Week 3: More View Controllers

- **Topics covered:**
  - UITableView and UITableViewController (BNR Ch. 8)
  - Editing UITableView (BNR Ch. 9)
  - UINavigationController (BNR Ch. 10)
- **In-class:** I will review the implementation of the UITableViews and their custom cells and controllers as well as how to implement a model class.

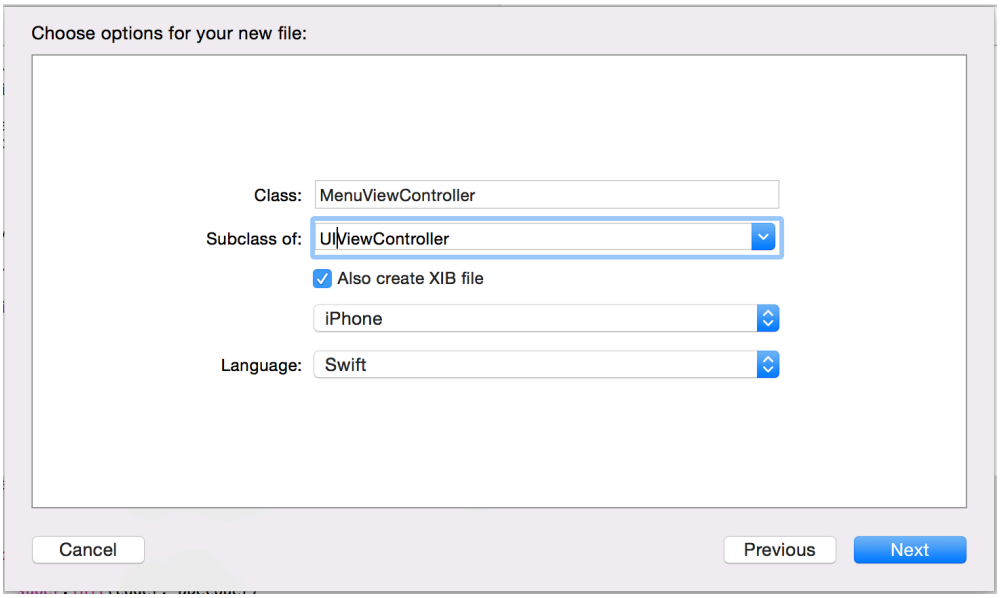
Homework:

- **Reading:**
  - BNR Ch. 7, 8, 9 and 19
  - Please read the section entitled "Optional Chaining" in The Swift Programming Language
- **Coding:** Students will modify their application from Week 2 by replacing the tab-based navigation interface with a website-style menu using UITableViewController and also provide a 'Favorites' Table View to allow users to save some favorite conversions to gain experience creating a true MVC application with a proper Model class
- **How to accomplish this:**

1. **Project Setup**
   a. Create a copy of your project from Week 2 and rename it.
   b. Open the new project in Xcode.
2. **Replace the Navigation UI -**
   Let's get some practice using UITableView and UITableViewController. We will replace the UITabBarController with a UITableViewController that contains static cells for the three view controllers that were housed in the

tabs. This UITableViewController subclass should replace the tab bar controller as the first thing the user sees when the application starts.

