iOS Application Development

Day 1

Chris Zelenak

06/22/2010

Outline

Outline

Intro to class

Welcome to the 4-Day iOS Application Development class. Over the course of these next 4 days, we're going to be reviewing Objective C, UIKit, Core Animation, Core Data, and a host of other technologies you may only know by name.

Intro to class

Email me two pictures of you - one serious and one silly - to chris - at - fastestforward.com

Outline

Lab 1

2. Lab 1

Create a new iPhone Project

Create a new View based application in XCode called Lab1

Create a new iPhone Project

Open Lab1ViewController.xib in Interface Builder

Add a button

Add a button with the text "Hello iPhone"

Connect the button to the view controller

Add an IBAction selector to the view controller

Connect the button to the view controller

Make it open an alert view

Build and run in simulator

Yay!

Outline

Objective C

3. Objective C

Overview

Objective C is a superset of C. It tacks on to C a message-passing object model similar to Smalltalk's (see Ruby), with a reflective class model built in.

Overview

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

Overview

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

Overview

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

Review of standard C elements

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

```
typedef enum {
  NONE = 0,
  SOME,
  ALL
} ThoseWhoPreferItHot;

ThoseWhoPreferItHot = SOME; // Use of enumerated constant
```

Listing 1: C Language elements - Enumerations

Review of standard C elements

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

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

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

Listing 2: C Language Elements - Structs

Review of standard C elements

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

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

// C99 Variable length array declaration
float aListOfFloats[someQualifier];

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

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

Listing 3: C Language Elements - Arrays and Pointers

Review of standard C elements

If you've not kept abreast of changes between ANSI-92 C (a common version of C taught in universities and in many books about C) and C99, read more at `http://home.datacomm.ch/t_wolf/tw/c/c9x_changes.html` [?].

Syntax

```
NSString * anExampleString = @"This is a pointer to an NSString object";
int strLen = [anExampleString length];

NSString * aSecondString = [anExampleString
  stringByAppendingString:@" that had a second string appended to it"];

if([anExampleString
     compare:@"this is a pointer to an nsstring object"
     options:NSCaseInsensitiveCompare] == NSOrderedSame){
  NSLog(@"The string %@ was case insensitive equal", anExampleString);
}
```

Listing 4: Some typical Objective C code

Calling functions

Given some Objective C object, like:

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

you can invoke functionality on that object via the form:

```
[anObject theMethodName];
```

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

.

Messages vs. Methods

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

nil, NULL

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

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

Listing 5: nil object responding to any message

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

Classes

Memory management

Objective C's memory management model is a reference counting system that you can think of as an integer counter assigned to each object. Each object, when initialized, starts with a counter equal to 1. When this counter reaches 0, the cleanup functions (dealloc) on the object are called and the memory for the object is released back to the system.

Memory management

```
// Retain count of 1
NSString * foo = [[NSString alloc] initWithString:@"Foo"];
[foo retain]; // Retain count of 2
NSLog(@"Foo is %@", foo);
[foo release]; // Retain count of 1
NSLog(@"Foo is still %@", foo);
[foo release]; // Retain count of 0,
               // foo is released back to the system

// This will crash the program (EXC_BAD_ACCESS)
NSLog(@"Foo crashes %@", foo);
```

Listing 6: Memory management in Objective C

EXC BAD ACCESS ? What?

EXC BAD ACCESS is the error message your program will output when it attempts to do something to a pointer to a memory location whose contents have previously been released to the system.

Memory management

You can indicate you don't want to manage the release of an object by indicating an object should be autoreleased objects. Autoreleased objects are managed by an object called an NSAutoreleasePool, and are intermittently checked to see how many references have been claimed on them. The pool will release the object "at some unknown point in the future"* once all retains on the object have been released.

Memory management

```
// Indicate that the foo object should be managed by
// the current NSAutoreleasePool
NSString * foo = [[[NSString alloc]
                            initWithString:@"Foo"]
                              autorelease];
NSLog(@"Foo will be good for now %@", foo);

[foo retain];
[foo release];
NSLog(@"Foo is still good, %@", foo);
// You may retain and release on an autoreleased object as normal

[foo release];
// Calling an 'unbalanced release' like this will cause a crash
// at some unknown point in the future
```

Listing 7: Memory management in Objective C

Memory management

Different classes in Cocoa / Objective C have different rules about how they retain objects. The documentation is instructive in telling you what memory management semantics you should expect from given functionality.

Mac Programmers Have All The Luck

Should you be programming Mac OS applications as opposed to iOS applications, you can take advantage of garbage collection memory management which has been available since Mac OS Leopard (10.5); iOS programmers still have to manage their memory themselves, however.

