iOS 101

Hands-On Labs

Lab #1: Chuck Norris Joke Generator Class

Chuck Norris does not need Twitter... he is already following you.
*Chuck Norris doesn't flush the toilet, he scares the sh*t out of it.*
Chuck Norris is the reason why Waldo is hiding.

If you've been on the Internet in the last couple years, you've probably seen a bunch of these Chuck Norris jokes flying around. Well, now's your chance to make an app for that... or, an Objective-C class at least.

You will create a simple Objective-C class that will store an array of jokes, and a method to return a random joke. You will then write some code to test this out when the app starts up, and print the random jokes out to the console.

In the process, you'll get some hands-on experience with Objective-C, and will have the data model in place for the next lab! :]

Basic Level Walkthrough

Start up Xcode and choose **File\New\Project** from the main menu. Select **iOS \Application\Single View Application**, and click **Next**. Enter **ChuckNorris** for the Product Name and select **iPhone** for Devices. Click **Next**, choose a folder to save your project, and click **Create**.

In the top toolbar, select the **iPhone Retina (4-inch)** simulator**,** and click **Run**. Verify a blank white screen appears.

Next, let's create a new class for the joke generator. Control-click the **ChuckNorris** yellow folder (this is called a "group" in Xcode) and click **New File**. Select **iOS\Cocoa Touch\Objective-C** class, and click **Next**. Enter **JokeGenerator** for Class and **NSObject** for Subclass, click **Next**, and then **Create**.

Open up **JokeGenerator.h** and replace it with the following:

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

@interface JokeGenerator : NSObject

@property (strong) NSArray * jokes;

- (NSString *)randomJoke;

@end
```

This defines the following:

- A class named JokeGenerator, that derives from NSObject (the base class in Objective-C).
- A property for an NSArray of jokes. This is not the mutable variant, so jokes cannot be added after it is initialized.
- By creating a property, the compiler will automatically create an instance variable called _jokes for you. It will also automatically create the methods jokes and setJokes: behind the scenes.
- Defines a method to return a random joke.

Next, open **JokeGenerator.m** and replace it with the following:

```
#import "JokeGenerator.h"

@implementation JokeGenerator

- (id)init {
    if ((self = [super init])) {
        self.jokes = @[
        @"They once named a street after Chuck Norris, but they had to
close it down because no one dared cross Chuck Norris.",
        @"The truth hurts because Chuck Norris roundhouse kicked it.",
        @"Chuck Norris doesn't cheat death, he beats it fair and
square.",
        @"Ghosts sit around the campfire and tell Chuck Norris
stories.",
        @"Chuck Norris only uses stunt doubles for crying scenes.",
        @"That's not an eclipse - it's the sun hiding from Chuck
Norris."];
    }
    return self;
}

- (NSString *)randomJoke {
    int randomIdx = arc4random() % _jokes.count;
    return [_jokes objectAtIndex:randomIdx];
}

@end
```

This does the following:

- Creates an initializer to initialize the array of jokes.
- Note that it uses the "array literal" syntax @[a, b, ...], which is a shortcut for calling [NSArray arrayWithObjects:a, b, ..., nil];
- Creates a method to return a random joke. arc4random() returns a number between 0 and a super-large integer, so you use the modulus operator to chop it down between 0

and the size of the jokes array (non-inclusive). You then use NSArray's objectAtIndex method to pull out a random joke.

Finally, switch to **AppDelegate.m** and add the following to the top of the file:

```
#import "JokeGenerator.h"
```

Then find the **application:didFinishLaunchingWithOptions** method. Add the following test code at the beginning of this method:

