 ppi – 640x960 pixels, 326 ppi
  - Connectivity: GSM/UMTS, Wi-Fi (802.11b/g/n), Bluetooth
- SDK available since spring 2008

SDK Options

- Official iPhone SDK
  - Requires Mac to develop (IDE/compiler/debugger only for Mac)
  - Requires registration as developer ($99 per year)
  - Official support
  - Possibility to release on Apple App Store
  - http://developer.apple.com/devcenter/ios/

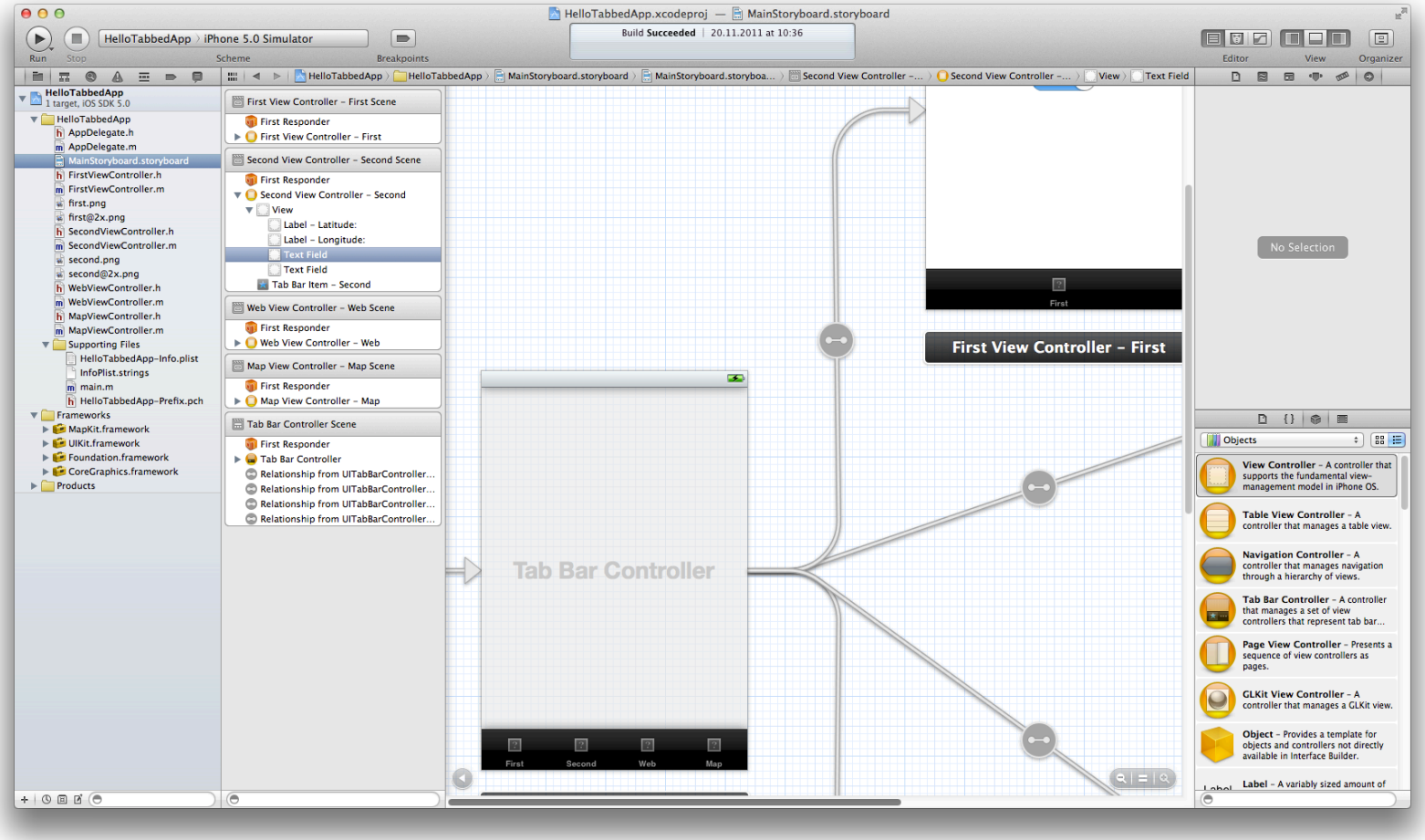
- iPhone toolchain SDK
  - Unofficial SDK
  - Available for Mac, Linux, PC (with varying comfort)
  - Command line gcc compiler (on-device compiling also possible)
  - All features of the phone actually accessible (even closed ones)
  - Requires “jailbreaking” the phone
  - May be legally questionable
  - http://code.google.com/p/iphone-dev/

Development Environment

- Xcode: IDE + integrated compiler, run-time debugger

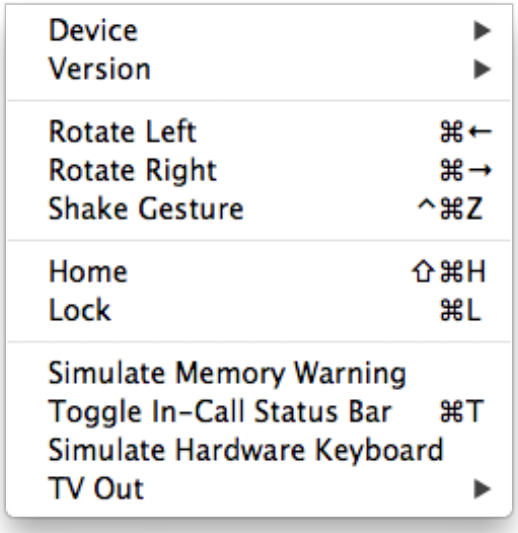
Development Environment

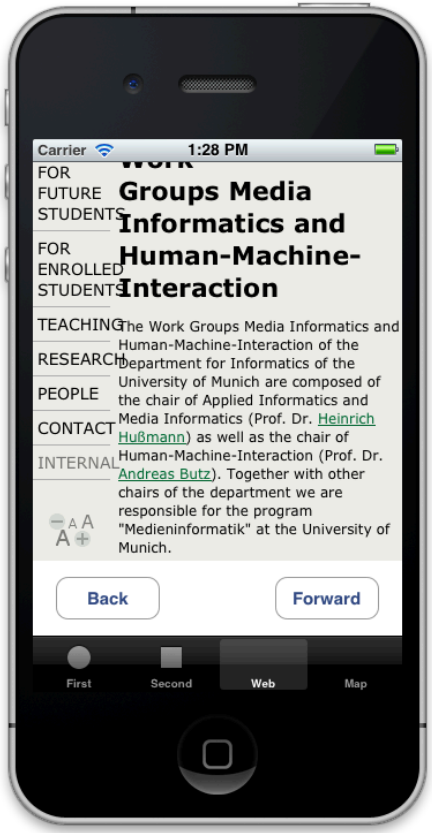
- Interface Builder: Graphical interface layouter



Development Environment

- iPhone Simulator: Mac simulator of iPhone
  - Most features except tilt, simulated multitouch





IOS TECHNICAL BACKGROUND

Philosophy of the API

- Compatibility with Mac OS X
  - Foundation frameworks: shared, Cocoa Touch: iPhone-only
- Maintains general framework structure
- Benefit
  - Shared code development between iPhone and OS X
  - Rapid porting of applications
  - Developer familiarity (for previous Mac developers)
- Preferred language
  - Objective C (implementation language of the SDK)
  - C/C++ work
- Protective
  - Some APIs are privileged and cannot be accessed
  - Example: AudioCore, LayerKit (direct access to framebuffers)

Cocoa Touch Architecture

- Cocoa Touch
  - High level architecture for building iOS applications
- Cocoa Touch consists of:
- UIKit
  - User interface elements
  - Application runtime
  - Event handling
  - Hardware APIs
- Foundation
  - Utility classes
  - Collection classes
  - Object wrappers for system services
  - Subset of Foundation in Cocoa

Objective C

- Objective C is superset of C, with OO constructs
  - Unusual Syntax, rarely used outside Apple realm, inspired by SmallTalk
- General syntax for method calls (“messages”):

  object.method(parameter1, parameter2); becomes:

  [object method:parameter1 parameterkey:parameter2];
- Example

  employee.setSalary(100,20); // arguments base_salary, bonus

  [employee setSalary:100 withBonus:20];
- Learn more at

  developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC

Objective C - Methods

- Method declaration syntax

  ± (type) selector:(type)param paramkey:(type)param2;

  Instance methods: - (void) myInstanceMethod;

  Class methods: + (void) myClassMethod;

- Example

  - (void) setSalary:(int)income withBonus:(int)bonus;

- Basic classes, examples
  - NSObject is root class (basics of memory management)
  - NSString
    - Example: s = [NSString stringWithFormat: @"The answer is: %@", myObject];
    - Constant strings are @"this is a constant string"
  - NSLog(NSString); (NSLog is your friend…)
  - NS… also offers collections (NSArray, NSDictionary etc) and other basic language service functionality
    - Prefix "NS" is derived from OS X predecessor, NextStep

Objective C – Features and Pitfalls

- Dynamically typed objects (or hard to find bugs)
  - id someObject
  - id is generic “pointer” without type (“void*”)
  - introspection allows finding out type at runtime
- Nil object pointers (or how to make really hard to find bugs)

  ```
  object = nil;
  [object setProperty: nil];
  ```

  - Will send message to nil, hard to find if objects didn’t get proper assignment
- id, nil and dynamic typing enable message-passing paradigm

Memory Management By Hand

- Don’t create memory leaks! ← now: ARC (later)
- Object reference life cycle:

```
myobject = [[MyClass alloc] init];          // reference count = 1 after alloc
[myobject retain];       // increment reference count (retainCount == 2)
[myobject release];      // decrement reference count (retainCount == 1)
[myobject release];      // decrement reference count (retainCount == 0)
// at this point myobject is no longer valid, memory has been reclaimed
[myobject someMethod]; // error: this will crash!
```

- Can inspect current reference count:

```
NSLog(@"retainCount = %d", [textField retainCount]);
```

- Can autorelease (system releases at some point in future)

```
[myobject autorelease];
```

Used when returning objects from methods.

Memory Management By Hand

- Memory rule: You are responsible for objects you allocate or copy (i.e. “allocate” or “copy” is some part of the name)!
- Not responsible:

```
NSData *data = [NSData dataWithContentsOfFile:@"file.dat"];
```

- Responsible:

```
NSData *data = [[NSData alloc] initWithContentsOfFile:@"file.dat"];
```

- Responsible:

```
NSData *data2 = [data copy];
```

- Never release objects you are not responsible for!

Objective C – Practical Aspects

- Based file extension .m
- Header file extension .h (expects Objective-C base file)
- Base file extension for Objective C++ is .mm (not .cpp)
- #import <...> (automatic redundancy check)

Objective C - Class

In .h file:

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

@interface Employee : NSObject
{ //Instance vars here
    NSString *name;
    int salary;
    int bonus;
}
// methods outside curly brackets
-  (void)setSalary:(int)cash withBonus:(int)extra
@end
```

Objective C - Class

In .m file:

```
#import ”Employee.h”

@implementation Employee
- (void)setSalary:(int)cash withBonus:(int)extra
{
   salary = cash;
   bonus = extra;
}
@end
```

Objective C - Protocols

- Used to simulate multiple inheritance and add functionality on top of existing objects (i.e. for delegates), similar to interfaces in Java:

```
@protocol Locking
- (void)lock;
- (void)unlock;
@end
```

- Denotes that there is an abstract idea of „Locking“
- Classes can state that they implement „Locking“ by declaring the following:

```
@interface SomeClass : SomeSuperClass <Locking>
@end
```

Objective C Properties

- .h file:

```
@interface MyDetailViewController : UIViewController {
        NSString *labelText;
}
@property (nonatomic, strong) NSString *labelText;
@end
```

- .m file:

```
@synthesize labelText;
        -(void)someMethod {
        self.labelText = @”hello”;
}
```

creates accessor methods: setLabelText (retains/releases) and getLabelText.

dot-syntax means: use property’s setLabelText accessor method, will retain the object

equivalent to [self setLabelText:@”hello”];

Implicit Setter/Getter Accessor Methods

- .h file: @property (nonatomic, strong) NSString *labelText;
- .m file: @synthesize labelText;
- Automatic creation of accessor methods:

