HIT3329 / HIT8329
Creating Data Driven Mobile Applications

Lecture 4
UIKit

Presented by Paul Chapman

Adjunct Industry Fellow F-ICT
Director, Long Weekend LLC
Swinburne University of Technology

SWINBURNE UNIVERSITY OF TECHNOLOGY

Last Lecture Reviewed

1. Protocols
2. Delegates & Datasources
3. Interface Builder Connections
4. Application Object
5. Collections
6. File IO

Questions?

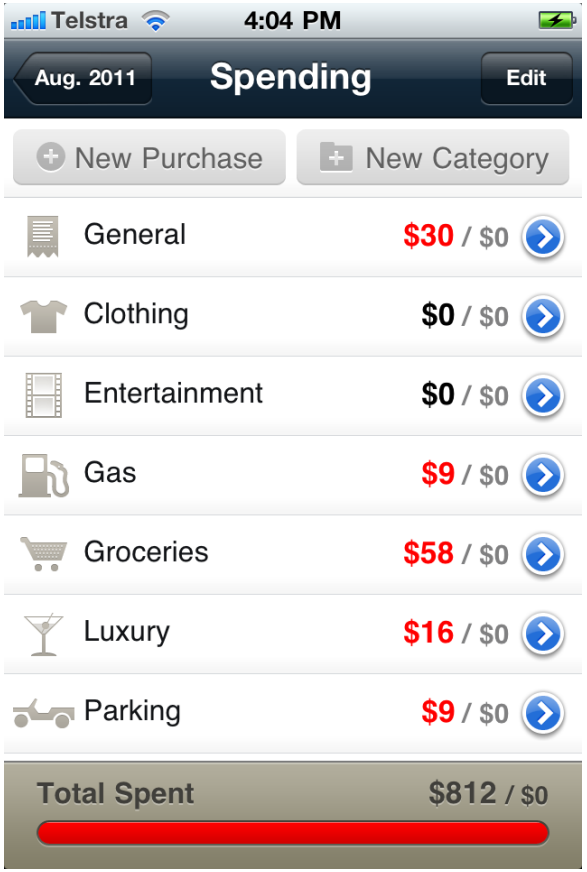
What is UIKit?

- UIKit is the Apple framework that draws "views" to the screen in iOS applications
- Today we learn how to draw views, and how to navigate between them

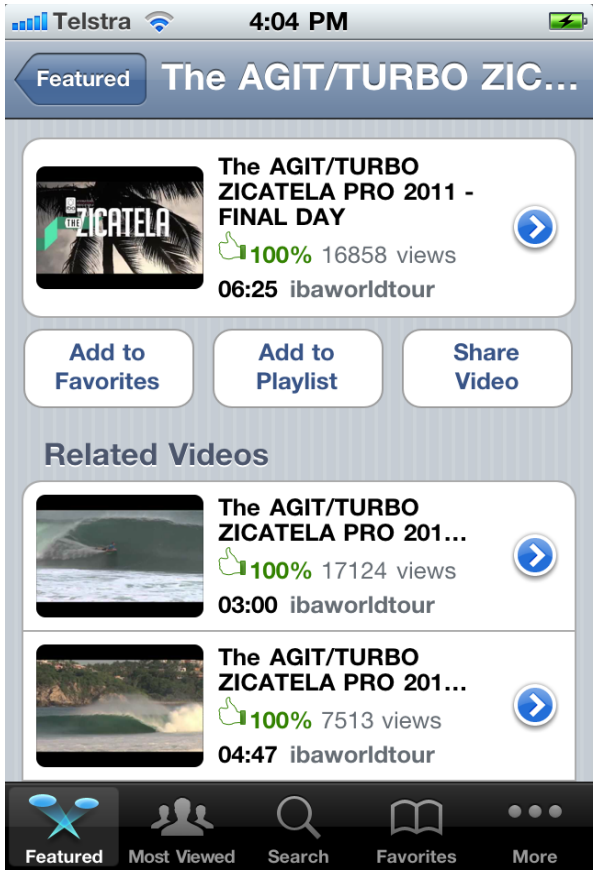
What's On For Today?

1. Views & View Controllers
2. Table Views
3. Navigation Controllers
4. Tab Bar Controllers
5. Unit Testing

1.0 What is a UIViewController?



(Back In Black)
Table View Controller



(Youtube)
Table View Controller

(App Store)
Custom View

Controller

1.1 View Controllers

UIViewController objects control how their contained UIView objects are displayed and behave.

- Subclassed from UIViewController
- VC responsible for all *views* in given *view hierarchy*

- Common parts of a VC sub class
  - Member properties (data store, view objects)
  - Action methods for responding to input
  - Any additional methods to control VC behaviour

- **iPhone**: single view hierarchy covers whole screen
- **iPad**: single view hierarchy covers part / whole screen

1.2 What is a UIView?

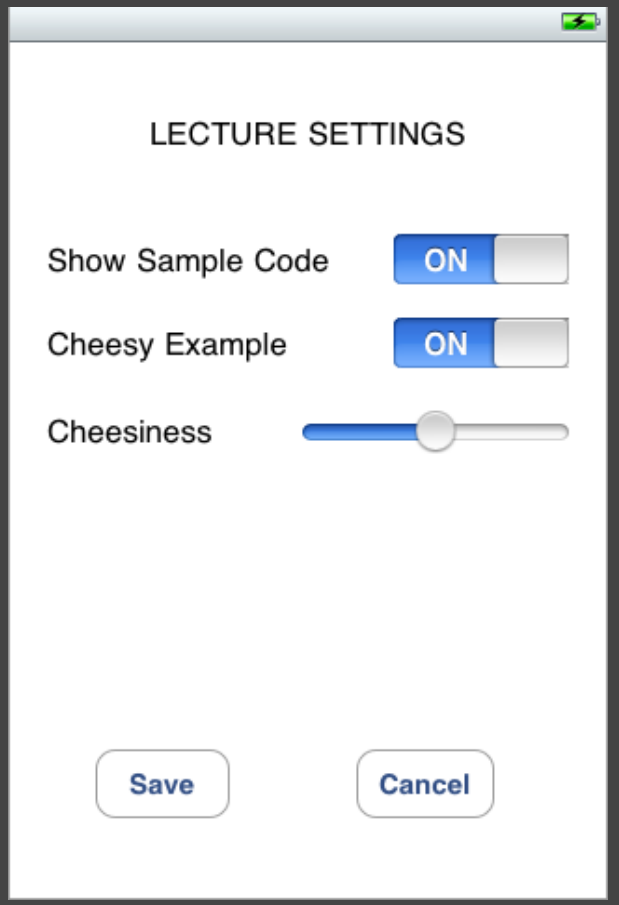
- A UIView object defines rectangular regions of the screen that can be drawn to
- It doesn't contain program logic or program flow - just your user interface

1.3 View Controller Interactivity

Q: Who's responsibility is it to set the *"Cheesiness"* slider to disabled when *"Cheesy Example"* is set to "OFF"??

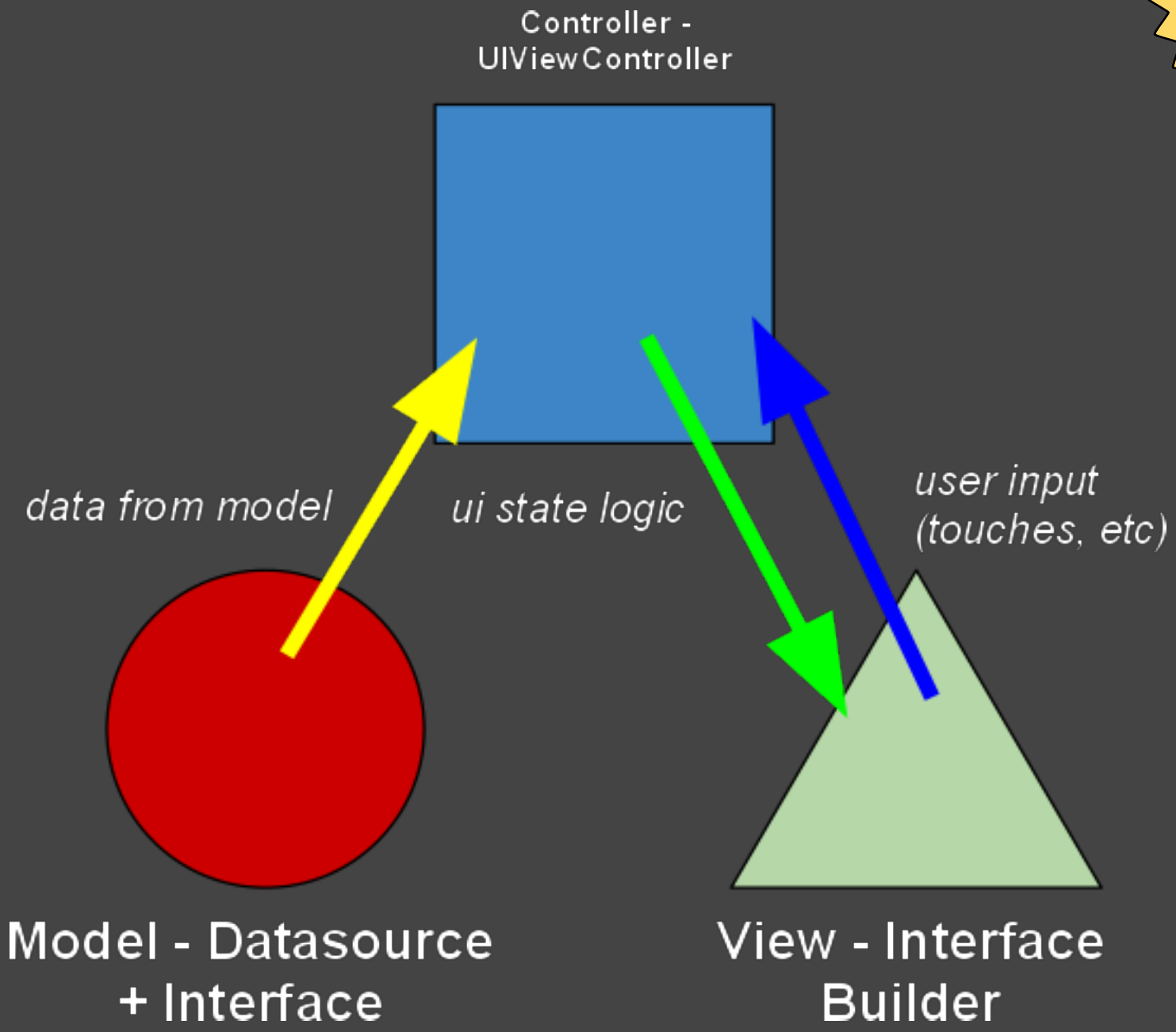
A: Not the UIView instances but the controller that controls them.

The UIViewController, or one of its subclasses, is the *"C"* in *"MVC"* ("Model-View-Controller" pattern).