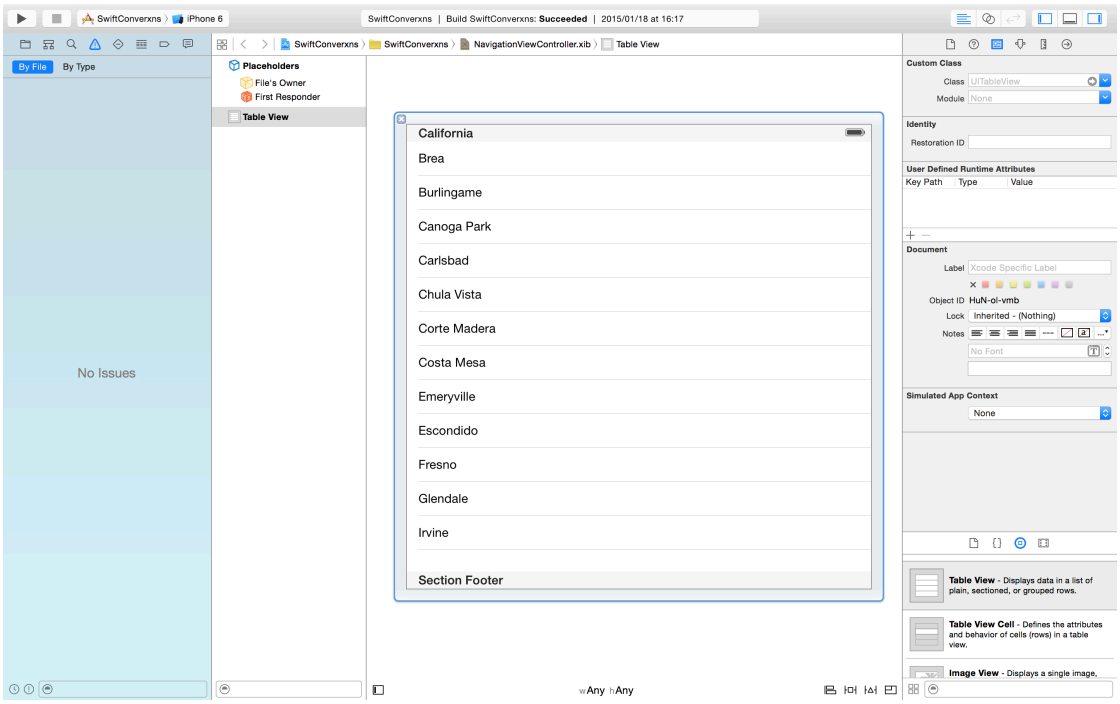
a. Before we can replace the tab bar controller, we need to create to subclasss UITableViewController and customize the content of the table view to represent our navigation scheme.
   i. Goto File>New… then select 'CocoaTouch Class' from the iOS>Source' section in the wizard that appears.
   ii. In the dialogue that appears, configure your new class as follows:



   iii. Select 'Create' in the dialogue that follows.
   iv. Xcode created a Class for us and also a XIB with a simple UIView inside of it. But we want at UITableView, so we need to exchange this UIView for a UITableView. Open the new XIB we just created, select the main 'View" object in Interface Builder, and then delete it

by pressing the backspace key. Then, drag a Table View (not a Table View Controller!) onto the blank canvas.

Your Interface Builder window should now look like this:



b. Xcode provided our custom UIViewController subclass as the type for the File's Owner object. (See this for yourself by clicking the File's Owner object in Interface Builder, then looking at its 'Class' property in the 'Identity Inspector'.)

Recall that in order to use a UITableView, the table view must 1) be owned by an instance of UITableViewController or 2) be owned by another class that conforms to the UITableViewDataSource and UITableViewDelegate protocols.

Let's get some practice implementing a table view using this second approach. Later, we will use the first approach to create our Favorites

MVC system.

Before we can set the datasource and delegate connections for our new table view, MenuViewController needs to conform to the appropriate protocols.

In MenuViewController.swift, add the code to specify that your class conforms to these protocols as follows. (You will get an error but we will address that in the following steps.)

```
class MenuViewController: UIViewController, UITableViewDataSource,
UITableViewDelegate {
```

c. We received an error stating “Type 'MenuViewController' does not conform to protocol 'UITableViewDataSource'”. This tells us that we need to implement the required methods of the protocols we just conformed to. Add the following code to specify our table view will have one section.

```
    func numberOfSectionsInTableView(tableView: UITableView) -> Int {
        return 1
    }
```