```
JokeGenerator * generator = [[JokeGenerator alloc] init];
for(int i = 0; i < 3; ++i) {
    NSString *joke = [generator randomJoke];
    NSLog(@"Random joke %d: %@", i, joke);
}
```

Run your app, and look inside the console. You should see some random jokes print out!

Uber Haxx0r Level Challenge

Modify your joke generator so that it stores multiple kinds of jokes - "Chuck Norris jokes", "Knock knock jokes", and "Yo Momma jokes."

The Joke Generator class should contain a NSDictionary instead of the current NSArray. The key for the dictionary should be the joke type ("Chuck", "Knock", or "YoMomma"), and the value should be the NSArray of jokes.

The randomJoke method should take an NSString parameter of the joke type to return a joke for ("Chuck", "Knock", or "YoMomma").

You should also return a new method called jokeTypes that returns an NSArray of available joke types. Hint: there's a method in NSDictionary that will help with this!

Once you have made these modifications, test out the new functionality with test code and congrats - you have achieved the Uber Haxx0r level!

And by the way - here's some jokes you can use to test with, feel free to add your own as well!

"Knock, Knock! Who's there? Says! Says who? Says me, that's who?"
"Knock, Knock! Who's there? Cows go. Cows go who? No, cows go moo!"
"Knock, Knock! Who's there? Spell. Spell who? OK, W_H_O."

"Yo momma so dumb she studied for a drug test!"

"Yo momma so fat she jumped in the air and got stuck."
"Yo momma so poor when she goes in her front door she's in her back yard."

Lab #2: Chuck Norris Complete App

I don't think Chuck Norris would have approved of your previous lab project.

First of all, the only way to see the output is in the console, and secondly how can you have an app about Chuck Norris without showing a bad-ass picture of him?!

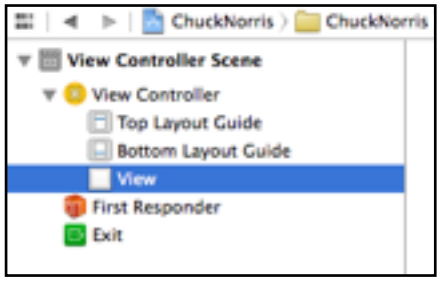
So in this lab, you're going to wrap up your Chuck Norris App by making a simple user interface!

If you did not finish the lab from last time, you can either keep working through it, or just copy the finished lab from the resources folder.

Basic Level Walkthrough

Open your **ChuckNorris** project where you left it off last time.

Then open **MainStoryboard.storyboard** and select the View in Interface Builder:



Make sure the Utilities pane is visible (the third tab on the far right of the toolbar), and in the top of the pane (the Inspector area), make sure the fourth tab is selected (the Attributes Inspector).

In the Attributes Inspector, set the Background to Black.

In the Utility Pane's Library panel, make sure the third tab is selected (the Objects Library). Drag an Image View into the view. With the image view selected, select the fifth Inspector tab (the Size Inspector) and set X=10, Y=10, Width=300, and Height=300.

Next find the **ChuckNorris.png** image inside the Art folder for this section, and drag it into your project. Make sure that **Copy items into destination group's folder (if needed)** is checked, and click Finish.

Select the image view again, and select the fourth Inspector tab (the Attributes inspector). Set the image to ChuckNorris.png, and set the mode to Aspect Fill.

Next, drag a Label from the Objects Library into your view. With it selected, go to the Size Inspector and set X=10, Y=318, Width=300, Height=77. Then go back to the Attributes Inspector and set the Text to “Joke Goes Here”, the Lines to 0 (means unlimited) and the Text Color to White. Also set the Alignment to Center.

Finally, drag a Button from the Objects library into your view. With it selected, go to the Size Inspector and set X=10, Y=403, Width=300, Height=37. Then go back to the Size Inspector and set the Title to “Roundhouse Kick!”

Finally, select the view and set the Tint color to an orange color (this will affect the button’s color).

At this point your layout should look like the following:

Now it’s time to hook up the label to an property on our view controller, and make clicking the button call a method in our view controller.

Select the second tab under the Editor section of the toolbar to bring up the Assistant Editor. Make sure that it is set to Automatic, and that it displays ViewController.m.

Then, control-drag from the Label down to the Assistant Editor’s ViewController.m’s private @interface, right before the @end. Make sure the Connection Type is set to Outlet, enter jokeLabel for the Name, and click Connect.

Next, control-drag from the Button down to the Assistant Editor's ViewController.m's @implementation, right before the @end. Make sure the Connection Type is set to Action, enter buttonTapped for the Name, and click Connect.

Next, you need to set up the class to create a JokeGenerator when it loaded, and use it to generate a random joke when the button is tapped.

Open up **ViewController.m** and make the following changes:

```
// Add to top of file
#import "JokeGenerator.h"

// Inside private @interface
@property (strong) JokeGenerator * jokeGenerator;
```

Here you just import the JokeGenerator header file, and create a property for the JokeGenerator.

Then open up **ViewController.m** and make the following changes:

```
// Add inside viewDidLoad
self.jokeGenerator = [[JokeGenerator alloc] init];

// Modify buttonTapped
- (IBAction)buttonTapped:(id)sender {
    self.jokeLabel.text = [self.jokeGenerator randomJoke];
}
```

Very simple stuff here - you create the JokeController class in viewDidLoad and store it in our property/instance variable. When buttonTapped is called, you call randomJoke to get a random joke, and set the joke label's text to the result.

Almost done - for a final finishing touch step, open Images.xcassets and select the AppIcon entry. Then find Icon.png inside the Art folder for this section and drag it the area where the boxes are. Note this is just one of the required icon sizes - for a real app you'd have to make the other variant sizes as well.

At this point your screen should look like this:

Guess what - that's it! Build and run, and you have your own Chuck Norris joke generator!

Note: You may have to delete your app off the simulator and re-install it for the icon to show up.

Uber Haxx0r Level Challenge

First of all, if you are a member of the iOS developer program, get this app running on your own iPhone, iPod, or iPad! If you haven't done this already, in the Xcode Organizer you can select your device and set it up for development by simply clicking a button. If you have any questions on how to do this, feel free to ask.

Secondly, finish the Uber Haxx0r challenge from last time if you haven't already, so your joke generator can make multiple kinds of jokes - Chuck Norris, Knock Knock, and Yo Momma.

Third, add buttons at the top of the app to allow the user to select the joke set they want to use. When the user clicks a button, store the joke type in a NSString on your class and set the image view to the appropriate image (I have put some more images inside the Art folder for you). Finally, when the user clicks the joke button, display the appropriate joke!

Finally, set the icon of your app to the Icon2.png file in your Art folder, and set the Bundle Display Name (in your Info.plist) to JokeGenerator.

Once you've done all this, congrats - you have achieved the Uber Haxx0r level!

Lab #3: Tip Calculator

Do a search for tip calculator on the App Store - and prepared to be amazed by the number of results! It seems like everybody out there has made a tip calculator - and you are about to join the ranks.

Why make a tip calculator? First, it's one of the easiest exercises you can do, so it is a great way to get started with iOS development. Secondly, it's actually quite handy so you may actually find yourself using it! I use a tip calculator I wrote pretty often when I'm feeling lazy about doing math.

In the process of making this tip calculator, you'll get some good practice using some of the most UIKit controls and Interface Builder.

So let's get started, and get tipping!

Basic Level Walkthrough

Start up Xcode and choose **File\New\Project** from the main menu. Select **iOS \Application\Single View Application**, and click **Next**. Enter **TipCalc** for the Product Name and select **iPhone** for devices. Click **Next**, choose a folder to save your project, and click **Create**.

In the **Scheme** dropdown, select the **iPhone Retina (4-inch)** simulator, and click **Run**. Verify a blank white view appears.

Let's start by creating the Model for this app - a simple Objective-C class that can generate tips for us.

Control-click the **TipCalc** group and click **New File**. Select **iOS\Cocoa Touch \Objective-C** class, and click **Next**. Enter **TipCalc** for Class, **NSObject** for Subclass, click **Next**, and then **Create**.

Open up **TipCalc.h** and replace it with the following:

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

@interface TipCalc : NSObject

- (id)initWithAmountBeforeTax:(float)amountBeforeTax andTipPercentage:
(float)tipPercentage;
- (void)calculateTip;

@property (assign) float amountBeforeTax;
@property (assign) float tipPercentage;
```

```
@property (assign, readonly) float tipAmount;

@end
```

