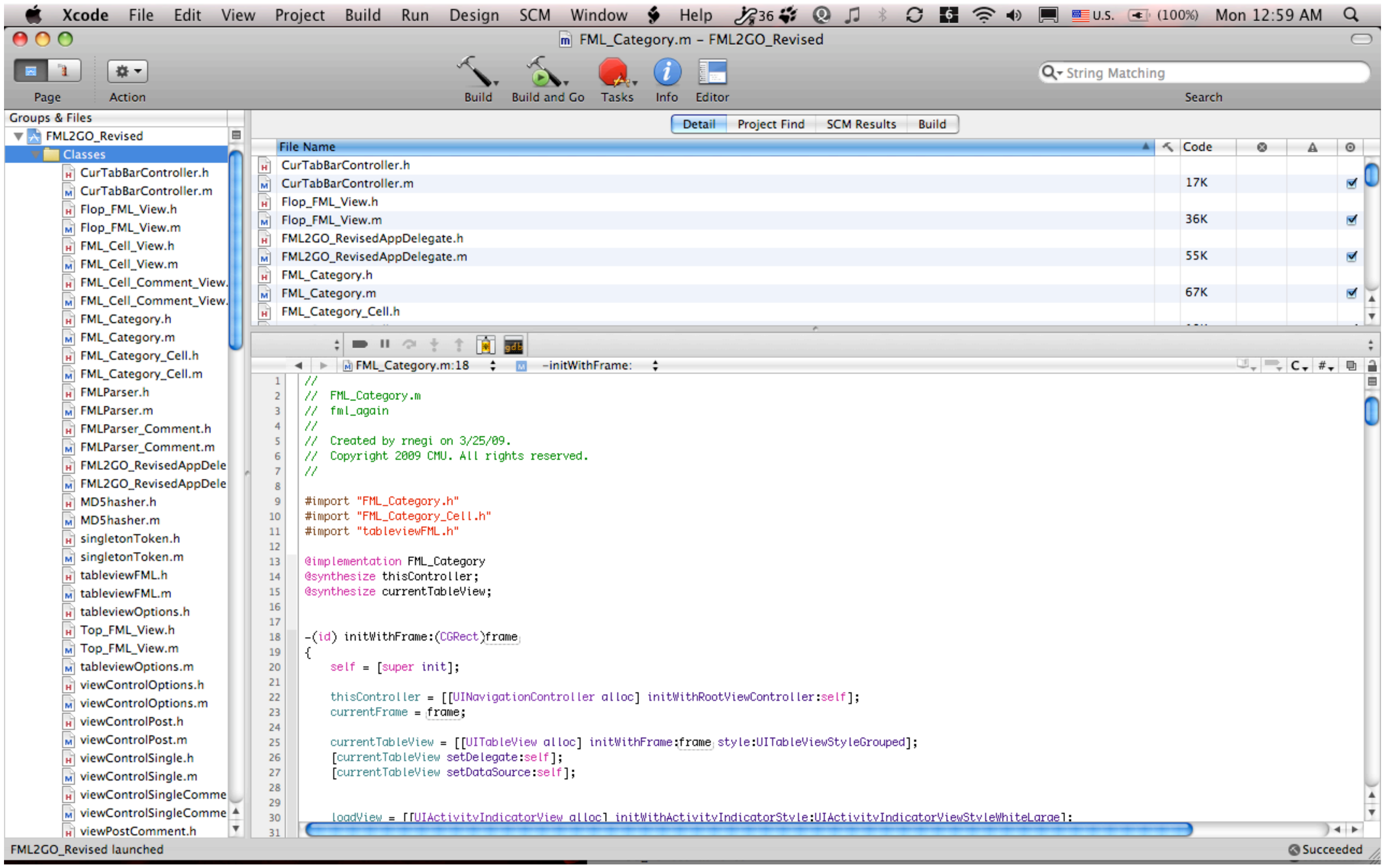
iPhone Development Basics

Do You have a Mac?

Basics for iPhone

- Need -> OS X operating system.
- Either a real Mac or a system that has OS X installed on it
- Need the iPhone SDK
- http://developer.apple.com
- This gives you the simulator and X Code installed on your computer

X Code



FML2GO_Revised
Classes
CurTabBarController.h
CurTabBarController.m
Flop_FML_View.h
Flop_FML_View.m
FML_Cell_View.h
FML_Cell_View.m
FML_Cell_Comment_View.
FML_Cell_Comment_View.
FML_Category.h
FML_Category.m
FML_Category_Cell.h
FML_Category_Cell.m
FMLParser.h
FMLParser.m
FMLParser_Comment.h
FMLParser_Comment.m
FML2GO_RevisedAppDele
FML2GO_RevisedAppDele
MD5hasher.h
MD5hasher.m
singletonToken.h
singletonToken.m
tableviewFML.h
tableviewFML.m
tableviewOptions.h
Top_FML_View.h
Top_FML_View.m
tableviewOptions.m
viewControlOptions.h
viewControlOptions.m
viewControlPost.h
viewControlPost.m
viewControlSingle.h
viewControlSingle.m
viewControlSingleComme
viewControlSingleComme
viewPostComment.h

File Name	Code			✓
CurTabBarController.h				
CurTabBarController.m	17K			✓
Flop_FML_View.h				
Flop_FML_View.m	36K			✓
FML2GO_RevisedAppDelegate.h				
FML2GO_RevisedAppDelegate.m	55K			✓
FML_Category.h				
FML_Category.m	67K			✓
FML_Category_Cell.h				

