CS193P – Lecture 18

iPhone Application Development

Unit Testing
Fun with Objective-C
Localization
Mailbag

Unit Testing

What Are Unit Tests?

- Test specific areas of functionality
- Minimal external dependencies
- Run frequently during development

Who Writes Unit Tests?

- **You do!**
- Ideally written along with new code
- Test-driven development
  - Write tests first
  - Fill in the implementation until tests pass
  - Rinse & repeat

Running Unit Tests

- Automate so you don't have to explicitly run tests

- Many testing frameworks can run tests **every time you build**

- Just as compiler checks **syntax**, unit tests check **semantics**

Why Unit Test?

Why Unit Test?

- **Fewer bugs**
  - More confidence that you're shipping a high quality product

Why Unit Test?

- **Fewer bugs**
  - More confidence that you're shipping a high quality product

- Find bugs **early**
  - Bugs are easier (and cheaper) to fix early in development

Why Unit Test?

- **Fewer bugs**
  - More confidence that you're shipping a high quality product

- Find bugs **early**
  - Bugs are easier (and cheaper) to fix early in development

- Avoid **regressions**
  - Ensure that changing one piece of code doesn't break another

Why Unit Test?

- **Fewer bugs**
  - More confidence that you're shipping a high quality product

- Find bugs **early**
  - Bugs are easier (and cheaper) to fix early in development

- Avoid **regressions**
  - Ensure that changing one piece of code doesn't break another

- **Document** your code
  - How is a method intended to be used? Check out the tests...

Why Unit Test?

- **Fewer bugs**
  - More confidence that you're shipping a high quality product

- Find bugs **early**
  - Bugs are easier (and cheaper) to fix early in development

- Avoid **regressions**
  - Ensure that changing one piece of code doesn't break another

- **Document** your code
  - How is a method intended to be used? Check out the tests...

- Encourage **good design**
  - Spaghetti code is hard to test! Design with **testability** in mind

Unit Testing Frameworks

- Family of similar frameworks for testing various languages
    - JUnit, NUnit, PyUnit...

- **OCUnit** for Objective-C
    - Ships with Mac OS X developer tools, integrates with Xcode
    - **Included with iPhone SDK as of 2.2**

Basics of OCUnit

- SenTestCase is abstract test case superclass
- Automatically runs methods that begin with "test"
- Macros for asserting conditions during tests
  - `STAssertNotNil(someObject, @"Some object was nil");`
  - See SenTestCase.h for more
- -setUp and -tearDown methods run before and after each test

Defining A New Test Case Class

Defining A New Test Case Class

```objc
#import <SenTestingKit/SenTestingKit.h>

@class Foo;

@interface FooTests : SenTestCase {
  Foo *foo;
}
@end
```

Preparing Tests

Preparing Tests

```objc
@implementation FooTests

- (void)setUp {
    // Every test will have its own Foo instance
    foo = [[Foo alloc] init];
}

- (void)tearDown {
    [foo release];
}

...

@end
```

Adding Tests

Adding Tests

```objc
@implementation FooTests

...

- (void)testCreateFoo {
  STAssertNotNil(foo, @"Couldn't create Foo");
}

- (void)testSetBar {
  Bar *bar = ...;
  foo.bar = bar;
  STAssertEqualObjects(foo.bar, bar, @"Couldn't set foo.bar");
}

...

@end
```

Testing Error Conditions

Testing Error Conditions

```objc
@implementation FooTests

...

- (void)testOutOfBoundsAccess {
    STAssertNil([foo barAtIndex:99], @"Index 99 should be nil");
}

...

@end
```

Demo:
Unit Testing an iPhone App

When Does Unit Testing Make Sense?

When Does Unit Testing Make Sense?

- Always be conscious of the return on investment
  - Benefit of the test versus time to create and maintain?

When Does Unit Testing Make Sense?