Properties

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

Properties

```
@interface MyObject : NSObject {
  NSString * instanceVariable;
  float numericVariable;
}
@property(readwrite, retain) NSString * instanceVariable;
@property(readonly, assign) float numericVariable;
@end

@implementation MyObject

@synthesize instanceVariable;
@synthesize numericVariable;

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

@end
```

Listing 8: Property interface declaration

Properties

```
MyObject * mo = [[MyObject alloc] init];
[mo setInstanceVariable:@"Test"];
[mo setNumericVariable:20.0f];
NSLog(@"The values I set were %@ and %f",
        [mo instanceVariable],
        [mo numericVariable]);
```

Listing 9: Property usage

Properties

You can specify the memory management semantics (retain, assign, copy) in the property declaration. You can also specify the atomicity (locking behavior) of the property, and more.

More to it..

Properties do not merely provide getter/setter functionality. The use of properties also implicitly adds Key-Value-Observing functionality to your class, letting you automatically monitor classes for change events and performing specific code in such cases. Read more at `http://developer.apple.com/mac/library/documentation/cocoa/conceptual/objectivec/articles/ocProperties.html`.

Selectors

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

Selectors

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

Listing 10: Selector usage

Selectors

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

```
NSString * aUserSuppliedString = MagicallyGetStringFromUserInput();
SEL aSelector = NSSelectorFromString(aUserSuppliedString);
if([someObject respondsToSelector:aSelector]){
  // NOTE: Surely nothing bad will happen
  [someObject performSelector:aSelector];
}
```

Listing 11: Other selector usage

Protocols

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

Protocols

```
@protocol TestProtocol

-(void) definitelyHasThisMethod;

@optional

-(void) mayHaveThisMethod;

@end
```

Listing 12: Protocol declaration

```
@interface SomeObject : NSObject<TestProtocol> {
}
@end
```

Listing 13: Protocol conformance

Protocols

```
SomeObject<TestProtocol> * oo = [[SomeObject alloc] init];

// We assume this because the object conforms to TestProtocol
[oo definitelyHasThisMethod];

if([oo respondsToSelector:@selector(mayHaveThisMethod)]){
  [oo mayHaveThisMethod];
}
```

Listing 14: Protocol usage

Categories

Categories allow you to mix in new code to existing classes without having to change the original source code for those classes. Examples of categories include automatically adding special serialization rules to NSObject, or special data structure Queue behavior to Cocoa collection classes.

Except for..

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

Categories

```
@interface NSString(FunkyStrings)

-(NSString *) getFunky;

@end

@implementation NSString(FunkyStrings)

-(NSString *) getFunky {
  return [self stringByAppendingString:@"...YEAH! ALL RIGHT! FEELS GOOD!"];
}

@end
```

Listing 15: Category declaration

Categories

```
#import "NSString+FunkyStrings.h"
void main(){
  NSLog(@"The funkiest string by far is %@", [@"James Brown" getFunky]);
}
```

Listing 16: Category usage

Review common useful classes in Objective C / Cocoa to accomplish certain tasks

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

Review common useful classes in Objective C / Cocoa to accomplish certain tasks

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

```
NSString * s = [[NSString alloc] initWithCString:"this is an ascii string"
                                         encoding:NSASCIIStringEncoding];
NSMutableString * so = [[NSMutableString alloc]
                                    initWithString:@"A mutable string with..."];

[so appendString:s];

NSLog(@"The mutable string is %@", so);

[so release];
[s release];
```

Listing 17: NSString usage

Review common useful classes in Objective C / Cocoa to accomplish certain tasks

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

```
NSNumber * n = [[NSNumber alloc] initWithFloat:20.0f];
double d = [n doubleValue];
int i = [n intValue];
[n release];
```

Listing 18: NSNumber usage

Review common useful classes in Objective C / Cocoa to accomplish certain tasks

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

Review common useful classes in Objective C / Cocoa to accomplish certain tasks

```
NSDictionary * a = [[NSDictionary alloc] initWithObjectsAndKeys:
                                        @"The foo string", @"Foo",
                                        @"The bar string", @"Bar",
                                        nil];

NSMutableDictionary * b = [[NSMutableDictionary alloc] init];
[b setObject:@"Another string" forKey:[NSNumber numberWithInt:20]];

NSLog(@"a's alue for Foo is %@, and b's value for 20 is %@",
                          [a objectForKey:@"Foo"],
                          [b objectForKey:[NSNumber numberWithInt:20]]);

for(id key in a){
  NSLog(@"The value for %@ is %@", key, [a objectForKey:key]);
}

[b release];
[a release];
```