As you can see, this is a very simple class with three variables - the amount before tax and the tip percentage (which the user will input), and the amount to tip. Note your'e making the properties public this time so you can access them from outside the class itself.

Next, open up **TipCalc.m** and replace it with the following:

```
#import "TipCalc.h"

@implementation TipCalc

- (id)initWithAmountBeforeTax:(float)amountBeforeTax andTipPercentage:
(float)tipPercentage {
    if ((self = [super init])) {
        self.amountBeforeTax = amountBeforeTax;
        self.tipPercentage = tipPercentage;
    }
    return self;
}

- (void)calculateTip {
    _tipAmount = self.amountBeforeTax * self.tipPercentage;
}

@end
```

You should also be well familiar with this by now - it creates a custom initializer, and creates a method that can be called to calculate the tip based on the amount before tax and tip percentage.

Next open up **Main.storyboard** and drag some Labels, Text Fields, and Buttons into the view to create the following layout:

Select both text fields (by clicking one, holding down shift, and clicking the other) and go to the Attributes Inspector. Set the Keyboard to Numbers and Punctuation, set the Clear Button to Appears While Editing, and check Clear when editing begins.

Then select just the first text field and set the Return Key to Next, and select just the second text field and set the Return Key to Done.

Now let's hook everything up to the view controller. Bring up the Assistant Editor, and make sure ViewController.m is visible. Then do the following:

- Control-drag from the first text field down to the private @interface. Make sure the Connection Type is set to **Outlet**, enter **amountBeforeTaxTextField** for the name, and click Connect.
- Repeat for the second text field, and name it **tipPercentageTextField**.
- Repeat for the label, and name it **resultLabel**.
- Repeat for the button, except drag it inside the @implementation right before the @end. Set set the Connection Type to **Action**, and name it **buttonTapped**.
- Control-click the first text field, and drag from the little dot to the right of delegate up to the View Controller and release to connect.
- Repeat for the second text field.

You're done with Interface Builder - onto code! Make the following changes to **ViewController.m**:

```
// Add to top of file
#import "TipCalc.h"

// Mark private @interface as implementing UITextFieldDelegate
@interface ViewController () <UITextFieldDelegate>

// Add inside private @interface
@property (strong) TipCalc * tipCalc;
@property (strong) NSArray * textFields;

// Add inside viewDidLoad
self.tipCalc = [[TipCalc alloc] initWithAmountBeforeTax:25.00
andTipPercentage:0.2];
self.amountBeforeTaxTextField.text = [NSString
stringWithFormat:@"%0.2f",self.tipCalc.amountBeforeTax];
self.tipPercentageTextField.text = [NSString
stringWithFormat:@"%0.2f",self.tipCalc.tipPercentage];
self.textFields = @[self.amountBeforeTaxTextField,
self.tipPercentageTextField];

// Add before buttonTapped
- (void)calcTip {
    self.tipCalc.tipPercentage = [self.tipPercentageTextField.text
```

```
      floatValue];
    self.tipCalc.amountBeforeTax =
    [self.amountBeforeTaxTextField.text floatValue];
    [self.tipCalc calculateTip];
    self.resultLabel.text = [NSString stringWithFormat:
      @"Your tip: %0.2f.", self.tipCalc.tipAmount];
}

// Add inside buttonTapped
[self calcTip];

// Add new method
- (BOOL)textFieldShouldReturn:(UITextField *)textField {

    int i = [self.textFields indexOfObject:textField];
    if (i < [self.textFields count] - 1) {
        UITextField *nextTextField =
          self.textFields[i+1];
        [nextTextField becomeFirstResponder];
    } else {
        [textField resignFirstResponder];
        [self calcTip];
    }
    return TRUE;
}
```

In viewDidLoad, you create a new TipCalc and populate it with default values. You set the user interface to match what the tip calc has.

When the user clicks the calcTip button (or taps done on the last text field), it calls calcTip. This updates the TipCalc model with what the user entered, calls calculateTip, and sets the result label to show the result.

Almost done - for a final finishing touch step, add the app icon that comes in the Art directory for this section to the project. If you forgot how, check the previous tutorial.

At this point your screen should look like this:

That's it - build and run, and you have your very own tip calculator!

Uber Haxx0r Level Challenge

As you can see, this tip calculator is very basic. Let's see if you're uber enough to make a tip calculator the way it should be!

Modify the tip calculator to include the following features (I suggest you add them one by one and see how far you can get):

- Modify the tip percentage field to be a slider rather than a text field, with a range of 15-25%.
- Remove the button, and have the tip automatically update based on the current values whenever you change the value of a field. Tip: use the Editing Changed action.
- Add another slider to represent tax percentage, with a range of 0-10%. Modify the TipCalculator to use this so it can figure out the total bill as well! The result label should now display the following based on the input: "Amount after tax: XX Your tip: XX Your total bill: XX"
- Finally, add one more slider for number of people in your party, and another output field for "per-person total."

If you have achieved all of the above, congrats - you have achieved the Uber Haxx0r level!

