iOS development in Objective-C

Assignment 1

Assignment Format

- 3 weeks of classes, 3 assignments
- 60 to 90 minute lecture, with some hands-on time
- Remainder of class is hands-on time
- Remainder of assignment is due during the following class
- Assignments will build upon each other, so not completing one will make future assignments harder

Alternatives to Objective-C & Cocoa

- OpenFrameworks (C++)
- Flash Builder (ActionScript)
- PhoneGap (JavaScript)
- Appcelerator Titanium (JavaScript)
- RubyMotion (Ruby)
- MonoTouch (C#)
- And more...

Alternatives to Objective-C & Cocoa

- Although less familiar and potentially more difficult, we’re going to show you Objective-C because it’s how professional developers make apps.

- For your personal projects and prototypes, other frameworks and languages may make more sense

If you already know some Objective-C...

- Come talk to me during the hands-on period today

- You'll be able to propose a simple project for me to grade, and be able to get help with your project

One more thing...

- This stuff is hard
- Pay close attention... missing a concept will hurt
- Help each other out!
- Your work must be your own

The iOS Development Process

Text Editor

Interface Builder

Compiler

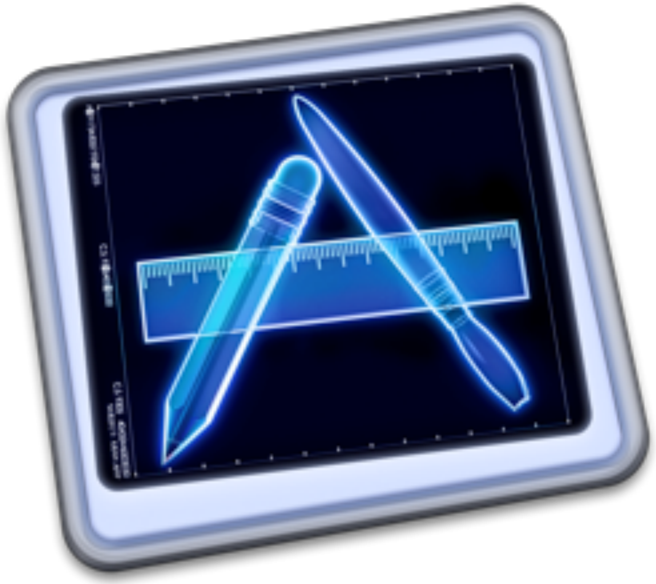
Debugger

Xcode

An **IDE** (Integrated Development Environment)

iPhone Simulator

For checking progress without a device



Instruments

For determining problems with memory or performance

Assignment

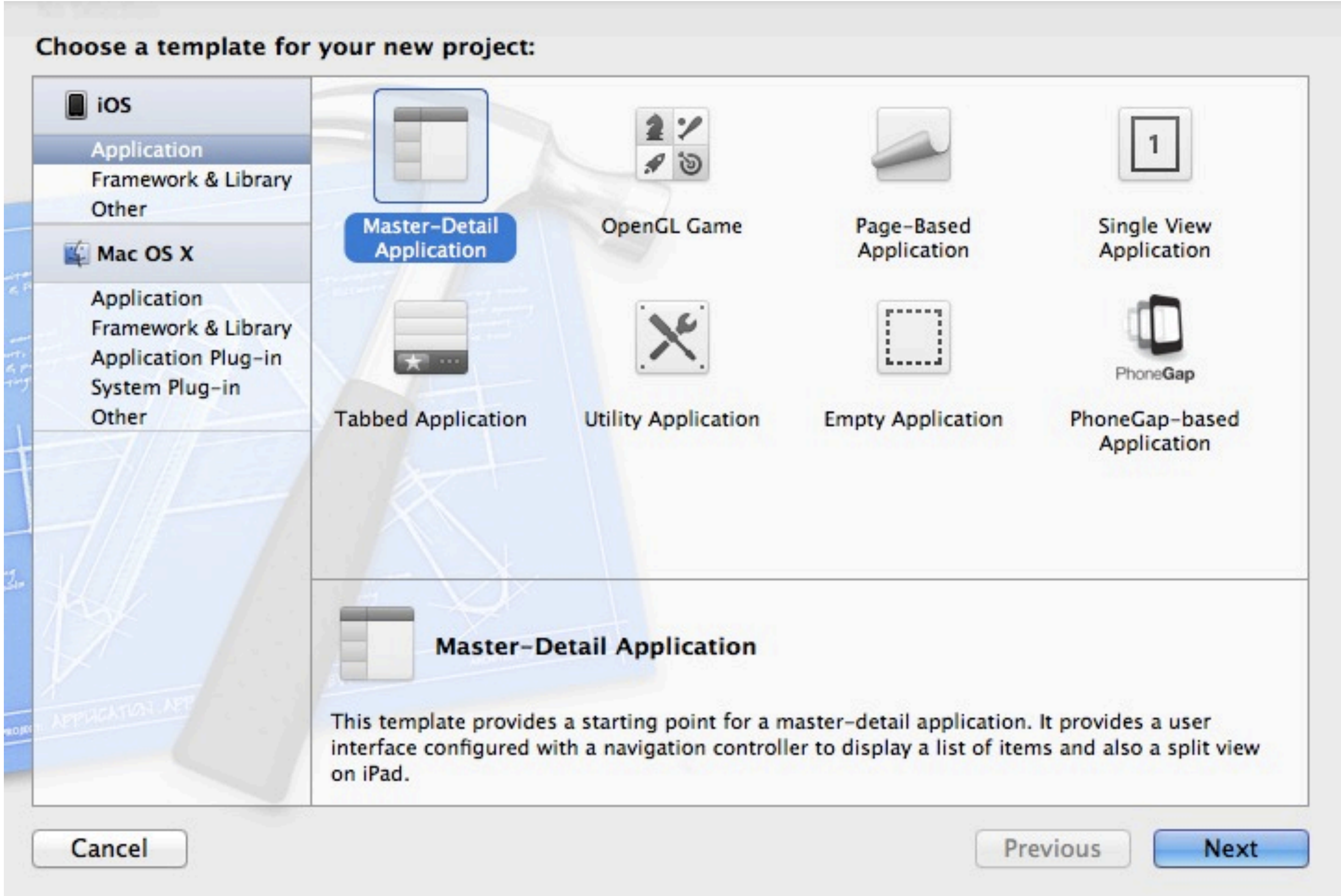
Getting started with Xcode

Goal

Create a basic view to display information about a restaurant.

Understanding Xcode

- We’ll start by creating a **New Project** from a **Master-Detail** template



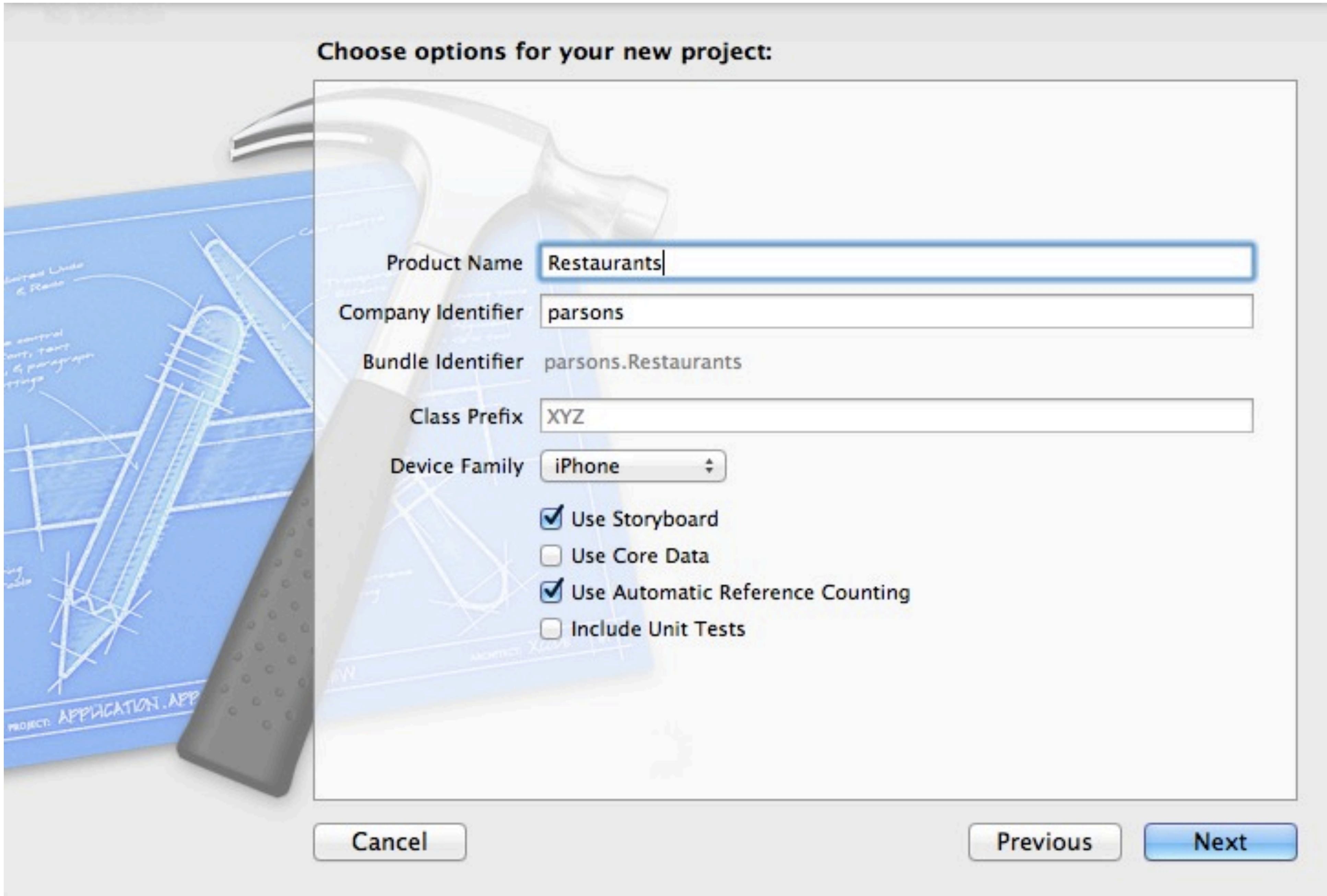
Choose a template for your new project:

Master-Detail Application

This template provides a starting point for a master-detail application. It provides a user interface configured with a navigation controller to display a list of items and also a split view on iPad.