```
- (void) setLabelText:(NSString *)newLabelText {
    [labelText release];
    labelText = newLabelText;
    [labelText retain];
}
- (NSString*) getLabelText {
    return labelText;
}
```

decrement reference counter on old object (if any)

increment reference counter on new object (if any)

- Properties are accessible from other classes, data members only if declared @public

Property Attributes

- Writability: readwrite (default), readonly
- Setter semantics: strong, weak ← with ARC
- Atomicity: atomic (default), nonatomic

- “readonly” means only a getter, but no setter accessor method is generated by @synthesize

Selectors

- Methods as arguments (useful for callback methods)
- Example: setting a button callback method
- .h file

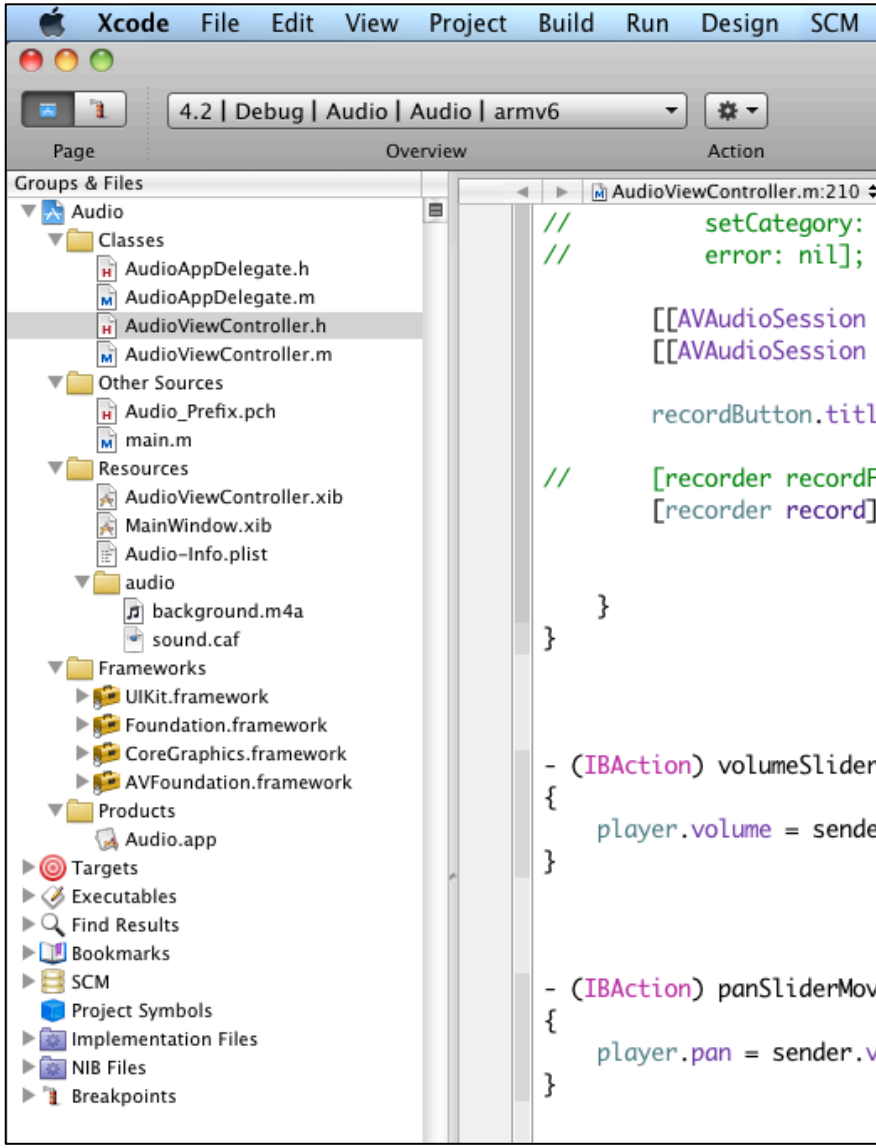
```
@interface MyDetailViewController : UIViewController {
    IBOutlet UIButton *newButton;
}
@property (nonatomic, retain) UIButton *newButton;
- (void) newButtonPressed:(id)source;
```

- .m file