Listing 19: NSDictionary usage

Review common useful classes in Objective C / Cocoa to accomplish certain tasks

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

```
NSArray * a = [[NSArray alloc] initWithObjects:@"Foo", @"bar", @"baz", nil];
NSMutableArray * b = [[NSMutableArray alloc] init];
[b addObject:@"Not foo"];
NSLog(@"The contents of a are %@ and b are %@", a, b);

for(NSString * obj in a){
  NSLog(@"The array contains %@", obj);
}

[a release];
[b release];
```

Listing 20: NSArray usage

Review common useful classes in Objective C / Cocoa to accomplish certain tasks

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

Review common useful classes in Objective C / Cocoa to accomplish certain tasks

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

Figure: Sample Property List

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

Listing 21: Property List Deserialization

More learning

Objective C is a rich, deep language. There are lots of resources on the web and your computer where you can learn more. The best is easily the XCode documentation, but there are blogs out there that can provide invaluable help: Mike Ash's NSBlog `http://www.mikeash.com/pyblog/`, Cocoa With Love `http://cocoawithlove.com/`, and the Apple Developer Forums `http://developer.apple.com/devforums/` can all provide invaluable help in learning new techniques and advanced usage for your apps.

Outline

iPhone App Layout Conventions

4. iPhone App Layout Conventions

MVC, as applied to an app

Apple strongly encourages you to adhere to Model View Controller pattern

Application -¿ Window -¿ View Controller(s) -¿ View(s) + Models

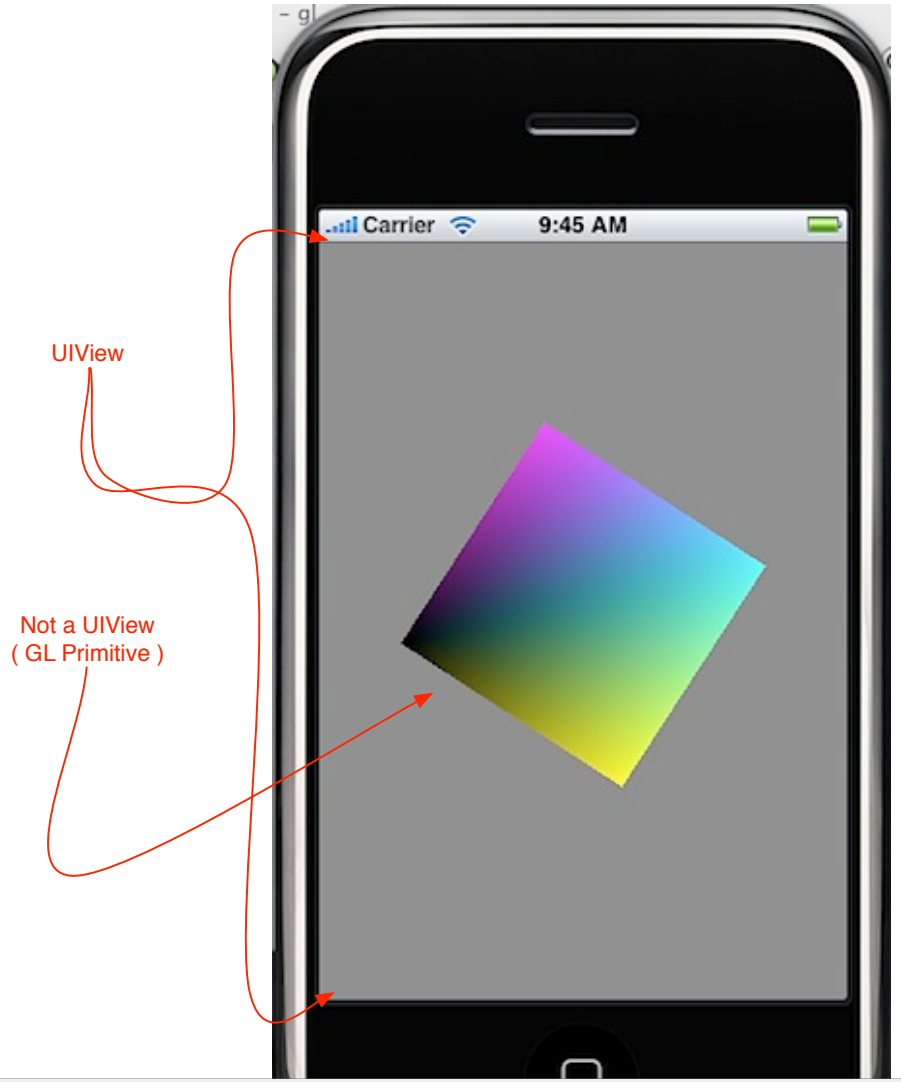
On iOS, a UIApplication (your app) typically has one UIWindow, which is the primary UIView upon which everything else displays. A UIView is just a rectangle upon which you draw things.