Understanding Xcode

- Fill in the following options



Choose options for your new project:

Product Name: Restaurants

Company Identifier: parsons

Bundle Identifier: parsons.Restaurants

Class Prefix: XYZ

Device Family: iPhone

☑ Use Storyboard

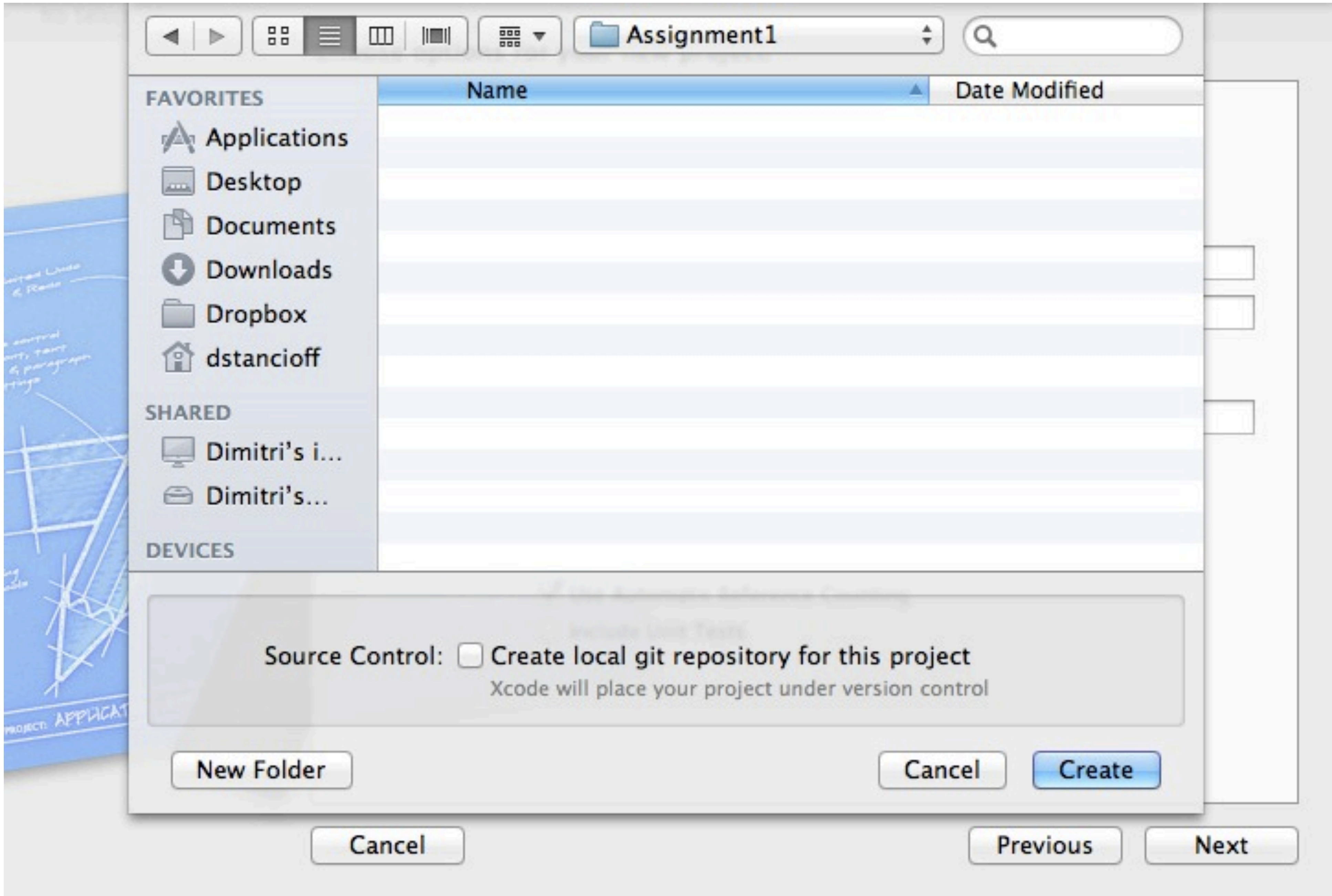
□ Use Core Data

☑ Use Automatic Reference Counting

□ Include Unit Tests

Understanding Xcode

- Save it

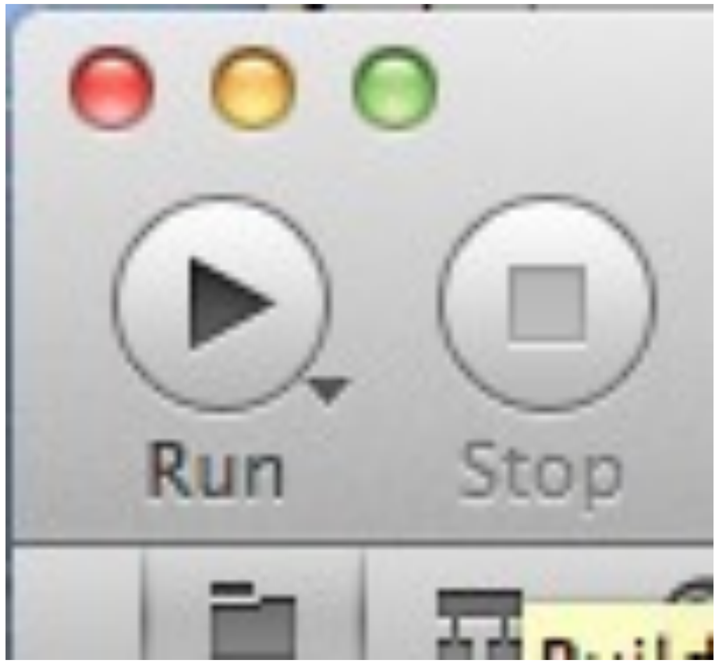


Source Control: □ Create local git repository for this project

Xcode will place your project under version control

Building

- Build and Run, and watch your app appear in the simulator



Reference

Files in Xcode

What are all these crazy files?

In the *Project Navigator* in Xcode you'll usually see various kinds of files that comprise various pieces of your application. Here are the most common ones.

- **.m** (Objective-C source code) These are the code files that are compiled by the compiler to make your program function. They will contain the logic that you will write.
- **.h** (header file) These are usually paired with a .m file. They represent the public interface for that .m file: i.e., they explain to other files how to communicate with it.
- **.storyboard** (Storyboard interface file) These files represent a series of views in your application, it's interface layout, and the transitions between interfaces. They are intended to draw parallels between storyboarding and wireframes.
- **.plist** (Property list file) These are useful files in that they represent arbitrary data: lists, dictionaries, strings, booleans, etc. They are usually used to either store data for an app or store configuration options for building that app.
- **.xcodeproj** You won't see this in the sidebar because you are already looking inside it using Xcode. These files are for specifying your project structure and settings.