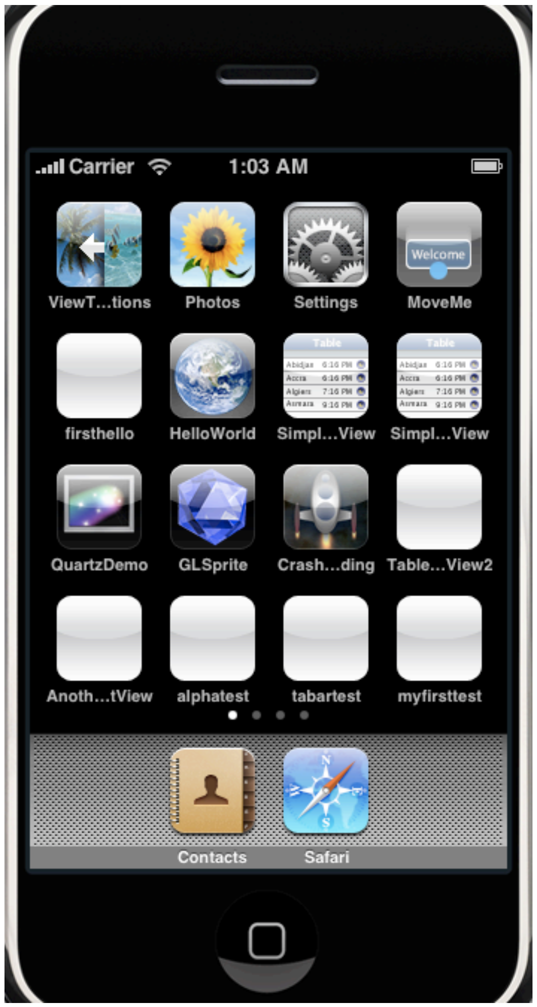
FML_Category.m:18   -initWithFrame:

```
//
//  FML_Category.m
//  fml_again
//
//  Created by rnegi on 3/25/09.
//  Copyright 2009 CMU. All rights reserved.
//

#import "FML_Category.h"
#import "FML_Category_Cell.h"
#import "tableviewFML.h"

@implementation FML_Category
@synthesize thisController;
@synthesize currentTableView;


-(id) initWithFrame:(CGRect)frame
{
    self = [super init];

    thisController = [[UINavigationController alloc] initWithRootViewController:self];
    currentFrame = frame;

    currentTableView = [[UITableView alloc] initWithFrame:frame style:UITableViewStyleGrouped];
    [currentTableView setDelegate:self];
    [currentTableView setDataSource:self];


    loadView = [[UIActivityIndicatorView alloc] initWithActivityIndicatorStyle:UIActivityIndicatorViewStyleWhiteLarge];
```

FML2GO_Revised launched

Succeeded

Iphone Simulator



←iPhone Simulator
-Looks like an iPhone
-Runs like an iPhone

-Not an iPhone
-Limited in certain areas
-Decieving

iPhone Simulator Limitations

- No Camera
- None. You cannot use the Mac webcam to take pictures.
- You are limited to the iPhone's photo library to process images
- Camera requires real hardware

iPhone Sim Limitations More

- The simulator is very fast. It runs faster than your iPhone
- Not as picky as real hardware
- Harder to process multi touches in simulator

iPhone Hardware

- Selections between 2G (First Gen), 3G (Second Gen), and 3GS(Second ½ Gen)
- BIG differences in the hardware limitations
- What will run well on the 3GS means nothing for the previous iPhones

iPhone Hardware continued

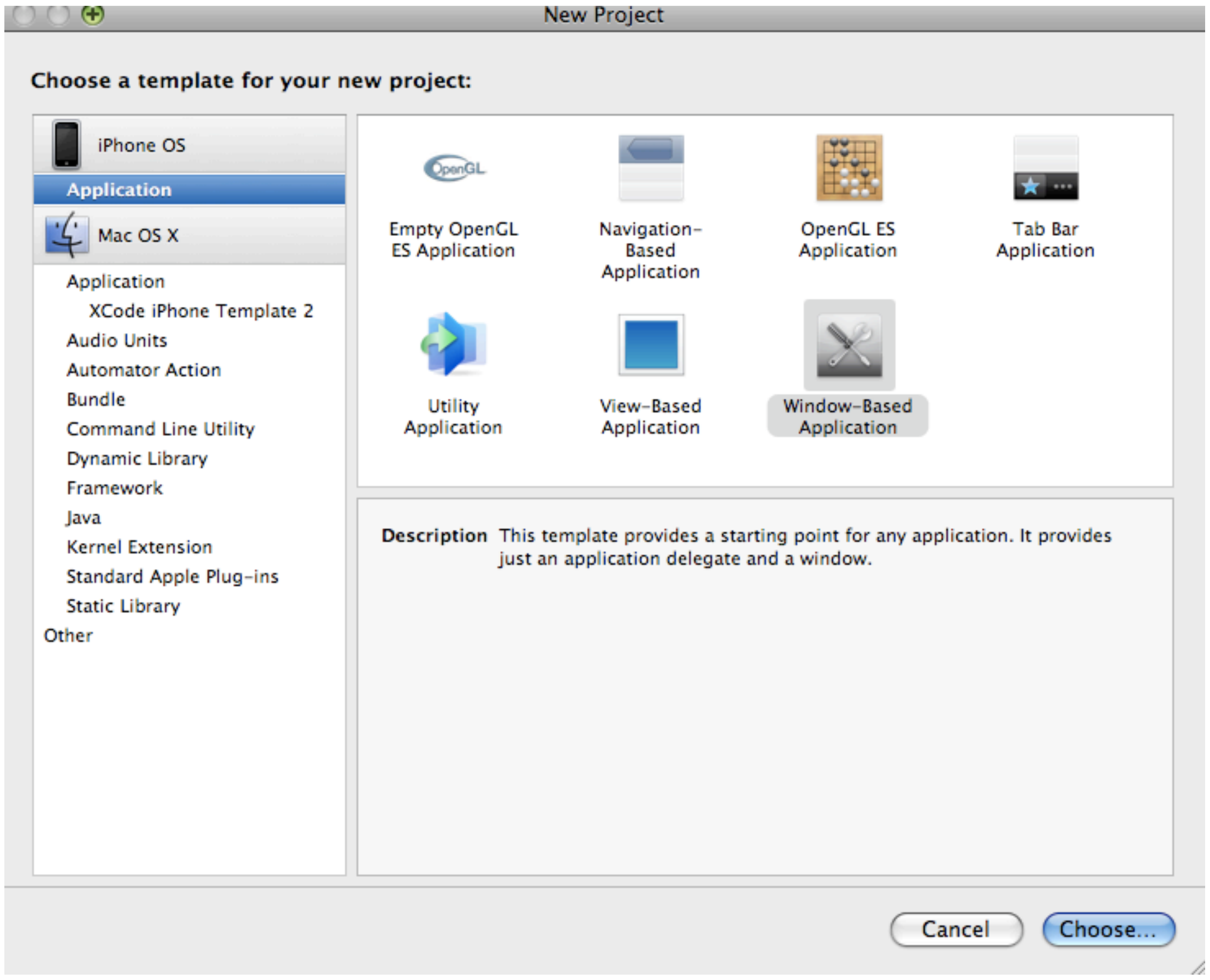
- iPod touches have no camera. But otherwise same conditions apply.
- First Gen iPod Touches have no bluetooth or speaker

Objective-C Basics

- It’s like a cross between java and C.
- Theres objects like java but C like syntax.