Application -¿ Window -¿ View Controller(s) -¿ View(s) + Models

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

Application -¿ Window -¿ View Controller(s) -¿ View(s) + Models

Application -¿ Window -¿ View Controller(s) -¿ View(s) + Models



Application -¿ Window -¿ View Controller(s) -¿ View(s) + Models

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

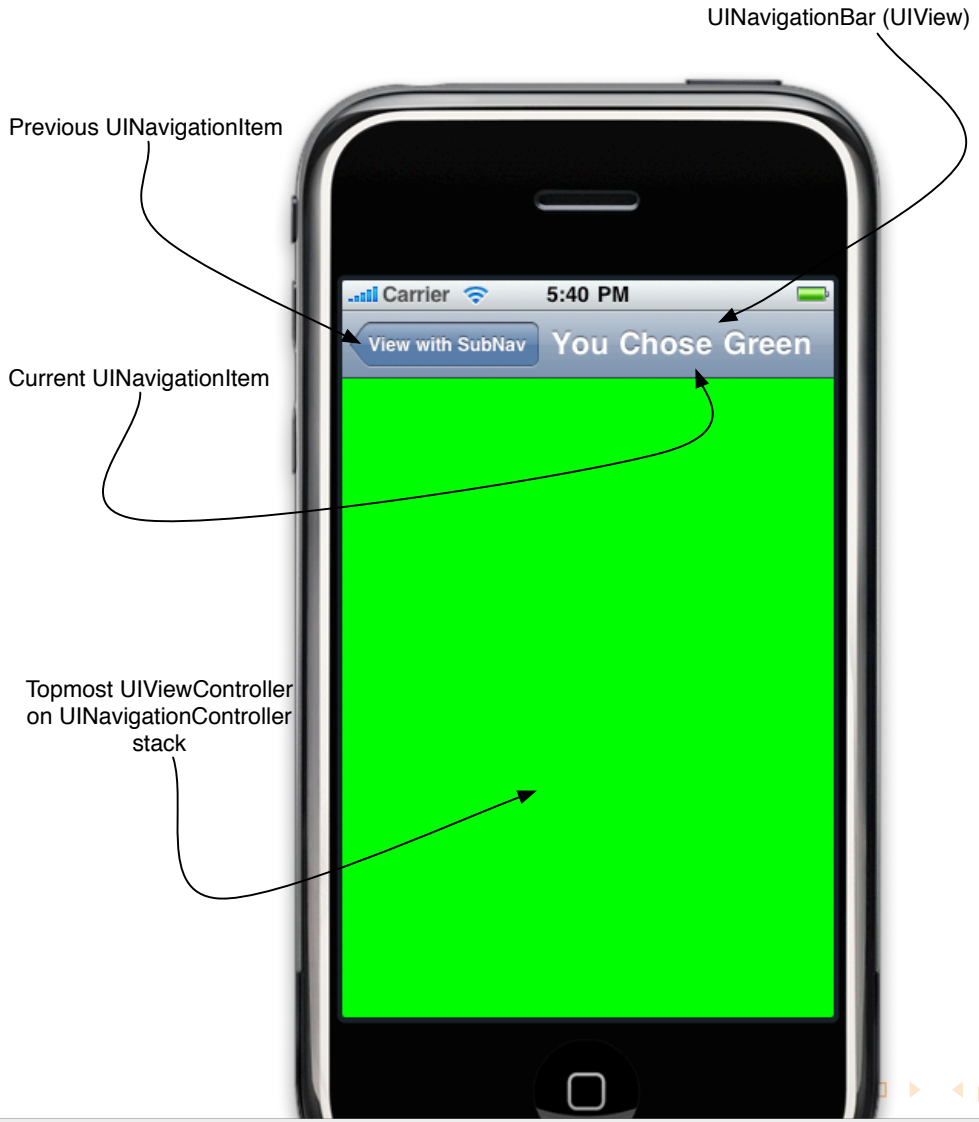
Application -¿ Window -¿ View Controller(s) -¿ View(s) + Models

You provide the model

UINavigationController

UINavigationController is a stack based manager of view controllers that the user can navigate through

UINavigationController



UINavigationController

Every UIViewController managed by a UINavigationController has a reference to that UINavigationController

UINavigationController

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

Listing 22: Adding another UIViewController to the UINavigationController stack

Lab 2

Create a new View based project

Lab 2

Add a UINavigationController to MainWindow.xib

Lab 2

Create a new view controller

Lab 2

Add a button to the first view controller and create an action to open the second view controller