Reference

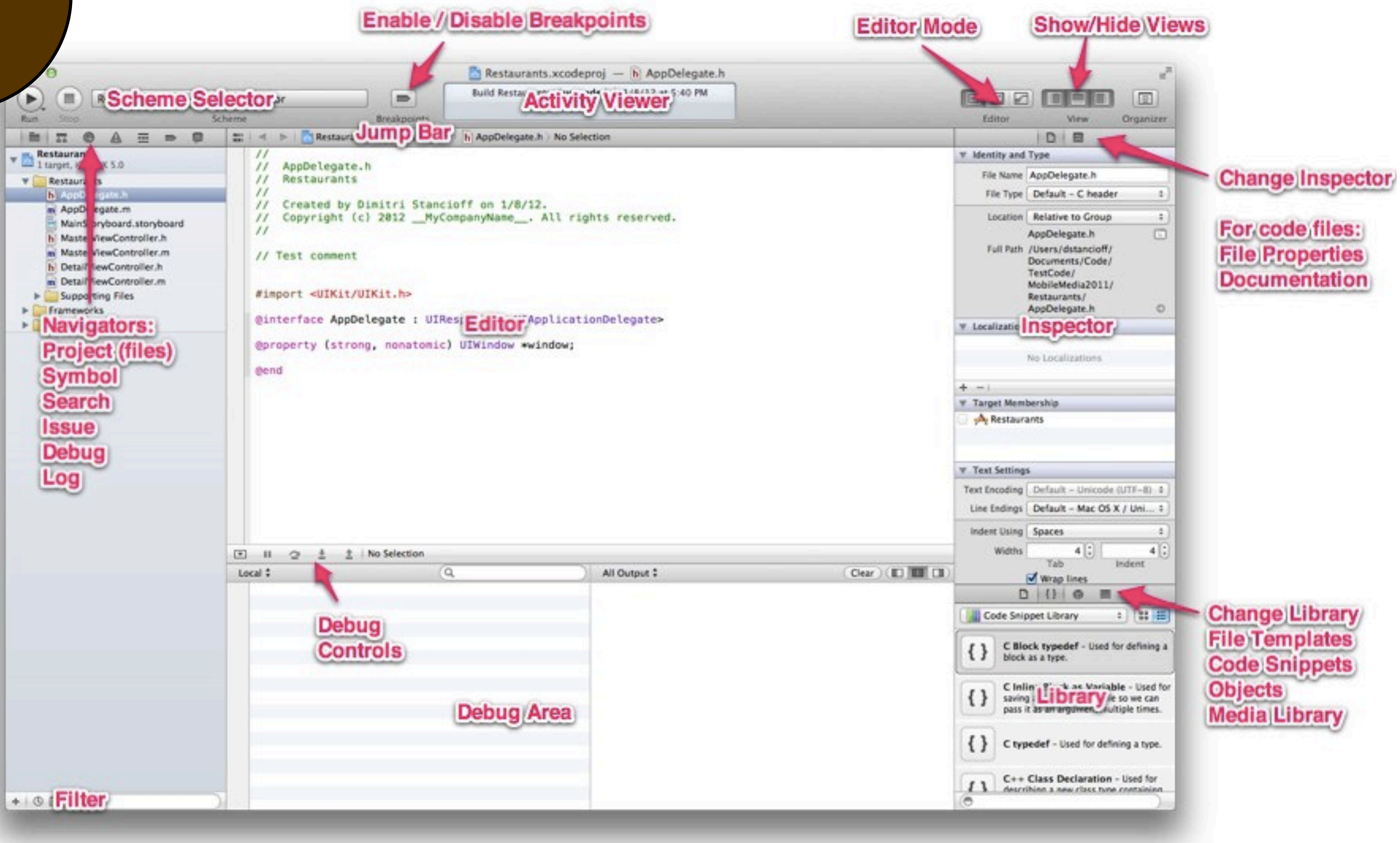
Additional Files

The following files we won't be editing in any of our assignments, although Xcode may perform tasks on them as we use it.

- **.strings** (Strings file) These won't be covered in this class, but they provide a place to put whatever user interface language is necessary in one place. When translating applications into other languages, strings files are used to completely replace the language while keeping the rest of the application the same.
- **.pch** (Precompiled header file) This file is referenced by the compiler. Anything included in this file will be available in any code file in your code without explicitly calling it. We won't need to modify this file in this course.
- **.framework** (Framework) Frameworks are bundles: special folders that act as files. They contain code that is related and that you might want to use. For example, to determine a user's location, the CoreLocation framework is used. Typically, framework files are managed through Xcode: you don't need to deal with them yourself.
- **.app** (Application Bundle) This is your app! Like a framework, it's a bundle, which means it contains a number of other files within it.

Reference

Xcode Interface



```
//
//  AppDelegate.h
//  Restaurants
//
//  Created by Dimitri Stancioff on 1/8/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

// Test comment

#import <UIKit/UIKit.h>

@interface AppDelegate : UIRes     ApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;

@end
```

File Name: AppDelegate.h
File Type: Default - C header
Location: Relative to Group
AppDelegate.h
Full Path: /Users/dstancioff/Documents/Code/TestCode/MobileMedia2011/Restaurants/AppDelegate.h

Text Encoding: Default - Unicode (UTF-8)
Line Endings: Default - Mac OS X / Uni...
Indent Using: Spaces
Widths: 4 (Tab) 4 (Indent)
☑ Wrap lines

C Block typedef - Used for defining a block as a type.

C Inlin... - Used for saving ... pass it as an argumen... multiple times.

C typedef - Used for defining a type.

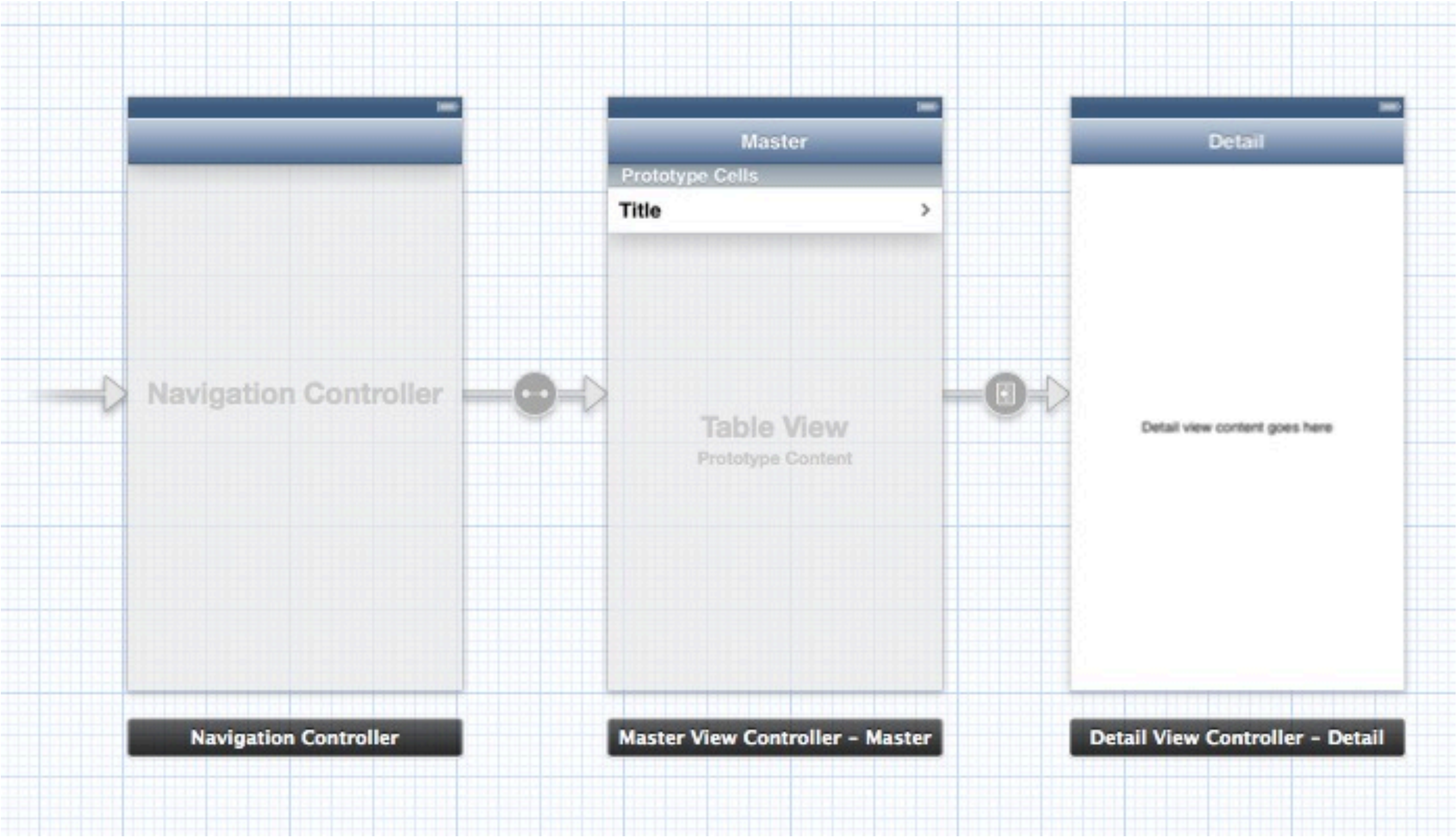
C++ Class Declaration - Used for describing a new class type containing

http://developer.apple.com/library/ios/#recipes/xcode_help-general/_index.html#//apple_ref/doc/uid/TP40010548

Starting with the User Interface

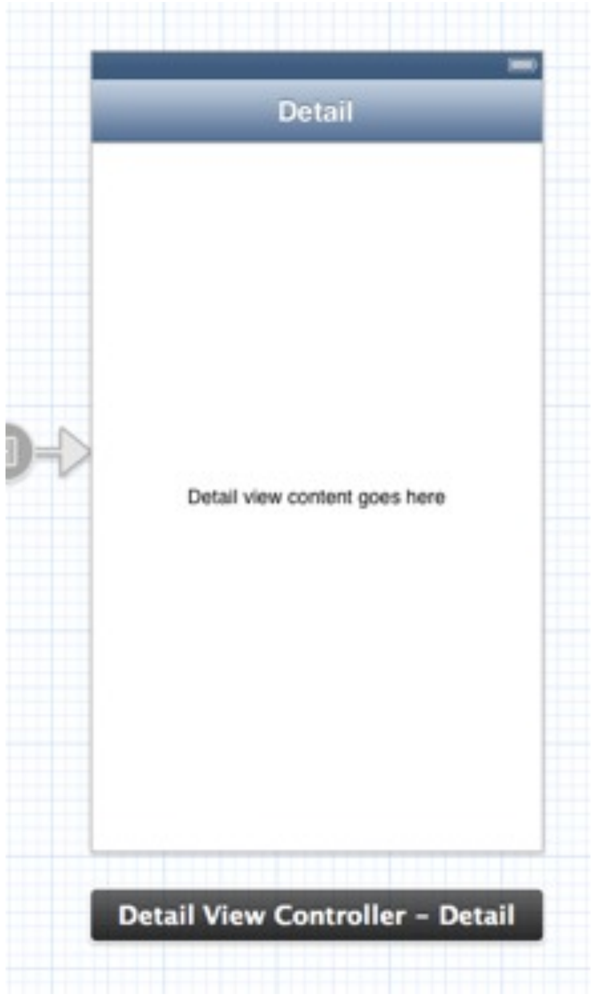
- We’ll start by designing **Storyboards**
- Created by **Interface Builder**
  - Built into Xcode
- Allows us to visually lay out some of the screens we are going to design.