Lab #4: Video Game Trivia

I am a huge gamer, ever since the good old days of the Atari and the NES. My favorite games include Final Fantasy II, Deus Ex, Diablo 2, Mega Man 2, and the Fallout Series.

In this lab, we'll see if there are any fellow gaming geeks in these class by creating a simple video game trivia app that test the user's knowledge of these games. And if you aren't into video games, feel free to replace the questions with trivia questions of your own. :]

In this app, you'll need two screens: a screen to ask the question and present multiple choice answers, and a screen to present the results. This is a good situation where you might want to use two view controllers!

Basic Level Walkthrough

Start up Xcode and choose **File\New\New Project** from the main menu. Select **iOS \Application\Single View Application**, and click **Next**. Enter **VGTrivia** for the Product Name and select **iPhone** for devices. Click **Next**, choose a folder to save your project, and click **Create**.

Select the **iPhone Retina (4-inch)** simulator, and click **Run**. Verify a blank white view appears.

As usual, you will start by creating the model for this class. Control-click the **VGTrivia** group and click **New File**. Select **iOS\Cocoa Touch\Objective-C** class, and click **Next**. Enter **Question** for Class, **NSObject** for Subclass, click **Next**, and then **Create**.

Open **Question.h** and replace it with the following:

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

@interface Question : NSObject

@property (strong) NSString * question;
@property (strong) NSString * answer1;
@property (strong) NSString * answer2;
@property (strong) NSString * answer3;
@property (strong) NSString * answer4;
@property (assign) int rightAnswer;

- (id)initWithQuestion:(NSString *)question answer1:(NSString
*)answer1 answer2:(NSString *)answer2 answer3:(NSString *)answer3
answer4:(NSString *)answer4 rightAnswer:(int)rightAnswer;
```

```
@end
```

This is just a simple class that keeps track of the question to ask, four possible answers, and an integer that corresponds to the correct answer.

Next switch to **Question.m** and replace it with the following:

```
#import "Question.h"

@implementation Question

- (id)initWithQuestion:(NSString *)question answer1:(NSString
*)answer1 answer2:(NSString *)answer2 answer3:(NSString *)answer3
answer4:(NSString *)answer4 rightAnswer:(int)rightAnswer {

    if ((self = [super init])) {
        _question = question;
        _answer1 = answer1;
        _answer2 = answer2;
        _answer3 = answer3;
        _answer4 = answer4;
        _rightAnswer = rightAnswer;
    }
    return self;
}

@end
```

Nothing surprising here - just a basic initializer.

Next, open **MainStoryboard.storyboard** and lay out the view controller like the following:

Everything should be as expected. For the question label, make it nice and wide/large and set the Lines to 0 (which enables unlimited lines).

Select the second tab under the Editor section of the toolbar to bring up the Assistant Editor. Make sure that it is set to Automatic, and that it displays **ViewController.m**.

Then connect the question label to a private outlet named questionLabel, and the answer buttons to outlets named answerButton1, answerButton2, and so on.

Also, connect all four answer buttons to an action named answerButtonTapped. Note that after you connect the first button to the action, you can just drag from the other buttons to the method you already created to connect it. But for this to work, you will have to predeclare the method in the @interface in **ViewController.h**:

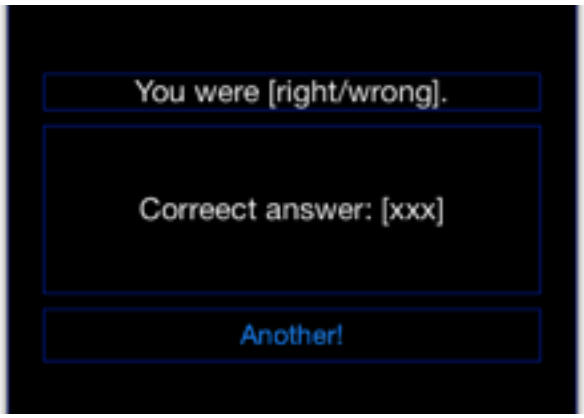
```
- (IBAction)answerButtonTapped:(id)sender;
```

Next open **ViewController.m** and make the following changes:

```
// Add at top of file
#import "Question.h"

// Add to private interface
@property (strong) NSArray * questions;
@property (strong) Question * curQuestion;

// Replace viewDidLoad and add viewWillAppear
- (void)viewDidLoad
{
    [super viewDidLoad];
    Question * question1 = [[Question alloc] initWithQuestion:@"Which
character was a twin in Final Fantasy II?" answer1:@"Cecil"
answer2:@"Kain" answer3:@"Palom" answer4:@"Tellah" rightAnswer:3];
    Question * question2 = [[Question alloc] initWithQuestion:@"Which
of these people was a game designer for Deus Ex?" answer1:@"JC Denton"
answer2:@"Shigeru Miyamoto" answer3:@"Matt Rix" answer4:@"Warren
Spector" rightAnswer:4];
    Question * question3 = [[Question alloc] initWithQuestion:@"Which
of these items was commonly used as currency in Diablo 2?"
answer1:@"Stone of Jordan" answer2:@"Duck of Doom" answer3:@"Ethereal
Shard" answer4:@"Tower Bux" rightAnswer:1];
    Question * question4 = [[Question alloc] initWithQuestion:@"Which
of these was a boss in Mega Man 2?" answer1:@"Snow Man" answer2:@"Wood
Man" answer3:@"Youda Man" answer4:@"Snake Man" rightAnswer:2];
```

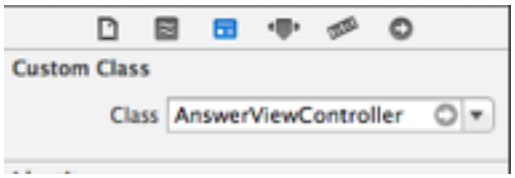
```
    Question * question5 = [[Question alloc] initWithQuestion:@"What
was the job of the main character in Fallout 3 New Vegas?" answer1:@"A
bounty hunter" answer2:@"A metalsmith" answer3:@"A courier"
answer4:@"A plumber" rightAnswer:3];
    self.questions = @[question1, question2, question3, question4,
question5];
}