d. Next, we need to specify how many rows will be each section (again, we only have one section in this example). Remember that the table view represents a list of objects – one object per row, so it makes good sense that its data be provided by an array. (Remember, we want to show a list of view controllers that the user can view). Declare this array at the top of the class and provide an initializer as follows (next page).

```
class MenuViewController: UIViewController, UITableViewDataSource,
UITableViewDelegate {

    var viewControllers: Array<UIViewController>?

    init(viewControllers: Array<UIViewController>) {

        super.init()

        self.viewControllers = viewControllers

    }

    required init(coder aDecoder: NSCoder) {

        super.init(coder: aDecoder)

    }

    override init(nibName nibNameOrNil: String?, bundle nibBundleOrNil:
NSBundle?) {
        super.init(nibName: nibNameOrNil, bundle: nibBundleOrNil)
    }
```

e. Now that we have our viewControllers array, we can tell our table view's dataSource how many rows we will have. Add the following function beneath the `numberOfSectionsInTableView()` function.

```
func tableView(tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
        return viewControllers!.count
    }
```

f. The last step we need to do to silence the error we got after Step 2b is to implement the `tableView:cellForRowAtIndexPath:` function to provide a UITableViewCell for our table view to reuse.

First, we need to register a UITableViewCell subclasss and provide a reuse identifier. Add the following line of code to your init() function.

```
init(viewControllers: Array<UIViewController>) {

        super.init()

        self.viewControllers = viewControllers

        (self.view as UITableView).registerClass(UITableViewCell.self,
forCellReuseIdentifier: "Cell")

    }
```

g. Next, add the following code below the tableView:numberOfRowsInSection: function.

```
func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath:
NSIndexPath) -> UITableViewCell {

        var cell = tableView.dequeueReusableCellWithIdentifier("Cell")
as UITableViewCell

        let viewController = viewControllers![indexPath.row]

        cell.textLabel?.text = viewController.title

        return cell

    }
```

YOUR CLASS SHOULD NOW LOOK AS FOLLOWS ON THE NEXT PAGE(S)

```
import UIKit

class MenuViewController: UIViewController, UITableViewDataSource, UITableViewDelegate {

    var viewControllers: Array<UIViewController>?

    init(viewControllers: Array<UIViewController>) {

        super.init()

        self.viewControllers = viewControllers

        (self.view as UITableView).registerClass(UITableViewCell.self, forCellReuseIdentifier:
"Cell")

    }

    required init(coder aDecoder: NSCoder) {

        super.init(coder: aDecoder)

    }

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    func numberOfSectionsInTableView(tableView: UITableView) -> Int {
        return 1
    }

    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return viewControllers!.count
    }

    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) ->
UITableViewCell {

        var cell = tableView.dequeueReusableCellWithIdentifier("Cell") as UITableViewCell

        let viewController = viewControllers![indexPath.row]

        cell.textLabel?.text = viewController.title

        return cell

    }

}
```