```
- (void)someInitializationMethod {
    [newButton addTarget:self action:@selector(newButtonPressed:)
                        forControlEvents:UIControlEventTouchUpInside];
}
- (void) newButtonPressed:(id)source { NSLog(@"newButtonPressed"); }
```

Contents of an Xcode iPhone Project

- Source files
- Compiled Code
- Framework code
  - E.g. UIKit.framework
- Nib file (extension .xib)
  - Contains interface builder data
- Resources
  - Media (images, icons, sound)
- Info.plist file
  - Application configuration data



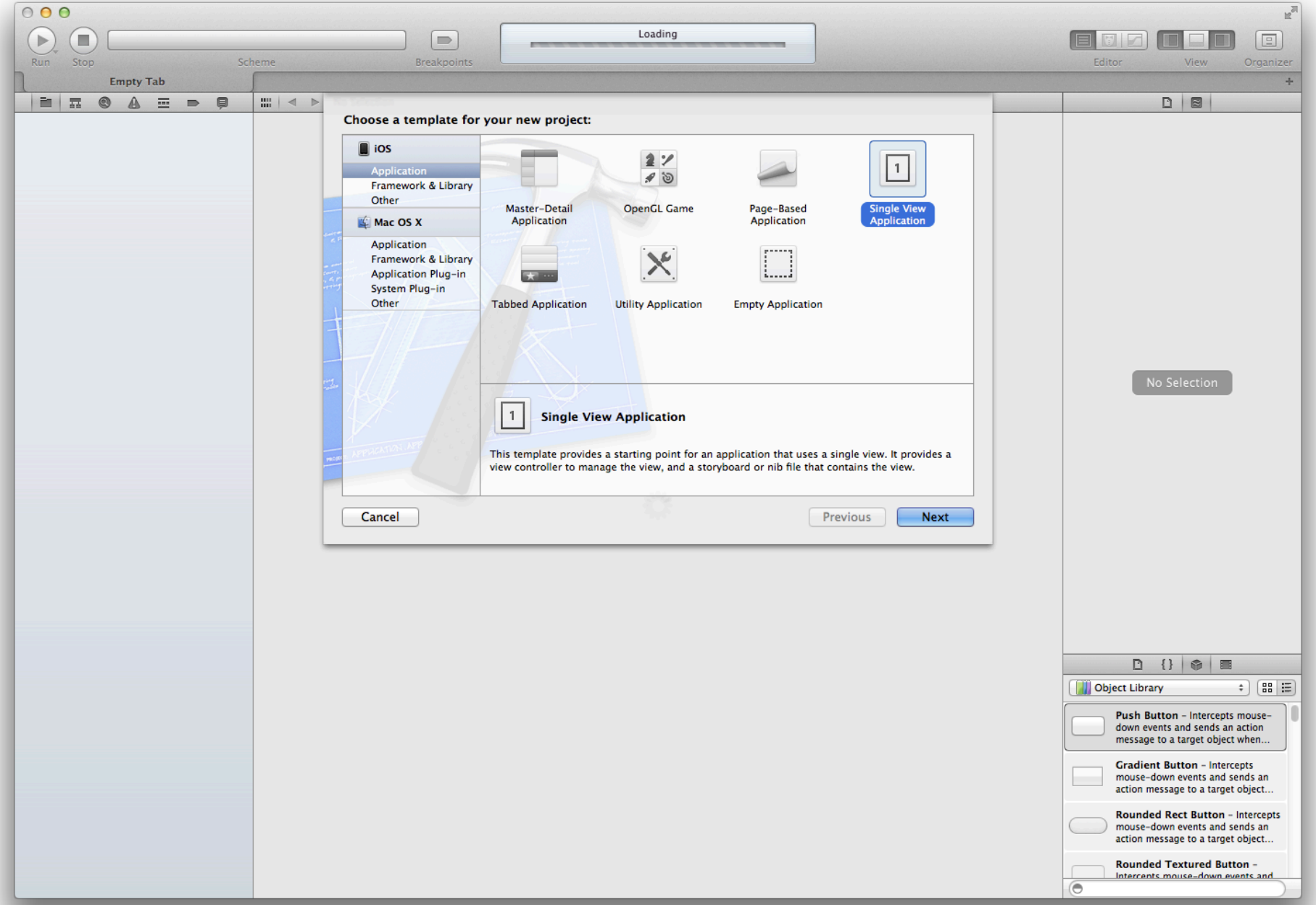
HELLO WORLD

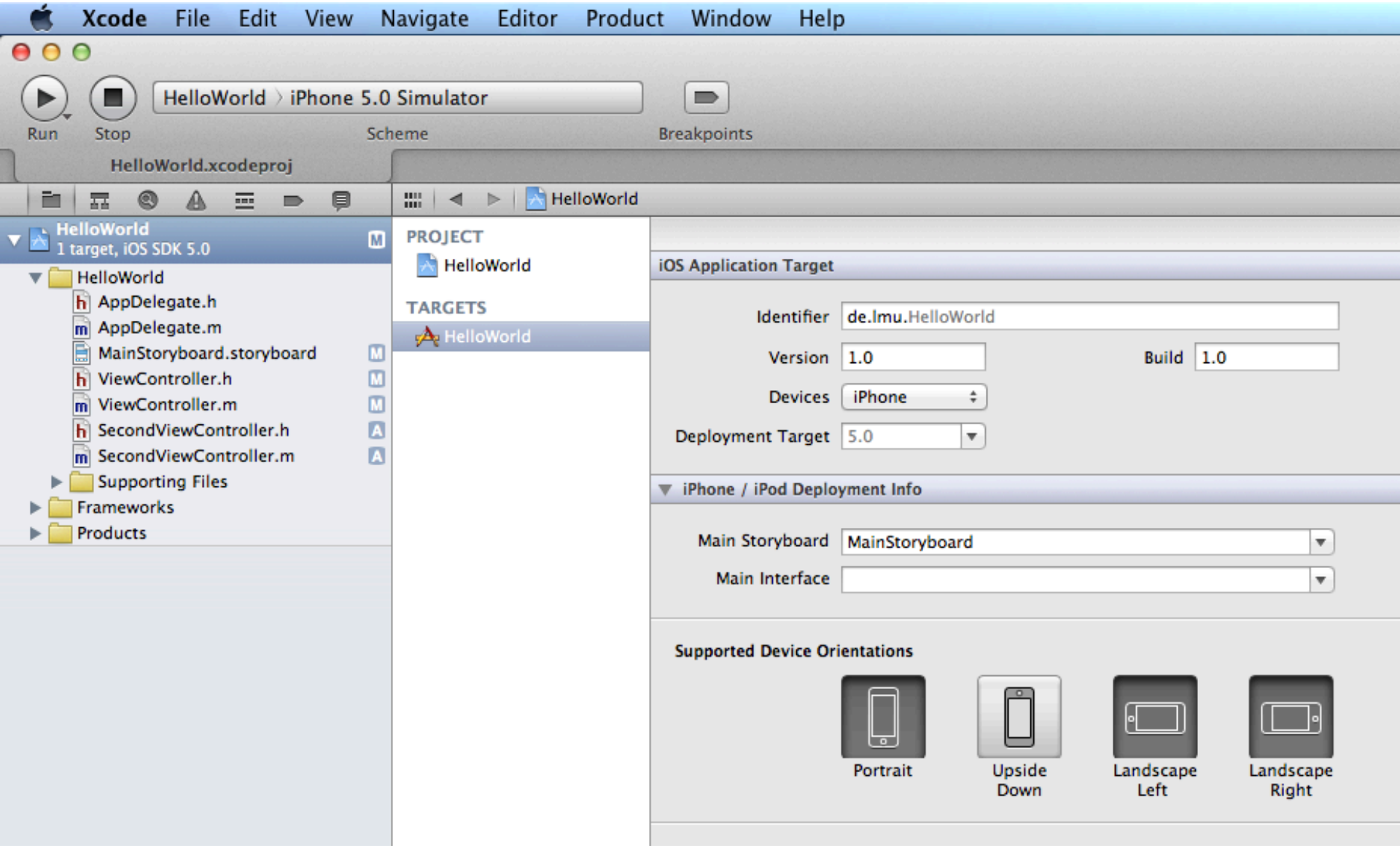
“Hello World” Steps

- Creating a project (“Single View Application”)
  - Use storyboards, use ARC, use Git
- Inspecting package contents
  - Navigator (left pane)
- Inspecting MainStoryboard
  - Utilities (right pane)
  - Roles of Views and ViewControllers
- Modify storyboard
  - Remove original view controller, add navigation view controller
  - Adding a label and a button
  - Add segue to another view controller

“Hello World” Steps

- Set label text when button was pressed
  - Add label outlet and property in .h file
  - Synthesize label property and set label text in .m file
- Increment counter when button was pressed
  - Add variable in .h file
  - Use NSString stringWithFormat in .m file
- Access label view using a tag (no IBOutlet required)
  - Define tag for label in Interface Builder (e.g. Tag = 100)
  - `UILabel *label = (UILabel*)[self.view viewWithTag:100];`
- Explain #pragma


Run
Stop
Scheme
Breakpoints
Loading
Editor
View
Organizer
Empty Tab
Choose a template for your new project:
iOS
Application
Framework & Library
Other
Mac OS X
Application
Framework & Library
Application Plug-in
System Plug-in
Other
Master-Detail Application
OpenGL Game
Page-Based Application
Single View Application
Tabbed Application
Utility Application
Empty Application
Single View Application
This template provides a starting point for an application that uses a single view. It provides a view controller to manage the view, and a storyboard or nib file that contains the view.
Cancel
Previous
Next
No Selection
Object Library
Push Button – Intercepts mouse-down events and sends an action message to a target object when...
Gradient Button – Intercepts mouse-down events and sends an action message to a target object...
Rounded Rect Button – Intercepts mouse-down events and sends an action message to a target object...
Rounded Textured Button –

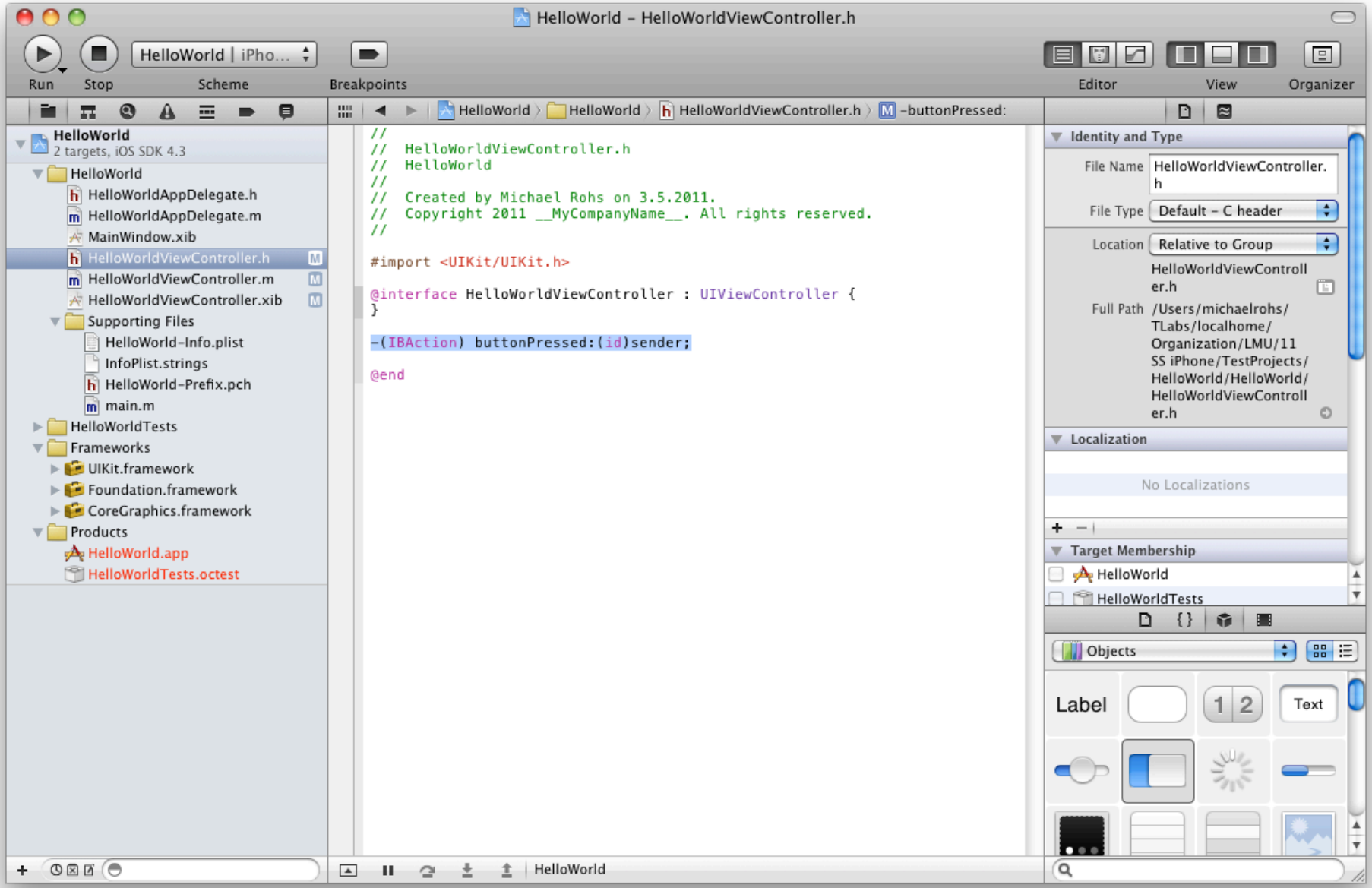

Xcode File Edit View Navigate Editor Product Window Help
Run
Stop
HelloWorld 〉 iPhone 5.0 Simulator
Scheme
Breakpoints
HelloWorld.xcodeproj
HelloWorld
1 target, iOS SDK 5.0
HelloWorld
AppDelegate.h
AppDelegate.m
MainStoryboard.storyboard
ViewController.h
ViewController.m
SecondViewController.h
SecondViewController.m
Supporting Files
Frameworks
Products
PROJECT
HelloWorld
TARGETS
HelloWorld
iOS Application Target
Identifier
de.lmu.HelloWorld
Version
1.0
Build
1.0
Devices
iPhone
Deployment Target
5.0
iPhone / iPod Deployment Info
Main Storyboard
MainStoryboard
Main Interface
Supported Device Orientations
Portrait
Upside Down
Landscape Left
Landscape Right

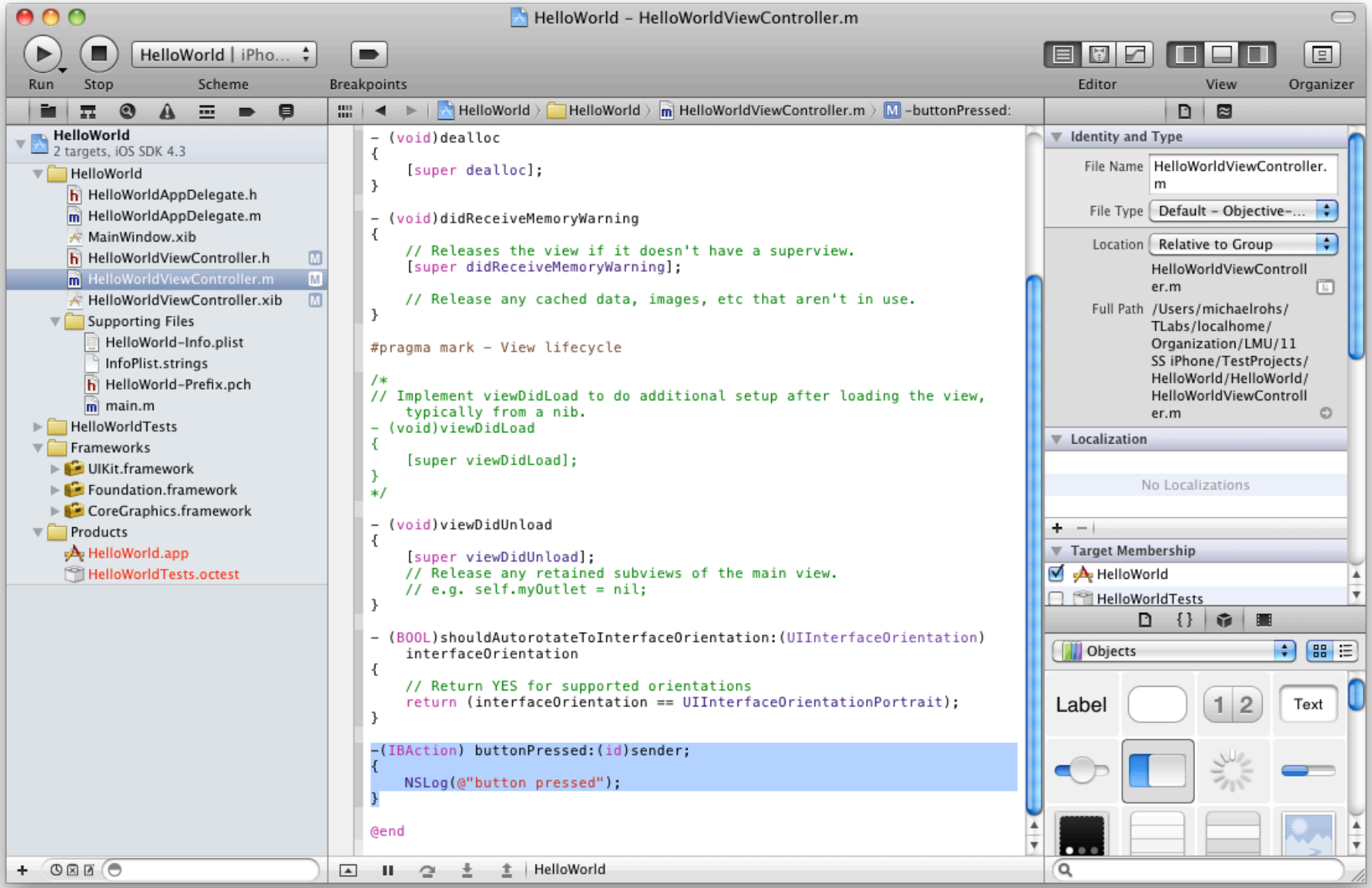
UIViewController subclasses

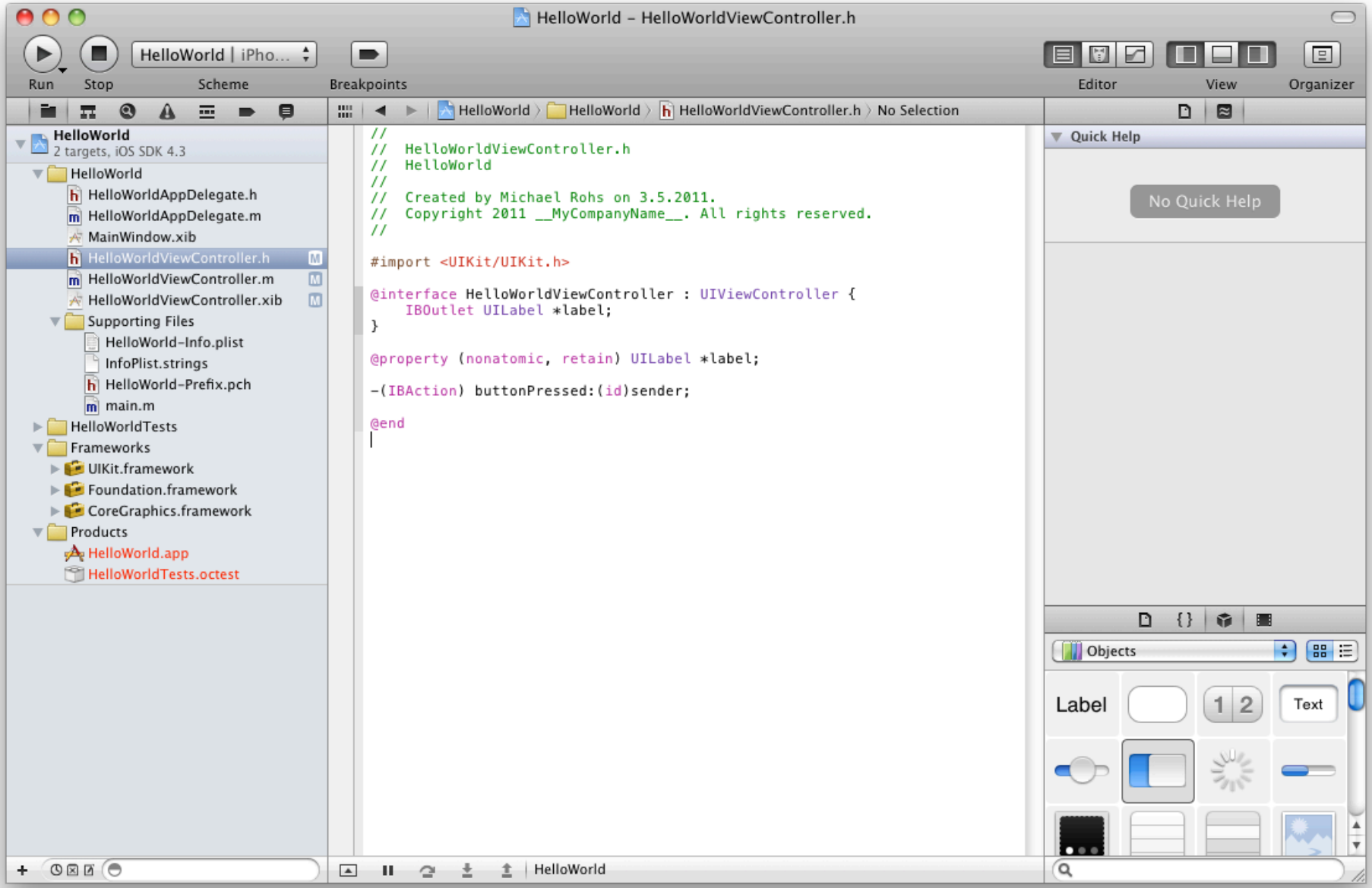
- View lifecycle
  - (void)viewDidLoad
  - (void)viewDidUnload
- View events
  - (void) viewWillAppear:(BOOL)animated
  - (void) viewWillDisappear:(BOOL)animated
  - (void) viewDidAppear:(BOOL)animated
  - (void) viewDidDisappear:(BOOL)animated
- Rotation settings and events

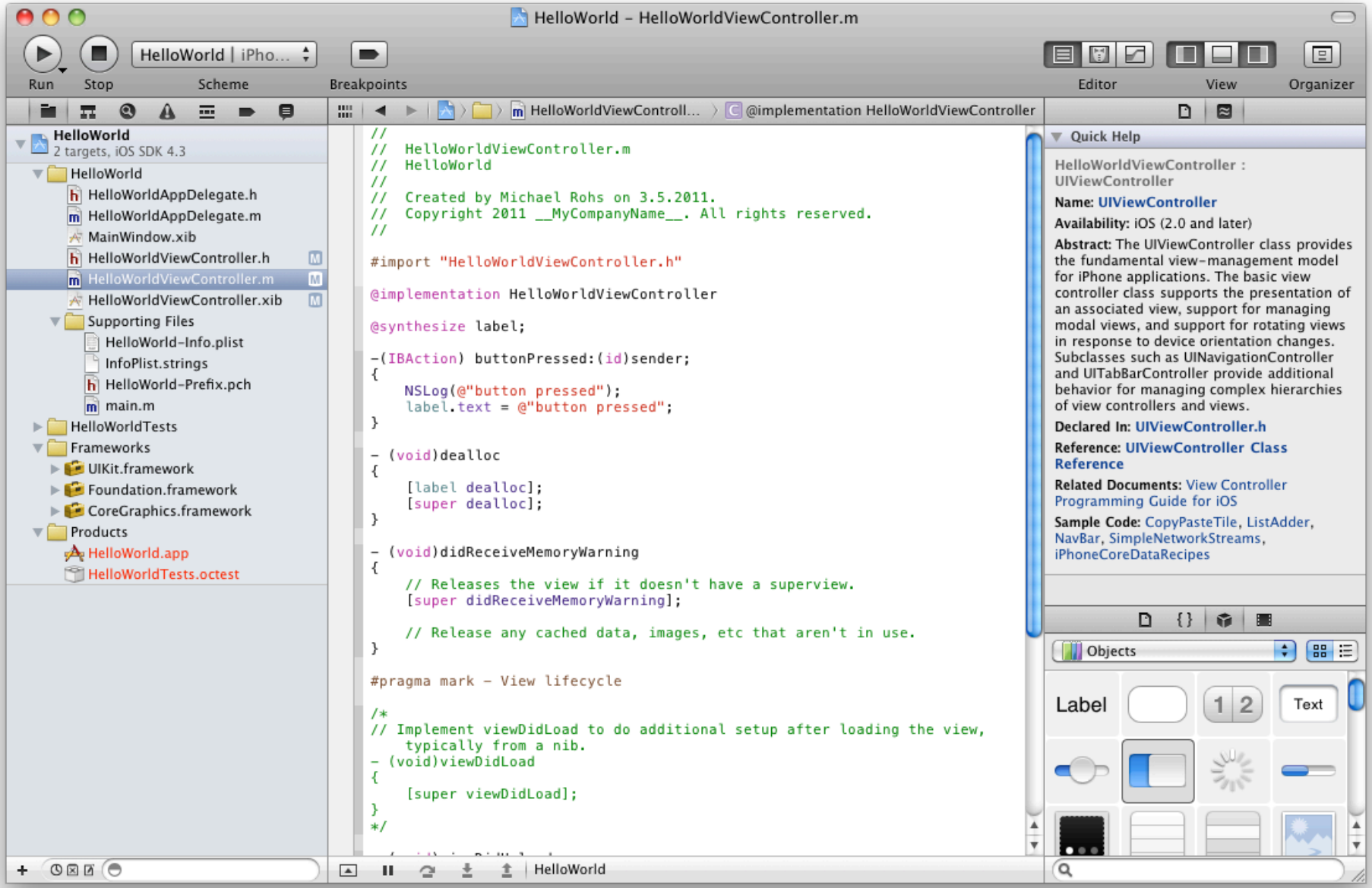
  interfaceOrientation property

  – shouldAutorotateToInterfaceOrientation:
- many more… → see documentation


HelloWorld – HelloWorldViewController.h
Run
Stop
HelloWorld | iPho...
Scheme
Breakpoints
Editor
View
Organizer
HelloWorld
2 targets, iOS SDK 4.3
HelloWorld
HelloWorldAppDelegate.h
HelloWorldAppDelegate.m
MainWindow.xib
HelloWorldViewController.h
HelloWorldViewController.m
HelloWorldViewController.xib
Supporting Files
HelloWorld-Info.plist
InfoPlist.strings
HelloWorld-Prefix.pch
main.m
HelloWorldTests
Frameworks
UIKit.framework
Foundation.framework
CoreGraphics.framework
Products
HelloWorld.app
HelloWorldTests.octest
HelloWorld › HelloWorld › HelloWorldViewController.h › -buttonPressed:
//
// HelloWorldViewController.h
// HelloWorld
//
// Created by Michael Rohs on 3.5.2011.
// Copyright 2011 __MyCompanyName__. All rights reserved.
//
#import <UIKit/UIKit.h>
@interface HelloWorldViewController : UIViewController {
}
-(IBAction) buttonPressed:(id)sender;
@end
Identity and Type
File Name
HelloWorldViewController.h
File Type
Default – C header
Location
Relative to Group
HelloWorldViewController.h
Full Path
/Users/michaelrohs/TLabs/localhome/Organization/LMU/11 SS iPhone/TestProjects/HelloWorld/HelloWorld/HelloWorldViewController.h
Localization
No Localizations
Target Membership
HelloWorld
HelloWorldTests
Objects
Label
1 2
Text
HelloWorld

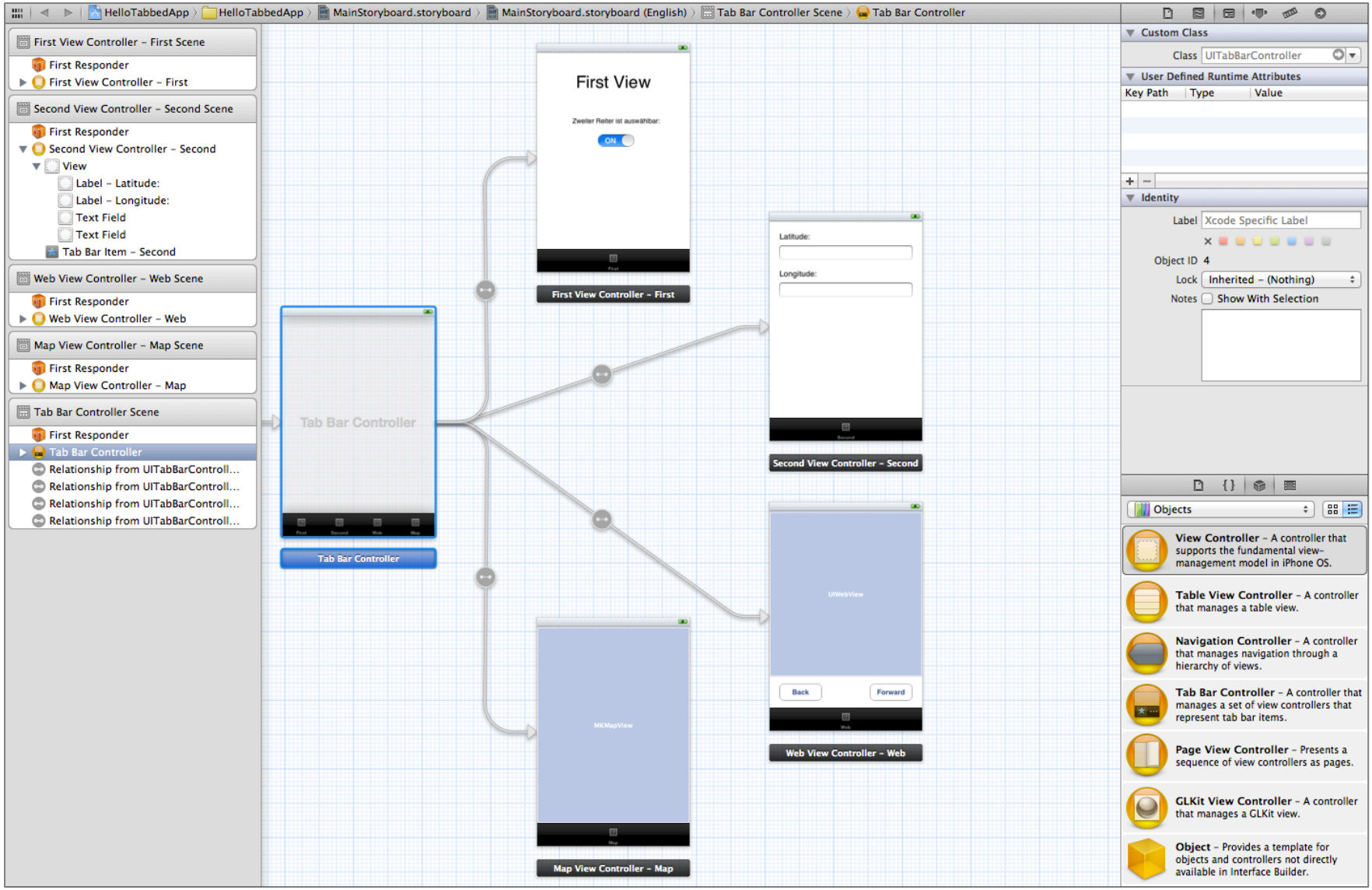

HelloWorld – HelloWorldViewController.m
Run
Stop
HelloWorld | iPho...
Scheme
Breakpoints
Editor
View
Organizer
HelloWorld
2 targets, iOS SDK 4.3
HelloWorld
HelloWorldAppDelegate.h
HelloWorldAppDelegate.m
MainWindow.xib
HelloWorldViewController.h
HelloWorldViewController.m
HelloWorldViewController.xib
Supporting Files
HelloWorld-Info.plist
InfoPlist.strings
HelloWorld-Prefix.pch
main.m
HelloWorldTests
Frameworks
UIKit.framework
Foundation.framework
CoreGraphics.framework
Products
HelloWorld.app
HelloWorldTests.octest
HelloWorld › HelloWorld › HelloWorldViewController.m › -buttonPressed:
- (void)dealloc
{
[super dealloc];
}
- (void)didReceiveMemoryWarning
{
// Releases the view if it doesn't have a superview.
[super didReceiveMemoryWarning];
// Release any cached data, images, etc that aren't in use.