- (void)viewWillAppear:(BOOL)animated {
    self.curQuestion = [self.questions objectAtIndex:arc4random() %
self.questions.count];
    self.questionLabel.text = self.curQuestion.question;
    [self.answerButton1 setTitle:self.curQuestion.answer1
forState:UIControlStateNormal];
    [self.answerButton2 setTitle:self.curQuestion.answer2
forState:UIControlStateNormal];
    [self.answerButton3 setTitle:self.curQuestion.answer3
forState:UIControlStateNormal];
    [self.answerButton4 setTitle:self.curQuestion.answer4
forState:UIControlStateNormal];
}
```

Build and run, and you should see a random question appear. So far so good - but now you want to create a new view controller to show the user if they were right or wrong!



Open up **MainStoryboard.storyboard** and drag a new View Controller into the storyboard to the right of the current view controller. Set the background to black, and drag two labels and a button in as you can see here:
You need to tie these labels to outlets so you can set them programmatically, but before you do that you need to create a new View Controller subclass for this new view controller. So control-click the **VGTrivia** group and click **New File**. Select **iOS\Cocoa Touch\Objective-C** class, and click **Next**. Enter **AnswerViewController** for Class, **UIViewController** for Subclass, click **Next**, and then **Create**.

Go back to **MainStoryboard.storyboard** and select the new view controller. Set the class to **AnswerViewController** in the Identity Inspector.



Select the new view controller, bring up the Assistant Editor and make sure AnswerViewController.m is displayed. Then connect the two labels to private outlets in AnswerViewController.m - **statusLabel** and **correctAnswerLabel**, respectively.

Next, you need to create a properties on AnswerViewController for the question to display the results for, and the answer the user guessed. The main view controller will send this information to the AnswerViewController when the user clicks a button.

Open **AnswerViewController.h** and make the following changes:

```
// Add to top of file
#import "Question.h"

@property (strong) Question * question;
@property (assign) int guessedAnswer;
```

Then switch to **AnswerViewController.m** and make the following changes:

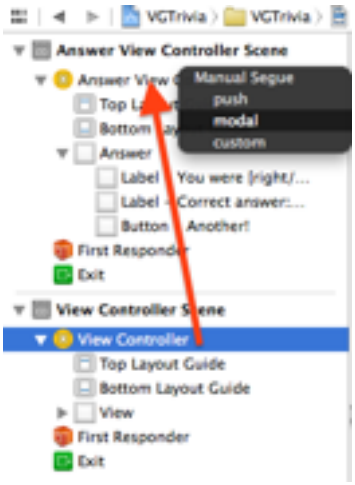
```
// Add new method
- (void)viewWillAppear:(BOOL)animated {
    if (self.guessedAnswer == self.question.rightAnswer) {
        self.statusLabel.text = @"Right answer, w00t!";
    } else {
        self.statusLabel.text = @"Wrong answer :[";
    }

    if (self.question.rightAnswer == 1) {
       self.correctAnswerLabel.text = [NSString
stringWithFormat:@"Right answer: %@", self.question.answer1];
    } else if (self.question.rightAnswer == 2) {
        self.correctAnswerLabel.text = [NSString
stringWithFormat:@"Right answer: %@", self.question.answer2];
    } else if (self.question.rightAnswer == 3) {
        self.correctAnswerLabel.text = [NSString
stringWithFormat:@"Right answer: %@", self.question.answer3];
    } else if (self.question.rightAnswer == 4) {
```

```
        self.correctAnswerLabel.text = [NSString
stringWithFormat:@"Right answer: %@", self.question.answer4];
    }

}
```

OK - now both view controllers are set up, it's just a matter of hooking them together with segues.

In this case, there are multiple buttons, each of which wants to execute the same transition. So the best thing to do is to create a single segue, and have each button call that segue with prepareSegue.

Open **MainStoryboard.storyboard** and control-drag from the first view controller to the second view controller:



Choose "modal" from the popup. Then select the icon for the segue and give it an Identifier "DisplayAnswer".

Then open **ViewController.m** and make the following changes:

```
// Add at top of file
#import "AnswerViewController.h"

// Replace answerTapped
- (IBAction)answerButtonTapped:(id)sender {
    [self performSegueWithIdentifier:@"DisplayAnswer" sender:sender];
}

// Add new method
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    if ([segue.identifier isEqualToString:@"DisplayAnswer"]) {
```

```
        AnswerViewController * answerViewController =
(AnswerViewController *)segue.destinationViewController;
        answerViewController.question = self.curQuestion;
        if (sender == self.answerButton1) {
            answerViewController.guessedAnswer = 1;
        } else if (sender == self.answerButton2) {
            answerViewController.guessedAnswer = 2;
        } else if (sender == self.answerButton3) {
            answerViewController.guessedAnswer = 3;
        } else if (sender == self.answerButton4) {
            answerViewController.guessedAnswer = 4;
        }
    }
}
```

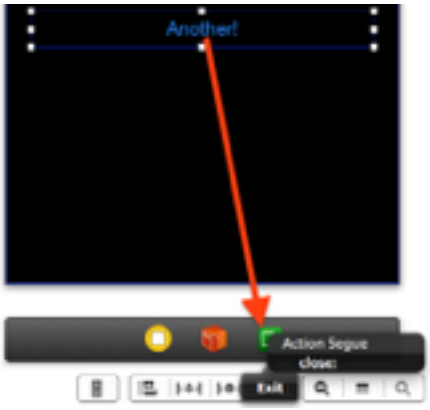
When answerButtonTapped is called, the sender variable will be whatever button is tapped. You pass that through to prepareForSegue, and it then sets the guessedAnswer appropriately based on what button is tapped.

That takes care of the segue to display the AnswerViewController. But what about going the other way, and closing the AnswerViewController? For this you need an exit segue.

For an exit segue to work, you first have to write a close method in the parent view controller. So add this to ViewController.m:

```
- (IBAction)close:(UIStoryboardSegue *)segue {
}
```

Finally, open MainStoryboard.storyboard one last time, and control-drag from the Another button to the exit segue indicator, and select close:



And that's it! Compile and run, and you have your own video game trivia app! How many did you know the answers of? :]

Uber Haxx0r Level Challenge

Modify the app to include the following features (I suggest you add them one by one and see how far you can get):

- Instead of presenting the answer view controller modally, embed the main view controller inside a navigation view controller and push it on instead. You can then remove the “another” button.
- Store the questions in a property list file instead of having them hardcoded in the app. Hints:
  - You can create a property list file with an Xcode template. It allows you to easily store standard data types like dictionaries, arrays, numbers, strings, etc.
  - The root object should be a dictionary. It should have a single entry for questions, which is an array of dictionaries. There should be five keys in each dictionary: question, answer1, answer2, answer3, answer4, rightAnswer.
  - You can read it in with NSDictionary’s dictionaryWithContentsOfFile method.
  - To get a path to a file, you can use NSBundle’s pathForResource method.
  - After reading the file you’ll need to pull the info out of the dictionary and populate the Question objects appropriately.

If you have achieved all of the above, congrats - you have achieved the Uber Haxx0r level!

Lab #5: Wish List

Have you ever been at a store or online and saw something that looked cool, but thought “meh, I’ll get that later when I have more money.”

And then have you ever been asked by a friend or relative what you want for Christmas, and have no ideas at all?

Well, now you’re going to make an app to help with that - Wish List! It will display a list of items you found and like, and allow you to select items for additional details.

Basic Level Walkthrough

Start up Xcode and choose **File\New\Project** from the main menu. Select **iOS \Application\Master-Detail Application**, and click **Next**. Enter **WishList** for the Product Name and select **iPhone** for devices. Click **Next**, choose a folder to save your project, and click **Create**.

Select the **iPhone Retina (4-inch)** simulator, and click **Run**. Verify a blank table view appears.

As usual, you will start by creating the model for this class. Control-click the **WishList** group and click **New File**. Select **iOS\Cocoa Touch\Objective-C** class, and click **Next**. Enter **WishListItem** for Class, **NSObject** for Subclass, click **Next**, and then **Create**.

Open up **WishListItem.h** and replace it with the following:

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

@interface WishListItem : NSObject

@property (strong) NSString * name;
@property (strong) UIImage * photo;
@property (assign) float price;
@property (strong) NSString * notes;

- (id)initWithName:(NSString *)name photo:(UIImage *)photo price:
(float)price notes:(NSString *)notes;

@end
```

As you can see, this is a plain old Objective-C class - you should be well familiar with this by now.

Next open **WishListItem.m** and replace it with the following implementation:

```
#import "WishListItem.h"

@implementation WishListItem

- (id)initWithName:(NSString *)name photo:(UIImage *)photo price:
(float)price notes:(NSString *)notes {
    if ((self = [super init])) {
        _name = name;
        _photo = photo;
        _price = price;
        _notes = notes;
    }
    return self;
}

@end
```

This is also quite straightforward. Now let's test out this model by creating some test items in our Application Delegate.

First you'll need some images, so drag the images inside the Art\Lab folder for this section into your project. Make sure that **Copy items into destination group's folder (if needed)** is checked, and click Finish.

Then make the following changes to **AppDelegate.m:**

```
// Add to top of file
#import "WishListItem.h"

// Add inside application:didFinishLaunchingWithOptions
WishListItem * battleMap = [[WishListItem alloc] initWithName:@"Battle
Map" photo:[UIImage imageNamed:@"BattleMap.png"] price:29.99
notes:@"Map drawing app for D&D geeks like Ray!"];
WishListItem * mathNinja = [[WishListItem alloc] initWithName:@"Math
Ninja" photo:[UIImage imageNamed:@"MathNinja.png"] price:1.99
notes:@"Fun castle defense style game that happens to teach you
math!"];
WishListItem * wildFables = [[WishListItem alloc] initWithName:@"Wild
Fables" photo:[UIImage imageNamed:@"WildFables.png"] price:0.00
notes:@"Aesop's Fables with physics-based interactivity on every
page!"];
NSMutableArray * wishListItems = [NSMutableArray
arrayWithObjects:battleMap, mathNinja, wildFables, nil];
for (int i = 0; i < wishListItems.count; ++i) {
    WishListItem * item = [wishListItems objectAtIndex:i];
    NSLog(@"Item %d: %@", i, item.name);
```

```
}
```

These are just examples - feel free to replace these with your own wish list items.

Build and run your app, and look in the console and verify you see the item names printing out.