- Objective-C is SUPER-SET of C
  - You can write pure C code for iPhone Apps
  - Problem: Will need to write Objective-C wrappers to do conversion between C data and Objective-C objects
  - Luckily Objective-C has many objects available for you to use

Objective-C Basics



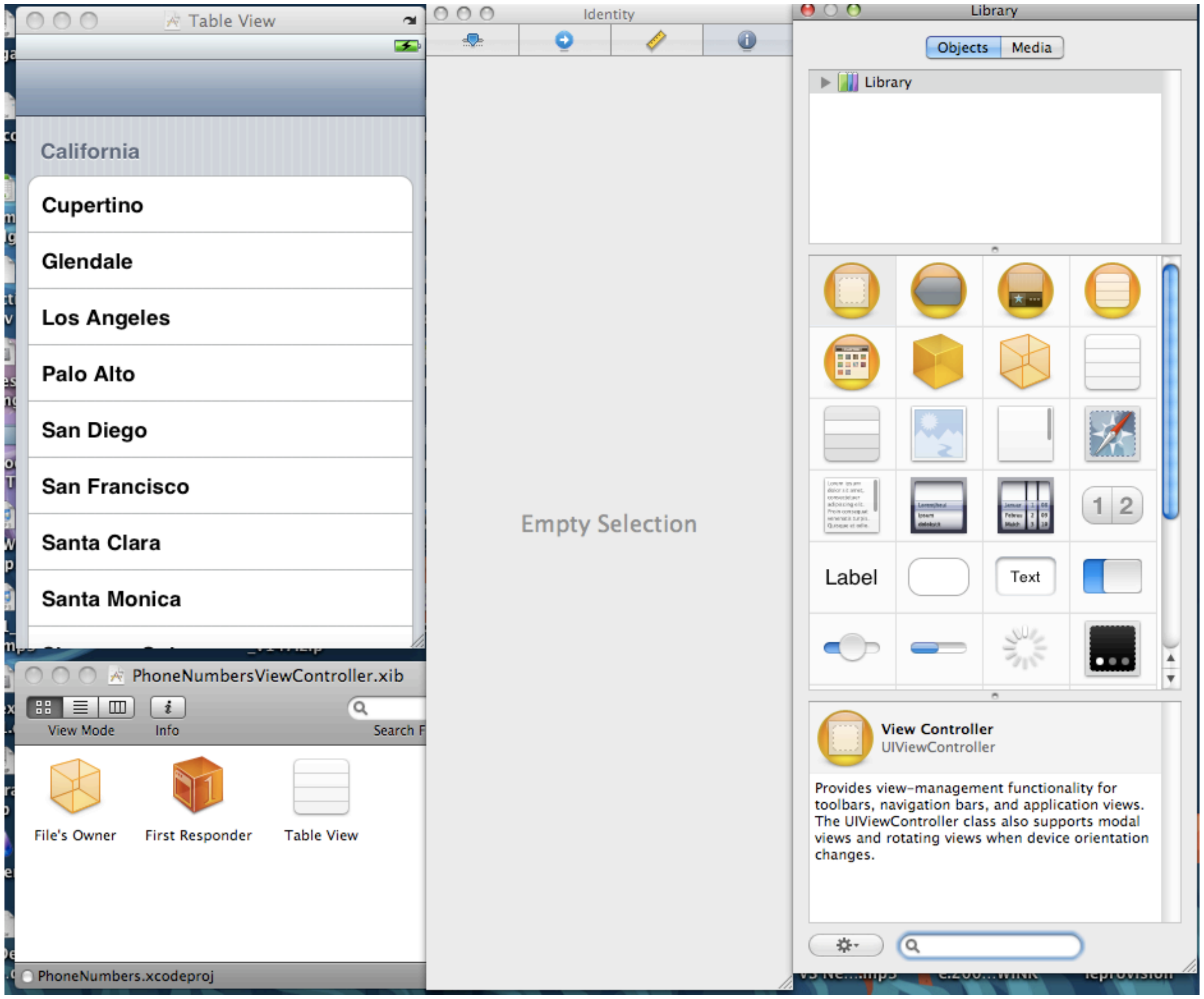
New Project

Choose a template for your new project:

- iPhone OS
  - Application
- Mac OS X
  - Application
    - XCode iPhone Template 2
  - Audio Units
  - Automator Action
  - Bundle
  - Command Line Utility
  - Dynamic Library
  - Framework
  - Java
  - Kernel Extension
  - Standard Apple Plug-ins
  - Static Library
- Other

Description This template provides a starting point for any application. It provides just an application delegate and a window.