- Always be conscious of the return on investment
  - Benefit of the test versus time to create and maintain?
- Some types of code are notoriously difficult to test
  - Networking
  - Databases
  - Often possible to **test a subset of behavior** and still benefit

Unit Testing Philosophy

Unit Testing Philosophy

- Keep tests short, lightweight, fast

Unit Testing Philosophy

- Keep tests short, lightweight, fast
- Test individual methods, not end-to-end behavior

Unit Testing Philosophy

- Keep tests short, lightweight, fast
- Test individual methods, not end-to-end behavior
- Find a new bug? Write a new test before you fix it

Unit Testing Philosophy

- Keep tests short, lightweight, fast
- Test individual methods, not end-to-end behavior
- Find a new bug? Write a new test before you fix it
- Complement (rather than replace) other types of tests
  - http://www.friday.com/bbum/2005/09/24/unit-testing/

Fun with Objective-C

The Objective-C Runtime

- How does OCUnit find all the methods that begin with "test"?
- Any other cool tricks?

/usr/include/objc

/usr/include/objc

- **<objc/objc.h>**
  - id, Nil, nil, BOOL, YES, NO

/usr/include/objc

- **\<objc/objc.h\>**
  - id, Nil, nil, BOOL, YES, NO

- **\<objc/message.h\>**
  - objc_msgSend() and friends

/usr/include/objc

- **<objc/objc.h>**
  - id, Nil, nil, BOOL, YES, NO

- **<objc/message.h>**
  - objc_msgSend() and friends

- **<objc/runtime.h>**
  - Inspect and manipulate classes, protocols, methods
  - **Add and replace methods at runtime**

Inspecting Methods

Inspecting Methods

- Copy all methods for a class

```
Method *class_copyMethodList(Class cls,
                            unsigned int *outCount);
```

Inspecting Methods

- Copy all methods for a class

```
Method *class_copyMethodList(Class cls,
                             unsigned int *outCount);
```

- Get attributes for a method

```
SEL method_getName(Method m);
IMP method_getImplementation(Method m);
char *method_copyReturnType(Method m);
...
```

Demo:
Inspecting Methods

Playing With Fire

Playing With Fire

- Adding a method to a class

```
BOOL class_addMethod(Class cls, SEL name, IMP imp,
                     const char *types);
```

Playing With Fire

- Adding a method to a class

```
BOOL class_addMethod(Class cls, SEL name, IMP imp,
                     const char *types);
```

- Replacing the implementation for a method

```
IMP method_setImplementation(Method method, IMP imp);
```

Playing With Fire

- Adding a method to a class

```
BOOL class_addMethod(Class cls, SEL name, IMP imp,
                     const char *types);
```

- Replacing the implementation for a method

```
IMP method_setImplementation(Method method, IMP imp);
```

- Method swizzling

```
void method_exchangeImplementations(Method m1, Method m2);
```

Method Swizzling

Method Swizzling

- What if you want to **override a method in a category** while still making use of the original method?

Method Swizzling

- What if you want to **override a method in a category** while still making use of the original method?
  - Can't call super, a category isn't a subclass

Method Swizzling

- What if you want to **override a method in a category** while still making use of the original method?
  - Can't call super, a category isn't a subclass
- Define a new method, swizzle it into place

Method Swizzling

- What if you want to **override a method in a category** while still making use of the original method?
  - ▪ Can't call super, a category isn't a subclass
- Define a new method, swizzle it into place

```
Method existingMethod = ...;
Method fancyNewMethod = ...;
method_exchangeImplementations(existingMethod, fancyNewMethod);
```

Method Swizzling

- What if you want to **override a method in a category** while still making use of the original method?
  - Can't call super, a category isn't a subclass

- Define a new method, swizzle it into place

```
Method existingMethod = ...;
Method fancyNewMethod = ...;
method_exchangeImplementations(existingMethod, fancyNewMethod);

- (void)fancyNewMethod
{
  // This looks like it will cause an infinite loop...
  // Once swizzled, it will actually invoke -existingMethod!
  [self fancyNewMethod];

  // Perform additional work here
}
```

Demo:
Method Swizzling

Why is this dangerous?

Why is this dangerous?

- Other code may be dependent on the original implementation
  - Perhaps code you didn't even write?

Why is this dangerous?