}
#pragma mark - View lifecycle
/*
// Implement viewDidLoad to do additional setup after loading the view, typically from a nib.
- (void)viewDidLoad
{
[super viewDidLoad];
}
*/
- (void)viewDidUnload
{
[super viewDidUnload];
// Release any retained subviews of the main view.
// e.g. self.myOutlet = nil;
}
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
{
// Return YES for supported orientations
return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
-(IBAction) buttonPressed:(id)sender;
{
NSLog(@"button pressed");
}
@end
HelloWorld
Identity and Type
File Name
HelloWorldViewController.m
File Type
Default – Objective-...
Location
Relative to Group
HelloWorldViewController.m
Full Path
/Users/michaelrohs/TLabs/localhome/Organization/LMU/11 SS iPhone/TestProjects/HelloWorld/HelloWorld/HelloWorldViewController.m
Localization
No Localizations
Target Membership
HelloWorld
HelloWorldTests
Objects
Label
1 2
Text


HelloWorld – HelloWorldViewController.h
Run
Stop
HelloWorld | iPho...
Scheme
Breakpoints
Editor
View
Organizer
HelloWorld 〉 HelloWorld 〉 HelloWorldViewController.h 〉 No Selection
HelloWorld
2 targets, iOS SDK 4.3
HelloWorld
HelloWorldAppDelegate.h
HelloWorldAppDelegate.m
MainWindow.xib
HelloWorldViewController.h
HelloWorldViewController.m
HelloWorldViewController.xib
Supporting Files
HelloWorld-Info.plist
InfoPlist.strings
HelloWorld-Prefix.pch
main.m
HelloWorldTests
Frameworks
UIKit.framework
Foundation.framework
CoreGraphics.framework
Products
HelloWorld.app
HelloWorldTests.octest
//
// HelloWorldViewController.h
// HelloWorld
//
// Created by Michael Rohs on 3.5.2011.
// Copyright 2011 __MyCompanyName__. All rights reserved.
//
#import <UIKit/UIKit.h>
@interface HelloWorldViewController : UIViewController {
IBOutlet UILabel *label;
}
@property (nonatomic, retain) UILabel *label;
-(IBAction) buttonPressed:(id)sender;
@end
Quick Help
No Quick Help
Objects
Label
1 2
Text
HelloWorld


HelloWorld – HelloWorldViewController.m
Run
Stop
HelloWorld | iPho...
Scheme
Breakpoints
Editor
View
Organizer
HelloWorld
2 targets, iOS SDK 4.3
HelloWorld
HelloWorldAppDelegate.h
HelloWorldAppDelegate.m
MainWindow.xib
HelloWorldViewController.h
HelloWorldViewController.m
HelloWorldViewController.xib
Supporting Files
HelloWorld-Info.plist
InfoPlist.strings
HelloWorld-Prefix.pch
main.m
HelloWorldTests
Frameworks
UIKit.framework
Foundation.framework
CoreGraphics.framework
Products
HelloWorld.app
HelloWorldTests.octest
HelloWorldViewControll...
@implementation HelloWorldViewController
//
// HelloWorldViewController.m
// HelloWorld
//
// Created by Michael Rohs on 3.5.2011.
// Copyright 2011 __MyCompanyName__. All rights reserved.
//
#import "HelloWorldViewController.h"
@implementation HelloWorldViewController
@synthesize label;
-(IBAction) buttonPressed:(id)sender;
{
NSLog(@"button pressed");
label.text = @"button pressed";
}
- (void)dealloc
{
[label dealloc];
[super dealloc];
}
- (void)didReceiveMemoryWarning
{
// Releases the view if it doesn't have a superview.
[super didReceiveMemoryWarning];
// Release any cached data, images, etc that aren't in use.
}
#pragma mark - View lifecycle
/*
// Implement viewDidLoad to do additional setup after loading the view,
typically from a nib.
- (void)viewDidLoad
{
[super viewDidLoad];
}
*/
Quick Help
HelloWorldViewController : UIViewController
Name: UIViewController
Availability: iOS (2.0 and later)
Abstract: The UIViewController class provides the fundamental view-management model for iPhone applications. The basic view controller class supports the presentation of an associated view, support for managing modal views, and support for rotating views in response to device orientation changes. Subclasses such as UINavigationController and UITabBarController provide additional behavior for managing complex hierarchies of view controllers and views.
Declared In: UIViewController.h
Reference: UIViewController Class Reference
Related Documents: View Controller Programming Guide for iOS
Sample Code: CopyPasteTile, ListAdder, NavBar, SimpleNetworkStreams, iPhoneCoreDataRecipes
Objects
Label
1 2
Text
HelloWorld

STORYBOARDS

Storyboards = Scenes + Segues

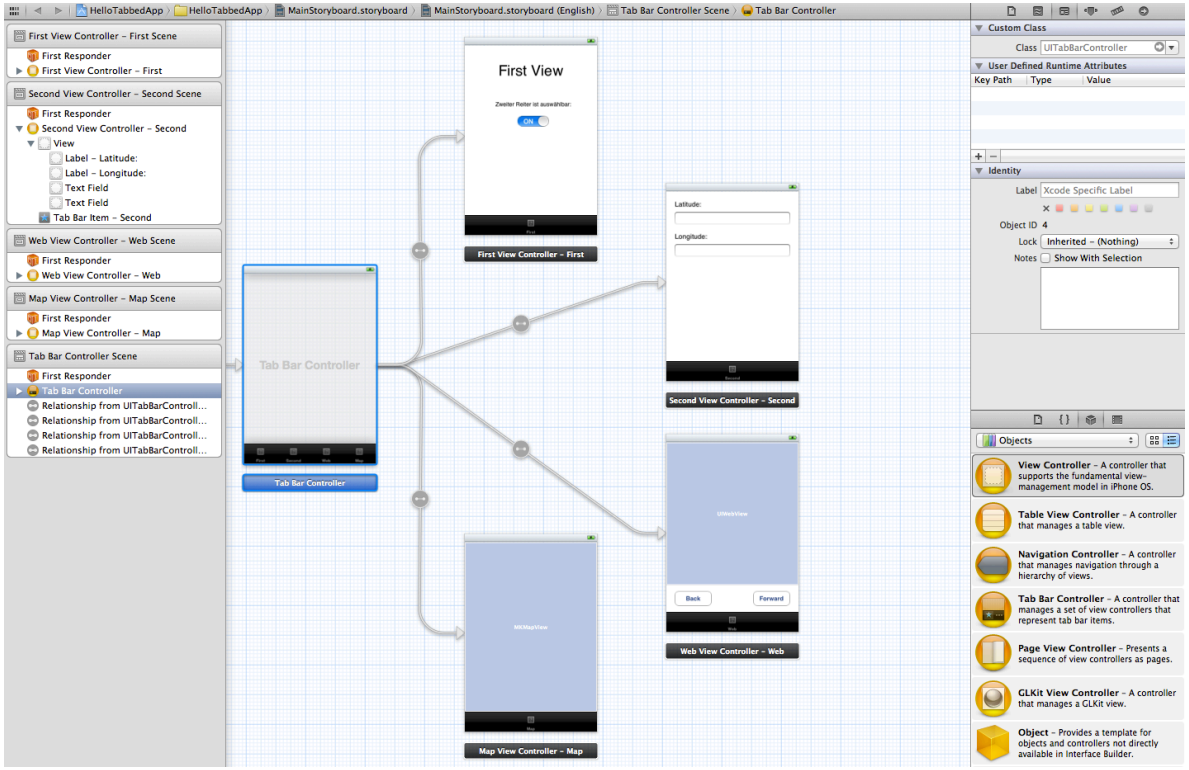


Storyboards = Scenes + Segues

- Scene: A single screen of content
- Segue: Transition between scenes
- One storyboard per project, contains all scenes

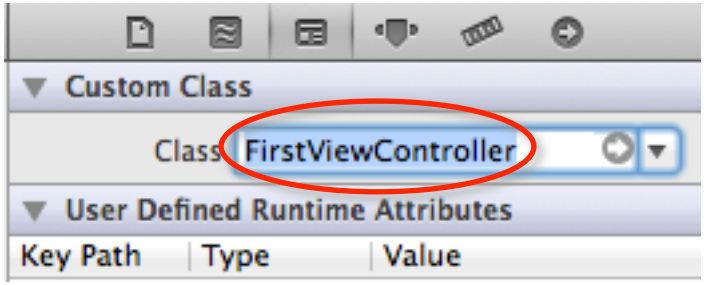
- Zoom in-out: double-click background
- Zoom in to edit scene

- class: UIStoryboard

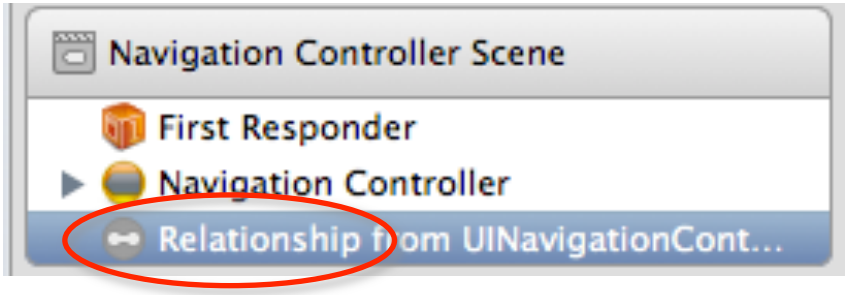


Storyboards = Scenes + Segues

- Scene: A single screen
  - UIViewController subclass
  - Create subclass: File | New File… | UIViewController subclass
  - Set new subclass to scene



- Segue: Transition between scenes
  - UIStoryboardSegue: transitions are objects, too!
    - Performs the visual transition between two view controllers
  - Types: Push, Modal, Custom
  - Relationships link containers (Tab Bar Controller, Navigation Controller) to content views



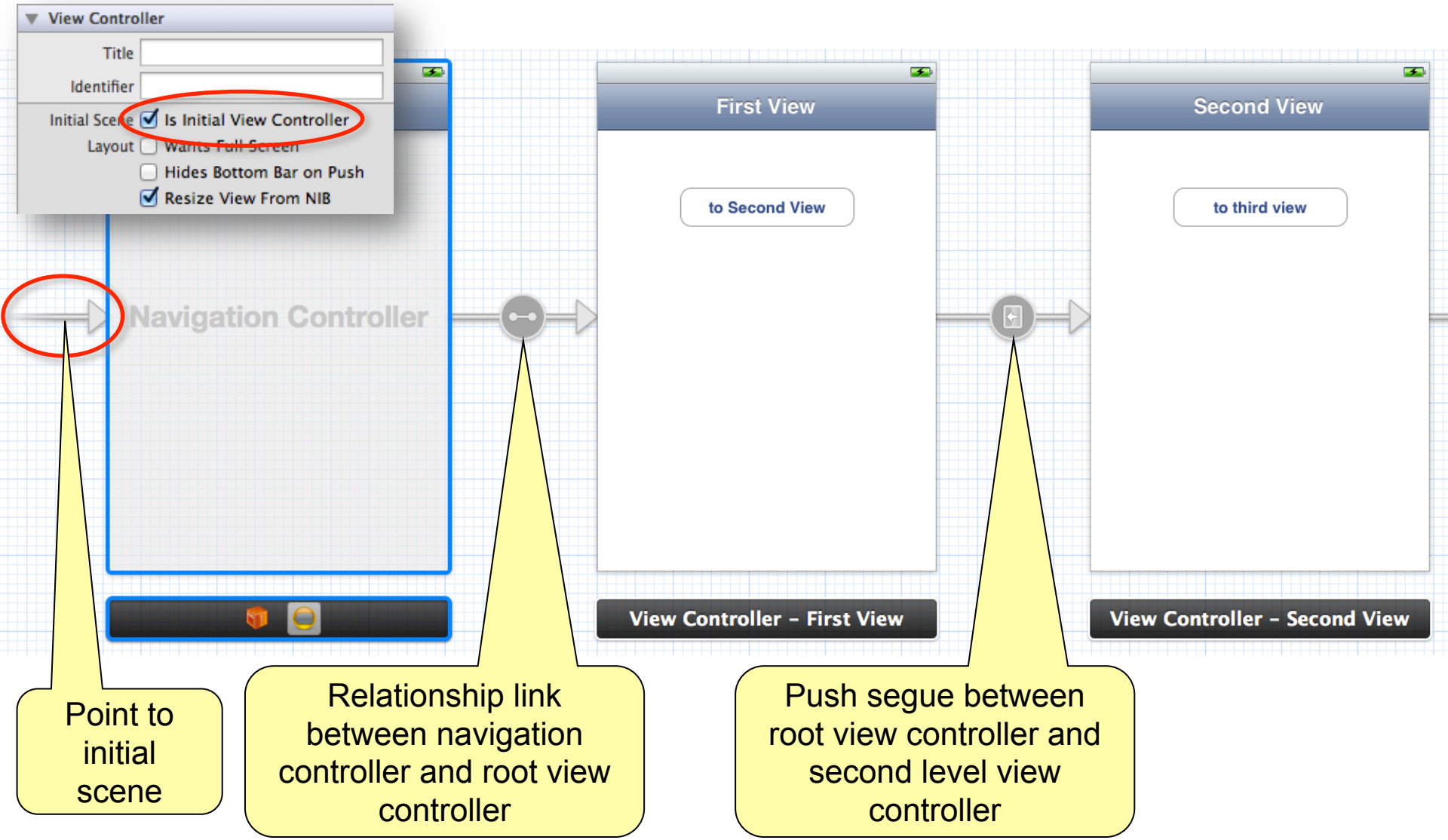
No More MainWindow.xib

- Can still configure projects to use xib-files in iOS 5
  - Adding items from library, IBOutlets, IBActions have not changed
- If using storyboards, then …-Info.plist shows:

Bundle version	String	1.0
Application requires iPhone environmer	Boolean	YES
▶ Required device capabilities	Array	(1 item)
▶ Supported interface orientations	Array	(3 items)
Main storyboard file base name	String	Storyboard

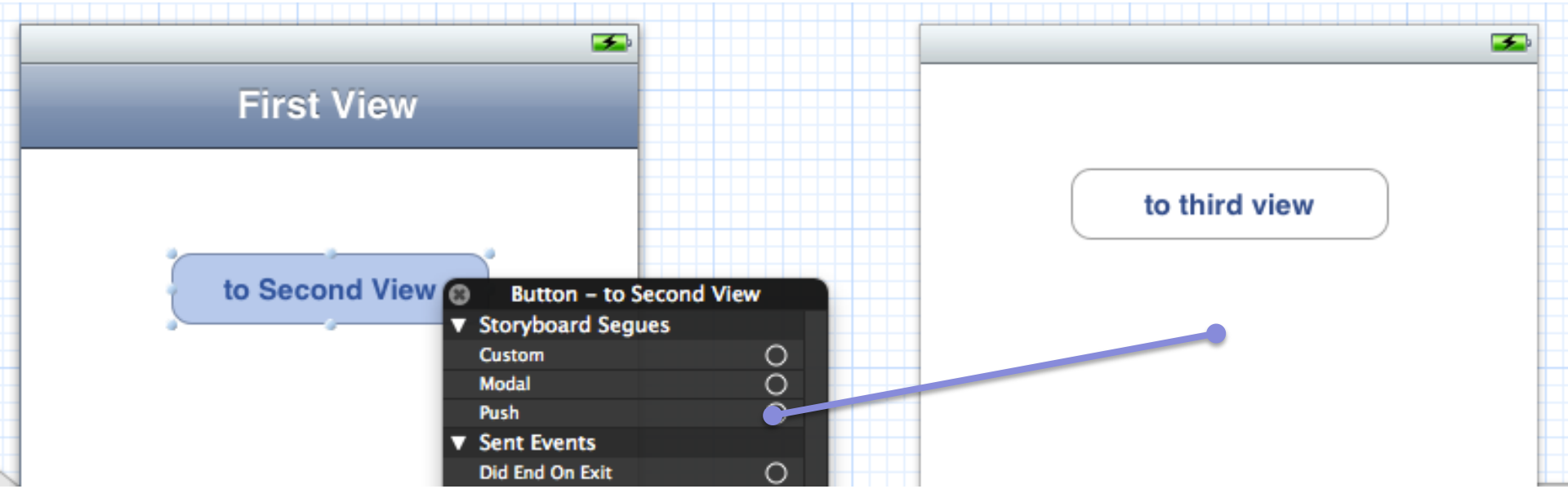
- Automatically instantiates “initial” view controller:

Scenes and Segues



Creating Segues

- Ctrl-click element that invokes second scene (e.g. button)
- Or right-click this element

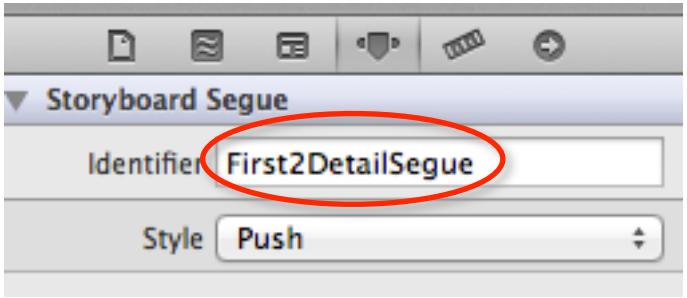


Pass Data Between Scenes (Up)

- Implement method prepareForSegue:sender: in source view controller