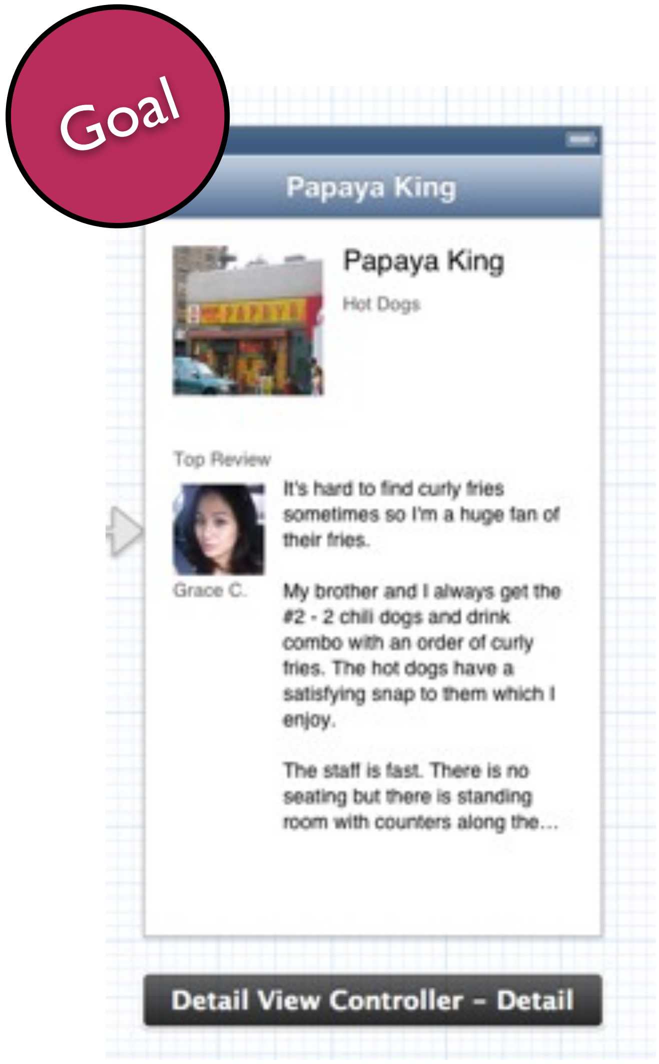
Storyboards



- Visually lay out views
- Visually connect views to other views

Laying out the Detail View


Detail
Detail view content goes here
Detail View Controller – Detail



It's hard to find curly fries sometimes so I'm a huge fan of their fries.

My brother and I always get the #2 - 2 chili dogs and drink combo with an order of curly fries. The hot dogs have a satisfying snap to them which I enjoy.

The staff is fast. There is no seating but there is standing room with counters along the...

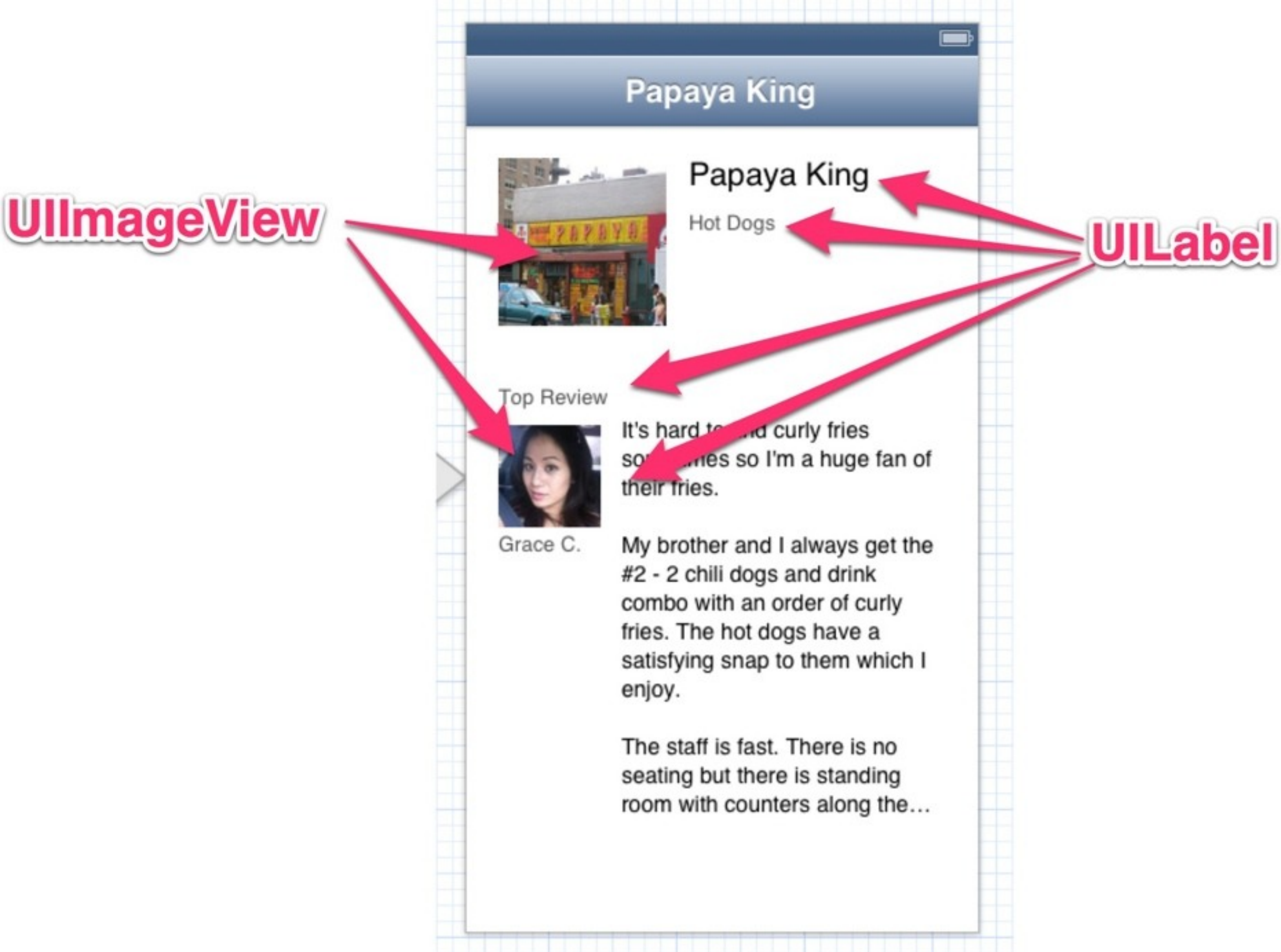
- Visually lay out views
- Visually connect views to other views

A couple of things to keep in mind...

- No code yet, just layout with static text and images
- It's not Photoshop

Goal Get familiar with Interface Builder and interface elements.

Labels and Images



Papaya King

Papaya King

Hot Dogs

Top Review

Grace C.

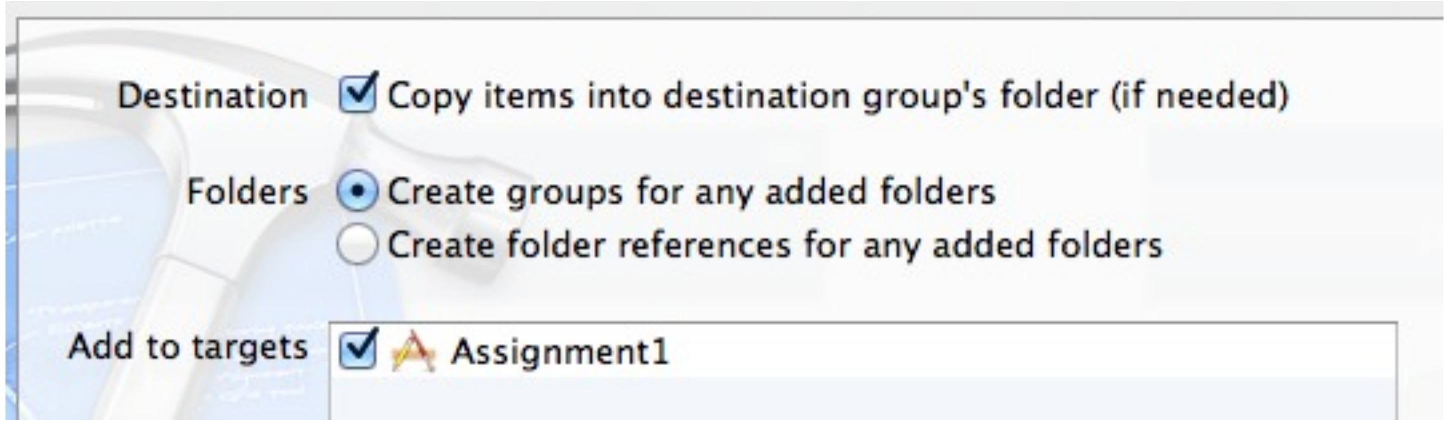
It's hard to find curly fries sometimes so I'm a huge fan of their fries.

My brother and I always get the #2 - 2 chili dogs and drink combo with an order of curly fries. The hot dogs have a satisfying snap to them which I enjoy.