- Other code may be dependent on the original implementation
  - Perhaps code you didn't even write?
- Can cause unexpected behavior, bizarre bugs, crashes
  - This has caused some popular apps to break on iPhone OS 3.0

Why is this dangerous?

- Other code may be dependent on the original implementation
  - Perhaps code you didn't even write?
- Can cause unexpected behavior, bizarre bugs, crashes
  - This has caused some popular apps to break on iPhone OS 3.0
- Writing "clever" code is **fun until you have to debug it**

Why is this dangerous?

- Other code may be dependent on the original implementation
  - Perhaps code you didn't even write?
- Can cause unexpected behavior, bizarre bugs, crashes
  - This has caused some popular apps to break on iPhone OS 3.0
- Writing "clever" code is **fun until you have to debug it**
- **Never ship an app that swizzles methods on system classes**

Objective-C 2.0 Runtime Reference

• http://developer.apple.com/DOCUMENTATION/Cocoa/
Reference/ObjCRuntimeRef/Reference/reference.html

class-dump

class-dump

• Inspect the classes and methods of an Objective-C binary

class-dump

- Inspect the classes and methods of an Objective-C binary
- Fascinating to see how a complex application is architected
  - Especially one that you didn't write!

class-dump

- Inspect the classes and methods of an Objective-C binary
- Fascinating to see how a complex application is architected
  - Especially one that you didn't write!
- As usual, this can be used for evil purposes as well
  - Discover and use private methods in a framework

"Calling unpublished APIs is like jaywalking..."

"Calling unpublished APIs is like jaywalking across 280"

The Problem with Using Private APIs

The Problem with Using Private APIs

- Framework APIs are kept private for one of a few reasons:

The Problem with Using Private APIs

- Framework APIs are kept private for one of a few reasons:
  - They're not done yet (and **will probably change**)

The Problem with Using Private APIs

- Framework APIs are kept private for one of a few reasons:
  - They're not done yet (and **will probably change**)
  - They're never going to be public (and **may disappear**)

The Problem with Using Private APIs

- Framework APIs are kept private for one of a few reasons:
  - They're not done yet (and **will probably change**)
  - They're never going to be public (and **may disappear**)
- Not just because Apple wants to hide cool stuff from you

The Problem with Using Private APIs

- Framework APIs are kept private for one of a few reasons:
  - They're not done yet (and **will probably change**)
  - They're never going to be public (and **may disappear**)
- Not just because Apple wants to hide cool stuff from you
- If your app depends on a private API that goes away...

The Problem with Using Private APIs

- Framework APIs are kept private for one of a few reasons:
    - They're not done yet (and **will probably change**)
    - They're never going to be public (and **may disappear**)
- Not just because Apple wants to hide cool stuff from you
- If your app depends on a private API that goes away...
    - At best, your app won't work correctly anymore

The Problem with Using Private APIs

• Framework APIs are kept private for one of a few reasons:
  ▪ They're not done yet (and **will probably change**)
  ▪ They're never going to be public (and **may disappear**)

• Not just because Apple wants to hide cool stuff from you

• If your app depends on a private API that goes away...
  ▪ At best, your app won't work correctly anymore
  ▪ More often, your app will just crash

Localization

Your International Application

- Multiple languages and locales in a **single built application**
- Keep localized resources separate from everything else
    - Strings
    - Images
    - User interfaces (in NIBs)

Where Do Localized Resources Go?

- **MyApp.app/**
  - MyApp
  - **English.lproj/**
    - Localizable.strings
    - MyView.nib
  - **Japanese.lproj/**
    - Localizable.strings
    - MyView.nib

Two Steps

- Internationalization (i18n)
  - Prepare your app to be used in different languages and locales
- Localization (l10n)
  - Add localized data for specific languages and locales

NSString to the Rescue

- Interconverts with dozens of encodings

- Saves you from having to deal with complexities of text

- Remember encoding when reading data from disk or web

```
- (id)initWithData:(NSData *)data
       encoding:(NSStringEncoding)encoding;
```

Localized Strings

- For user-visible strings in your application code
- Map from an unlocalized key to a localized string
- Stored in .strings files
    - Key-value pairs
    - Use UTF-16 for encoding