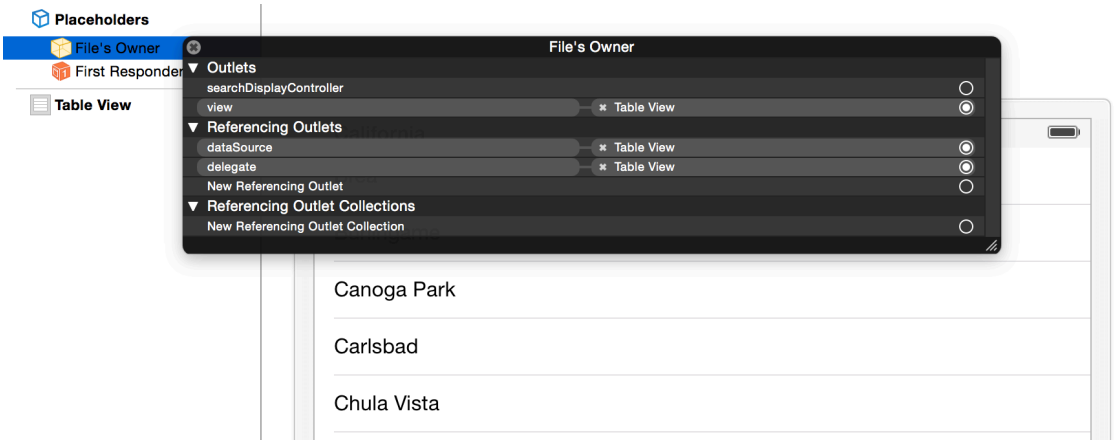
THIS CLASS CONTINUED ON NEXT PAGE

Note that since we did not specify the UITableViewCellStyle of our cells, they are initialized with `UITableViewCellStyle.Default` which gives a plain cell with a single UILabel (cell.textLabel). We set this text of the label to the viewController.title to be the value of viewController.title, so we have to make sure to provide this or our table view cell will appear blank. (Luckily, we did this inside our AppDelegate last week. If you missed this, compare your code to the AppDelegate file in step 2.i of this guide.)

h. Now, let's return to MenuViewController.xib and set the delegate and dataSource outlets by Control-dragging from the table view to the File's Owner object an selecting each outlet (you have to do this separately).

Also set the view outlet by Control-dragging from the File's Owner object to the table view.

If you have done this correctly, your outlets should appear as follows if you right-click the File's Owner Object to view the outlets.



i. Now our table view UI is ready for use but we need to do some work so that we can add it to our view hierarchy.

In your AppDelegate.swift file, find the application:didFinishLaunchingWithOptions: function. Let's replace the tab bar controller with an instance of our UIViewController subclass (i.e. MenuViewController). Modify your code as follows (next page).

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject: AnyObject]?) -> Bool {
      // Override point for customization after application launch.
      // 1st
      let temperatureConverterViewController = TemperatureConverterViewController (nibName:
"TemperatureConverter", bundle: NSBundle.mainBundle())

      temperatureConverterViewController.title = "Temperature"

      let firstNavController = UIMenuController(rootViewController:
temperatureConverterViewController)

      firstNavController.tabBarItem.title = "Temperature"

      // 2nd
      let distanceConverterViewController = DistanceConverterViewController (nibName:
"DistanceConverter", bundle: NSBundle.mainBundle())

      distanceConverterViewController.title = "Distance"

      let secondNavController = UINavigationController(rootViewController:
distanceConverterViewController)

      secondNavController.tabBarItem.title = "Distance"

      // 3rd
      let volumetricConverterViewController = VolumetricConverterViewController (nibName:
"VolumetricConverter", bundle: NSBundle.mainBundle())

      volumetricConverterViewController.title = "Volumetric"

      let thirdNavController = UINavigationController(rootViewController:
volumetricConverterViewController)

      thirdNavController.tabBarItem.title = "Volumetric"

      let tabBarController = UITabBarController()

      tabBarController.viewControllers = [firstNavController, secondNavController,
thirdNavController]

      self.window = UIWindow(frame: UIScreen.mainScreen().bounds)

      self.window?.rootViewController = tabBarController

      self.window?.addSubview(tabBarController.view)

      let menuVC = MenuViewController(viewControllers: [temperatureConverterViewController,
distanceConverterViewController, volumetricConverterViewController])

      menuVC.title = "Menu"

      let mainNavigationController = UINavigationController(rootViewController: menuVC)

      self.window?.rootViewController = mainNavigationController

      self.window?.addSubview(mainNavigationController.view)

      self.window?.makeKeyAndVisible()

      return true
}
```

Note that we initialized the MenuViewController with the array of converter view controllers, *not the navigation controllers as we did with the UITabBarController*. Instead, we created a new navigation controller (mainNavigationController), set that as the rootViewController of the window, and then put our MenuViewController instance inside this new navigation controller.

j. There is one last step to add MenuViewController's view to the view hierarchy—we have to tell MenuViewController where to get its view from. Notice that in Step 2.i, we initialized MenuViewController using our convenience initializer, not the designated initializer initWithNibName:bundle:. To provide the view, override the loadView() function in MenuViewController.swift as follows.