The staff is fast. There is no seating but there is standing room with counters along the…

Tips:

- You can add interface elements using the panel on the bottom right
- Add images to Xcode by dragging them into the project navigator on the left
- Ensure you copy images to the project folder, like this:



Destination ☑ Copy items into destination group's folder (if needed)

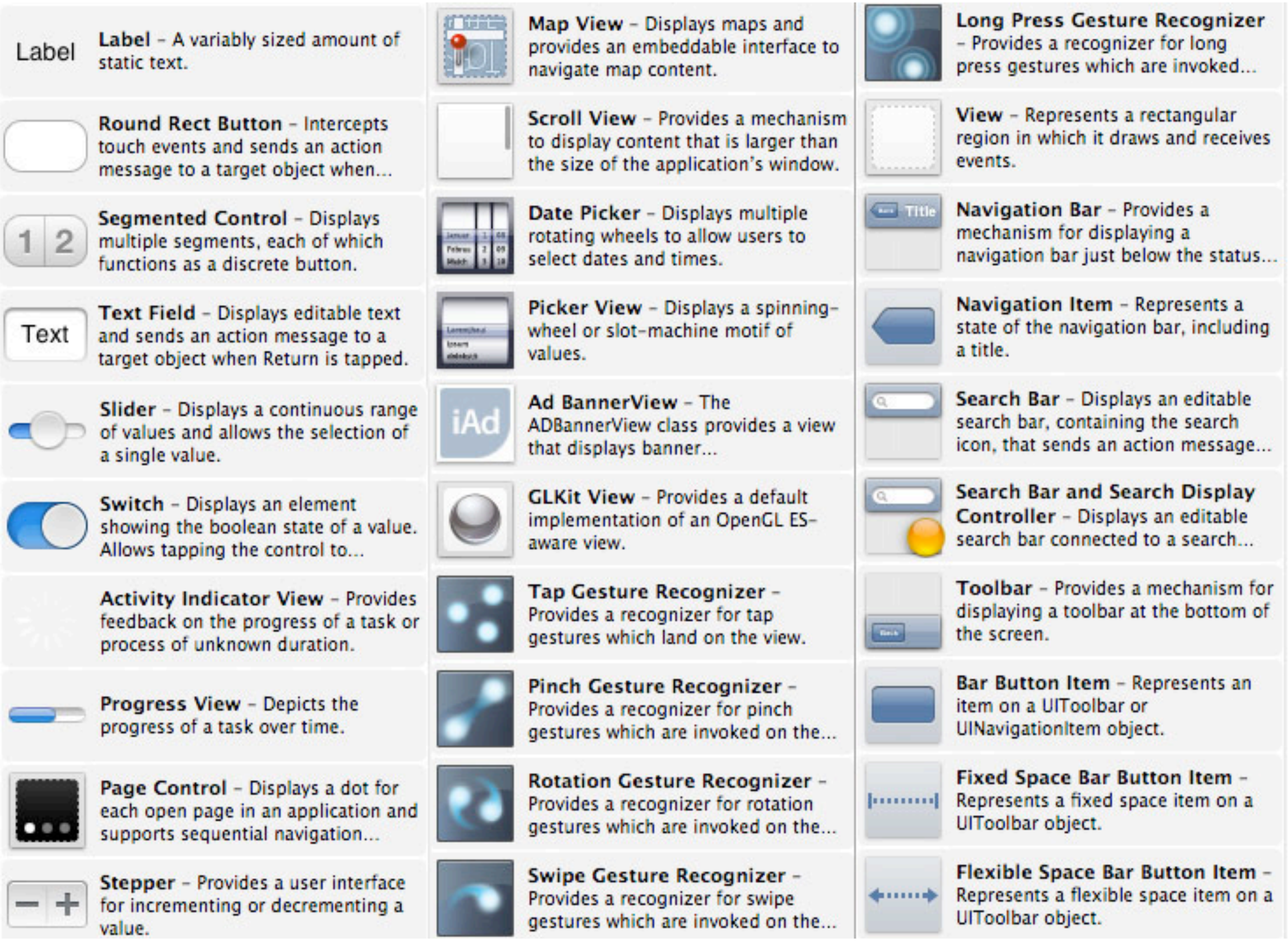
Folders ● Create groups for any added folders
○ Create folder references for any added folders

Add to targets ☑ Assignment1

- You can change properties using the panel on the right

Reference

Interface Builder Elements



Label – A variably sized amount of static text.

Round Rect Button – Intercepts touch events and sends an action message to a target object when...

Segmented Control – Displays multiple segments, each of which functions as a discrete button.

Text Field – Displays editable text and sends an action message to a target object when Return is tapped.

Slider – Displays a continuous range of values and allows the selection of a single value.

Switch – Displays an element showing the boolean state of a value. Allows tapping the control to...

Activity Indicator View – Provides feedback on the progress of a task or process of unknown duration.

Progress View – Depicts the progress of a task over time.

Page Control – Displays a dot for each open page in an application and supports sequential navigation...

Stepper – Provides a user interface for incrementing or decrementing a value.

Map View – Displays maps and provides an embeddable interface to navigate map content.

Scroll View – Provides a mechanism to display content that is larger than the size of the application's window.

Date Picker – Displays multiple rotating wheels to allow users to select dates and times.

Picker View – Displays a spinning-wheel or slot-machine motif of values.

Ad BannerView – The ADBannerView class provides a view that displays banner...

GLKit View – Provides a default implementation of an OpenGL ES-aware view.

Tap Gesture Recognizer – Provides a recognizer for tap gestures which land on the view.

Pinch Gesture Recognizer – Provides a recognizer for pinch gestures which are invoked on the...

Rotation Gesture Recognizer – Provides a recognizer for rotation gestures which are invoked on the...

Swipe Gesture Recognizer – Provides a recognizer for swipe gestures which are invoked on the...

Long Press Gesture Recognizer – Provides a recognizer for long press gestures which are invoked...

View – Represents a rectangular region in which it draws and receives events.

Navigation Bar – Provides a mechanism for displaying a navigation bar just below the status...

Navigation Item – Represents a state of the navigation bar, including a title.

Search Bar – Displays an editable search bar, containing the search icon, that sends an action message...

Search Bar and Search Display Controller – Displays an editable search bar connected to a search...

Toolbar – Provides a mechanism for displaying a toolbar at the bottom of the screen.

Bar Button Item – Represents an item on a UIToolbar or UINavigationItem object.

Fixed Space Bar Button Item – Represents a fixed space item on a UIToolbar object.

Flexible Space Bar Button Item – Represents a flexible space item on a UIToolbar object.

Reference

Interface Builder Elements, cont.

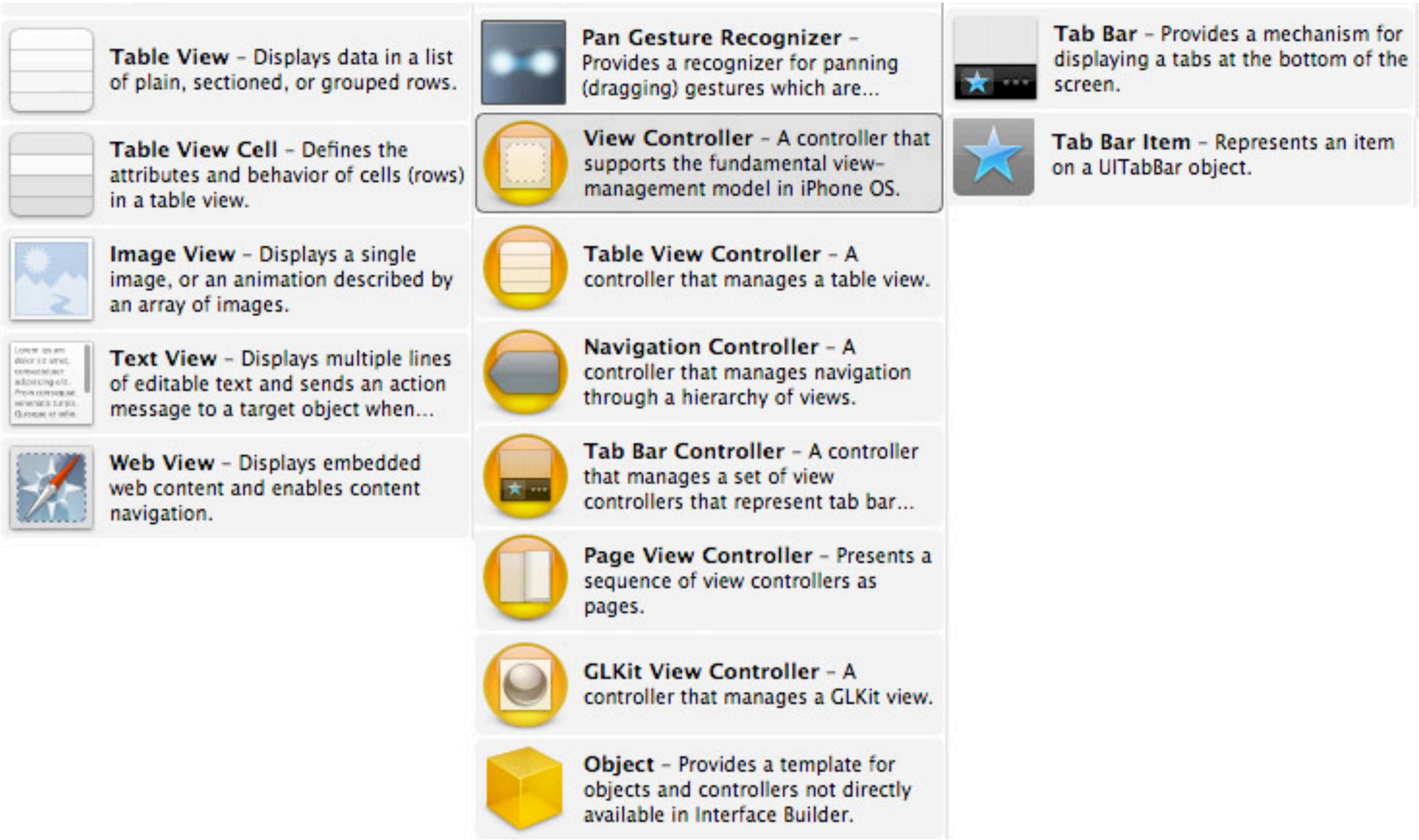


Table View – Displays data in a list of plain, sectioned, or grouped rows.