Strings File Example

- **en.lproj/Greetings.strings**

      "Hello" = "Hello";

      "Welcome to %@" = "Welcome to %@";

      "Blah %@ blah %@!" = "Blah %@ blah %@";

- **fr.lproj/Greetings.strings**

      "Hello" = "Bonjour";

      "Welcome to %@" = "Bienvenue a %@";

      "Blah %@ blah %@" = "Blah %2$@ %1$@ blah";

Accessing Localized Strings

```
// By default, uses Localizable.strings
NSLocalizedString(@"Hello", @"Greeting for welcome screen");

// Specify a table, uses Greetings.strings
NSLocalizedStringFromTable(@"Hello", @"Greetings",
                           @"Greeting for welcome screen");
```

genstrings

- Tool to scan your code and produce a .strings file
- Inserts comments found in code as clues to localizer
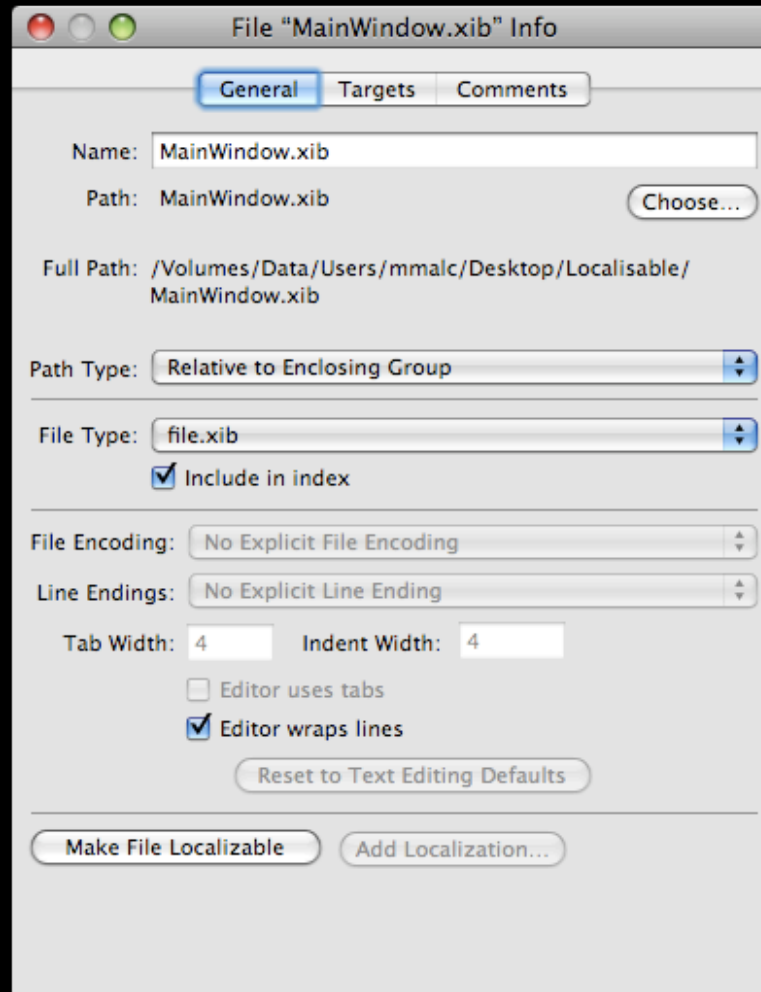- Run the tool over your *.m files