Interface Builder



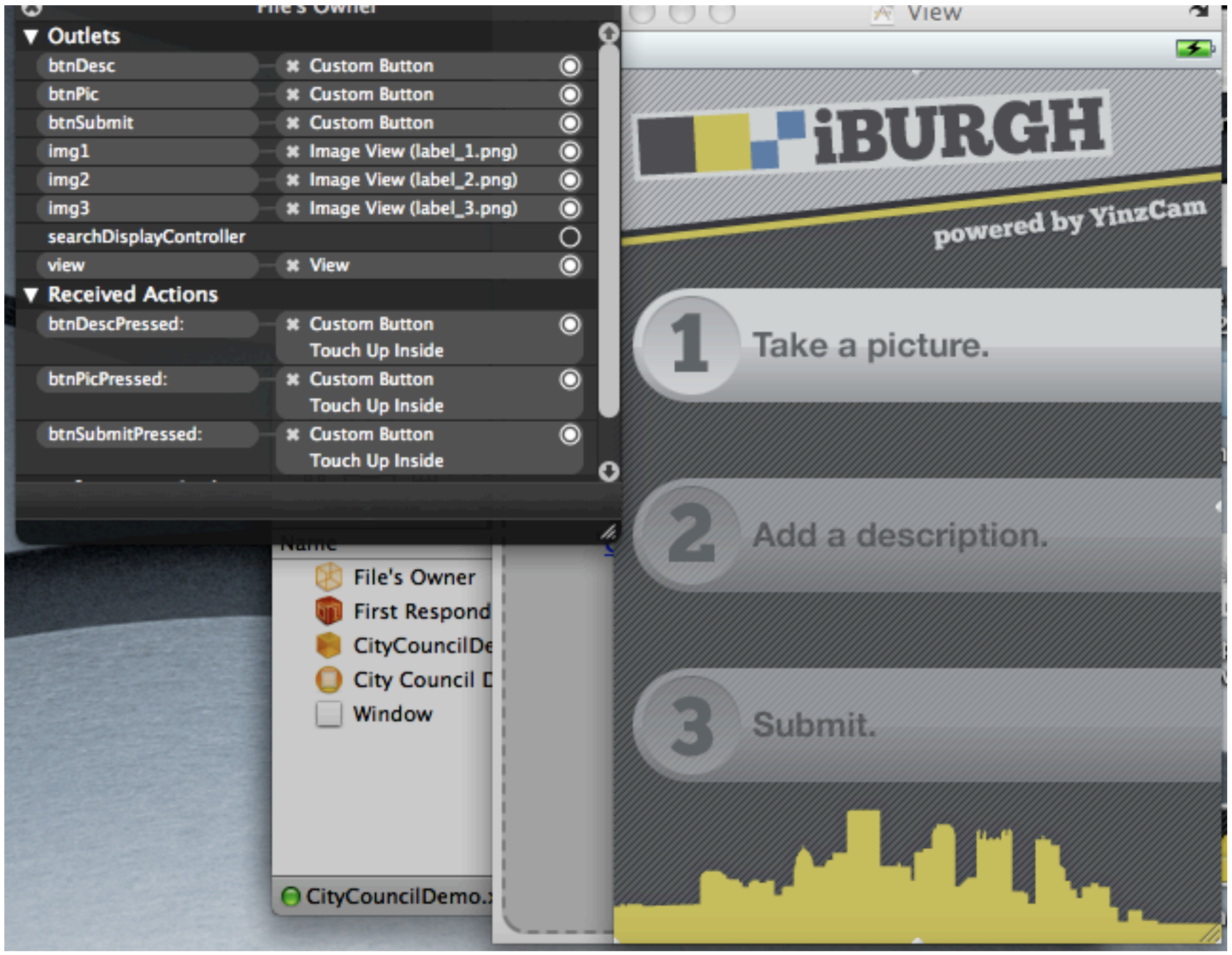
Provides view-management functionality for toolbars, navigation bars, and application views. The UIViewController class also supports modal views and rotating views when device orientation changes.

Interface Builder

- Very easy to use and powerful.
- Almost all the default Apple GUI elements included
- Easily integrated with code via ‘IBOutlet’ connections

X Code and Interface Builder

```
// UI components
IBOutlet UIButton *btnPic;
BOOL btnPic_selected;
IBOutlet UIButton *btnDesc;
BOOL btnDesc_selected;
IBOutlet UIButton *btnSubmit;
BOOL btnSubmit_selected;

IBOutlet UIImageView *img1;
IBOutlet UIImageView *img2;
IBOutlet UIImageView *img3;
```



File's Owner

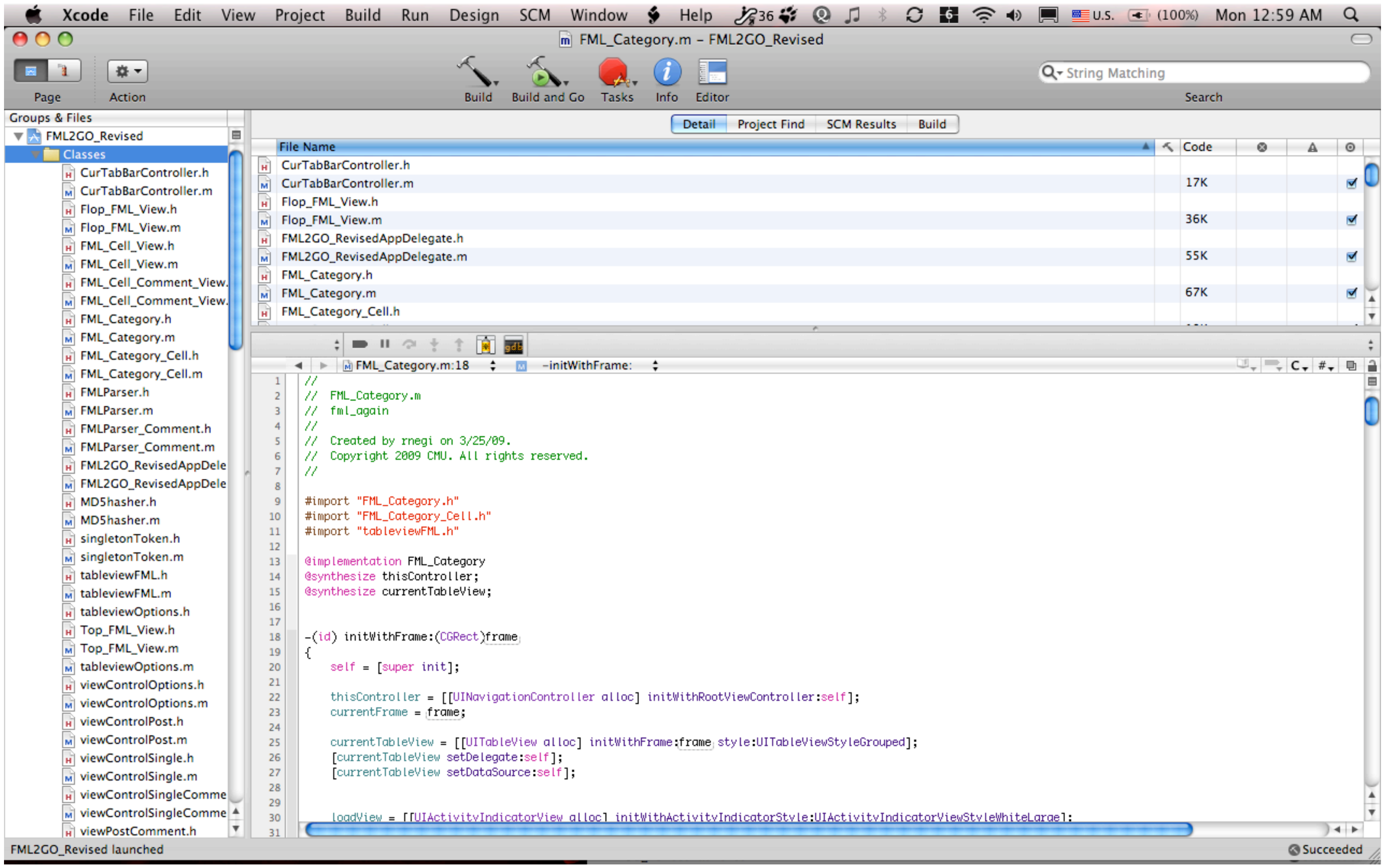
▼ Outlets

btnDesc	Custom Button
btnPic	Custom Button
btnSubmit	Custom Button
img1	Image View (label_1.png)
img2	Image View (label_2.png)
img3	Image View (label_3.png)
searchDisplayController	
view	View