1.4 Review Model-View-Controller

Design Pattern



1.5 Types of View Controllers

1. Custom VC
    - Subclasses UIViewController
    - One Custom VC for each "screen" in your app
    - Usually one XIB (Interface Builder) file per VC
    - VC handles the "unfreezing" of a XIB into UIViews

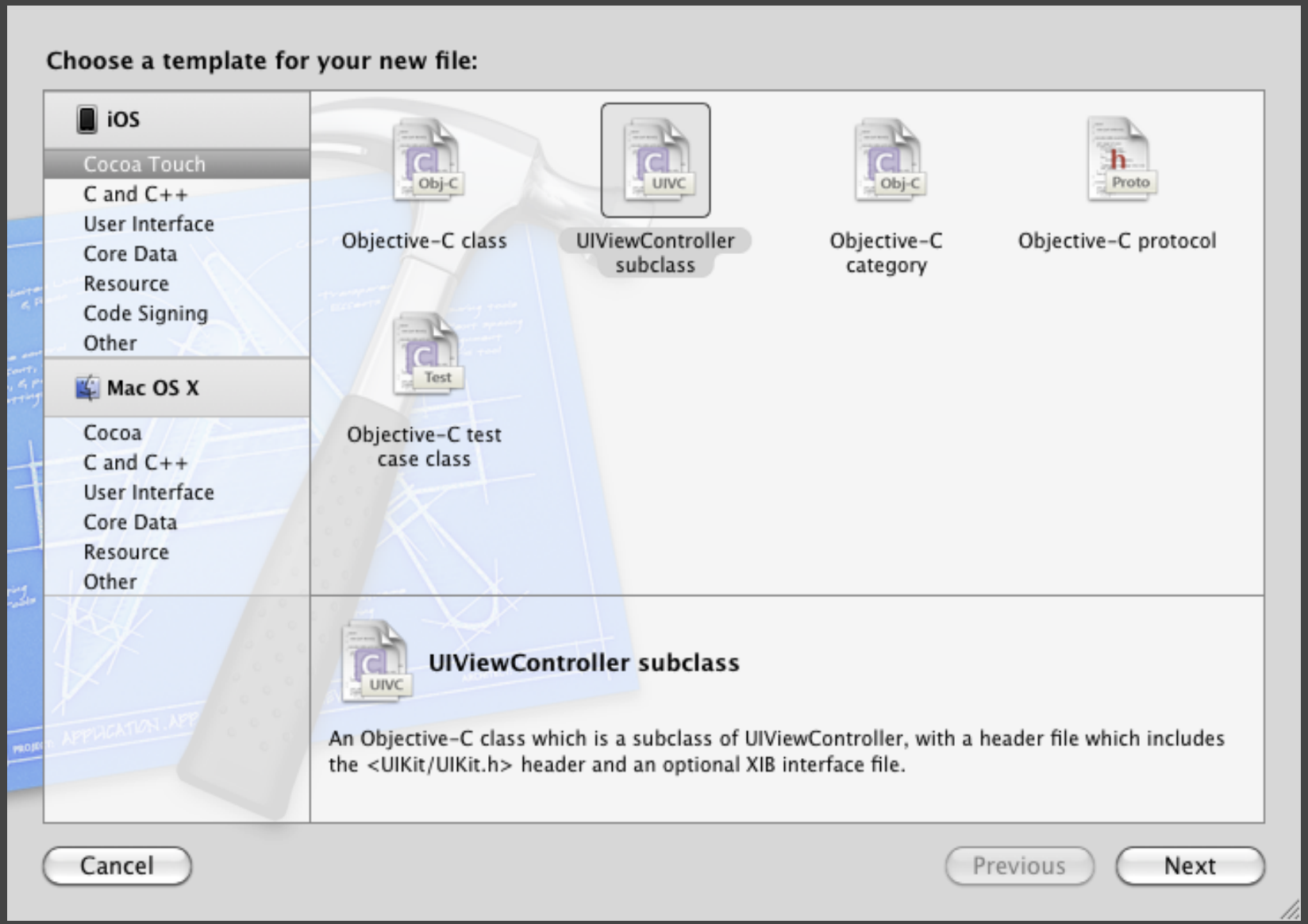
2. Navigation VC - more on these later!
    - UINavigationController
    - UITabBarController
    - UISplitViewController

3. Modal VC - a view presented modally

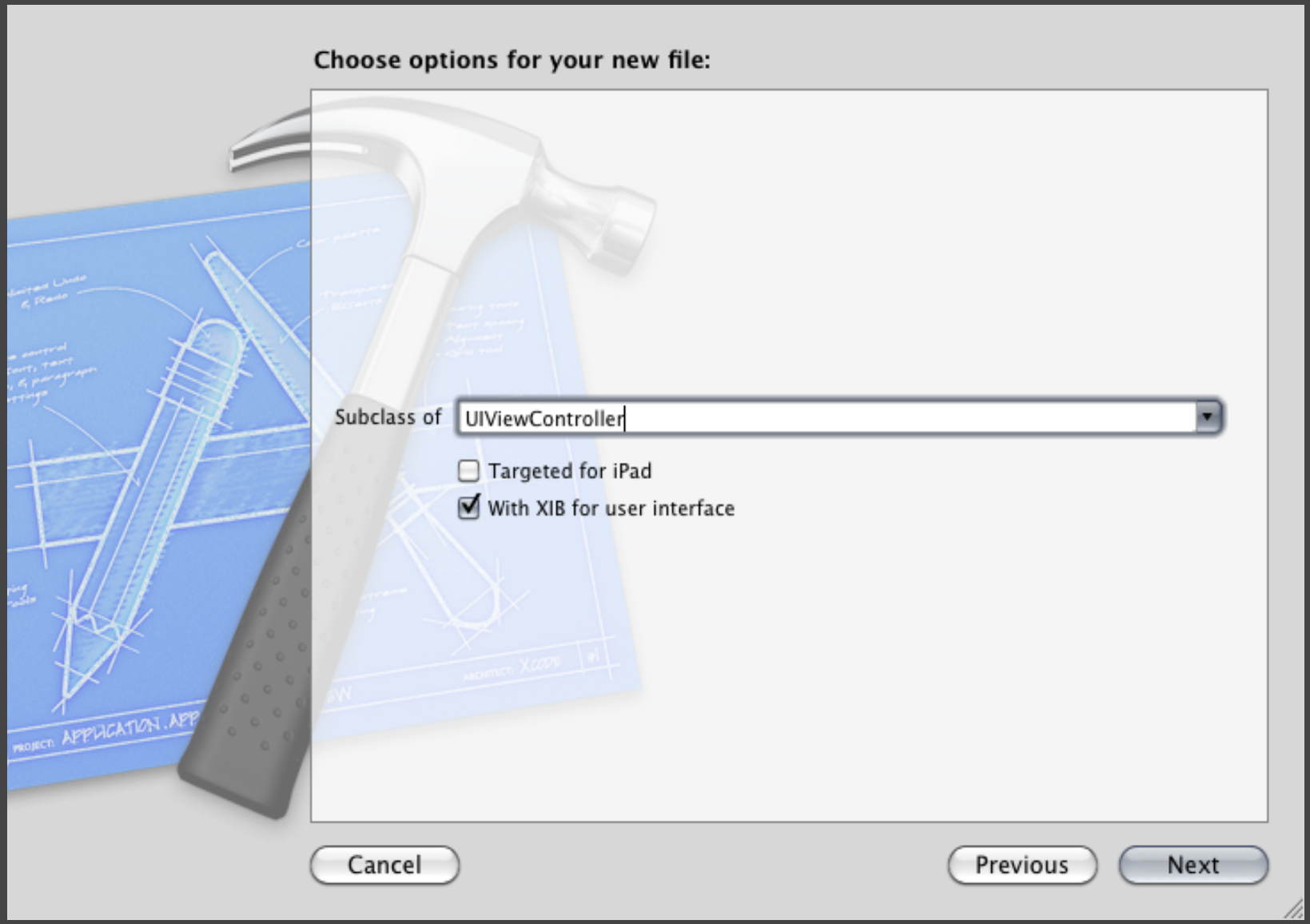
1.6 Creating a Custom VC - Step 1

Create Class Files

1. Menu: *Xcode* > *File* > *New File*
2. Select *Cocoa Touch* > *UIViewController subclass*
   press *Next*
3. Select "With XIB for user interface" box,
   press *Next*
4. Name It + Save It

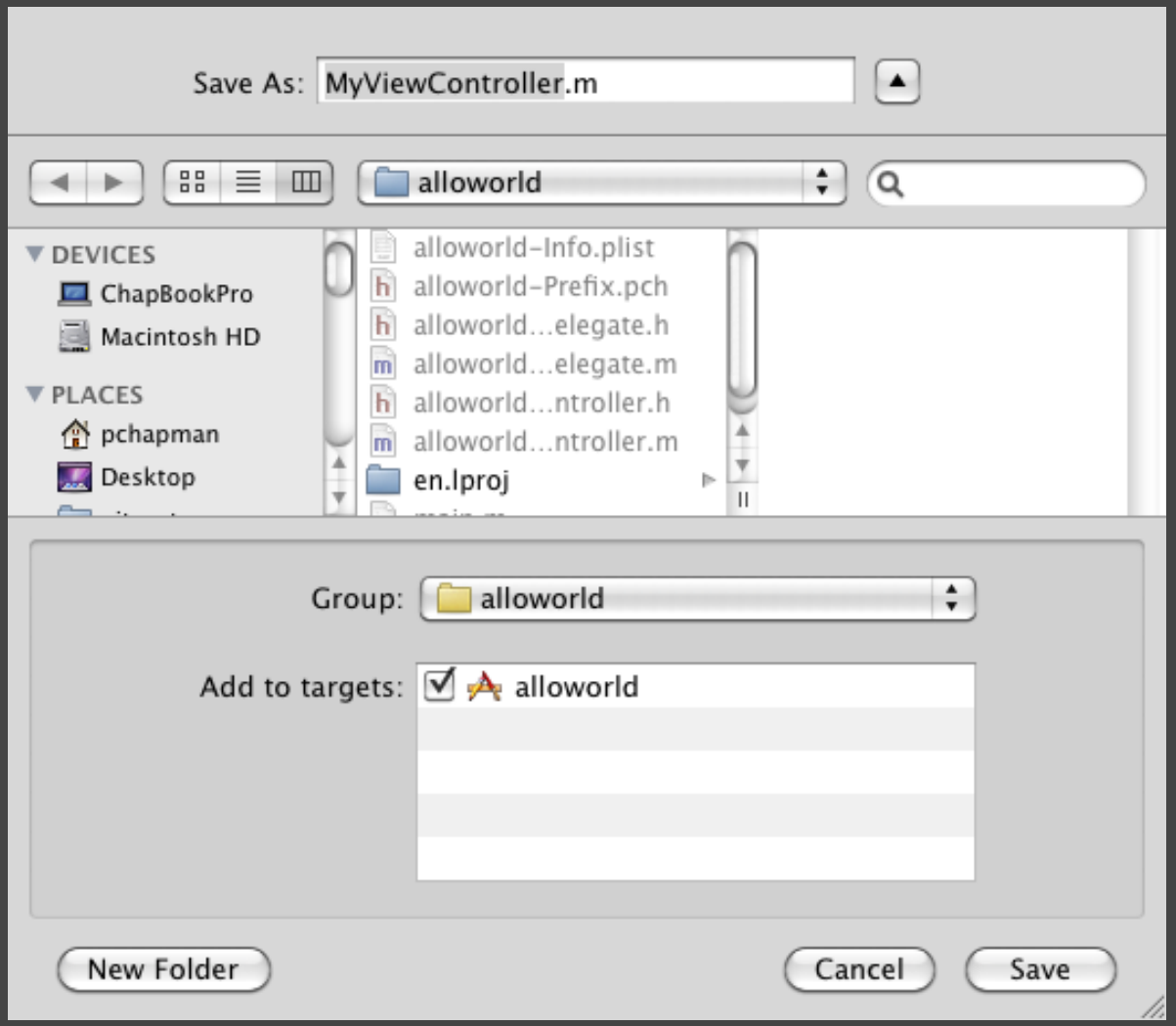


1.6.1 Create UIViewController Class



1.6.2 New Class Options

Swinburne University of Technology



1.6.3 Save File As

1.7 Creating a Custom VC - Step 2

Create View Controller Instance In App Delegate

1. alloc / initWithNibName
2. self.rootViewController = myVCInst;
3. Don't forget [myVCInst release];