```
- (void) prepareForSegue:(UIStoryboardSegue*)segue sender:(id)sender {
    if ([[segue identifier] isEqualToString:@"First2DetailSegue"]) {
        DetailViewController *dvc = (DetailViewController*) [
                                        segue destinationViewController];
        dvc.data = data;
    }
}
```

here the data is passed to the detail controller

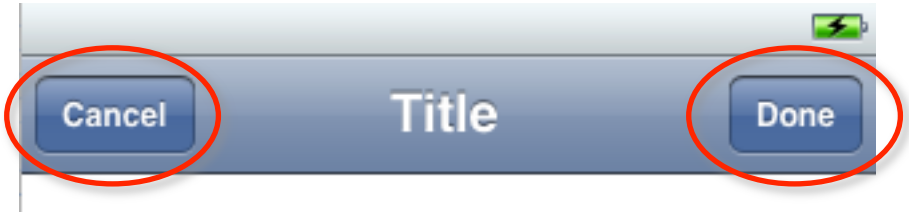


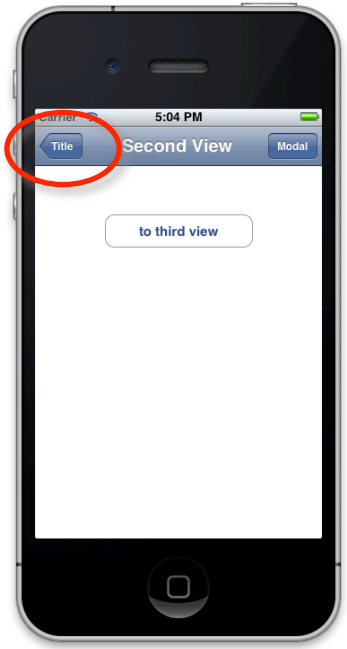
- Set segue identifier in storyboard
- Subclass UIStoryboardSegue for custom transitions
  - Specify as custom in storyboard
  - Override “perform” method

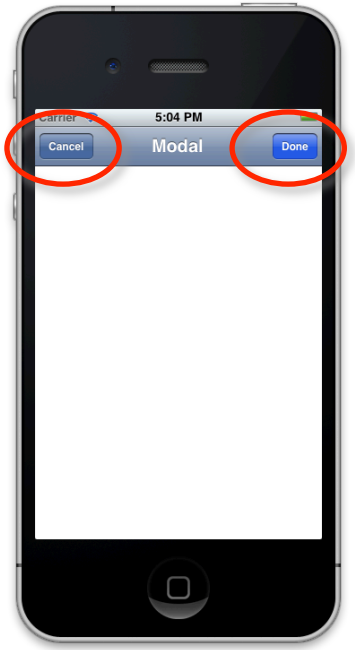
Pass Data Between Scenes (Back)

- For push segues 
  - Update data structure set in prepareForSegue
  - Back button automatically pops view controller

- For modal segues 
  - Use delegate object that processes done/cancel and dismisses modal view controller







Pass Data Back from Modal View

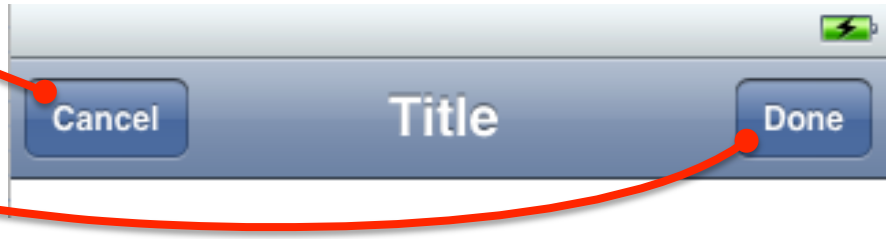
- Modal view defines delegate protocol