Other Resources

- NSBundle resource methods automatically use the best available localization
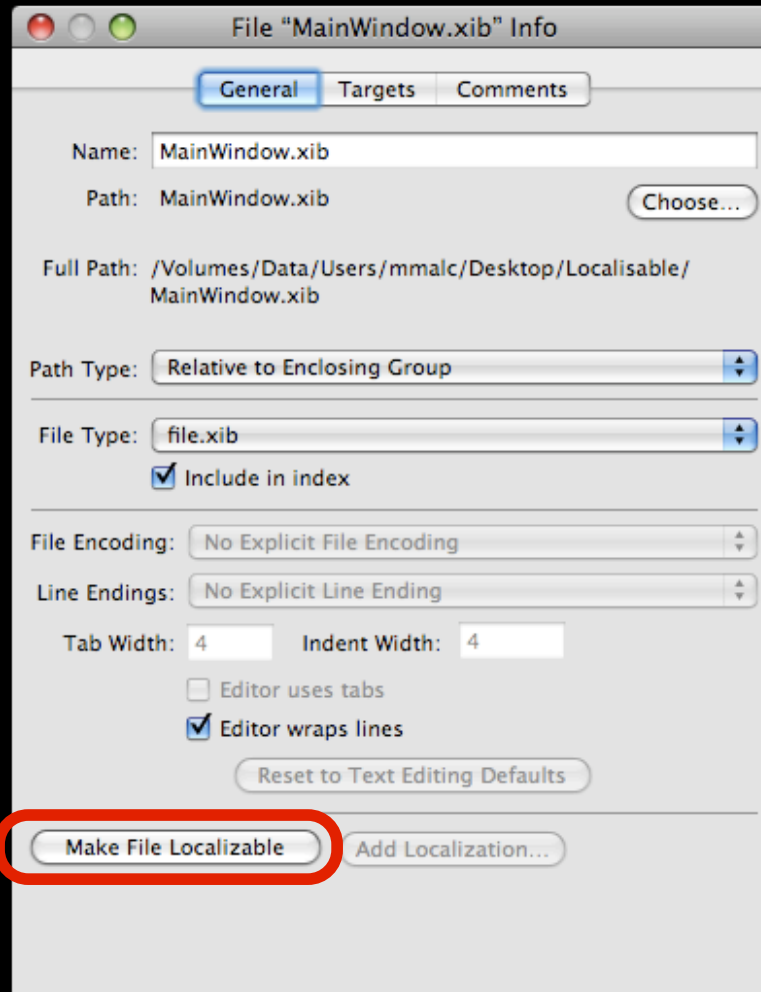
- Nib loading does the same

Internationalizing NIBs

- Plan for different string lengths in different languages
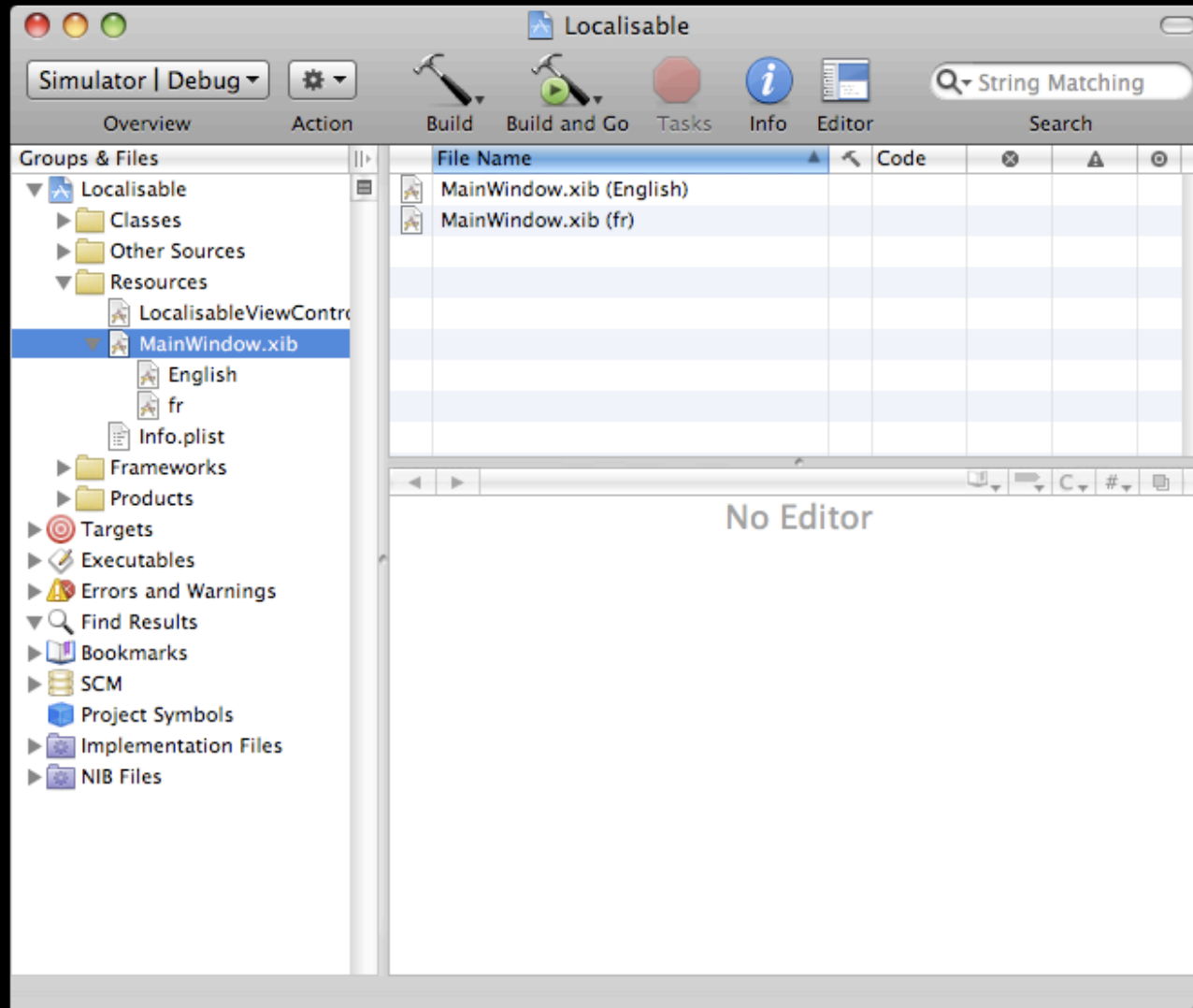 - Good idea to start with German...