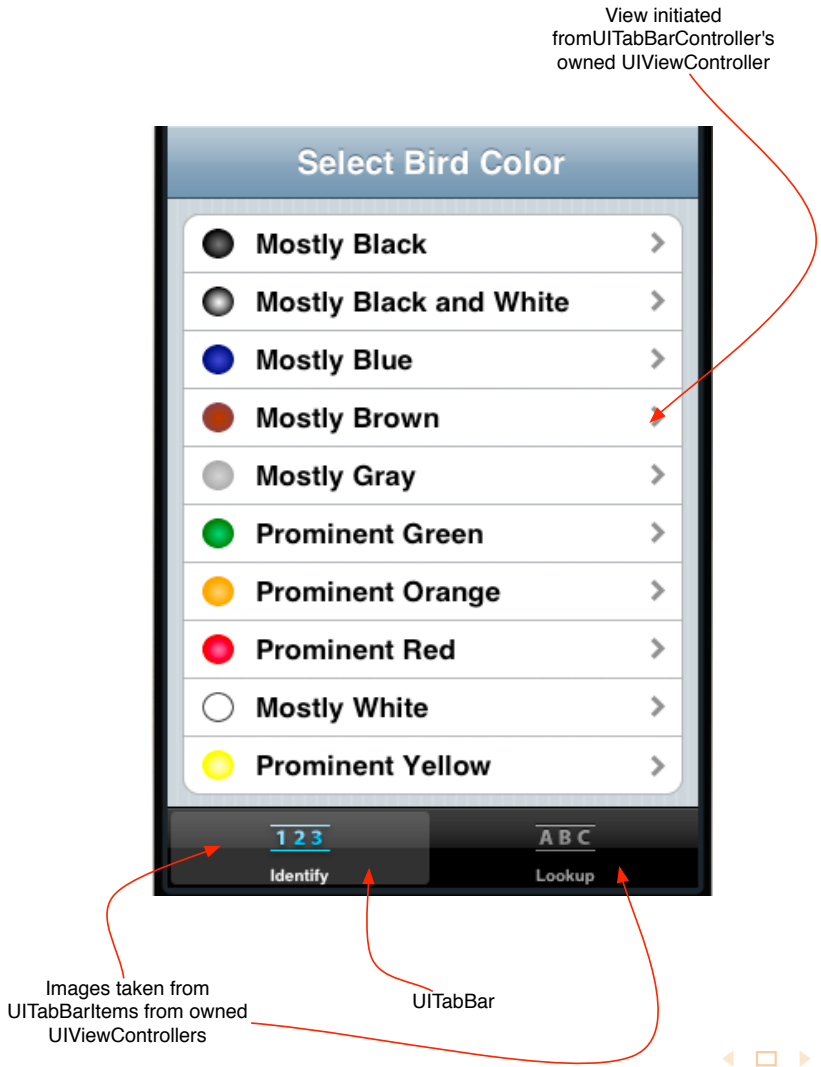
UITabBarController and UITableViewController

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

UITabBarController and UITableViewController

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

UITabBarController and UITableViewController



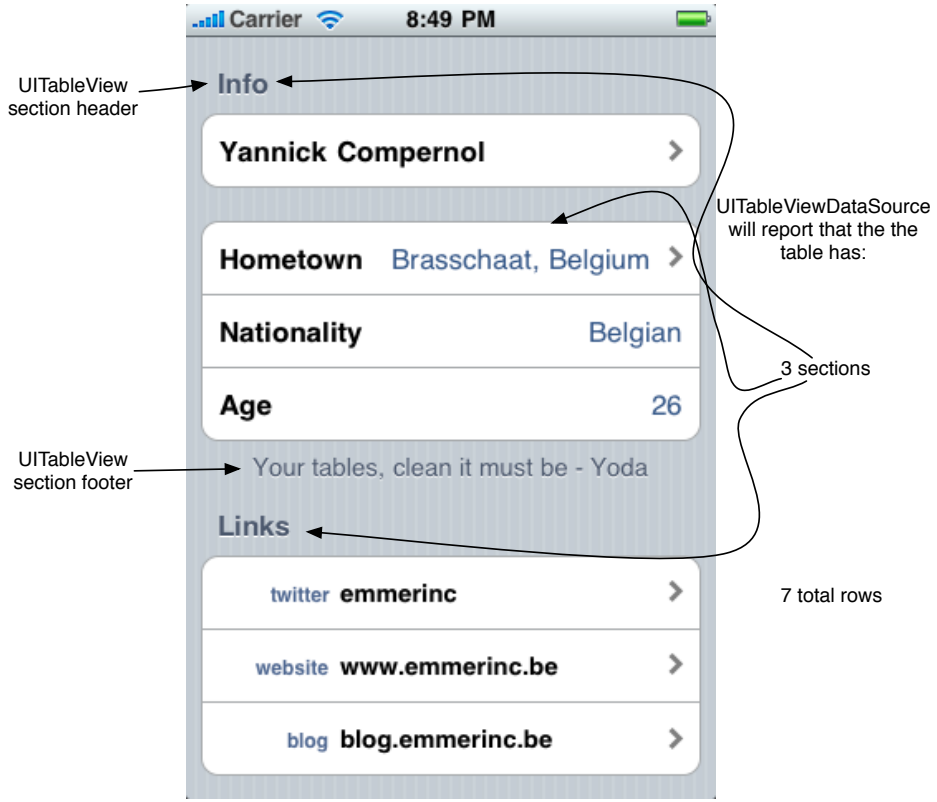
UITabBarController and UITableViewController