```
@class MyModalViewController; // forward declaration

@protocol MyModalViewControllerDelegate
- (void) myModalViewControllerDidCancel:(MyModalViewController*)controller;
- (void) myModalViewControllerDidSave:(MyModalViewController*)controller;
@end

@interface MyModalViewController : UIViewController
@property (nonatomic, strong) id <MyModalViewControllerDelegate> delegate;
- (IBAction)cancel:(id)sender;
- (IBAction)done:(id)sender;
@end
```

tells the compiler that My…Controller is a class (used before declared)

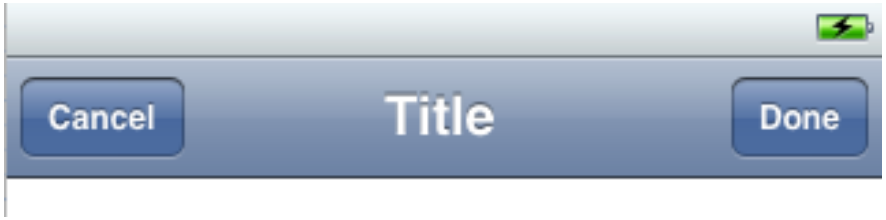


Pass Data Back from Modal View

- MyModalViewController.m, call the delegate

```
- (IBAction)cancel:(id)sender {
    [self.delegate myModalViewControllerDidCancel:self];
}

- (IBAction)done:(id)sender {
    [self.delegate myModalViewControllerDidSave:self];
}
```



Pass Data Back from Modal View

- FirstViewController.h

```
#import "MyModalViewController.h”
@interface FirstViewController : UIViewController
                                 <MyModalViewControllerDelegate>
@end
```

starting view controller implements delegate protocol

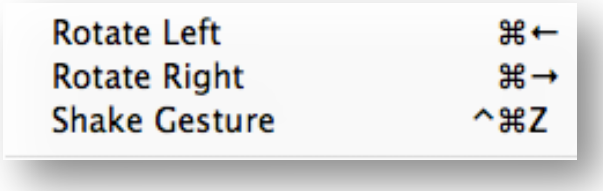
- FirstViewController.m

set segue identifier in storyboard

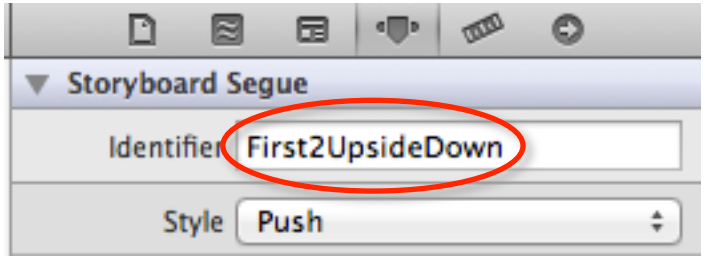
```
- (void) prepareForSegue:(UIStoryboardSegue*)segue sender:(id)sender {
    if ([[segue identifier] isEqualToString:@”First2Modal"]) {
        MyModalViewController *mvc = [segue destinationViewController];
        mvc.delegate = self;
} }
- (void) myModalViewControllerDidCancel:(MyModalViewController*)controller
{   [controller dismissModalViewControllerAnimated:YES];   }
- (void) myModalViewControllerDidSave:(MyModalViewController*)controller
{   [controller dismissModalViewControllerAnimated:YES];   }
```

Programmatically trigger Segues

- performSegueWithIdentifier
- Example: Show a scene when device upside down
  - Rotate simulator: ⌘←, ⌘→



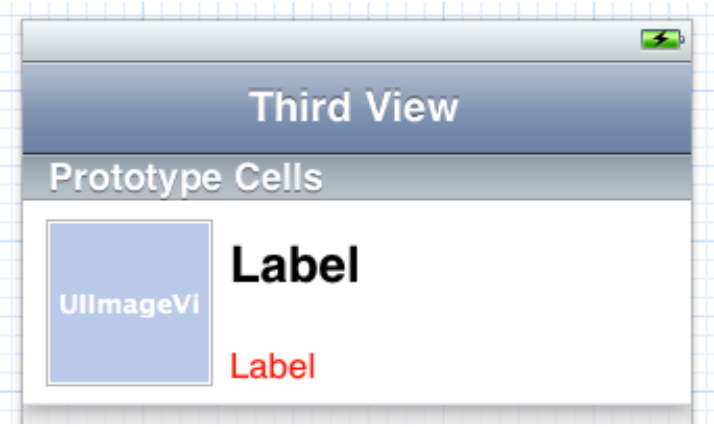
- In UIViewController subclass



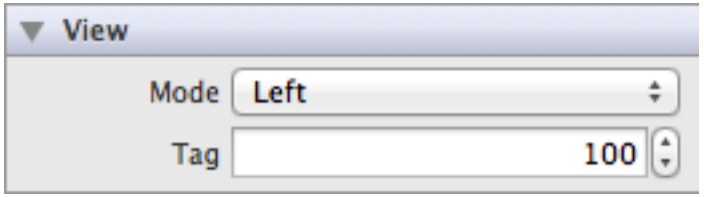

```
- (BOOL)shouldAutorotateToInterfaceOrientation:
            (UIInterfaceOrientation)interfaceOrientation
{
    if (interfaceOrientation == UIInterfaceOrientationPortraitUpsideDown) {
        [self performSegueWithIdentifier:@"First2UpsideDown" sender:self];
    }
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
```

Prototype Cells in Tables

- Prototype cells define the layout of table cells
- Set cell identifier in storyboard
- Use in cellForRowAtIndexPath

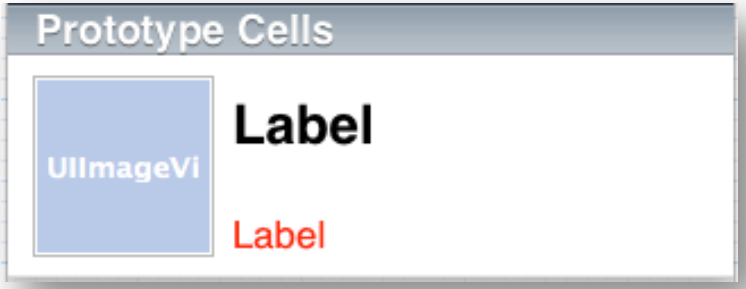