Localizing a Resource

Localizing a Resource

Localizing a Resource

NSLocale

- Measurements
- Currency
- Number formatting
- Calendar and date format
- Country information

Opening the Mailbag...

"How do I launch my app in landscape orientation?"

Launching Your App in Landscape

Launching Your App in Landscape

- Two steps

Launching Your App in Landscape

- Two steps
  - Specify **initial interface orientation** in your Info.plist

Key	Value
▼ Information Property List	(13 items)
Initial interface orientation	Landscape (right home button)
Main nib file base name	MainWindow
Localization native development re	English

Launching Your App in Landscape

- Two steps
  - Specify **initial interface orientation** in your Info.plist

Key	Value
▼ Information Property List	(13 items)
Initial interface orientation	Landscape (right home button)
Main nib file base name	MainWindow
Localization native development re	English

  - **Support the specified orientation** in your view controller

Launching Your App in Landscape

- Two steps
  - Specify **initial interface orientation** in your Info.plist

Key	Value
▼ Information Property List	(13 items)
Initial interface orientation	Landscape (right home button)
Main nib file base name	MainWindow
Localization native development re	English

  - **Support the specified orientation** in your view controller
    - Override -shouldAutorotateToInterfaceOrientation:

Launching Your App in Landscape

- Two steps
  - Specify **initial interface orientation** in your Info.plist

Key	Value
▼ Information Property List	(13 items)
Initial interface orientation	Landscape (right home button)
Main nib file base name	MainWindow
Localization native development re	English

  - **Support the specified orientation** in your view controller
    - Override -shouldAutorotateToInterfaceOrientation:
    - Return YES to indicate interface orientations that you support

Launching Your App in Landscape

- Two steps
  - Specify **initial interface orientation** in your Info.plist

Key	Value
▼ Information Property List	(13 items)
Initial interface orientation	Landscape (right home button)
Main nib file base name	MainWindow
Localization native development re	English

  - **Support the specified orientation** in your view controller
    - Override -shouldAutorotateToInterfaceOrientation:
    - Return YES to indicate interface orientations that you support
- Works on iPhone OS 2.1 or later

Demo:
Launching in Landscape

"How can I customize UIKit views and controls?"