1.8 Creating a Custom VC - Step 2 (Code)

```
//
// Inside your app delegate's .m file
//
- (BOOL)application:(UIApplication *)application
     didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
  MyViewController *myVCInst = [[MyViewController alloc]
              initWithNibName:@"MyViewController" bundle:nil];

  self.rootViewController = myVCInst;
  [myVCInst release];
  [window makeKeyAndVisible];
  return YES;
}
```

1.9 View Controller Classes

- UIView
  - UINavigationBar
  - UITabBar
- UIViewController
  - UITableViewController
  - UITabBarController
  - UISplitViewController
  - UIImagePickerController

Related:

- UIBarItem > UIButtonBarItem > UITabBarItem
- UINavigationItem
- UIPopOverController
- UIResponder

1.10 Custom VC: App Delegate Header

```
// MyAppDelegate.h

#import <UIKit/UIKit.h>
@class RootViewController; // forward class declaration
@interface MyAppDelegate : NSObject <UIApplicationDelegate>

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) RootViewController *rootVC;

@end
```

1.11 Custom VC: App Delegate Implementation

```
// MyAppDelegate.m

#import "MyAppDelegate.h"
#import "RootViewController.h";
@implementation MyAppDelegate
@synthesize window, rootVC;

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
  RootViewController *rootVC = [[RootViewController alloc]
              initWithNibName:@"RootViewController" bundle:nil];
  self.rootViewController = rootVC;
  [window makeKeyAndVisible];
  return YES;
}

// ... continued on next slide
```

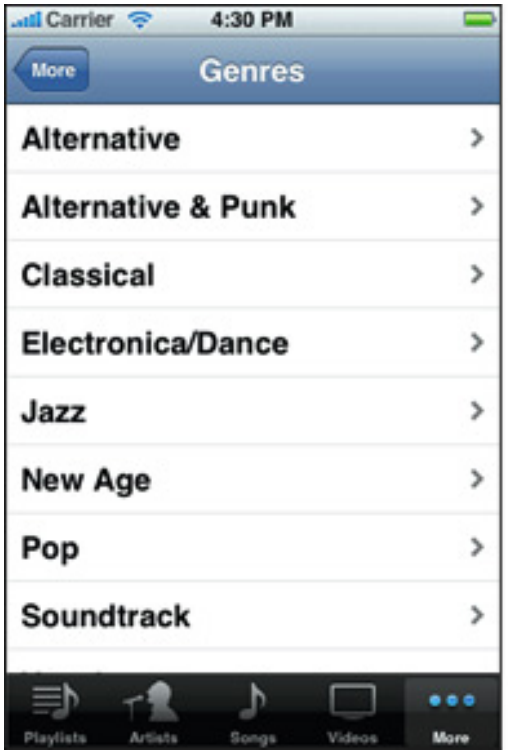
1.12 Custom VC: App Delegate (Continued)

```
// MyAppDelegate.m (continued)

- (void)dealloc {
    [rootVC release];
    [window release];
    [super dealloc];
}

@end
```

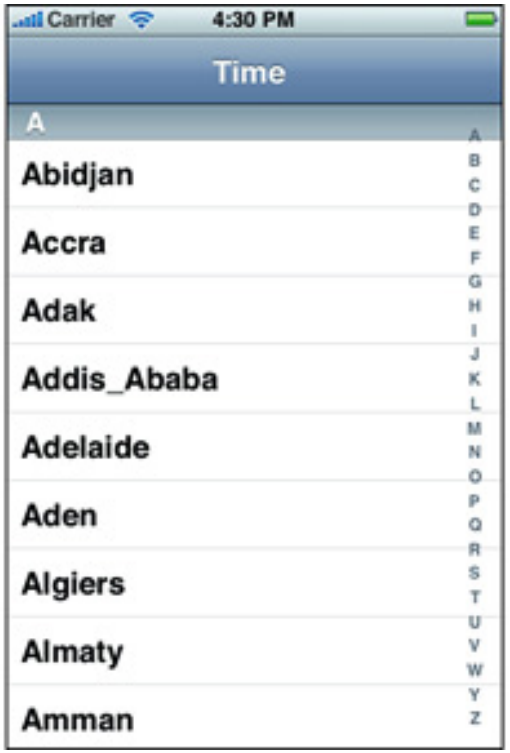
What's On For Today?

1. Views & View Controllers
2. **Table Views**
3. Navigation Controllers
4. Tab Bar Controllers
5. Unit Testing

2.0 What is a UITableViewController?



Standard



Sections + Index

Grouped

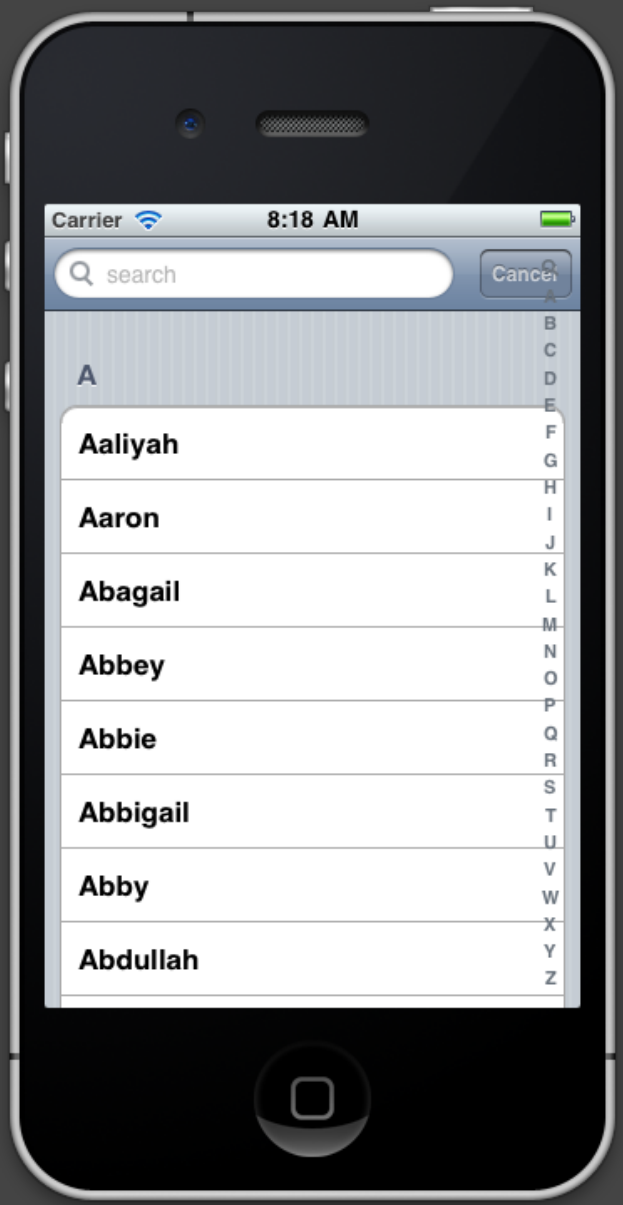
Source: http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/TableView_iPhone/AboutTableViewsiPhone/AboutTableViewsiPhone.html#//apple_ref/doc/uid/TP40007451

2.1 Table View Controllers

- Provides table-like behaviours:
  - Scrollable lists longer than screen
  - Table headers and footers
  - Section titles
  - Cell (row) styles
  - Indexes, grouping & sections
  - Row selection & editing
- Instantiate UITableView object

2.2 Fly Weight Pattern

Design Pattern



When top most cell scrolls off screen, cell object returned to queue of unused cells

Cell Objects In use

When next row of data is needed, VC requests a cell object from the cell queue

Queue of unused cell (table row) objects

dequeue cell

2.3 Creating a Table View Controller

1. Style UITableView object (in code or IB)
2. Set *data source* and *delegate*
3. Data Source conforms
   to UITableViewDataSource protocol
4. Delegate conforms
   to UITableViewDelegate protocol

NB: Uses *delegate & fly weight patterns*

2.4 Table VC: Header File

View controller conforms to the UITableViewDelegate and UITableViewDataSource protocols.

```
// FavAppTableViewContoller.h

@interface FavAppTableViewContoller : UIViewController
            <UITableViewDelegate, UITableViewDataSource> {}
@property (nonatomic, retain) NSArray *favAppsArray;
@end
```

2.5 Table VC: Implementation File

```
// FavAppTableViewContoller.m (PARTIAL)

@synthesize favAppsArray;

- (void)loadView
{
    UITableView *tableView = [[UITableView alloc] initWithFrame:
    [[UIScreen mainScreen] applicationFrame] style:
    UITableViewStylePlain];

    tableView.delegate = self;
    tableView.dataSource = self;
    [tableView reloadData];

    self.view = tableView;
    [tableView release];
}
```

2.6 Table VC: Populating Data I

Methods defined by the protocol are delegated responsibility to provide data for constructing the table.

```
// Return the number of sections
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
}

// Return the number of rows for each section
- (NSInteger)tableView:(UITableView *)tableView
     numberOfRowsInSection:(NSInteger)section
{
    return [favAppNames count];
}
```

2.7 Table VC: Populating Data II

```
- (UITableViewCell *)tableView:(UITableView *)tableView
          cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // Dequeue an available cell object for re-use
    // Fly-weight pattern: re-use small number of cell objects
    static NSString *MyIdentifier = @"MyCustomCellID";

    UITableViewCell *cell = [tableView
              dequeueReusableCellWithIdentifier:MyIdentifier];

    // Create a cell object if we didn't get an available one
    if (cell == nil) {
       cell = [[[UITableViewCell alloc]
              initWithStyle:UITableViewCellStyleDefault
              reuseIdentifier:MyIdentifier]
              autorelease];
    }

// ... continued on next slide
```

2.8 Table VC: Populating Data II (continued)

Method tablewView:cellForRowAtIndexPath: continued...

```
    // Decorate the cell's label
    cell.textLabel.text = [self.favAppsArray
                           objectAtIndex:indexPath.row];

    // Return the requested table cell object
    return cell;
}
```

2.9 Table VC: Populating Data III

Make sure we load the source data, for example, from a Property List XML file (PLIST).

```
// Called once when view loads
- (void)viewDidLoad {
   [super viewDidLoad]; // call super class' viewDidLoad

   // init empty array
   self.favAppsArray = [NSMutableArray array];

   // load the plist into an array
   NSString *dataFilePath = [[NSBundle mainBundle]
            pathForResource:@"fav-app-names" ofType:@"plist"];

   if (dataFilePath) {
     self.favAppsArray=[NSArray arrayWithContentsOfFile:thePath];
   }
}
```

2.10 Table VC: Other Options

- Custom header for the table
- Custom header for a section
- Custom indentation for each row
- Variable row heights or styling

Also See: http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/TableView_iPhone/CreateConfigureTableView/CreateConfigureTableView.html#//apple_ref/doc/uid/TP40007451-CH6-SW10

2.11 Fly Weight Pattern - Formal Definition

"A flyweight is an object that minimizes memory use by sharing as much data as possible with similar objects"

"... it is a way to use objects in large numbers when a simple repeated representation would use an unacceptable amount of memory"