UITableViewController instructs a UITableView about the number of rows it contains, what to do when clicked, the number of sections, and other information

UITabBarController and UITableViewController

UITableViewController is just a convenience class that saves you from manually conforming an object to the UITableViewDataSource and UITableViewDelegate protocols

UITabBarController and UITableViewController



Lab 3

Add a UITableViewController to the project

Lab 3

Add an NSArray containing names from the class to the controller

Lab 3

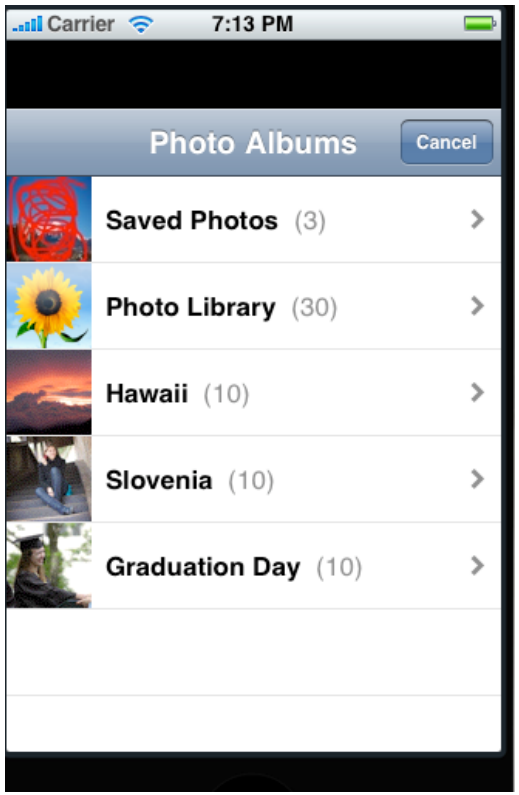
Change the Interface builder reference to the UITableViewController

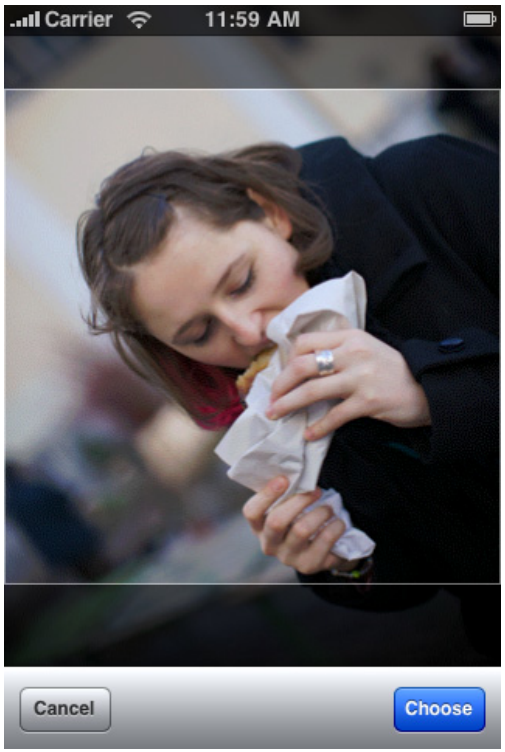
Modal view controllers

Not all view controllers fit within the "display in UINavigationController" or "display in UITabBarController" model.