Customizing UIKit Views

Customizing UIKit Views

- Some classes are designed to be **totally customizable**
  - UIButton
  - UITableView

Customizing UIKit Views

- Some classes are designed to be **totally customizable**
  - UIButton
  - UITableView
- Many classes have **limited customizability**
  - UINavigationBar
  - UISlider

Customizing UIKit Views

- Some classes are designed to be **totally customizable**
  - UIButton
  - UITableView
- Many classes have **limited customizability**
  - UINavigationBar
  - UISlider
- Other classes are **not customizable**
  - UISwitch
  - UITabBar

What's Safe to Customize?

- Look for methods for customizing appearance
  - UIButton: background image
  - UINavigationBar: style, tint, translucency
  - UITableView: delegate methods for appearance
- You can always create your own UIView or UIControl subclass
  - Handle touches
  - Custom drawing

Respecting the View Hierarchy

- Internal view hierarchies are always subject to change
  - Navigation bar
  - Navigation and tab bar controllers
  - Image picker controller
- Making assumptions is unsafe and **will likely break your app**
  - Don't manipulate undocumented subviews of system views
  - Don't add your own custom subviews
- You want your application to be **future-ready**

"Should I create my views using Interface Builder or in code?"

When to Use Interface Builder

- Almost always recommended
- Especially useful when creating a view with many subviews
- Not as useful when dealing with just a single view
  - Table views
- Remember: **one view controller subclass, one NIB**
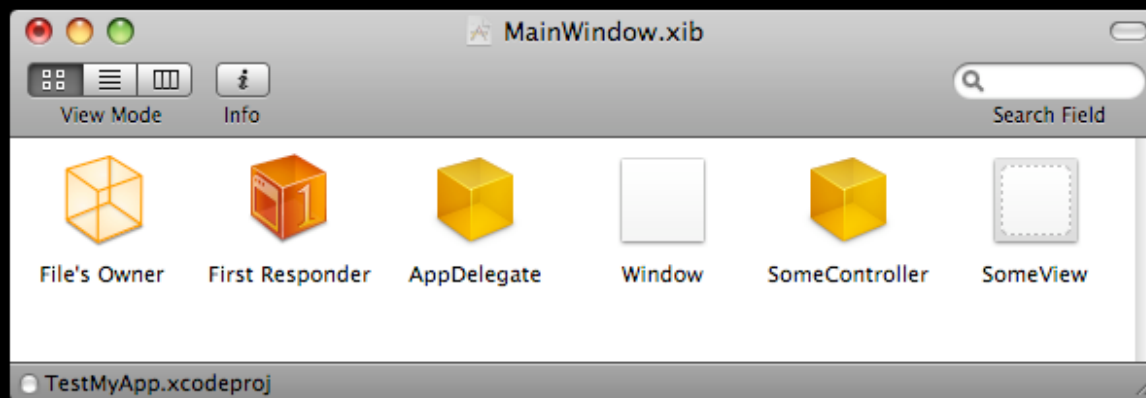 - Make connections to view controller via File's Owner
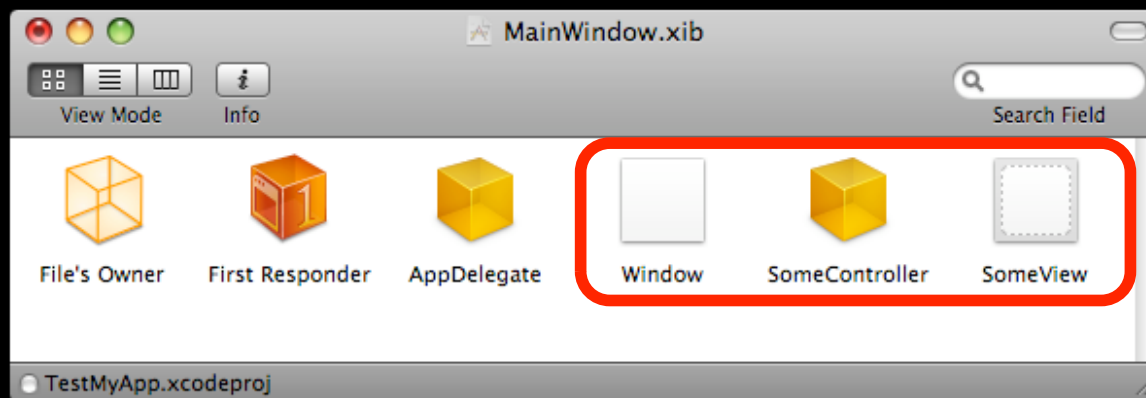
Nibs and Memory Management