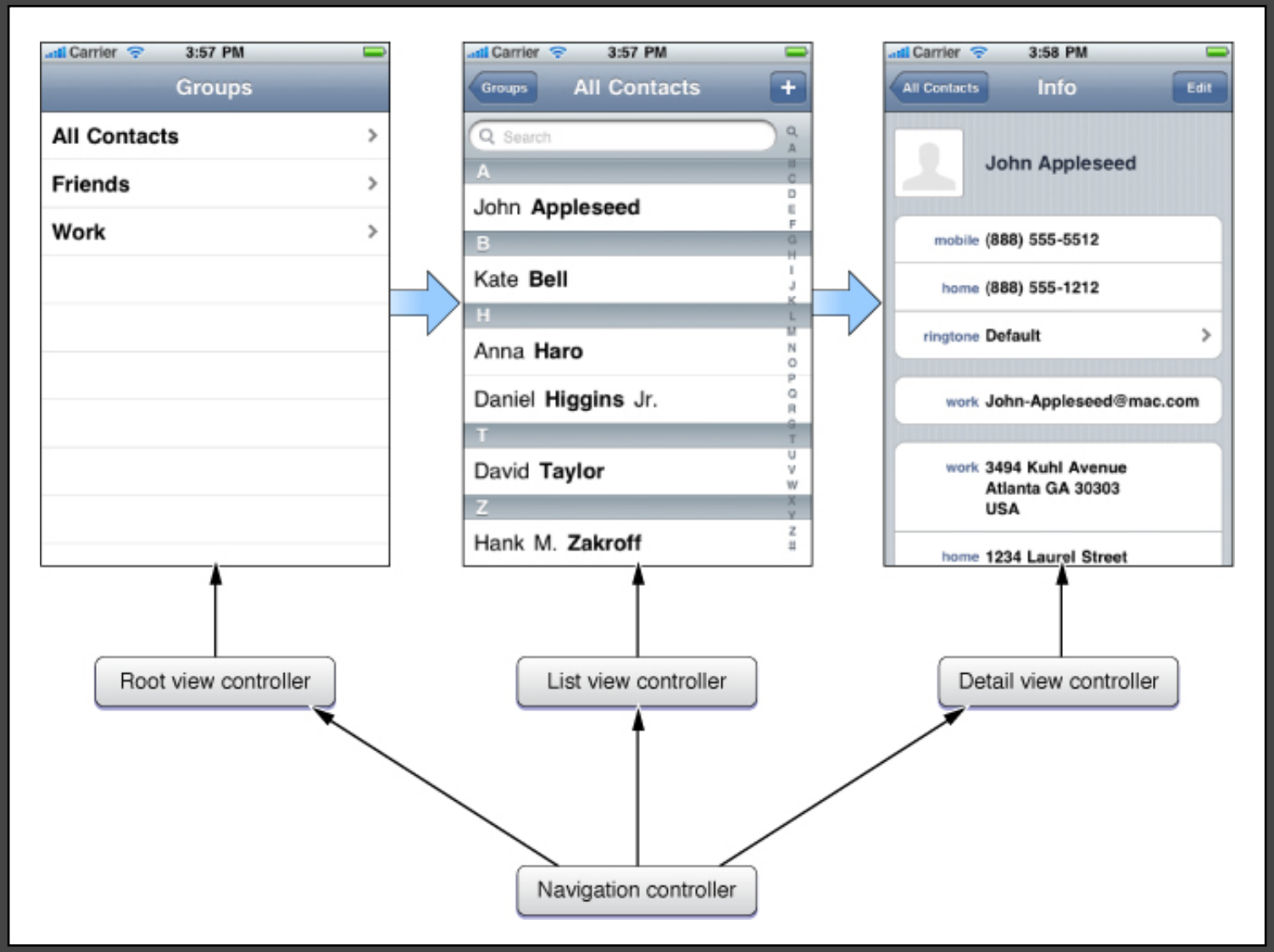
"The term is named after the boxing weight class."

Source: http://en.wikipedia.org/wiki/Flyweight_pattern

What's On For Today?

1. Views & View Controllers
2. Table Views
3. **Navigation Controllers**
4. Tab Bar Controllers
5. Unit Testing

3.0 What is a UINavigationController?



Source: http://developer.apple.com/library/ios/#featuredarticles/ViewControllerPGforiPhoneOS/AboutViewControllers/AboutViewControllers.html

3.1 Navigation Controllers

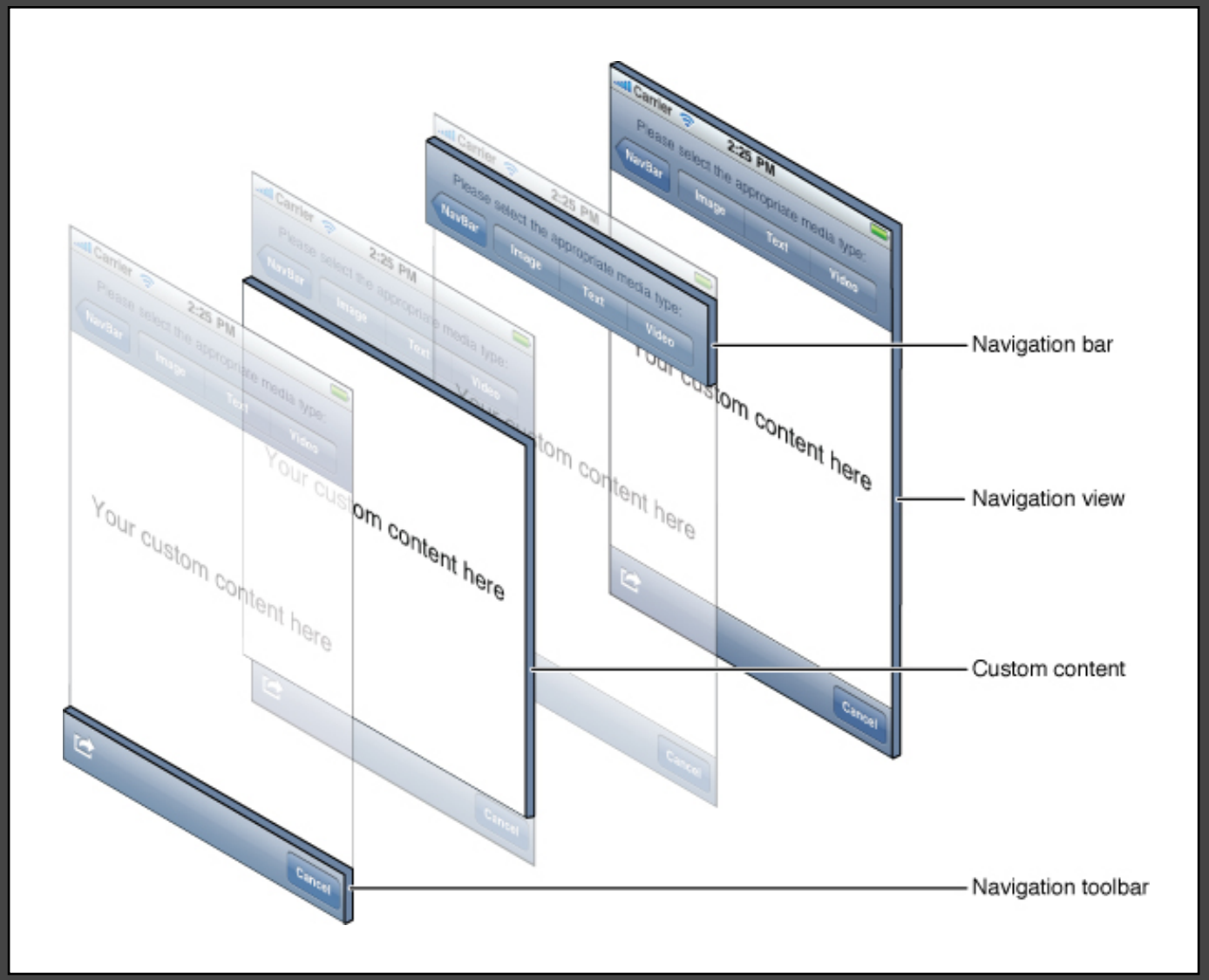
- Used to represent screens hierarchically
- Supports optional:
  - toolbar
  - title label
  - top back/left/right button(s)

- *Don't subclass UINavigationController*

3.2 The Navigation Stack

- A *Navigation Controller* manages a LIFO "stack" of *View Controllers*
- The stack represents user's path through the navigation hierarchy
- The **bottom** of stack is the starting point
- The **top** of stack is the current position

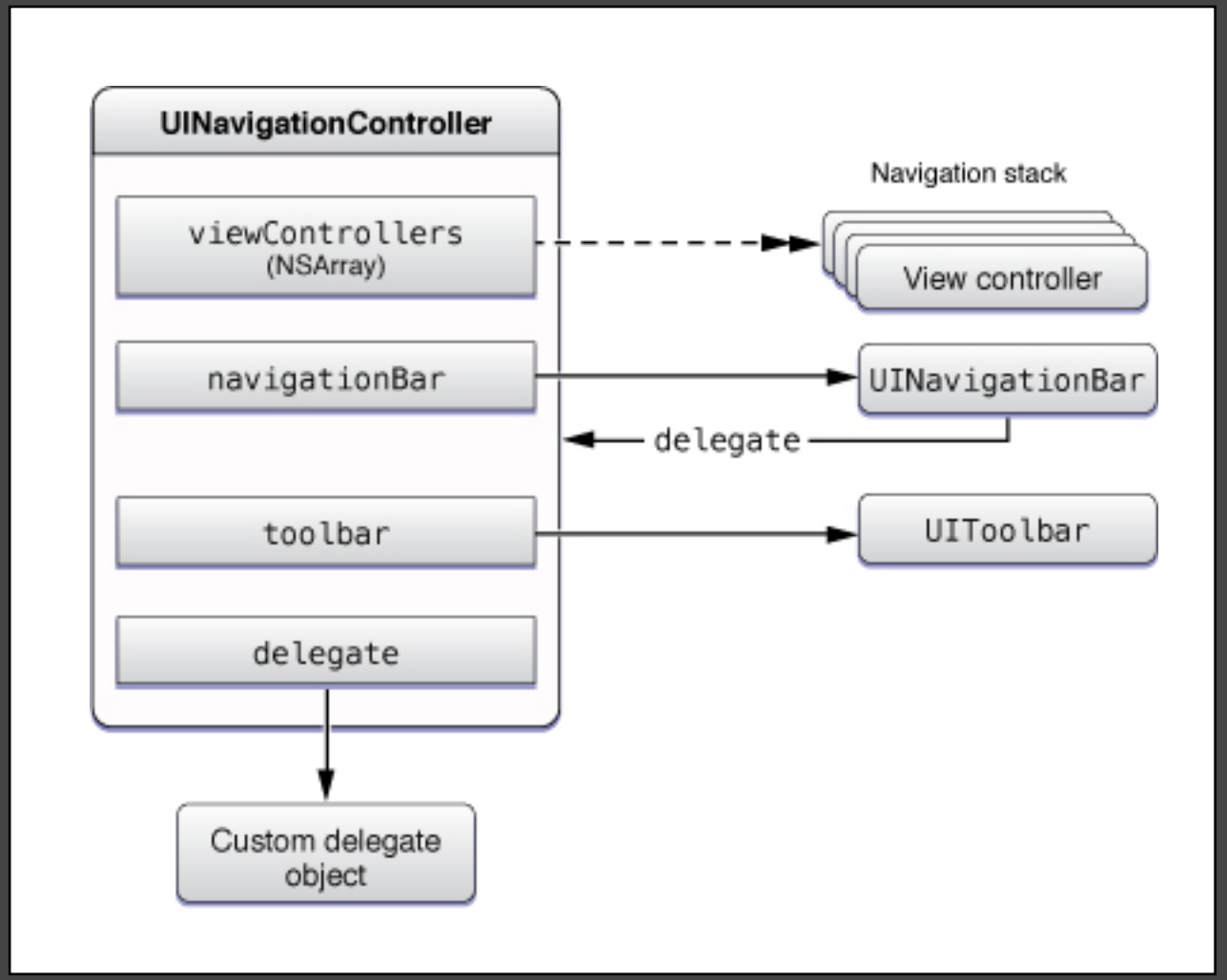
3.3 Navigation Controller in 3D



Source: http://developer.apple.com/library/ios/#featuredarticles/ViewControllerPGforiPhoneOS/NavigationControllers/NavigationControllers.html