Modal view controllers

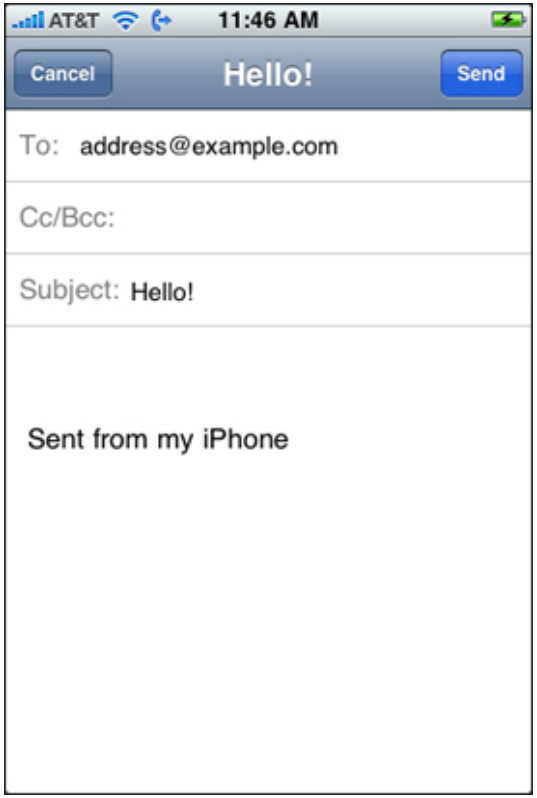
UIImagePickerController





Modal view controllers

MFMailComposeViewController



Modal view controllers

You can present a view controller modally by using the presentModalViewController:animated: message

```
-(IBAction) buttonClicked:(id) sender {
  MyViewController * mvc = [[MyViewController alloc]
                                    initWithNibName:@"TheNib"
                                    bundle:nil];
  [self presentModalViewController:mvc animated:YES];
  [mvc release];
}
```

Listing 23: Presenting a view controller modally

Modal view controllers

You can specify the transition style with which the view controller appears via the UIViewController message

```
setModalTransitionStyle:(UIModalTransitionStyle)style
```