▼ Received Actions

btnDescPressed:	Custom Button Touch Up Inside
btnPicPressed:	Custom Button Touch Up Inside
btnSubmitPressed:	Custom Button Touch Up Inside

X Code



Groups & Files

- FML2GO_Revised
  - Classes
    - CurTabBarController.h
    - CurTabBarController.m
    - Flop_FML_View.h
    - Flop_FML_View.m
    - FML_Cell_View.h
    - FML_Cell_View.m
    - FML_Cell_Comment_View.
    - FML_Cell_Comment_View.
    - FML_Category.h
    - FML_Category.m
    - FML_Category_Cell.h
    - FML_Category_Cell.m
    - FMLParser.h
    - FMLParser.m
    - FMLParser_Comment.h
    - FMLParser_Comment.m
    - FML2GO_RevisedAppDele
    - FML2GO_RevisedAppDele
    - MD5hasher.h
    - MD5hasher.m
    - singletonToken.h
    - singletonToken.m
    - tableviewFML.h
    - tableviewFML.m
    - tableviewOptions.h
    - Top_FML_View.h
    - Top_FML_View.m
    - tableviewOptions.m
    - viewControlOptions.h
    - viewControlOptions.m
    - viewControlPost.h
    - viewControlPost.m
    - viewControlSingle.h
    - viewControlSingle.m
    - viewControlSingleComme
    - viewControlSingleComme
    - viewPostComment.h

File Name	Code
CurTabBarController.h	
CurTabBarController.m	17K
Flop_FML_View.h	
Flop_FML_View.m	36K
FML2GO_RevisedAppDelegate.h	
FML2GO_RevisedAppDelegate.m	55K
FML_Category.h	
FML_Category.m	67K
FML_Category_Cell.h	

```
//
//  FML_Category.m
//  fml_again
//
//  Created by rnegi on 3/25/09.
//  Copyright 2009 CMU. All rights reserved.
//

#import "FML_Category.h"
#import "FML_Category_Cell.h"
#import "tableviewFML.h"

@implementation FML_Category
@synthesize thisController;
@synthesize currentTableView;


-(id) initWithFrame:(CGRect)frame
{
    self = [super init];

    thisController = [[UINavigationController alloc] initWithRootViewController:self];
    currentFrame = frame;

    currentTableView = [[UITableView alloc] initWithFrame:frame style:UITableViewStyleGrouped];
    [currentTableView setDelegate:self];
    [currentTableView setDataSource:self];


    loadView = [[UIActivityIndicatorView alloc] initWithActivityIndicatorStyle:UIActivityIndicatorViewStyleWhiteLarge];
```

FML2GO_Revised launched

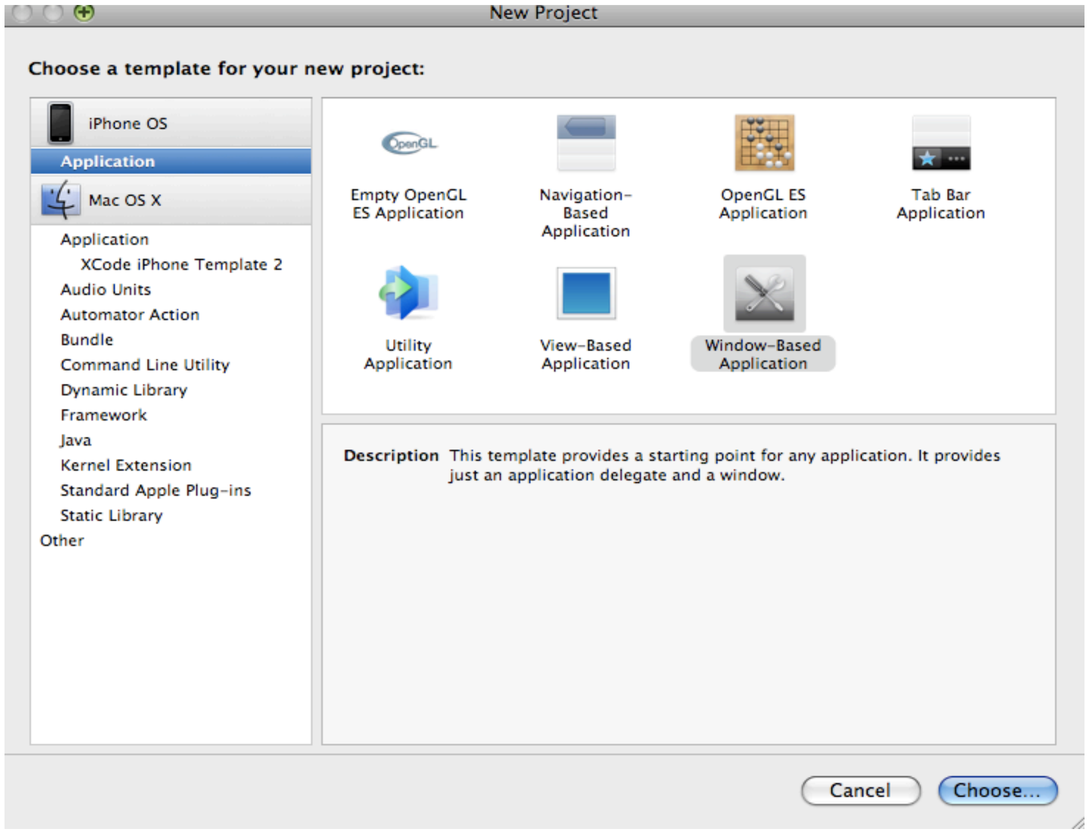
Succeeded

X Code & Objective-C

- Similar file definitions
  - .h for header files
  - .m for Objective-C files
  - Will compile *.c files as well

Creating a new project

- Open up XCode, Select File->New Project



Description This template provides a starting point for any application. It provides just an application delegate and a window.