```
- (UITableViewCell *)tableView:(UITableView *)tableView
                 cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    static NSString *CellIdentifier = @"MyCustomCell";
    UITableViewCell *cell = [tableView
                             dequeueReusableCellWithIdentifier:CellIdentifier];
    UILabel *label = (UILabel*) [cell viewWithTag:100];
    label.text = [data objectAtIndex:indexPath.row];
    return cell;
}
```



- Or: sublass UITableViewCell

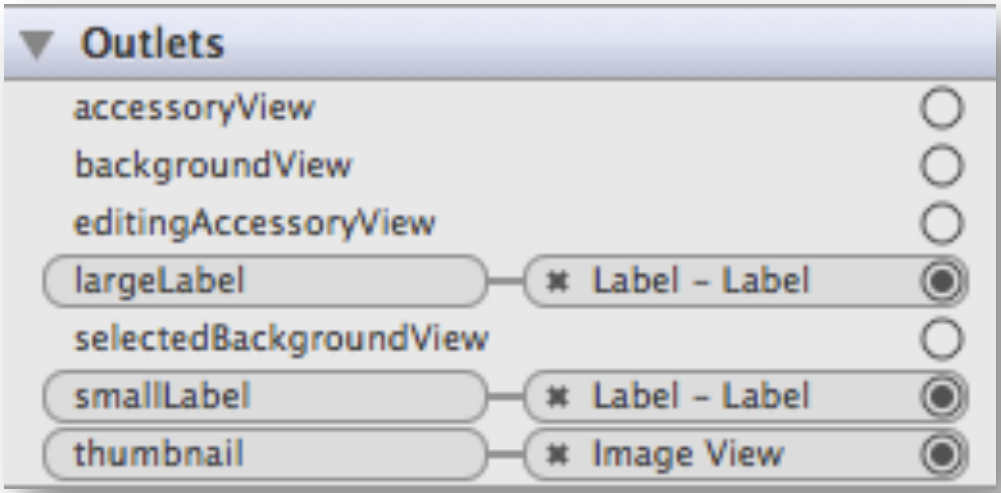
Custom TableViewCell



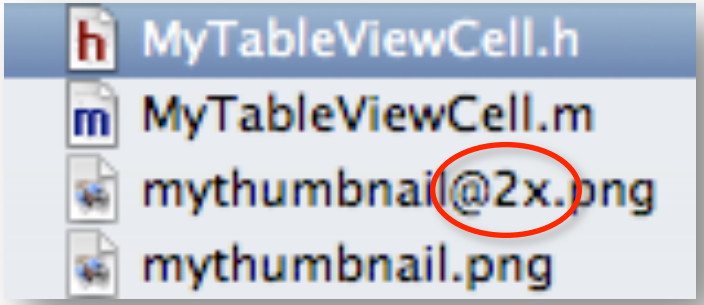
- MyTableViewCell.h

```
@interface MyTableViewCell : UITableViewCell
@property (nonatomic, strong) IBOutlet UILabel* largeLabel;
@property (nonatomic, strong) IBOutlet UILabel* smallLabel;
@property (nonatomic, strong) IBOutlet UIImageView* thumbnail;
@end
```

- Select prototype cell
- Connect objects to outlets



Custom TableViewCell

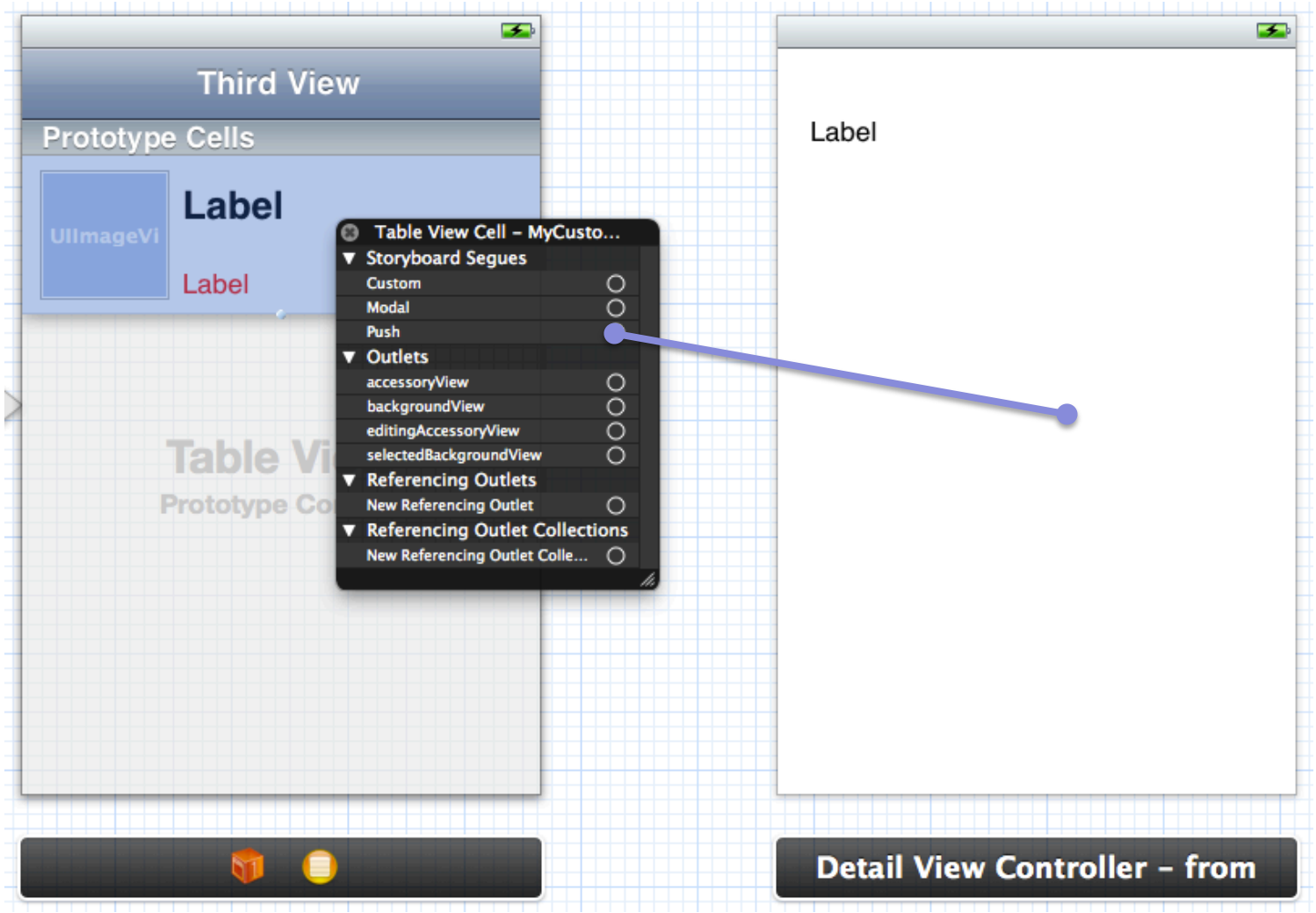



```
@interface MyTableViewCell : UITableViewCell
@property (nonatomic, strong) IBOutlet UILabel* largeLabel;
@property (nonatomic, strong) IBOutlet UILabel* smallLabel;
@property (nonatomic, strong) IBOutlet UIImageView* thumbnail;
@end
```

in MyTableViewController:

```
- (UITableViewCell *)tableView:(UITableView *)tableView
                 cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    MyTableViewCell *cell = [tableView
                    dequeueReusableCellWithIdentifier:@"MyCustomCell"];
    cell.largeLabel.text = [data objectAtIndex:indexPath.row];
    cell.smallLabel.text = @"this is a small label";
    cell.thumbnail.image = [UIImage imageNamed:@"mythumbnail"];
    return cell;
}
```

Link Prototype Cells to View Controllers



Passing Data from Table to Detail View

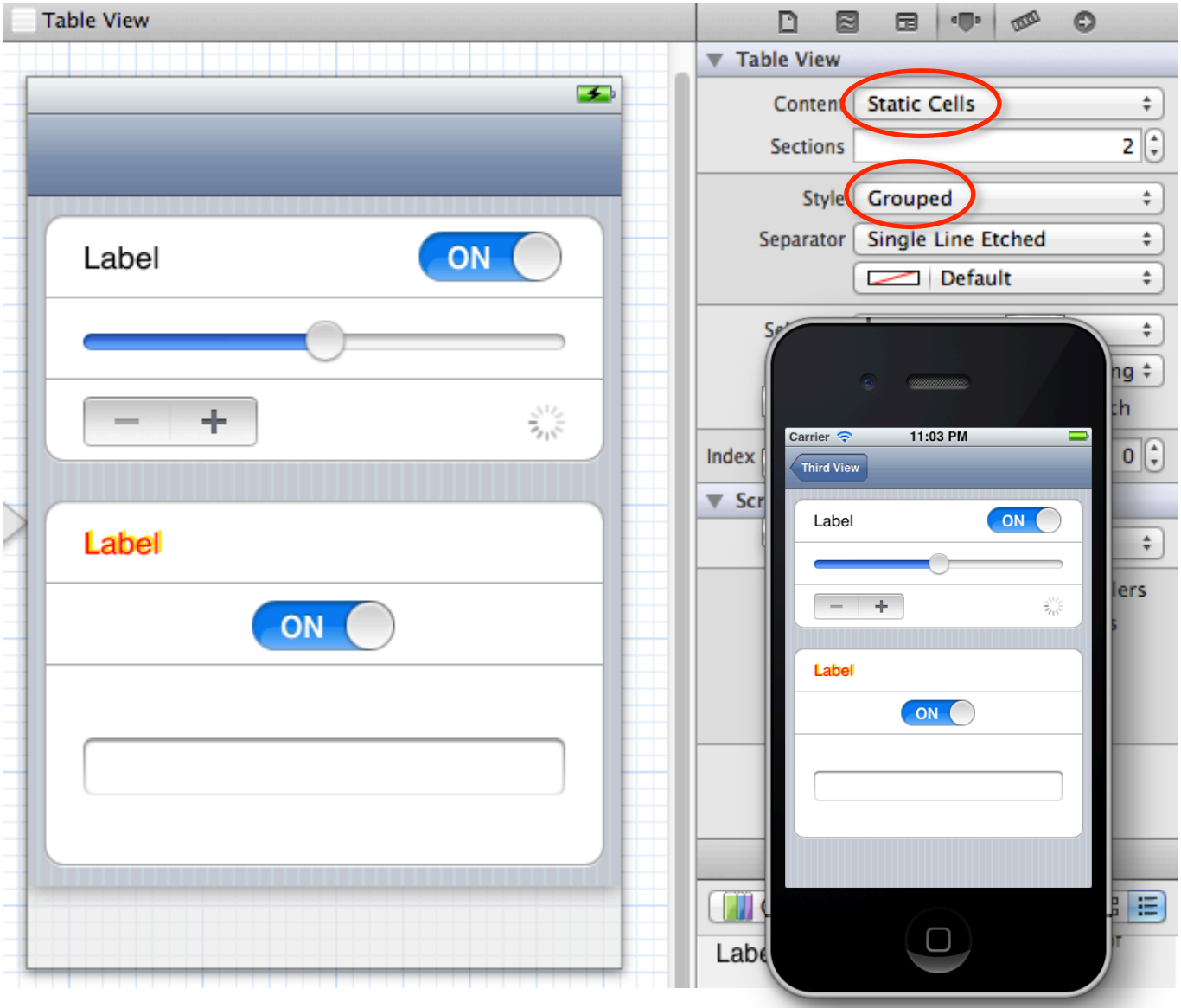
in MyTableViewController:

```
- (void) prepareForSegue:(UIStoryboardSegue*)segue sender:(id)sender
{
    if ([[segue identifier] isEqualToString:@"Third2Detail"]) {
        DetailViewController *dvc = (DetailViewController*)
                                            [segue destinationViewController];
        UITableViewCell *cell = sender;
        UILabel *label = (UILabel*) [cell viewWithTag:100];
        dvc.data = label.text;
    }
}
```

here the data is passed to the detail controller

Static Table Cells

- Fixed cell content
- Appears on device as is
- Nice for grouped tables (edit views)
- Hook up to controllers via outlets



AUTOMATIC REFERENCE COUNTING (ARC)

Automatic Reference Counting (ARC)

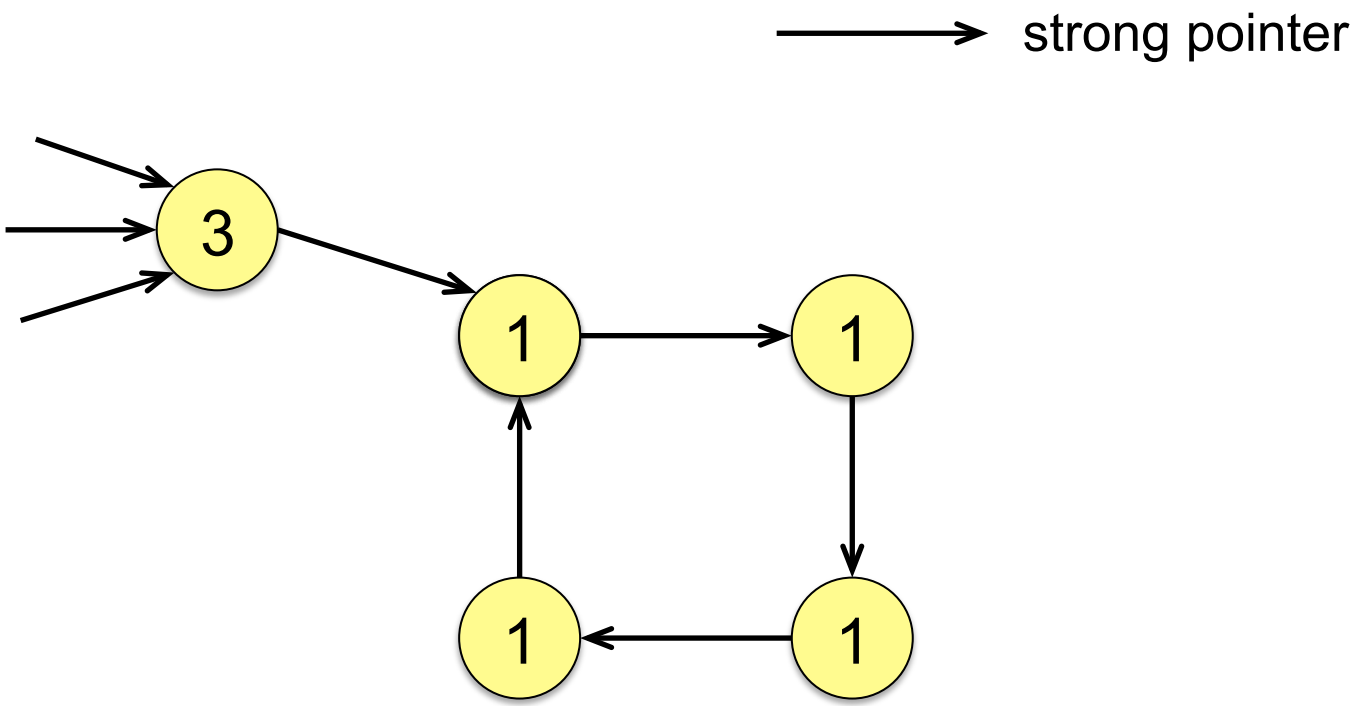
- Automates contain and release calls
  - Writing code without thinking about retain/release! ☺
  - Still uses reference counting internally
  - Retain, release, etc. not allowed; dealloc rarely necessary
- Objective-C language extensions
  - Automatic retain/release on entry/exit of scopes
  - Compiler knows about naming conventions (alloc, new, copy, …)
  - @autoreleasepool { … }
- ARC is a compile-time mechanism
  - Not a new runtime memory model
  - Not a garbage collector
  - Does not cover malloc/free, core foundation

Strong and Weak Pointers

- Strong pointers
  - Strong pointers keep objects alive
  - Strong pointers are like “retain” properties (+1 ref. count)
  - Default for all variables (instance variables, local variables, etc.)
  - Keyword: __strong
  - Example: __strong NSString *name;
- Weak pointers
  - Weak pointers do not keep objects alive
  - Weak pointers are like “assign” properties (+0 ref. count)
  - Weak pointers get nil when object is deallocated
  - Keyword: __weak
  - Example: __weak NSString *name;

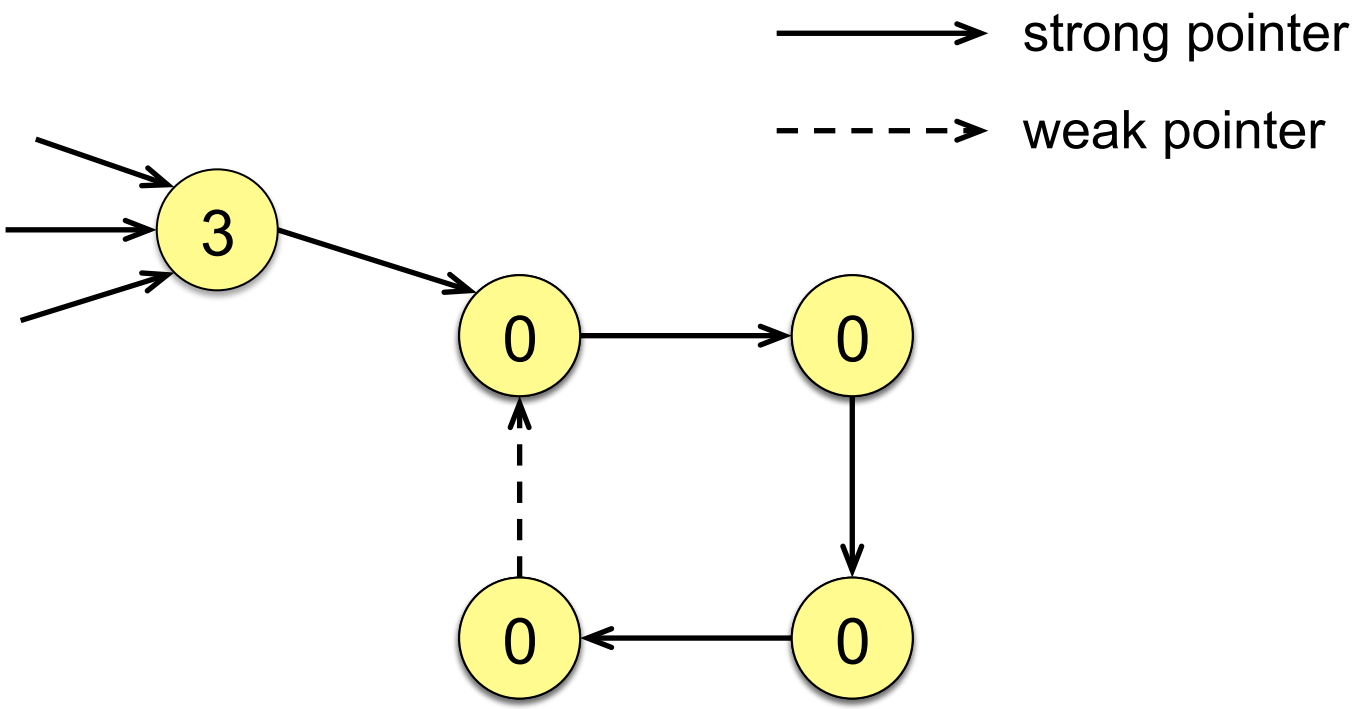
Cycles in Object Graphs

- Problem: Strong pointers keep objects in cycle alive
  - Memory leak!



Cycles in Object Graphs

- Problem: Strong pointers keep objects in cycle alive
- Solution: Weak pointers do not keep objects alive
  - A weak pointer gets nil when object it points to get deallocated



ARC Variable Modifiers

- __strong
  Strong pointer, the default
  Initialized to nil: NSString *name; → NSString *name = nil;
- __weak
  Weak pointer, initialized to nil, set to nil when object deallocated
- __unsafe_unretained
  Traditional variable, unretained, not set to nil
  Sometimes needed for non-Objective-C code
- __autoreleasing
  Out-parameters, not for general use

Writing a __strong Variable

- Code you write

```
name = newName
```

- What the compiler adds

```
[newName retain];
NSString *oldName = name;
name = newName;
[oldName release];
```

Scoping of __strong Variables

- Code you write

```
if (a < 10) {
    NSString *name =
                [[NSString alloc] init…];
    // using name…

}
```

- What the compiler adds

```
if (a < 10) {
    NSString *name =
                [[NSString alloc] init…];
    // using name…
    [name release];
}
```

ARC Deallocation Examples

- Automatic deallocation when variable goes out of scope
  - method exit
  - if-cause ends
  - etc.
- Object graph deallocation
  - Referenced object deallocated when root goes out of scope

Dealloc Object

```
- (void) myCallerMethod {
    [self myMethod];
    NSLog(@"after myMethod");
}

- (void) myMethod {
    MyObject *o = [[MyObject alloc] init];
    NSLog(@"%@", [o description]);
    NSLog(@"myMethod exit");
}
```

```
@interface MyObject : NSObject
@end

@implementation MyObject
- (void)dealloc {
    NSLog(@"MyObject::dealloc");
}
@end
```

Output:

<MyObject: 0x685b3f0>
myMethod exit
MyObject::dealloc
after myMethod

Dealloc Object after If-Clause

```
- (void) myCallerMethod {
    [self myMethod:TRUE];
    NSLog(@"after myMethod");
}

- (void) myMethod:(BOOL)condition {
    if (condition) {
        MyObject *o = [[MyObject alloc] init];
        NSLog(@"%@", [o description]);
    }
    NSLog(@"myMethod exit");
}
```

```
@interface MyObject : NSObject
@end

@implementation MyObject
- (void)dealloc {
    NSLog(@"MyObject::dealloc");
}
@end
```

Output:

<MyObject: 0x8844fd0>
MyObject::dealloc
myMethod exit
after myMethod

Dealloc Object in Array

```
- (void) myCallerMethod {
    [self myMethod];
    NSLog(@"after myMethod");
}

- (void) myMethod {
    MyObject *o = [[MyObject alloc] init];
    NSArray *a = [[NSArray alloc]
                  initWithObjects:o, nil];
    NSLog(@"%@", [a description]);
    NSLog(@"%@", [o description]);
    NSLog(@"myMethod exit");
}
```

```
@interface MyObject : NSObject
@end

@implementation MyObject
- (void)dealloc {
    NSLog(@"MyObject::dealloc");
}
@end
```

Output:

("<MyObject: 0x6831990>")
<MyObject: 0x685b3f0>
myMethod exit
MyObject::dealloc
after myMethod

…after If-Clause in Array

```
- (void) myCallerMethod {
    [self myMethod:TRUE];
    NSLog(@"after myMethod");
}

- (void) myMethod:(BOOL)condition {
    if (condition) {
        MyObject *o = [[MyObject alloc] init];
        NSArray *a = [[NSArray alloc]
                        initWithObjects:o, nil];
        NSLog(@"%@", [a description]);
    }
    NSLog(@"myMethod exit");
}
```

```
@interface MyObject : NSObject
@end

@implementation MyObject
- (void)dealloc {
    NSLog(@"MyObject::dealloc");
}
@end
```

Output:

```
( "<MyObject: 0x68748d0>" )
MyObject::dealloc
myMethod exit
after myMethod
viewDidLoad
```

Normal and Retained Returns

- ARC knows method naming conventions
  - first part of name (capitalization subdivides name parts)
- Transfer of ownership if first name part
  - alloc, init, copy, mutableCopy, new
  - returned objects are not autoreleased
  - “retained returns”
- Otherwise no transfer of ownership
  - returned objects are autoreleased
  - “normal returns”
  - @autoreleasepool { … } determines when autoreleased objects are deallocated

Normal (Autoreleased) Returns

- Code you write

```
- (NSString*) name {
    return myName;
}
```

- What the compiler adds

```
- (NSString*) name {
    return [[myName retain] autorelease];
}
```

Retained (Non-Autoreleased) Returns

- Code you write

```
- (NSString*) newName {
    return myName;
}
```

- What the compiler adds

```
- (NSString*) newName {
    return [myName retain];
}
```

Method name starts with “new” → transfer of ownership

Output of these programs?

```
@autoreleasepool {
    [self myMethod];
    NSLog(@"after myMethod");
}

- (MyObject*) myMethod {
    MyObject *s = [[MyObject alloc] init];
    NSLog(@"myMethod exit");
    return s;
}
```

autorelease pool emptied here

```
myMethod exit
after myMethod
MyObject::dealloc
```

```
@autoreleasepool {
    [self newMethod];
    NSLog(@"after newMethod");
}

- (MyObject*) newMethod {
    MyObject *s = [[MyObject alloc] init];
    NSLog(@"newMethod exit");
    return s;
}
```

```
newMethod exit
MyObject::dealloc
after newMethod
```

ARC Autoreleased Array

```
@interface MyObject : NSObject
@end
```

```
- (void) myCallerMethod {
    @autoreleasepool {
        [self myMethod];
        NSLog(@"after myMethod");
    }
}
- (void) myMethod {
    MyObject *o = [[MyObject alloc] init];
    NSArray *a = [NSArray arrayWithObject:o];
    NSLog(@"%@", [a description]);
    NSLog(@"%@", [o description]);
    NSLog(@"myMethod exit");
}
```

autorelease pool emptied here

```
@implementation MyObject
- (void)dealloc {
    NSLog(@"MyObject::dealloc");
}
@end
```

Output:

```
( "<MyObject: 0x6d3f5d0>" )
<MyObject: 0x6d3f5d0>
myMethod exit
after myMethod
MyObject::dealloc
```

Example: Weak Pointers

- Output of this program?

```
__weak MyObject *o = [[MyObject alloc] init];
NSLog(@"MyObject is: %@", [o description]);
```

- Output:

```
MyObject::dealloc
MyObject is: (null)
```

- Why?
  - Weak pointer does not keep object alive

```
@interface MyObject : NSObject
@end

@implementation MyObject
- (void)dealloc {
    NSLog(@"MyObject::dealloc");
}
@end
```