```
    override init(nibName nibNameOrNil: String?, bundle nibBundleOrNil:
NSBundle?) {
        super.init(nibName: nibNameOrNil, bundle: nibBundleOrNil)
    }

    required init(coder aDecoder: NSCoder) {

        super.init(coder: aDecoder)

    }

    override func loadView() {
        self.view = NSBundle.mainBundle().loadNibNamed("MenuViewController",
owner: self, options: nil)[0] as UITableView
    }
```

Now, MenuViewController will always load with the assigned nib. Note that since we overrode the loadView() function, we should *not* try to initialize instances of this class via the designated initializer initWithNibName:bundle:.

k. Run the application. You should see the menu with a list of converters instead of the tab bar. But if we tap on the rows in the "Menu" table view, nothing happens! Why?

We need to implement the delegate function `tableView: didSelectRowAtIndexPath:` to push the view controller onto the navigation stack. In MenuViewController.swift, add the following function below the tableView:cellForRowAtIndexPath: function.

```
func tableView(tableView: UITableView, didSelectRowAtIndexPath
indexPath: NSIndexPath) {

    let selectedViewController = viewControllers![indexPath.row]

    self.navigationController?.pushViewController(selectedViewController
, animated: true)

}
```

l. Run the application again. Now when you tap on the rows of the "Menu", the corresponding converter should appear on screen. Assuming your converters from Week 2 worked as expected, you should be able to perform conversions without having to do anything else at this point. Go ahead and give it a try.

3. **Let users save their favorite conversions.**

   Now that we know how to use UITableView inside of a UIViewController, let's practice using a UITableView with UITableViewController (the easier way of doing things). We will also practice creating a true MVC application by creating a model class called Favorites, which will represent the conversions' type, input and output values and units of measurement. Of course, we will need to somehow save these to disk so that our favorites persist between app launches and we will do that, too.

   a. **Create a Favorites model class.** Create a new Swift file and call it Favorite.swift. Declare the properties of this class and provide an initializer as follows.

```
import Foundation

class Favorite: NSObject, NSCoding {

    var conversionType: String

    var inputValue: Double

    var inputUnits: String

    var outputValue: Double

    var outputUnits: String

    required init(conversionType: String, inputValue: Double, inputUnits: String, outputValue:
Double, outputUnits: String) {

        self.conversionType = conversionType

        self.inputValue = inputValue

        self.inputUnits = inputUnits

        self.outputValue = outputValue

        self.outputUnits = outputUnits

        super.init()

    }

}
```

This will allow us create our Favorites and later save them into an array that will drive the Favorites table. Then, we will save that array to disc and write to it when we create new favorites.

b. **Create the 'Favorite' button.** We will add a favorites button to our converters that create an instance of the Favorites model and save it to a location we can later reach from the FavoritesTableViewController we will create in Step 4 to that we can view the list of favorite conversions.

In TemperatureConverterViewController, override the viewDidLoad() function as follows. You will get an error, but we will resolve that in the steps that follow.

```
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.

        let favoriteBarButton = UIBarButtonItem(image: UIImage(named: "favoritesButton_60x60"),
style:
            UIBarButtonItemStyle.Plain, target: self, action: "favoriteButtonPressed:")

        self.navigationItem.rightBarButtonItem = favoriteBarButton

    }
```

Note that the argument for the action argument is a string. In Swift, we reference functions by name (Cf. using @selector in Objective-C). make sure to include the ":" (colon) in the name of functions that take arguments.

i. **Include assets.** We specified the button should be initialized with the image called "favoritesButton_60x60". We need to include this file in our project. Download it from the class home page. Select the folder called "Supporting Files" in the file navigator. Click the

"+" button in the bottom left corner of the file navigator and select "Add Files to 'SwiftConvervns'…". In the dialogue, select the file called favoritesButton_60x60@2x.png, ensure the 'Copy items if needed' checkbox is checked and click 'Add'.

c. **Get ready to save the conversion as a 'Favorite'.** We need to take the result of the conversion calculation and turn it into an instance of the Favorite model class and temporarily store in an array called 'conversions' so that it is available if a user decides to favorite it.

In TemperatureConverterViewController.swift, declare the 'conversions' array and modify your calculateCelsius() and calculateFahrenheit() functions as follows. (We will resolve the error in the next step).