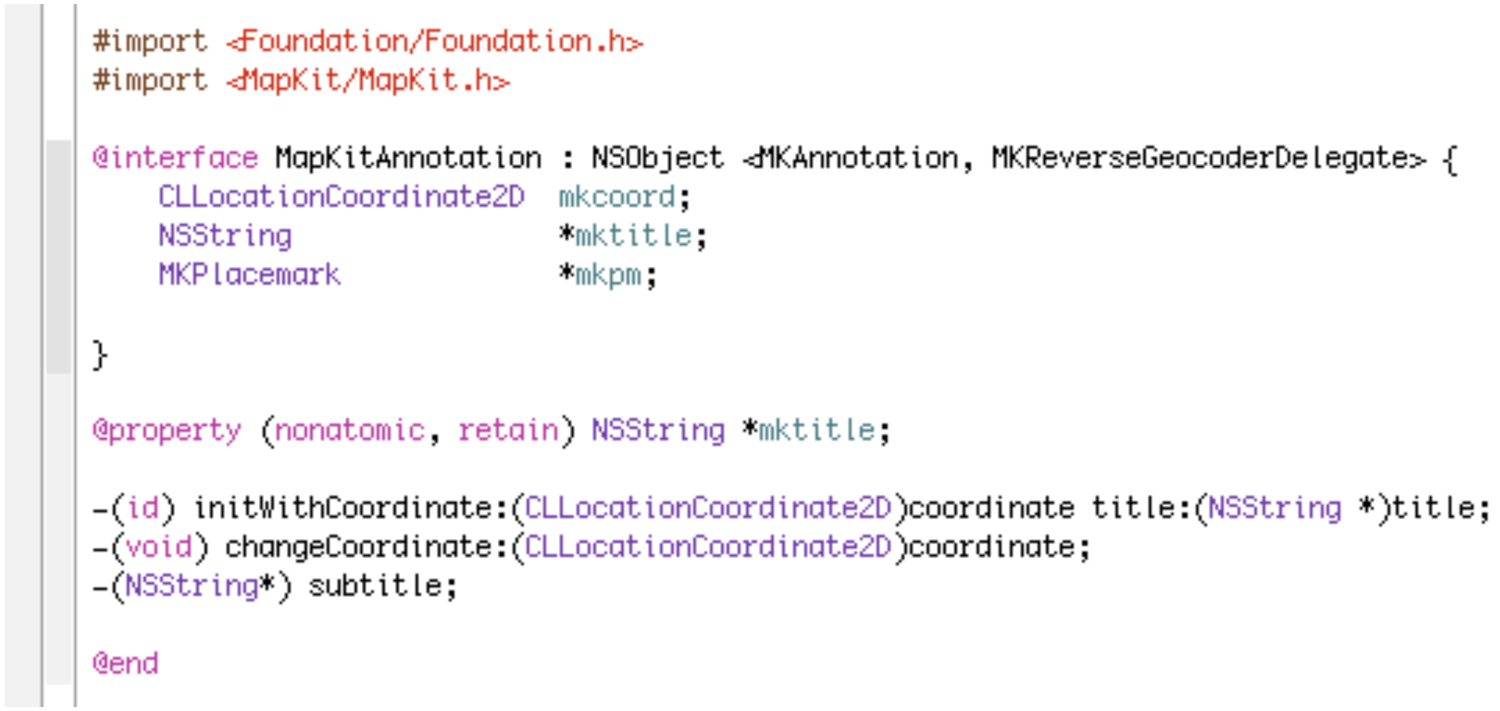
Objective-C basics

- C’s #include is Objective-C’s #import
  - Like Java but with C syntax
  - i.e. #import “MyheaderFile.h”

  - Classes are outlined in header files as such
    - @interface CLASS_NAME : SUPER_CLASS

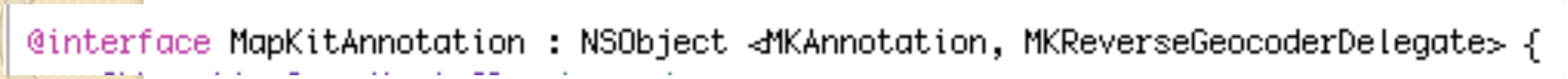
      //fill stuff in here with variables and types
      @end

Similar too…



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

@interface MapKitAnnotation : NSObject <MKAnnotation, MKReverseGeocoderDelegate> {
    CLLocationCoordinate2D  mkcoord;
    NSString                *mktitle;
    MKPlacemark             *mkpm;

}

@property (nonatomic, retain) NSString *mktitle;

-(id) initWithCoordinate:(CLLocationCoordinate2D)coordinate title:(NSString *)title;
-(void) changeCoordinate:(CLLocationCoordinate2D)coordinate;
-(NSString*) subtitle;

@end
```

Confused?



```
@interface MapKitAnnotation : NSObject <MKAnnotation, MKReverseGeocoderDelegate> {
```

- Note: @interface <Class_Name>
- The <Class_Name> : NSObject indicates you are subclassing NSObject
- Technically all Classes you write will subclass somethin
- NSObject <stuff?> - this indicates what delegate methods this class will implement/override

Delegates?

- Everytime you want to write your own custom subclass of something Apple already wrote it's really easy.
- Just override the delegate methods needed by the class with your own versions
- Like Java overloading

@property?

- @property (nonatomic, retain) <Class_type> <class_name>
  - i.e. @property (nonatomic, retain) NSString *astring;
  - @property indicates you are setting properties for this variable
  - The (nonatomic, retain) indicate type of properties to set.
    - MORE in Apple documentation.
    - (nonatomic, retain) is one of the most common ones

  - By properties I mean GET/SET methods for these values.
  - Objective-C is a OOP language -> prefers you to use get/set methods to assign/read values.

@property?

- @property (#properties you set#) sets the properties for the variable.
- Objective-C allows get/set methods to be generated automatically for variables
- Such as: NSString *mystring;
- Set property: @property (nonatomic, retain) NSString* mystring
- This says to the compiler: set GET/SET methods for mystring and keep the variable in memory until I release it
- Require a corresponding @synthesize mystring method

@synthesize?

- When you have a @property you must have a corresponding @synthesize in the *.m file to INIT the properties of the value.
- NOTE: @synthesize only inits the properties, not the value itself.
- Still must allocate memory for the value or assign a initial value

Like this:

- Header.h will be like so:
  - #import "someother headerfile.h"