Table View Cell – Defines the attributes and behavior of cells (rows) in a table view.

Image View – Displays a single image, or an animation described by an array of images.

Text View – Displays multiple lines of editable text and sends an action message to a target object when...

Web View – Displays embedded web content and enables content navigation.

Pan Gesture Recognizer – Provides a recognizer for panning (dragging) gestures which are...

View Controller – A controller that supports the fundamental view-management model in iPhone OS.

Table View Controller – A controller that manages a table view.

Navigation Controller – A controller that manages navigation through a hierarchy of views.

Tab Bar Controller – A controller that manages a set of view controllers that represent tab bar...

Page View Controller – Presents a sequence of view controllers as pages.

GLKit View Controller – A controller that manages a GLKit view.

Object – Provides a template for objects and controllers not directly available in Interface Builder.

Tab Bar – Provides a mechanism for displaying a tabs at the bottom of the screen.

Tab Bar Item – Represents an item on a UITabBar object.

Objective-C

Part 1: What it is, Types and Operators, Logging, and Debugging

What is Objective-C?

- A superset of C, a 40+ year-old powerful programming language
- Any C code is also valid Objective-C code
- Adds classes and objects (along with much more)
- The primary language for iOS and Mac OS development
- Used by Next for the NextStep operating system, which was then purchased by Apple and became OS X

Why did Apple choose it?

- It’s fast
- It’s powerful
- It’s (fairly) easy to read

What does it look like?

	Objective-C	JavaScript
Variable assignment	`int x = 5 + 3;`	`var x = 5 + 3;`
Creating a string	`NSString *name = @”Dimitri”;`	`var name = “Dimitri”;`
Calling a property on a class	`person.name;`	`person.name;`
Calling a method on a class	`[joe addChild:mary];`	`joe.addChild(mary);`

Types

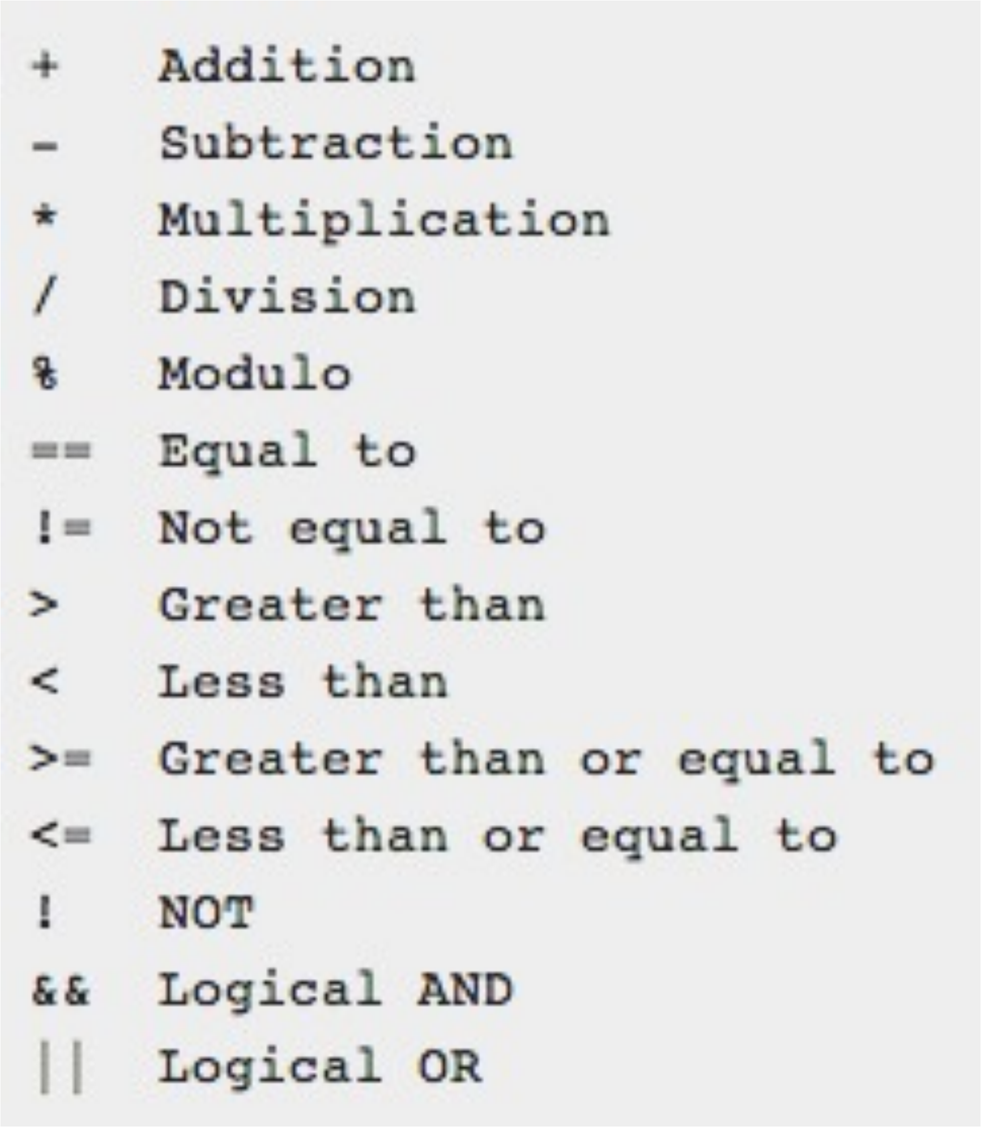
- Objective-C, like C, requires *Static Types* for basic types
- This means that you will need to tell the compiler what a variable holds
    - Declare a variable: int x;
    - Use that variable: x = 12;
- We are letting the compiler know that we are going to store an integer number in the variable x.

More types

- The most common ones we will use will be int, float, and BOOL
- You can declare a variable and assign it in one line:
  - `int x = 4;`
- If you try to store a variable in a type it doesn’t expect, strange things happen:
  - `int x = 0.9; // x will be 0`
  - `int x = YES; // x will be 1`
  - `int x = @”yellow”; // your app will crash (but it will warn you first)`

Operators

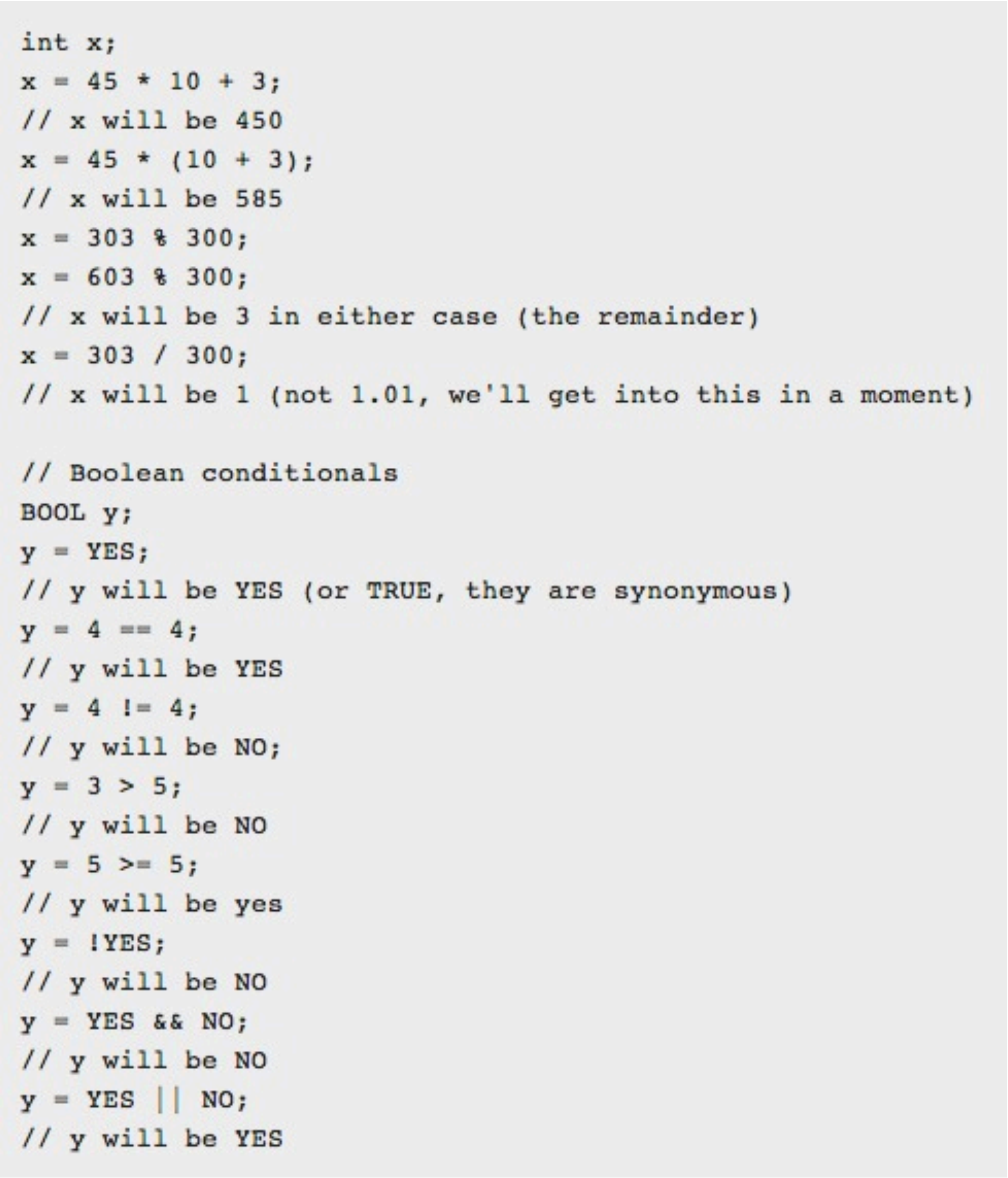
- Operators come from C, here are some common ones:



+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
!	NOT
&&	Logical AND
||	Logical OR

Reference

Operator Reference



```
int x;
x = 45 * 10 + 3;
// x will be 450
x = 45 * (10 + 3);
// x will be 585
x = 303 % 300;
x = 603 % 300;
// x will be 3 in either case (the remainder)
x = 303 / 300;
// x will be 1 (not 1.01, we'll get into this in a moment)

// Boolean conditionals
BOOL y;
y = YES;
// y will be YES (or TRUE, they are synonymous)
y = 4 == 4;
// y will be YES
y = 4 != 4;
// y will be NO;
y = 3 > 5;
// y will be NO
y = 5 >= 5;
// y will be yes
y = !YES;
// y will be NO
y = YES && NO;
// y will be NO
y = YES || NO;
// y will be YES
```

A tricky problem

```
 int x = 1;
x = x + 1;
float y = 2.0;
float result = x / y;
```

What is the value of `result`?

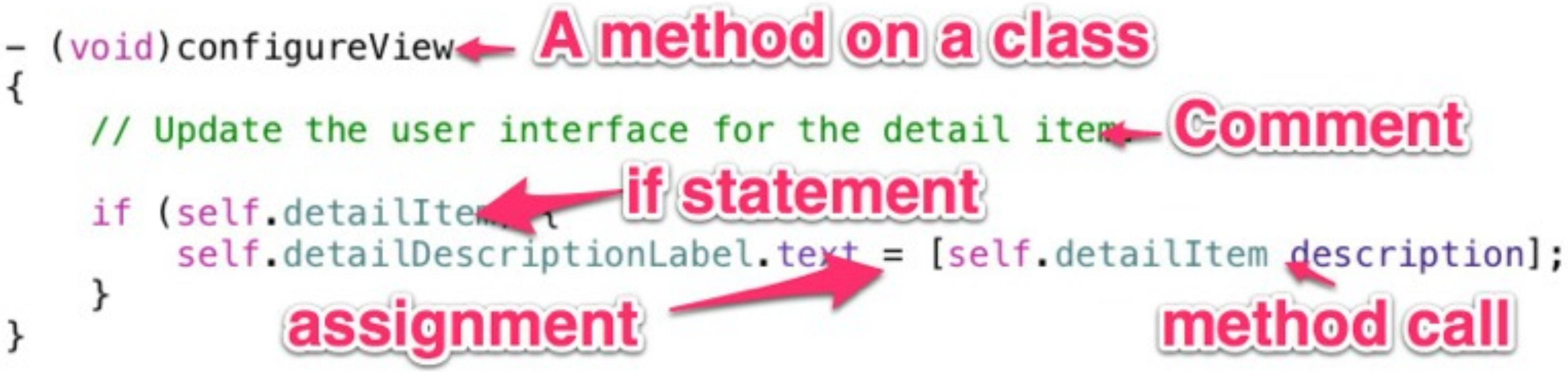
a) .5 (1/2)
b) 0
c) 1

Sample

```
- (void)configureView
{
    // Update the user interface for the detail item.

    if (self.detailItem) {
        self.detailDescriptionLabel.text = [self.detailItem description];
    }
}
```

Sample



```
- (void)configureView
{
    // Update the user interface for the detail item.

    if (self.detailItem) {
        self.detailDescriptionLabel.text = [self.detailItem description];
    }
}
```

A method on a class

Comment

if statement

assignment

method call

First lines of Objective-C

Goal

- Create a UILabel with the price of a meal displayed
- The price of a meal will be a calculation we write in code.

Setup

1. Create the UILabel in the Storyboard
2. Click on the DetailViewController below the view
3. Drag a connection from **detailDescriptionLabel** to your new UILabel
4. Build & Run. Your label should show with the date in it.

Let’s get coding!

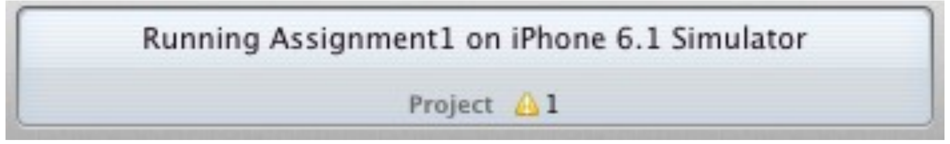
- Open DetailViewController.m
- Modify the configureView method
- Create a local variable to hold the price of dinner (for now lets just set it to 42.09)
- Change the line where the date is being set:

  -
    ```
    self.detailDescriptionLabel.text = [self.detailItem description];
    ```
- to

  -
    ```
    self.detailDescriptionLabel.text = [NSString stringWithFormat:@"$%.2f",dinnerPrice];
    ```
- Don’t worry about the details of this line yet, we’ll get to that later!

Testing your code

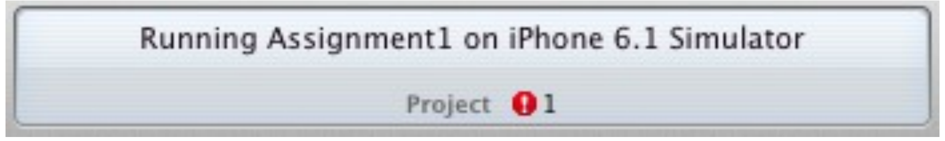
- Build and Run
- Your build should be error and warning free.



Running Assignment1 on iPhone 6.1 Simulator

Project ⚠ 1

Warnings



Running Assignment1 on iPhone 6.1 Simulator

Project ● 1

Errors

- You should see the value $42.09 on your UILabel

Calculating the price of dinner

Goal Calculate the price of dinner for 4 people.

Rules:

- Entrees cost $21.50, and each person buys one
- Appetizers cost $8.00, and 2 people split one
- Wine costs $43.00, and 4 people split a bottle
- Dessert costs $4.75, and each person buys one
- The restaurant does not sell half bottles of wine
- Express all prices and counts as variables

Methods

- Methods are similar to functions, but are performed on a class

- They are used to enhance readability and reduce repetition in code.

- They encapsulate functionality into one place.

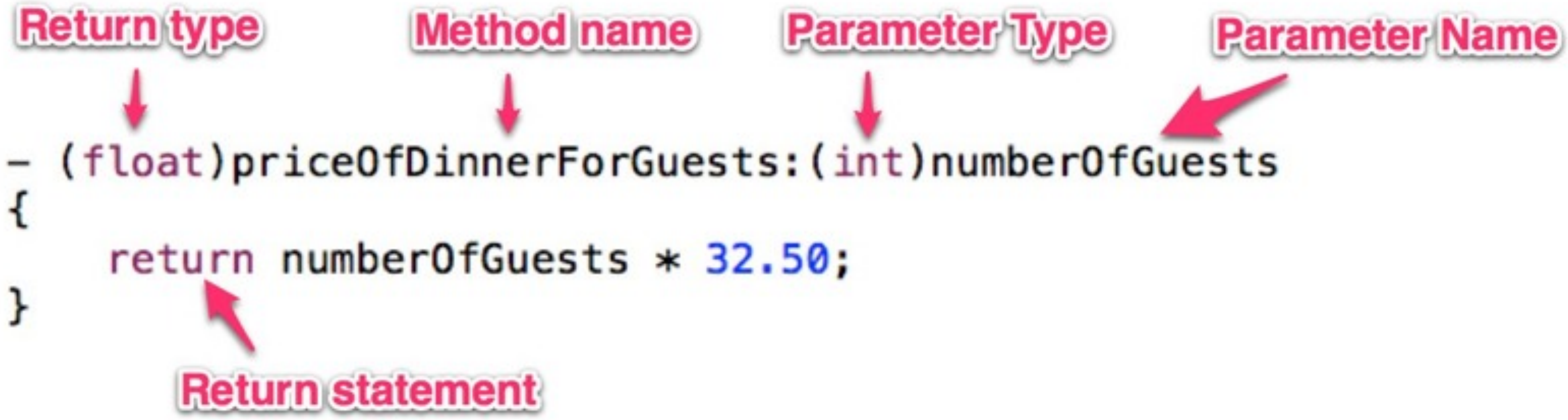
Methods

- What if we wanted to change the number of people coming to dinner?

- What if we wanted to expose a control that the user could change that number?