```
class TemperatureConverterViewController: UIViewController, UITextFieldDelegate {

    @IBOutlet weak var fahrenheitTextField: UITextField!

    @IBOutlet weak var celsiusTextField: UITextField!

    var conversions: Array<AnyObject> = []
```

```
    func calculateCelsius(fahrenheitTemp: Double) -> String
    {

        return "\((fahrenheitTemp - 32) * 5 / 9)"

        var celsiusTemp = (fahrenheitTemp - 32) * 5 / 9

        self.createConversionObject(fahrenheitTemp, inputUnits: "ºF", outputValue: celsiusTemp,
outputUnits: "ºC")

        return "\(celsiusTemp)"

    }

    func calculateFahrenheit(celsiusTemp: Double) -> String
    {

        return "\(celsiusTemp * 9 / 5 + 32)"

        var fahrenheitTemp = celsiusTemp * 9 / 5 + 32

        self.createConversionObject(celsiusTemp, inputUnits: "ºC", outputValue: fahrenheitTemp,
outputUnits: "ºF")

        return "\(fahrenheitTemp)"

    }
```

d. **Now, create the Favorite.** We received an error because we called createConversionObject() but we have not implemented it yet. Let's do that now. Implement the following function above calculateCelsius(). (We do it this way to make sure that every conversion performed by the TemperatureConverterViewController gets marked as having a conversionType of "Temperature" when it is saved as a favorite.)

```
    func createConversionObject(inputValue: Double, inputUnits: String, outputValue: Double,
outputUnits: String)
    {

        let favoriteConversion = Favorite(conversionType: "Temperature", inputValue: inputValue,
inputUnits: inputUnits, outputValue: outputValue, outputUnits: outputUnits)

        conversions += [favoriteConversion]

    }
```

e. **Provide the target action function for the 'Favorite' button.** Add the following function above viewDidLoad().

```
func favoriteButtonPressed(sender: AnyObject) {

     // save the favorite instance here...

     // get the last favorite we prepped and store it

     if let data: NSData =
NSUserDefaults.standardUserDefaults().objectForKey("FavoriteConversions") as? NSData {

         var savedFavorites: Array<AnyObject> = NSKeyedUnarchiver.unarchiveObjectWithData(data) as
Array
         if conversions.count > 0 {

             savedFavorites.append(conversions.last!)

             let updatedFavorites = NSKeyedArchiver.archivedDataWithRootObject(savedFavorites)

             NSUserDefaults.standardUserDefaults().setValue(updatedFavorites, forKey:
"FavoriteConversions")

             NSUserDefaults.standardUserDefaults().synchronize()

         }

     }

}
```

(NB: With Obj-C 2.0 and Swift we do not *have* to define function above where they are called in code, but it is still good practice).

Note how we accomplish this. In step 3d, we wrapped our completed conversion into an instance of the Favorite class and added it to the array of conversions that have been performed. Now, when the favorite button is pressed, we look in NSUserDefaults for the data that represents our saved Favorites, use NSKeyedUnarchiver to unarchive that data into an array called savedFavorites, update this array by adding (appending) the most recent conversion (already prepared as an instance of our Favorite class). Then we use NSKeyedArchiver to archive the updated array into a data representation again and store that back into NSUserDefaults.

f. **Conform to the NSCoding protocol.** We have set up our model class, Favorites, now me must provide the mapping so that the coder objects know what to do with the values when they are converted to/from data. Conform to the NSCoding protocol and implement the following methods in Favorite.swift.