3.4 Navigation Controller Objects

Provided by NC object



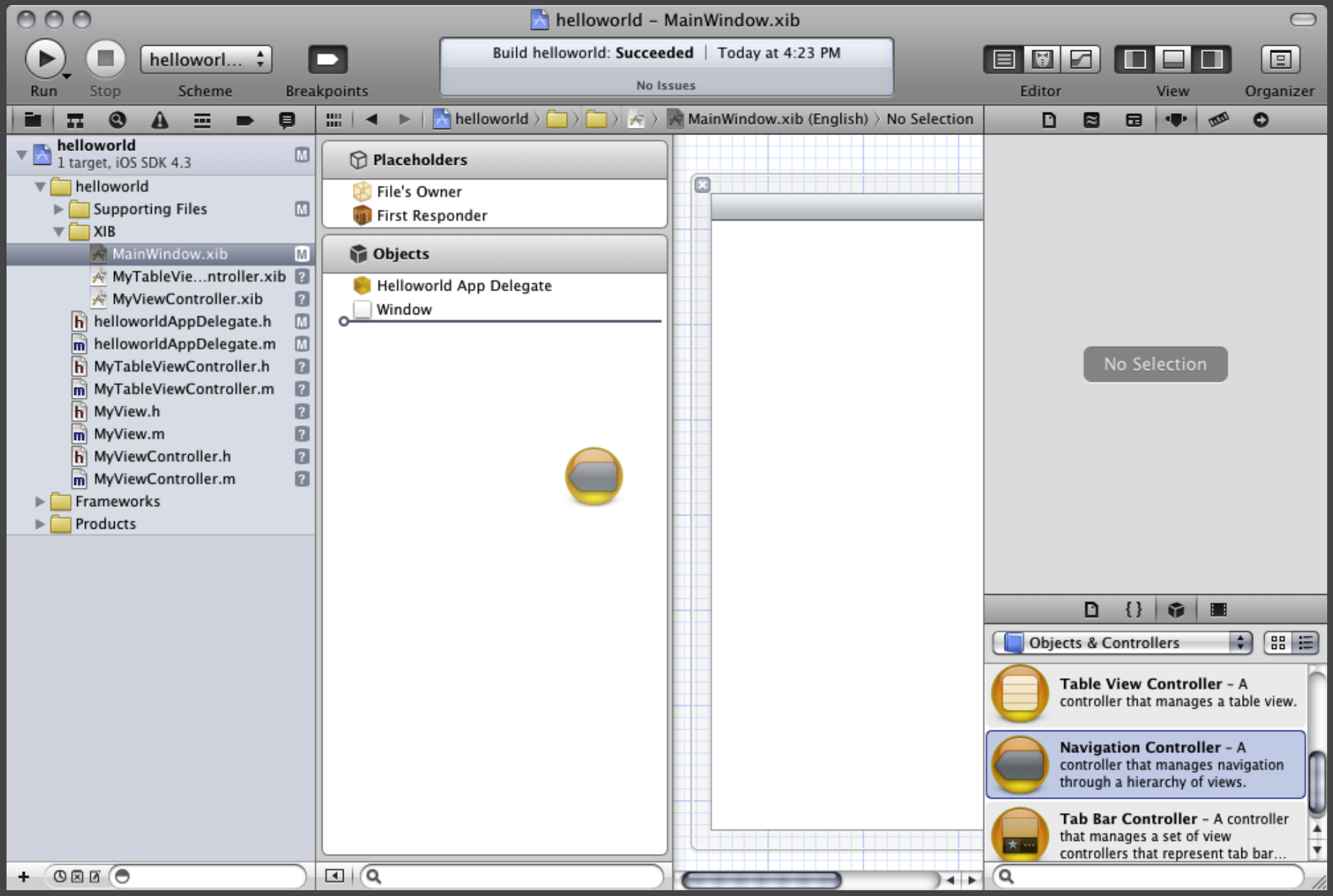
Provided by your code

Provided by your code

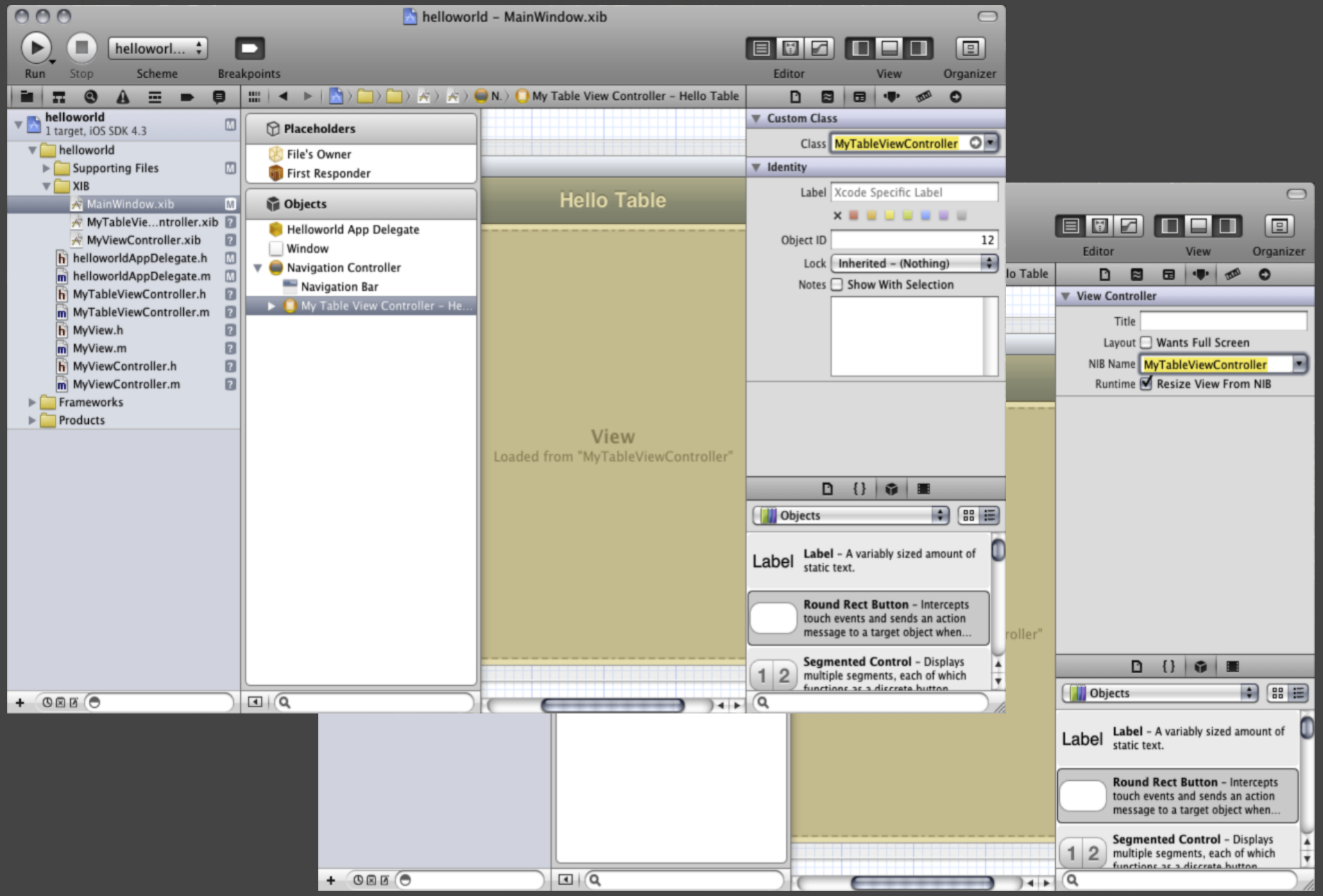
Source: http://developer.apple.com/library/ios/#featuredarticles/ViewControllerPGforiPhoneOS/NavigationControllers/NavigationControllers.html

3.5 Creating a Navigation Controller with IB

1. Add UINavigationController object to XIB file
2. Set *Class* and *NIB* properties of UINavigationController object's top-most *View Controller object*
3. In your class header, add IBOutlet declared property for the UINavigationController object
4. Connect the IBOutlet to the UINavigationController object using Interface Builder
5. In your implementation file (.m), add the navigation controller's view as a subview



3.5.1 Add UINavigationController object to XIB

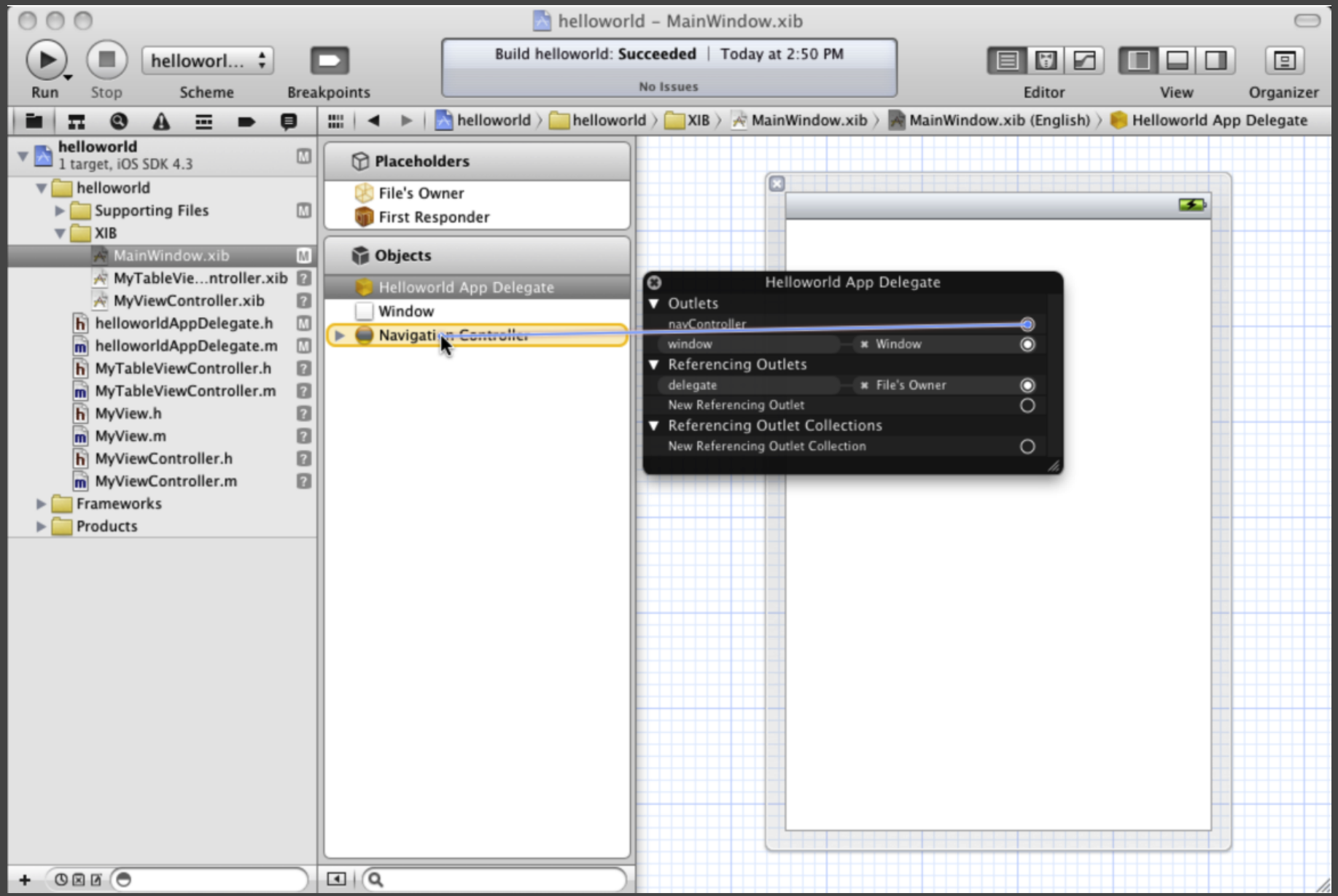


3.5.2 Set Class & NIB File Properties

Swinburne University of Technology

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

@interface MyAppDelegate : NSObject <UIApplicationDelegate>

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet UINavigationController
                                        *navController;
@end
```

3.5.3 Add IBOutlet for Nav Controller



3.5.4 Connect IBOutlet

Swinburne University of Technology