So far so good! Now let's make our table view display these items. Open up **MasterViewController.h** and replace it with the following:

```
#import <UIKit/UIKit.h>
#import "WishListItem.h"

@interface MasterViewController : UITableViewController

@property (strong) NSMutableArray * wishListItems;

@end
```

Here you simply added an instance variable and property to store the wish list items. Next switch to **MasterViewController.m** and make the following changes:

```
// Replace return statement in numberOfRowsInSection with this
return self.wishListItems.count;

// Replace tableView:cellForRowAtIndexPath with this
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:@"Cell" forIndexPath:indexPath];

    WishListItem * item = self.wishListItems[indexPath.row];
    cell.textLabel.text = item.name;
    cell.imageView.image = item.photo;
    return cell;
}
```

Here you specify the number of rows to be equal to the number of elements in the array. You leave each cell with the default cell style, and set the title and image to match what's set for the wish list item for the row.

The RootViewController is all set up to display a list of items - you just need to pass it that list! Go back to **AppDelegate.m** and make the following changes:

```
// Add to top of file
#import "MasterViewController.h"
```

```
// Add inside application:didFinishLaunchingWithItems,
// right before the return YES
UINavigationController * navigationController =
  (UINavigationController *) self.window.rootViewController;
MasterViewController * masterViewController =
    navigationController.viewControllers[0];
masterViewController.wishListItems = wishListItems;
```

Compile and run, and you should see the table view populated with your wish list items!

Now let's set up the detail view controller so you can see and edit the rest of the info.

Open up **MainStoryboard.storyboard** and find the detail view controller. Delete the placeholder label, and drag some labels, text fields, an image view, and a text view into the view to create the following layout:

Then do the following:

- Connect the first text field to an outlet called nameTextField.
- Connect the second text field to an outlet called priceTextField.
- Connect the text view to an outlet called notesTextView.
- Connect the image view to an outlet called photoImageView.

- Set the delegate of the first text field to the Detail View Controller.
- Also set the delegate of the second text field to the Detail View Controller.
- Change the Image View Mode to Aspect Fit.

Then open up **DetailViewController.h** and make the following changes:

```
// Add to top of file
#import "WishListItem.h"

// Mark as implementing UITextFieldDelegate
@interface DetailViewController : UIViewController
<UITextFieldDelegate>

// Add after @interface
@property (strong) IBOutlet WishListItem * item;
```

Here you just add a property to keep track of the WishListItem we're showing, and mark the class as implementing UITextFieldDelegate.

Next open **DetailViewController.m** and make the following changes:

```
// Add new methods
- (void)viewWillAppear:(BOOL)animated {
    self.nameTextField.text = self.item.name;
    self.priceTextField.text = [NSString stringWithFormat:@"%0.2f",
      self.item.price];
    self.notesTextView.text = self.item.notes;
    self.photoImageView.image = self.item.photo;
}

- (void)viewWillDisappear:(BOOL)animated {
    self.item.name = self.nameTextField.text;
    self.item.price = [self.priceTextField.text floatValue];
    self.item.notes = self.notesTextView.text;
    self.item.photo = self.photoImageView.image;
}

- (BOOL)textFieldShouldReturn:(UITextField *)textField {
    [textField resignFirstResponder];
    return YES;
}
```

So when the view is about to appear, you populate all of the controls with the current item's info. Similarly, when the view is about to disappear, you store the current values on the screen back into the item.

Almost done - just need to hook this up to the MasterViewController. Open up **MasterViewController.m** and make the following changes:

```
// Replace prepareForSegue
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    if ([[segue identifier] isEqualToString:@"showDetail"]) {
        NSIndexPath *indexPath = [self.tableView
          indexPathForSelectedRow];
        WishListItem * item = self.wishListItems[indexPath.row];
        DetailViewController * detailViewController =
          (DetailViewController *)segue.destinationViewController;
        detailViewController.item = item;
    }
}

// Add new method
- (void)viewWillAppear:(BOOL)animated {
    [self.tableView reloadData];
}
```

When the user selects a row, the storyboard editor is set up to perform a segue called showDetail. You receive notification that this is about to occur in prepareForSegue, and you pass the wish list item to display to the detail view controller.

Note that you need to call reloadData in viewWillAppear in case the user edited the name of the wish list item, so that the table stays consistent. You also need to set the title in viewDidLoad so the back button appears OK.

Compile and run your app, and you have your very own wish list tracker!

Uber Haxx0r Level Challenge

This app is a good start, but there's a few features this would need before it could actually be shippable. Let's see if you can add these in!

- The user should be able to change the picture by tapping it, and selecting from their photo library.
- Make the "Add" button (the plus on the toolbar) work. It should create a new default WishListItem, add it to the array, and present it in the Detail View for editing.
- Make the app work on both 4-inch and non 4-inch displays using Auto Layout.
- Make the "Edit" button work. Here's some hints on that:
    - Table views have built in edit support (setEditing:animated)
    - Even has a built in button for navigation bar (editButtonItem)
    - Each cell shows appropriate controls for tableView:editingStyleForRowAtIndexPath
    - Implement commitEditingStyle to support delete

If you have achieved all of the above, congrats - you have achieved the Uber Haxx0r level!

Lab #6: Saving Your Wish List

Your Wish List app is coming along well so far, but there's one major problem - if you edit your wish list and terminate the app, your changes are lost forever!

Obviously that will not do. So let's add some simple NSCoding saving for the win!

If you did not finish the lab from last time, you can either keep working through it, or just copy the finished lab from the resources folder.