- We use methods with parameters to be able to accomplish these goals

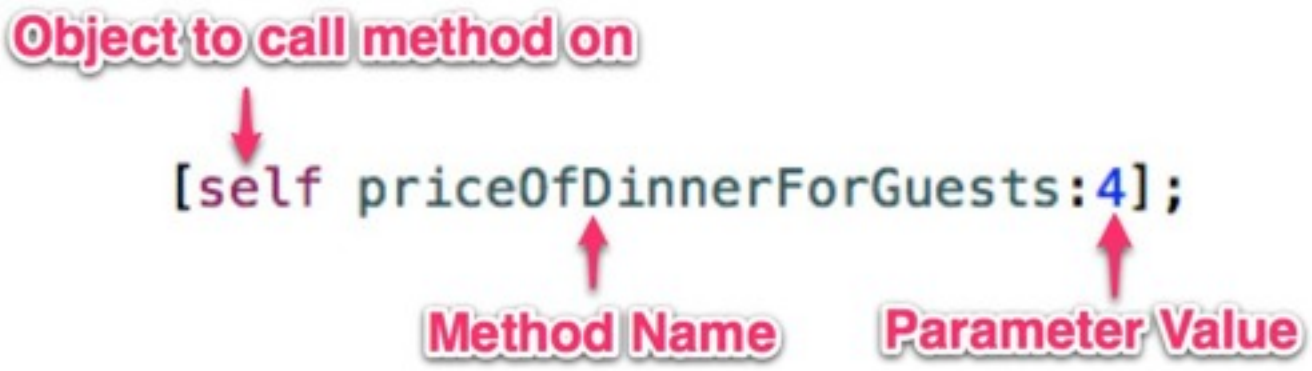
Writing a method



```
- (float)priceOfDinnerForGuests:(int)numberOfGuests
{
    return numberOfGuests * 32.50;
}
```

- The number of guests is an int, because you can’t have 3.4 guests.
- The return price is a float, since your dinner can cost $97.50.

Calling a Method



```
[self priceOfDinnerForGuests:4];
```

- `self` is a special keyword which simply means "this object", in this case the DetailViewController object.

Calculating the price of dinner

Goal Create a method to calculate dinner for an arbitrary number of people, and call it with 2, 4, 5 and 6, checking your answer each time.

when you are done...

Goal Add 20% tip and 8.875% tax to the price of dinner

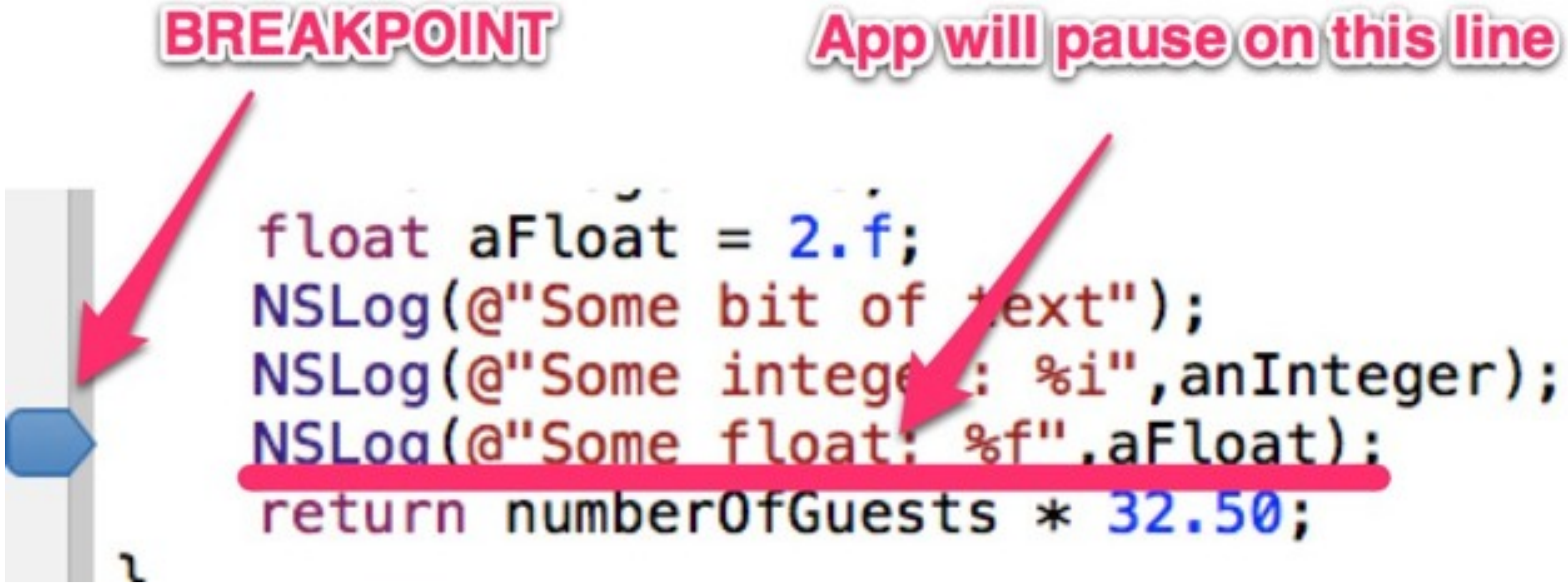
Logging

- It's often useful to be able to see the result of some calculation without putting it on the screen
- To do this, we call the function (it's not a method) `NSLog`

```
NSLog(@"Some bit of text");
NSLog(@"Some integer: %i",anInteger);
NSLog(@"Some float: %f",aFloat);
```

Debugging and Breakpoints

- It’s also useful to be able to stop your app at a specific point to see what’s wrong.
- You need a **Breakpoint**.
- To make, just click in the gutter next to your code



BREAKPOINT

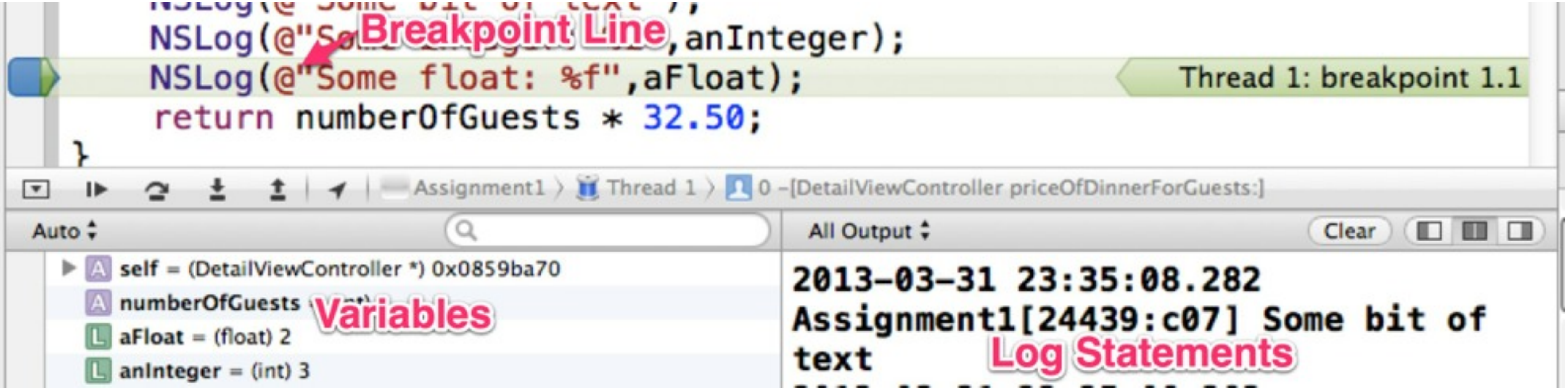
App will pause on this line

```
    float aFloat = 2.f;
    NSLog(@"Some bit of text");
    NSLog(@"Some integer: %i",anInteger);
    NSLog(@"Some float: %f",aFloat);
    return numberOfGuests * 32.50;
}
```

Breakpoints

- Beware you don’t set them accidentally!
- Your app will appear to be frozen
    - You should check Xcode if this happens
- In the bottom pane you can inspect the current value of all of your variables

Debugging



```
    NSLog(@"Some bit of text");
    NSLog(@"Some integer: %i",anInteger);
    NSLog(@"Some float: %f",aFloat);
    return numberOfGuests * 32.50;
}
```

Thread 1: breakpoint 1.1

Assignment1 › Thread 1 › 0 -[DetailViewController priceOfDinnerForGuests:]

```
self = (DetailViewController *) 0x0859ba70
numberOfGuests = (int)
aFloat = (float) 2
anInteger = (int) 3
```

```
2013-03-31 23:35:08.282
Assignment1[24439:c07] Some bit of
text
```

Breakpoint Line

Variables

Log Statements

General Tips

- Take it slow!
- Don’t move on until you correctly complete where you are
- Build as frequently as you can
- Never let errors or warnings linger: fix them!

Language Tips

- All statements must end with a semi-colon
- All parentheses and curly braces must match
- Don’t forget the @ sign before a string: `@”test”`
- Make sure you understand what types you are using and why

Assignment

- Ensure you have an error-free and warning free app
- Your detail screen should show both static text and images similar to the example, but feel free to get creative
- Your detail screen should show the correct price of dinner for 4 people, or for 5 or 6 if you choose, including tip & tax
- You should be able to debug a running application to help narrow down where a problem is occurring

Additional Assignment

- Compile a list of questions that you have for next class.
- We’ll review some of these questions at the beginning of class and discuss them as a group

Turn In

- Zip your Assignment1 folder and email to:

  stanciod@newschool.edu

  DUE: Before class on Monday

  Bring your questions to class next week!

Optional Work

These aren't required, and will take some research to be able to answer

- Try adding more labels to the view, and populating them from the result of other methods.
- Try writing a method that takes 2 parameters.
  - Hint: `[self insertItem:item atIndex:index]`

Next Class

Objects and Classes