Nibs and Memory Management

- Top-level nib objects are autoreleased
  - Retain them if they should stick around after loading

Nibs and Memory Management

- Top-level nib objects are autoreleased
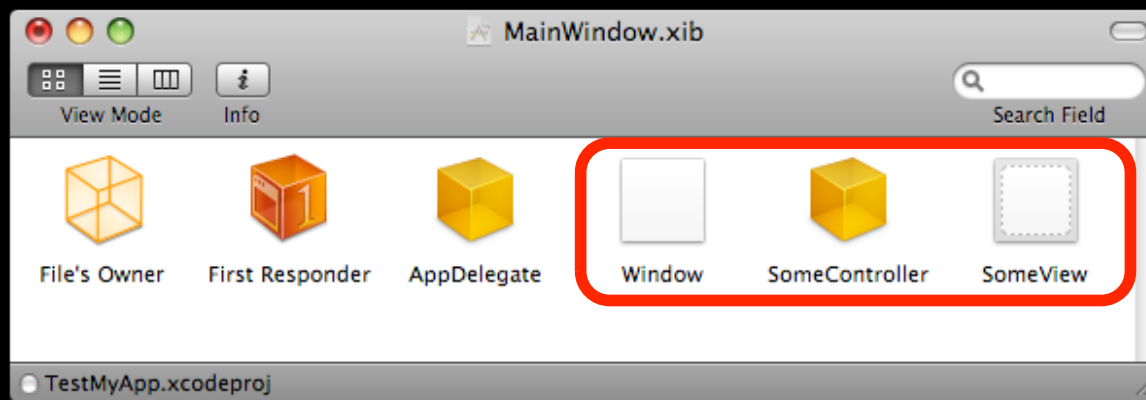 - Retain them if they should stick around after loading

Nibs and Memory Management

- Top-level nib objects are autoreleased
  - Retain them if they should stick around after loading

Nibs and Memory Management

- Top-level nib objects are autoreleased
  - Retain them if they should stick around after loading

- IBOutlets are **retained by default**
  - Release them in -dealloc even if you don't have a setter!
  - Implement or synthesize a non-retained setter if desired

"Where can I get cool icons for my toolbar or tab bar items?"

Toolbar and Tab Bar Images

- Images should be about 30 x 30 pixels
- PNG with alpha channel
- Used as a mask by UIKit for drawing with system colors

Some Great Free* Icons

- Glyphish, by Joseph Wain: http://www.glyphish.com
- Creative Commons License
  - *Free to use, share, or remix with attribution

Some Great Free* Icons

- Glyphish, by Joseph Wain: http://www.glyphish.com
- Creative Commons License
  - *Free to use, share, or remix with attribution

"Can I build Mac apps now that I know iPhone development?"

Building iPhone & Mac OS X Apps

Building iPhone & Mac OS X Apps

- At this point, you're familiar with
  - Objective-C language
  - Cocoa Touch frameworks
  - Object-oriented design patterns
  - Interface Builder and NIBs

Building iPhone & Mac OS X Apps

- At this point, you're familiar with
    - Objective-C language
    - Cocoa Touch frameworks
    - Object-oriented design patterns
    - Interface Builder and NIBs
- Developing for the Mac desktop is within your reach!
    - First few lectures of CS193P and CS193E are identical

Building iPhone & Mac OS X Apps

- At this point, you're familiar with
  - Objective-C language
  - Cocoa Touch frameworks
  - Object-oriented design patterns
  - Interface Builder and NIBs
- Developing for the Mac desktop is within your reach!
  - First few lectures of CS193P and CS193E are identical
- Big difference is Cocoa vs. Cocoa Touch
  - Many UIKit and AppKit classes are similar