```
//
// MyAppDelegate.m - snippet
//
@implementation MyAppDelegate
@synthesize window=_window, navController;

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
  [_window addSubview:navController.view];
  [_window makeKeyAndVisible];
  return YES;
}

// other methods ... //
@end
```

3.5.5 Add Nav Controller's View as Subview

3.6 Adding a Navigation Controller

- Navigation Controllers can be added to:
  - the App Delegate
  - a View Controller

3.7 Navigation Root View Controller

- Each Navigation Controller stack has a *root view controller*
- A navigation controller's root view controller (RVC) is the start of a navigation hierarchy

3.8 Displaying a Navigation Controller

- To display a Navigation Controller:
  - Add it as a subview to the *Window, or*
  - Display modally via *Root View Controller*

3.9 Create a Modal Navigation Controller

```
// create a view controller to push
UIViewController *tmpVC =
  [[UIViewController alloc] initWithNibName:@"ModalAlertVC"];

// create a nav controller to manage it
UINavigationController *modalNavCon =
[[UINavigationController alloc] initWithRootViewController:tmpVC];

// Get pointer to our App Delegate
OurAppDelegate *appDel=[[UIApplication sharedApplication] delegate];

// Use app delegate's RVC to display it modally
[appDel.rootViewController presentModalViewController:
                        modalNavController animated:YES];
// relinquish ownership
[modalNavCon release];
[tmpVC release];
```

3.10 Manipulating the Stack

```
// Add a new VC to the stack (e.g. go forward)
[self.navCon pushViewController:myViewCon animated:YES];

//Remove the topmost VC (e.g. go back)
[self.navCon popViewControllerAnimated:YES];

//Jump to bottom-most VC (e.g. back to start)
[self.navCon popToRootViewControllerAnimated:YES];
```

3.11 Manipulating the Stack (continued)

pushViewController:animated:
Display the next level of hierarchical data

popViewControllerAnimated:
Back up one level in the hierarchy

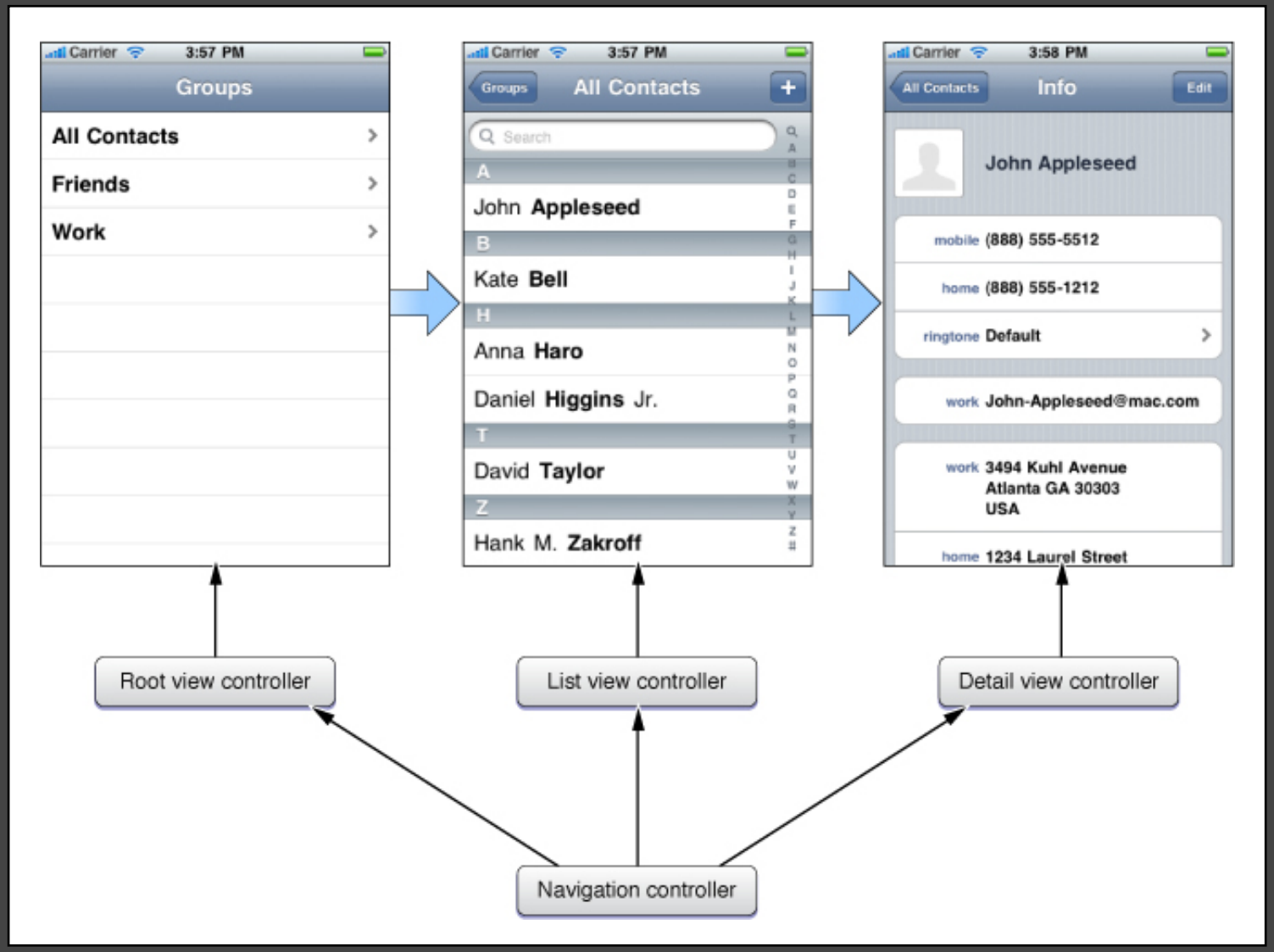
popToRootViewControllerAnimated:
Return the user to the root view controller

setViewControllers:animated:
Restore the navigation stack to a previous state

popToViewController:animated:
Back up an arbitrary number of levels in the hierarchy

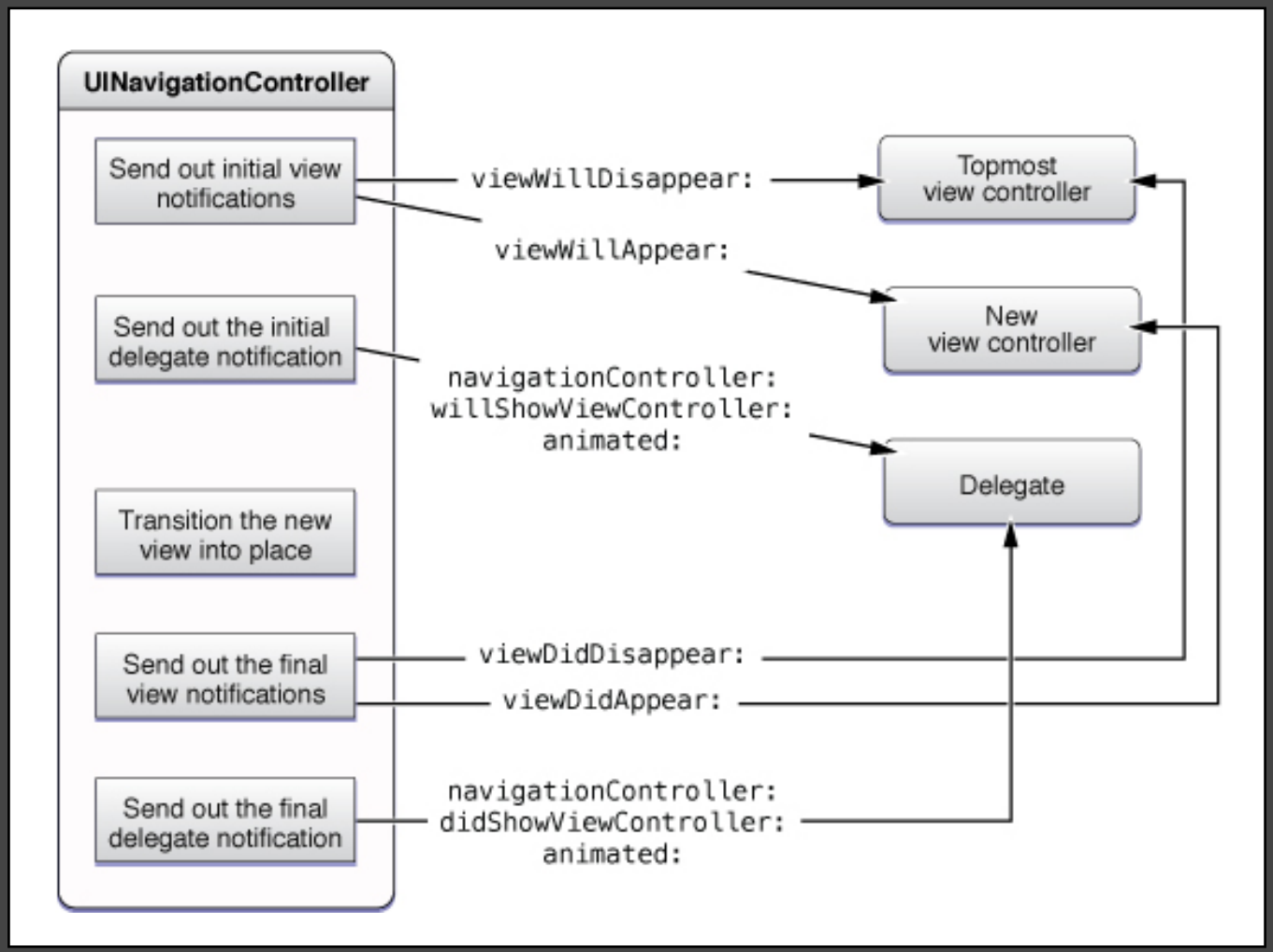
setViewControllers:animated:
Jump to an arbitrary location in your data hierarchy

3.12 Review the Navigation Stack



Source: http://developer.apple.com/library/ios/#featuredarticles/ViewControllerPGforiPhoneOS/AboutViewControllers/AboutViewControllers.html

3.13 Observing the Stack



Source: http://developer.apple.com/library/ios/#featuredarticles/ViewControllerPGforiPhoneOS/NavigationControllers/NavigationControllers.html

3.14 Nav Controller Delegate Methods

The UINavigationController object sends messages to its delegate, allowing you to respond to life cycle events.

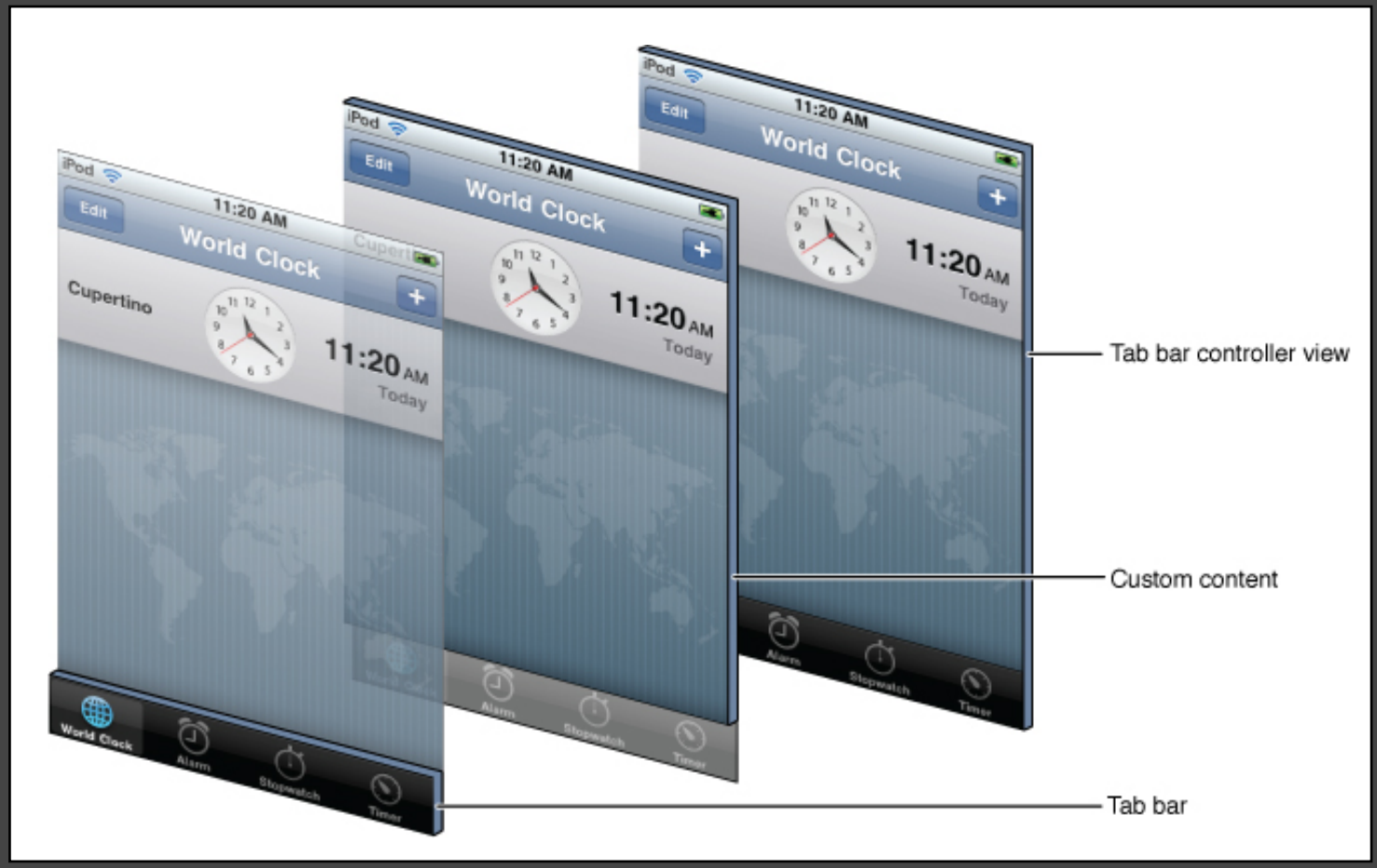
For Example:

```
// Called when NC is about to display
navigationController:willShowViewController:animated:

// Called directly after NC appears
navigationController:didShowViewController:animated:
```

What's On For Today?

1. Views & View Controllers
2. Table Views
3. Navigation Controllers
4. **Tab Bar Controllers**
5. Unit Testing

4.0 What is a UITabBarController?



Source: http://developer.apple.com/library/ios/#featuredarticles/ViewControllerPGforiPhoneOS/TabBarControllers/TabBarControllers.html

4.0 Tab Bar Controllers

- A tab bar controller provides a *tab bar interface* along the bottom of the screen
- Each tab has a different:
  - navigation path / navigation controller
  - root view controller
- Tab bar controller has an optional delegate

4.1 Placing a Tab Bar Controller

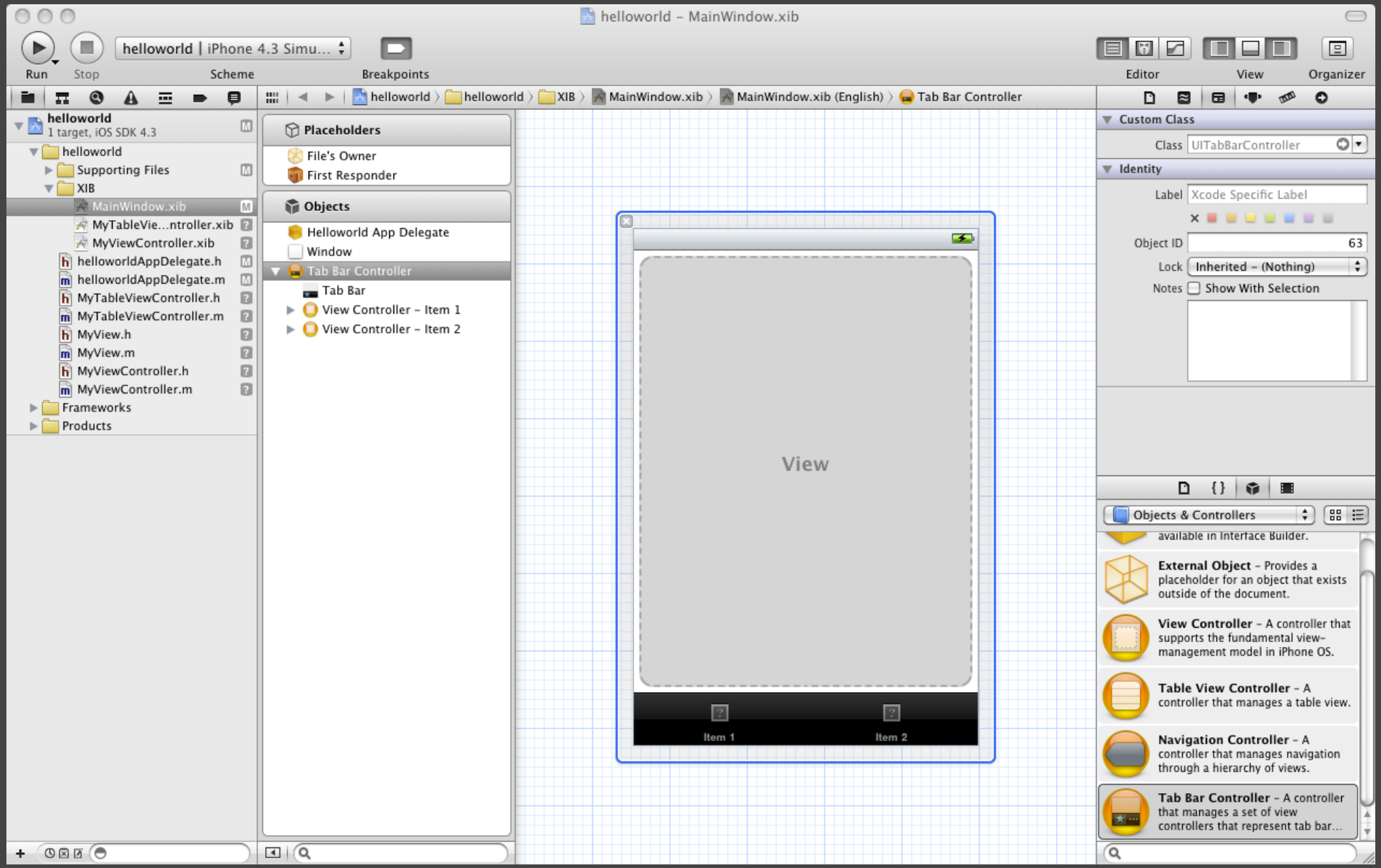
A *Tab Bar Controller* can only be added:

- To the MainWindow
- Modally
- To a Split View Interface (iPad)
- To a Popover Control (iPad)

Why does this limitation exist?

4.2 Creating a Tab Bar Controller with IB

1. Add Tab Bar Controller to *MainWindow.xib* (note, it automatically adds 1x tab bar, 2x root VC)
2. Add a tab bar controller IBOutlet to your App Delegate's header file (.h)
3. Connect the outlet to UITabBarController object in IB
4. Customise your view controllers as needed
5. Add tab bar controller's view to the window

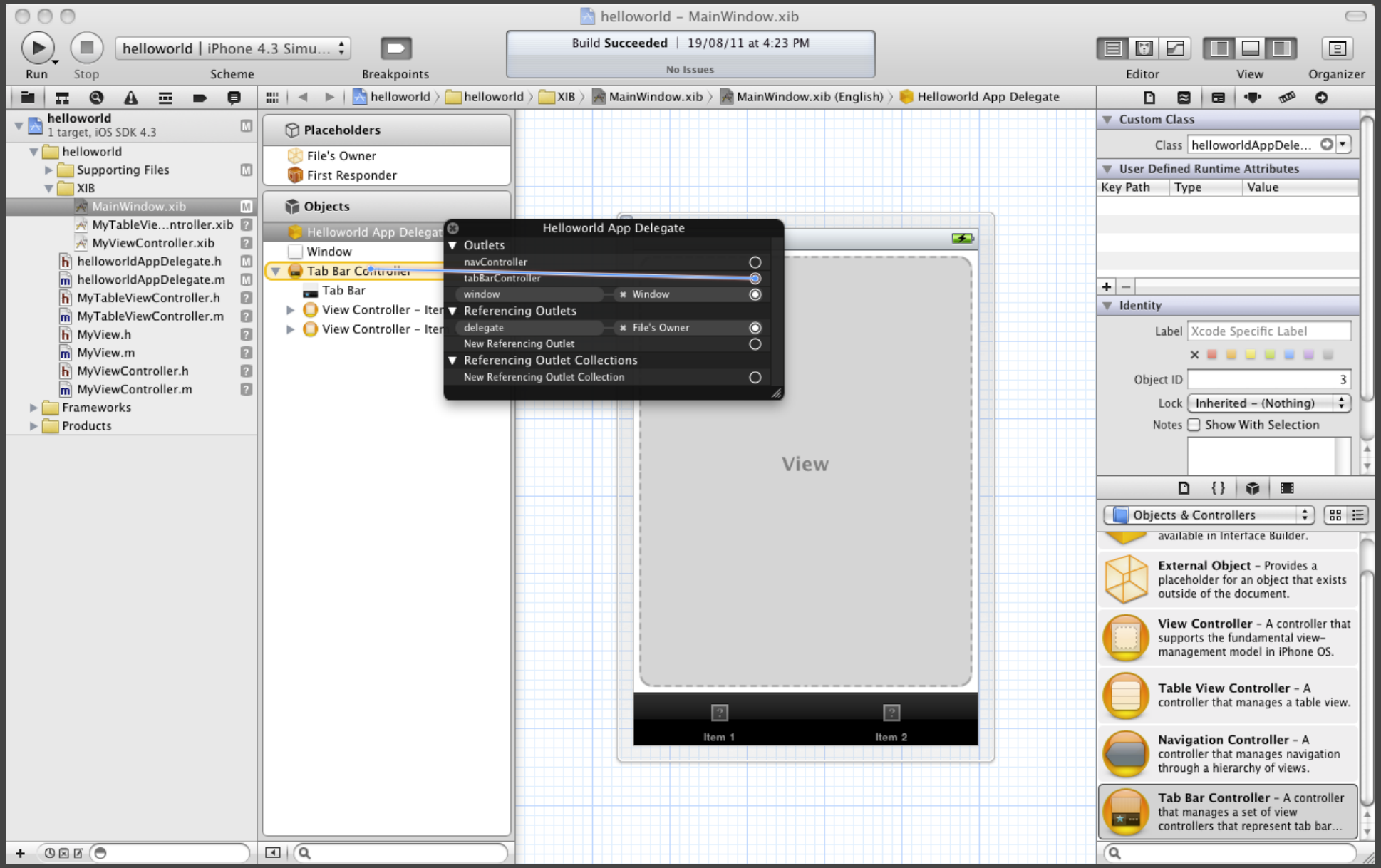


4.2.1 Add Tab Bar Controller Object

Swinburne University of Technology