```
@interfaceTest_Class : NSObject
{
    NSString *mystring;
}
@propert (nonatomic, retain) NSString *mystring;
@end
```

Header.m

- @implementation Test_Class
  @synthesize mystring

  //other stuff here

  @end

- Simple right?
- How to initialize the string?
- Most Objective-C objects will require the following initializing code:
  object = [[object_class alloc] init];

How to set value?

- Now if you want to set the value you can just do this
- [self setMystring:@"Hello World"]
- Self is Java's 'this' in reference to the current object class
- If another class holds the object, reference that object with that class's object name instead of self
- Same principles apply to function calling
- [self functionToCall:Parameters];

wait? What? The?

- [object_owner setMystring:@"Hello World"]
- Confused? This is how Objective-C calls functions, in a [ ] fashion.
- Always [object_owner function:<parameters>];

- Object_owner is the owner of the function/value you want to call/set
- Same type of call is used for return values:
  - New_value = [object_owner <value_name>]; will return that value

  - Can be kind of annoying and useful at the same time

Functions

- Header.h will be like so:
  - #import "someother headerfile.h"

```
@interfaceTest_Class : NSObject
{
    NSString *mystring;
}
@propert (nonatomic, retain) NSString *mystring;

- (void) letsprint : (NSString*)astring;
@end
```

Header.c

- Within Header.c between @implementation and @end you fill out the function prototype just like C

-
  ```
  - (void) printstring : (NSString *)astring
  {

          //print the string to the 'console'
          NSLog(@"%@", astring);
  }
  ```

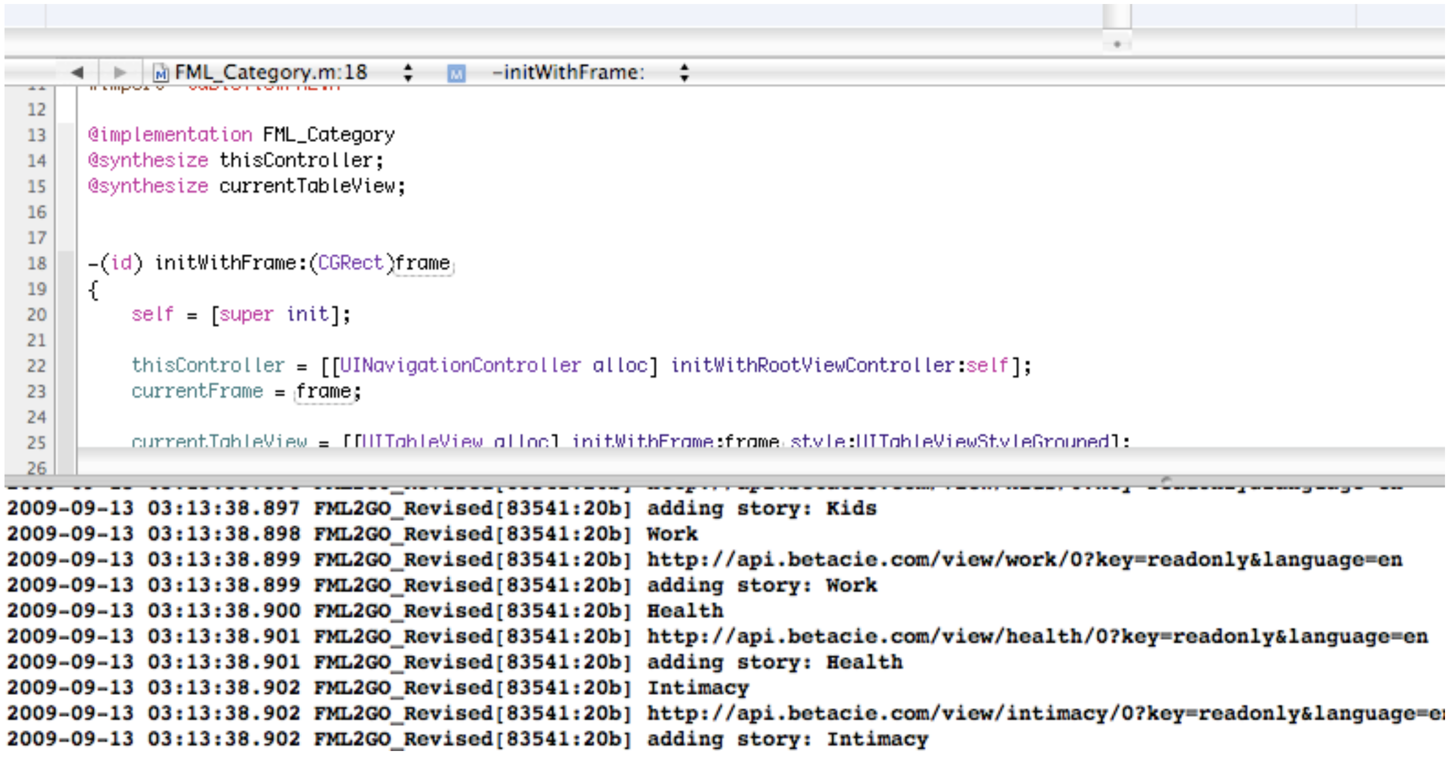
NSLog? %@?

- NSLog is the function that prints to the Xcode console
- iPhone apps are GUI apps so this is the only way to log out information to the GDB console
- %@ is the %s equivalent in Objective-C
- NOTE: ALL STRINGS MUST HAVE A '@' in front

Calling the function

- The function can be called somewhere in another function like thus [self letsprint:@"hey"];

- And the GDB console will print "hey";



```objc

@implementation FML_Category
@synthesize thisController;
@synthesize currentTableView;


-(id) initWithFrame:(CGRect)frame
{
    self = [super init];

    thisController = [[UINavigationController alloc] initWithRootViewController:self];
    currentFrame = frame;

    currentTableView = [[UITableView alloc] initWithFrame:frame style:UITableViewStyleGrouped];
```