Basic Level Walkthrough

Open up your WishList project and open **WishListItem.h**. Mark it as implementing NSCoding as follows:

```
@interface WishListItem : NSObject <NSCoding>
```

Then switch to **WishListItem.m**, and implement the NSCoding protocol as follows:

```
- (void)encodeWithCoder:(NSCoder *)aCoder {
    [aCoder encodeObject:self.name forKey:@"name"];
    NSData * photoData = UIImagePNGRepresentation(self.photo);
    [aCoder encodeObject:photoData forKey:@"photo"];
    [aCoder encodeFloat:self.price forKey:@"price"];
    [aCoder encodeObject:self.notes forKey:@"notes"];
}

- (id)initWithCoder:(NSCoder *)aDecoder {
    NSString *name = [aDecoder decodeObjectForKey:@"name"];
    NSData *photoData = [aDecoder decodeObjectForKey:@"photo"];
    UIImage *photo = [UIImage imageWithData:photoData];
    float price = [aDecoder decodeFloatForKey:@"price"];
    NSString *notes = [aDecoder decodeObjectForKey:@"notes"];
    return [self initWithName:name photo:photo price:price
notes:notes];
}
```

To encode/decode the strings, you just call encodeObject:forKey and decodeObjectForKey.

You do the same for the image, except you have to convert it to/from NSData by calling UIImagePNGRepresentation, and imageWithData.

To encode the float, which is a primitive, you use the encodeFloat:forKey and decodeFloatForKey methods.

Next switch to **AppDelegate.m**, and modify the beginning of **application:didFinishLaunchingWithOptions** to look like the following:

```
NSMutableArray * wishListItems = [NSMutableArray array];
NSArray * docDirs =
NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
NSString *docDir = [docDirs objectAtIndex:0];
NSString *wishListItemsPath = [docDir
stringByAppendingPathComponent:@"WishListItems.plist"];
if ([[NSFileManager defaultManager]
fileExistsAtPath:wishListItemsPath]) {

    // File exists, let's load it
    NSData *data = [NSData dataWithContentsOfFile:wishListItemsPath];
    NSArray * wishListItemsArray = [NSKeyedUnarchiver
unarchiveObjectWithData:data];
    [wishListItems addObjectsFromArray:wishListItemsArray];

} else {
    // Old code to add default wish list items...
    // Replace this with your own items if you've modified this
    WishListItem * battleMap = [[WishListItem alloc]
initWithName:@"Battle Map" photo:[UIImage imageNamed:@"BattleMap.png"]
price:29.99 notes:@"Map drawing app for D&D geeks like Ray!"];
    WishListItem * mathNinja = [[WishListItem alloc]
initWithName:@"Math Ninja" photo:[UIImage imageNamed:@"MathNinja.png"]
price:1.99 notes:@"Fun castle defense style game that happens to teach
you math!"];
    WishListItem * wildFables = [[WishListItem alloc]
initWithName:@"Wild Fables" photo:[UIImage
imageNamed:@"WildFables.png"] price:0.00 notes:@"Aesop's Fables with
physics-based interactivity on every page!"];
    [wishListItems addObject:battleMap];
    [wishListItems addObject:mathNinja];
    [wishListItems addObject:wildFables];
}
for (int i = 0; i < wishListItems.count; ++i) {
    WishListItem * item = [wishListItems objectAtIndex:i];
    NSLog(@"Item %d: %@", i, item.name);
}
```

This code does the following:

- Creates an empty array to store the wish list items.
- Finds the Documents directory for this app.
- Creates a path to the file where the bugs should be saved in (Documents \WishListItems.plist).

- Uses the NSFileManager to see if this file exists. If it doesn't, it just loads some default wish list items.
- Reads the contents of the file into an NSData.
- Uses the NSKeyedUnarchiver to "unserialize" the contents of the file using the NSCoding protocol. You know this is an array of wish list items (because that's what you write out to it).
- This returns an array (not a mutable array), so you add the contents of the array into the mutable array. You do this because you want an array that you can add items to later.

Loading done - now onto saving! Modify the **applicationDidEnterBackground** method as follows:

```
- (void)applicationDidEnterBackground:(UIApplication *)application
{
    UINavigationController * navigationController =
(UINavigationController *) self.window.rootViewController;
    MasterViewController * masterViewController =
[navigationController.viewControllers objectAtIndex:0];

    NSArray * docDirs =
NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
    NSString *docDir = [docDirs objectAtIndex:0];
    NSString *wishListItemsPath = [docDir
stringByAppendingPathComponent:@"WishListItems.plist"];

    NSData * data = [NSKeyedArchiver
archivedDataWithRootObject:masterViewController.wishListItems];
    [data writeToFile:wishListItemsPath atomically:YES];
}
```

This simply gets the array of wish list items from the RootViewController, uses NSKeyedArchiver to "serialize" them into an NSData using the NSCoding protocol, and writes the data out to a file.

Compile and run your app, and modify the names of one of the wish list items. Go back to the table view to verify the name saved properly. Then hit the home button to make the app enter the background. Stop the app in the simulator, and start it again. Your new name should still show up!

Uber Haxx0r Level Challenge

Before proceeding to the Uber Haxx0r level for this lab, it's recommended that you finish the Uber Haxx0r challgenge for last lab first, as those new features really improve the program.

Once you reach that point, make the following improvements to the app:

- Read the default Wish List items from a property list, rather than being hardcoded into the app.
- Keep track of the number of times the app has run using NSUserDefaults, and display the current number in the navigation bar's title.
- Right now the app only saves when the app enters the background, which isn't great as if your app crashed after a ton of edits, the user would lose all of their changes. So Modify the code to load/save as the data is changed, by doing the following:
  - Create a new singleton class called WishListDatabase. It should have two methods:
    - - (NSMutableArray *) loadWishList; (this should contain the code that loads the wish list from disk. It should also store a reference to the array so it has it for saving later.)
    - - (void)saveWishList; (this should contain the code that saves the wish list from disk)
  - Refactor your code to use this and make sure everything still works.
  - Modify your Detail View Controller to call saveWishList on viewWillDisappear to save at that point. Edit an item, go back to the table view, and terminate your app in the debugger, and verify your data was still saved OK.

If you have achieved all of the above, congrats - you have achieved the Uber Haxx0r level!