Outline

XCode and Interface Builder

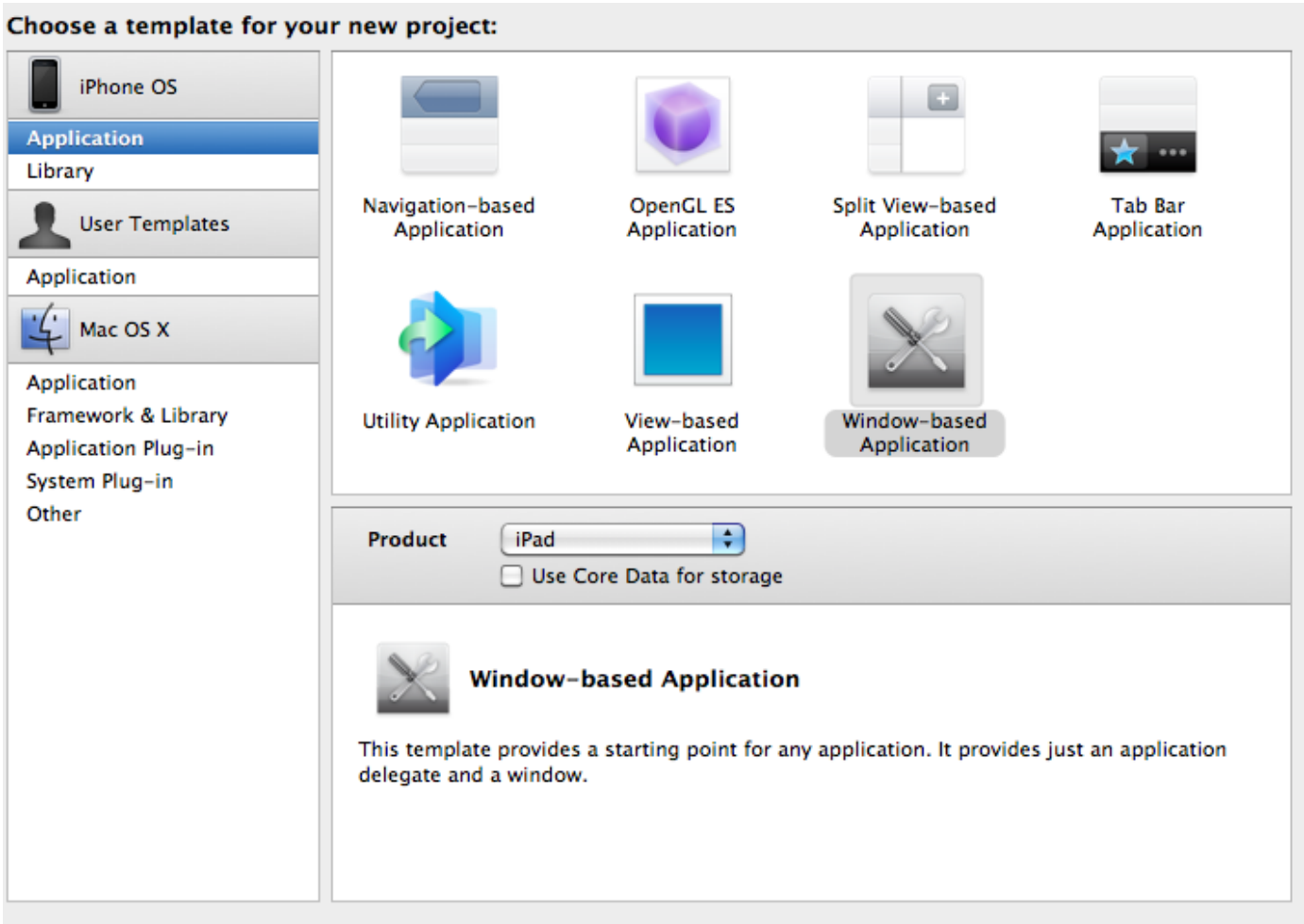
5. XCode and Interface Builder

Capabilities

XCode and Interface Builder combined provide a development, debugging and interface development tool for you to develop your iPhone apps. It can integrate with SVN, CVS or Perforce (and soon, Git!), it can run automated tests on your code, provide inline documentation assist, code completion and a lot more. It also provides a developer level interface to all the devices you will be debugging your code on; you can manage devices and device profiles through XCode as necessary.

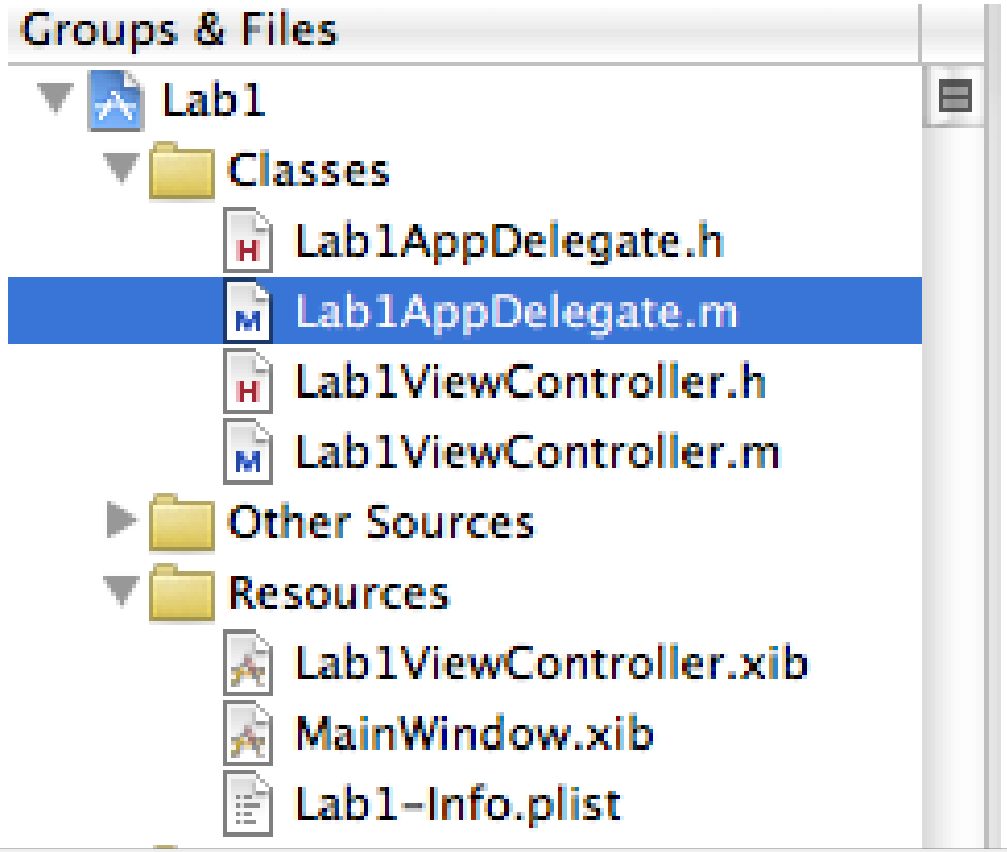
Creating an app

XCode provides a number of application templates for starting an app. Most basic types of applications can get a fast start by using one of these app templates.



Creating an app

Once you've selected the type of app you want to create, your project will be created with an application delegate and usually at least one view controller that will manage what goes on the screen. It will also have a .xib file for your application, and for each view controller in the app, that is used by Interface Builder to layout the screen contents.



Creating an app

Double clicking on a .xib file will open it in Interface Builder, where you can drag components from the Interface Builder toolbox into the view. You can easily change the orientation of objects.

Documentation usage

Help - Developer Documentation is an invaluable resource for researching Cocoa internals.

Protips!

Learn hotkeys! (CMD+SHIFT+D and CMD+0 will buy you hours)

Protips!

Make XCode fit you (Single window interface, XCode themes, application templates)

Protips!

XCode plugins (Code Pilot
`http://mac.brothersoft.com/code-pilot-for-xcode.html`,
Accessorizer
`http://www.kevincallahan.org/software/accessorizer.html`)

Outline

Questions