```
2009-09-13 03:13:38.897 FML2GO_Revised[83541:20b] adding story: Kids
2009-09-13 03:13:38.898 FML2GO_Revised[83541:20b] Work
2009-09-13 03:13:38.899 FML2GO_Revised[83541:20b] http://api.betacie.com/view/work/0?key=readonly&language=en
2009-09-13 03:13:38.899 FML2GO_Revised[83541:20b] adding story: Work
2009-09-13 03:13:38.900 FML2GO_Revised[83541:20b] Health
2009-09-13 03:13:38.901 FML2GO_Revised[83541:20b] http://api.betacie.com/view/health/0?key=readonly&language=en
2009-09-13 03:13:38.901 FML2GO_Revised[83541:20b] adding story: Health
2009-09-13 03:13:38.902 FML2GO_Revised[83541:20b] Intimacy
2009-09-13 03:13:38.902 FML2GO_Revised[83541:20b] http://api.betacie.com/view/intimacy/0?key=readonly&language=e
2009-09-13 03:13:38.902 FML2GO_Revised[83541:20b] adding story: Intimacy
```

Basic iPhone Development

- Whenever a new project is created.
  - A XXX_XXXAppDelegate.h & *.m file are created.
  - These are executed by the apps main.c (which you will never have to modify hopefully) and displays the initial screen
  - The loading function for every app is:
  - -(void)applicationDidFinishLaunching

Memory?

- Objects are created thus:
- new_object = [[A_CLASS alloc] init];

- To release this later you will call [new_object release];
- PLEASE be careful about memory leaks. On an iPhone can severly impact performance.
- You can usually release everything in an objects
- -(void)dealloc() method. This is the default method called when object is closed or 'released'
- Objective C is not too picky about memory initialization. Won't always crash but your application might not work. Make sure you always alloc and init your objects

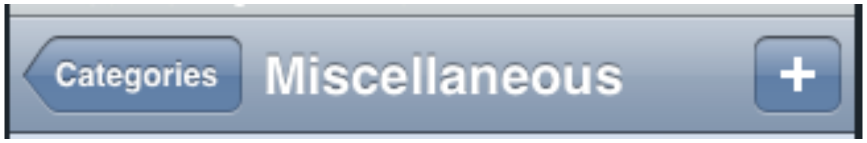
The MVC model

- Model-View-Control model
- Makes it so all elements are separate from each other

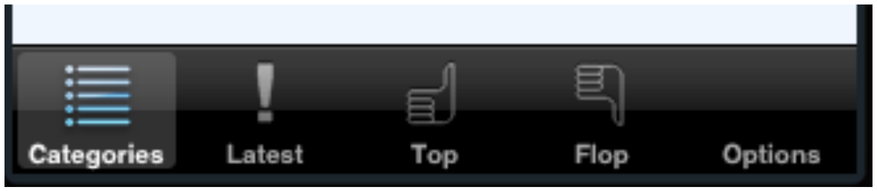
- The view (GUI), the data (Model) and the interaction between (Control)
- May fit you may not. Do what is best and doesn’t break your code easily or frustrate you

Basic Apple GUI elements to note

- UIView – basic view : shown to user; add gui elements to the objects view to show
- UIViewController -> Controls a 'view'. Think of this as literally the gaurdian of the UIView. Used with navigation controls
- UINavigationController -> that navigation bar on top with a 'Back' button

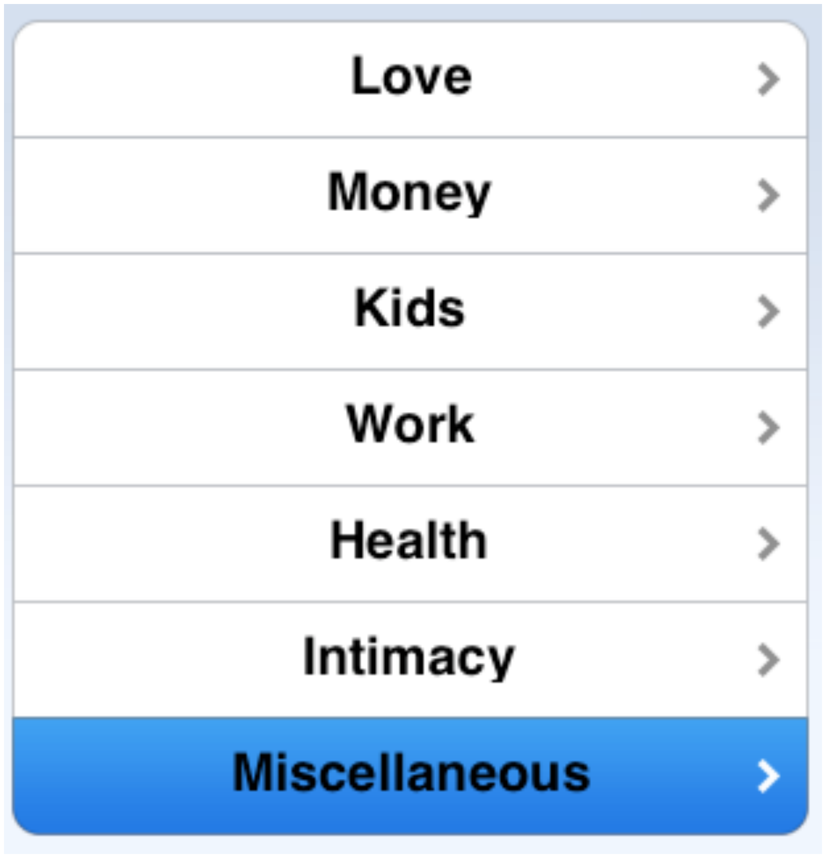


- UITabBarController -> bottom row tab bar with buttons



More GUI elements

- UIWebView -> similar to safari html browser
- UITableViewController -> lists data in a cell like fashion
- UITextField
- UITextArea
- UILabel
- UIButton
- Etc…



Any questions?

- Hmm.
- Google is your friend.
- www.iphonedevsdk.com is a good resource.
- http://developer.apple.com is very good.

http://www.iphonedevcentral.org/home.php

Site with lots of video tutorials. Interface Builder is confusing at first, so all the basic tutorials here can get you up to speed.

http://www.iphonedevsdk.com

great site for beginners with a forum for questions.

http://developer.apple.com/iphone/library/documentation/iPhone/Conceptual/iPhone101/Articles/00_Introduction.html

basic introduction from apple to application programming

http://developer.apple.com/iphone/library/referencelibrary/GettingStarted/Learning_Objective-C_A_Primer/

basic objective-C introduction; useful for the little introduction stuff and a primer

http://developer.apple.com/iphone/index.action

where the above links came from; Don't need an account to download the SDK or view the documents. Accounts only good for real iphone device development