```
//
// MyAppDelegate.h
//
@interface MyAppDelegate : NSObject <UIApplicationDelegate> {
   IBOutlet UITabBarController*  tabBarController;
}
@end
```

4.2.2 Add IBOutlet to App Delegate Header



4.2.3 Connect the IBOutlet

Swinburne University of Technology

```
//
// MyAppDelegate.m - partial
//
@implementation MyAppDelegate
@synthesize window=_window;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    [_window addSubview: tabBarController.view];
}

/* ... other stuff ... */

@end
```

4.2.2 Add Tab Bar Controller's View to Window

4.3 Creating a Tab Bar Controller in code

1. Add Tab Bar Controller object to App Delegate
2. Create Root View Controllers for each tab
3. Put the view controllers in an array
4. Add tab bar controller's view to the window

```
// MyAppDelegate.m - snippet
@implementation MyAppDelegate
@synthesize window=_window;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    tabBarController = [[UITabBarController alloc] init];

    MyViewCtlr* vc1 = [[MyViewCtlr] init];
    MyOtherViewCtlr* vc2 = [[MyOtherViewCtlr alloc] init];

    NSArray* controllers = [NSArray arrayWithObjects:vc1,vc2,nil];
    tabBarController.viewControllers = controllers;

    // Add tab bar controller's view as subview of window
    [_window addSubview:tabBarController.view];
}
```

4.3.1 Create Tab Bar Controller Object

4.4 Creating a Tab Bar Item in code

During the 'init' method of each view controller:

1. Create a UITabBarItem object
2. Add an image for the tab
3. Assign it to the VC's *tab bar item*
4. Release the local pointer

```
- (id)init {
   if (self = [super initWithNibName:@"MyViewCtlr" bundle:nil]) {
      self.title = @"My View Controller";
      UIImage* buttonImage = [UIImage imageNamed:@"MyBtn.png"];
      UITabBarItem* theItem = [[UITabBarItem alloc]
            initWithTitle:@"Tab One" image:buttonImage tag:0];
      self.tabBarItem = theItem;
      [theItem release];
   }
   return self;
}
```

4.4.1 Create Tab Bar Item Object

What's On For Today?

1. Views & View Controllers
2. Table Views
3. Navigation Controllers
4. Tab Bar Controllers
5. **Unit Testing**

5.0 Unit Testing

- **Question:**
  How do you test if your code works?

- **Answer:**
  Run it

5.1 The Problem

- **Question:**
  Your app has 100 features. How do you test if it all works correctly?

- **Answer:**
  Run it ... and test 100 features.

One problem, this isn't practical!

5.2 The Solution

- Your computer can test your code for you if you write *unit tests*.
- Unit tests are called this because ideally you are testing ***one unit of code*** (i.e. a single method or similar)

5.3 Unit Testing Theory

Textbooks Say:
You should test each feature every time you change *one line of code*.

This also isn't practical! So untested code is delivered in the final product, eventually causing headaches for everyone!

5.4 Unit Testing Example

- Our app has a UITextField for the user to enter their e-mail address

- We require our code to detect if the user enters a valid e-mail address or not, so we might do this (assuming -isValidEmail: exists)

```
BOOL isValidEmail = [self isValidEmail:textField.text];
if (isValidEmail) {
  // save the email
}
else {
  // show an alert
}
```

5.5 Human vs. Unit Testing

- A **human** might test this by typing an invalid email addresses to confirm an alert is shown, and then typing a valid address to check it passes.

- We run these tests in code using *2 unit tests*

```
- (void) testBadEmail {
  BOOL res = [testObj isValidEmail:@"swinburne"];
  STAssertEquals(res,NO,@"Should return NO here");
}

- (void) testGoodEmail {
  BOOL res = [testObj isValidEmail:@"paul@swin.edu.au"];
  STAssertEquals(res,YES,@"Should return YES here");
}
```

5.6 Unit Testing Example

- Congratulations! After releasing our app for testing a user reports the following bug:

> "I tried to type my email address, $A123@example.com, but ure app wouldn't let me !!!! PLZ FIX IT ASAP K THXBYE"

- We can write this as a unit test:

```
- (void) testWonkyButStillValidEmail {
  BOOL res = [testObj isValidEmail:@"$A123@example.com"];
  STAssertEquals(res,YES,@"should be valid!");
}
```

5.7 Unit Testing Example (continued)

1. We write a test that will *fail* -- our code is broken so we want to write a test to prove this
2. Next we fix our logic in the code to squash the bug
3. Run the test again and it passes, confirming the fix

This process becomes more important the bigger and more complex a software project becomes.

5.8 Creating Tests in Xcode

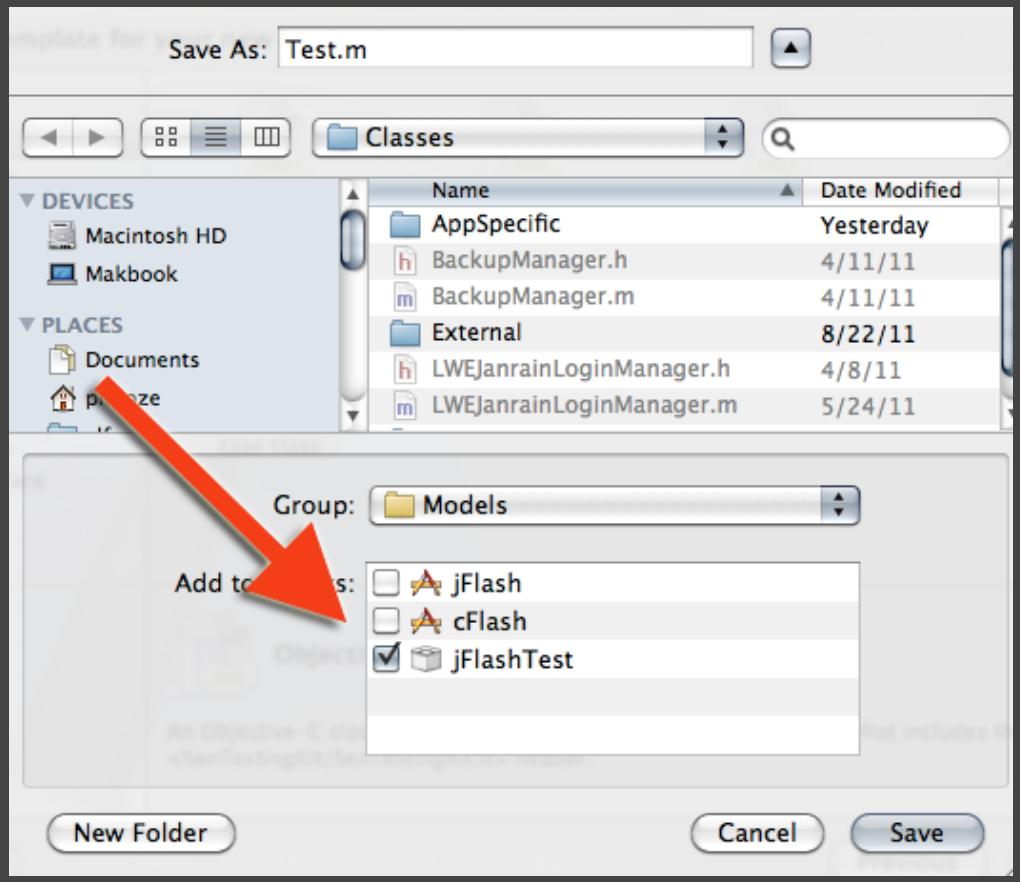
1. Tick "**Use Unit Tests**" box when you creating a new iOS project in Xcode

2. Create a new test class, *File* > *New* > *New File*

3. Select "Cocoa Touch" from the left panel, and "Objective-C test case class" from the right panel

5.9 Creating Tests in Xcode

Finally, on the Save dialog, tick the "Test" target, not any other target . This is what makes it run.



5.10 Inside Unit Test Classes

You need to define your *test cases* in *test classes*.
A unit test class might look like:

```
#import "<class whose methods you will test>.h"
@implementation ValidEmailTests
- (void) testBadEmail { // example as before }
- (void) testGoodEmail { // example as before }
- (void) testWonkyButGoodEmail { // example as before }
@end
```

5.11 Assert Yourself

- Test methods use ***assertions*** to confirm things are operating as expected

- ObjC's built-in assertion syntax look like this:

```
STAssertEqual(object, message, ...)
STAssertNotNil(object, message, ...)
STAssertTrue(expression, message, ...)
STAssertFalse(expression, message, ...)
```

5.12 Example Assertion Code

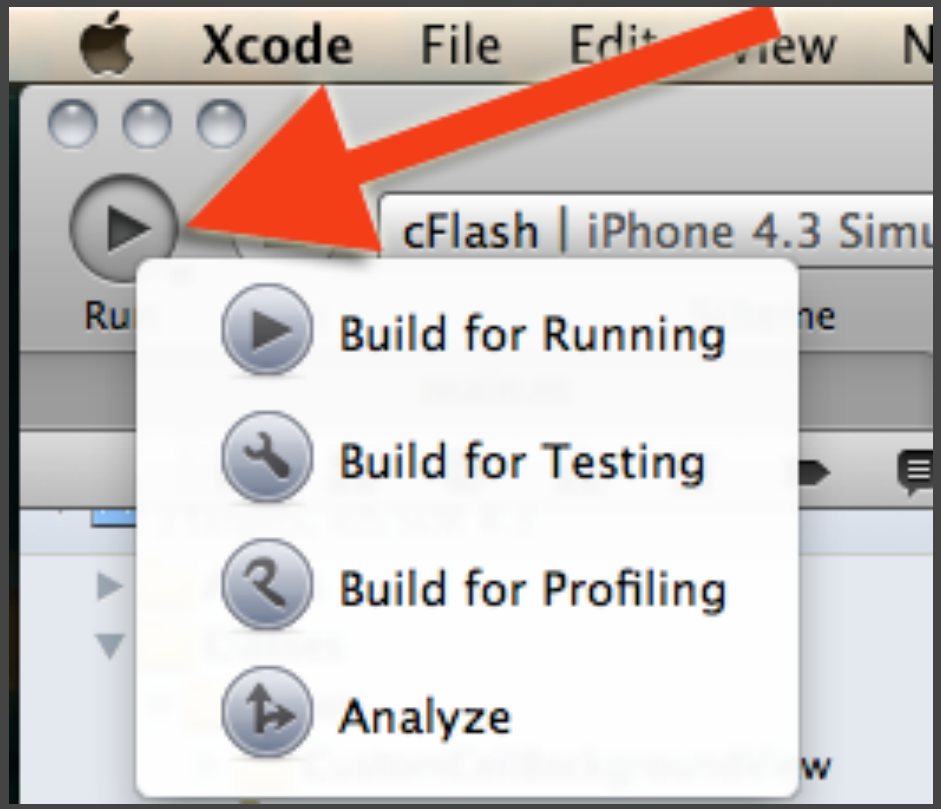
```
// Fictional test to confirm gender is male
- (void) testPersonIsMale:(NSString*)gender {
    STAssertEquals(@"male", gender,
                   @"Expected gender is 'male'");
}

// Fictional test to confirm type of love
- (void) testLoveIsTrue:(BOOL)type {
    STAssertTrue(type, @"TRUE is the only valid love!");
}
```

Apple's documentation has more details

See: *http://developer.apple.com/tools/unittest.html*

5.13 Running Your Tests

- Click-and-hold the Run button in the upper left:
- Tests are debug-able using Xcode -- very helpful!



What We Covered Today

1. Views & View Controllers
2. Table Views
3. Navigation Controllers
4. Tab Bar Controllers
5. Unit Testing

End of Lecture 4

1. Lab
2. Assignments
3. Practice Makes Less Imperfect