```
class Favorite: NSObject, NSCoding {


    required init(coder aDecoder: NSCoder) {

        self.conversionType = aDecoder.decodeObjectForKey("conversionType") as String
        self.inputValue = aDecoder.decodeDoubleForKey("inputValue")
        self.inputUnits = aDecoder.decodeObjectForKey("inputUnits") as String
        self.outputValue = aDecoder.decodeDoubleForKey("outputValue")
        self.outputUnits = aDecoder.decodeObjectForKey("outputUnits") as String

    }
     func encodeWithCoder(aCoder: NSCoder) {

        aCoder.encodeObject(self.conversionType, forKey: "conversionType")
        aCoder.encodeDouble(self.inputValue, forKey: "inputValue")
        aCoder.encodeObject(self.inputUnits, forKey: "inputUnits")
        aCoder.encodeDouble(self.outputValue, forKey: "outputValue")
        aCoder.encodeObject(self.outputUnits, forKey: "outputUnits")

    }
}
```

Ensure there is a location for us to store our Favorites. There is a problem: in the code from Step 3e, we try to load our saved Favorites array so that we can add our new favorites to it – but what it no one has saved any favorites yet, there will be no object in NSUserDefaults for the "FavoriteConversions" key and since we used *optional binding*, the code in the if statement will not be executed and our Favorite will not get saved.

Fortunately, we can resolve this easily by registering default values into NSUserDefaults. Let's add this protection now. In AppDelegate.swift, add the following code inside of application:didFinishLaunchingWithOptions:.

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject: AnyObject]?) -> Bool {
        // Override point for customization after application launch.

        if let data: NSData =
NSUserDefaults.standardUserDefaults().objectForKey("FavoriteConversions") as? NSData {

        }
        else
        {
            let defaultFavorites = NSKeyedArchiver.archivedDataWithRootObject([])

            NSUserDefaults.standardUserDefaults().registerDefaults(["FavoriteConversions" :
defaultFavorites])

            NSUserDefaults.standardUserDefaults().synchronize()

        }
```

Now, when our application starts, it will check if there are pre-existing user defaults for the "FavoriteConversions" key. If not, it will store default value for that key and thus guarantee that location is accessible so that our CRUD (Create, Read, Update, Delete) operations will always succeed.

TemperatureConverterViewController.swift FILE SHOULD APPEAR AS FOLLOWS (NEXT PAGE)

```
import UIKit

class TemperatureConverterViewController: UIViewController, UITextFieldDelegate {

    @IBOutlet weak var fahrenheitTextField: UITextField!

    @IBOutlet weak var celsiusTextField: UITextField!

    var conversions: Array<AnyObject> = []

    override init(nibName nibNameOrNil: String?, bundle nibBundleOrNil: NSBundle?) {

        super.init(nibName: nibNameOrNil, bundle: nibBundleOrNil)

    }

    required init(coder aDecoder: NSCoder) {

        super.init(coder: aDecoder)

    }

    func favoriteButtonPressed(sender: AnyObject) {

        // save the favorite instance here...

        // get the last favorite we prepped and store it

        if let data: NSData =
NSUserDefaults.standardUserDefaults().objectForKey("FavoriteConversions") as? NSData
        {

            var savedFavorites: Array<AnyObject> = NSKeyedUnarchiver.unarchiveObjectWithData(data)
as Array

            if conversions.count > 0
            {

                savedFavorites.append(conversions.last!)

                let updatedFavorites = NSKeyedArchiver.archivedDataWithRootObject(savedFavorites)

                NSUserDefaults.standardUserDefaults().setValue(updatedFavorites, forKey:
"FavoriteConversions")
                NSUserDefaults.standardUserDefaults().synchronize()

            }
        }

    }
```

CONTINUED ON NEXT PAGE

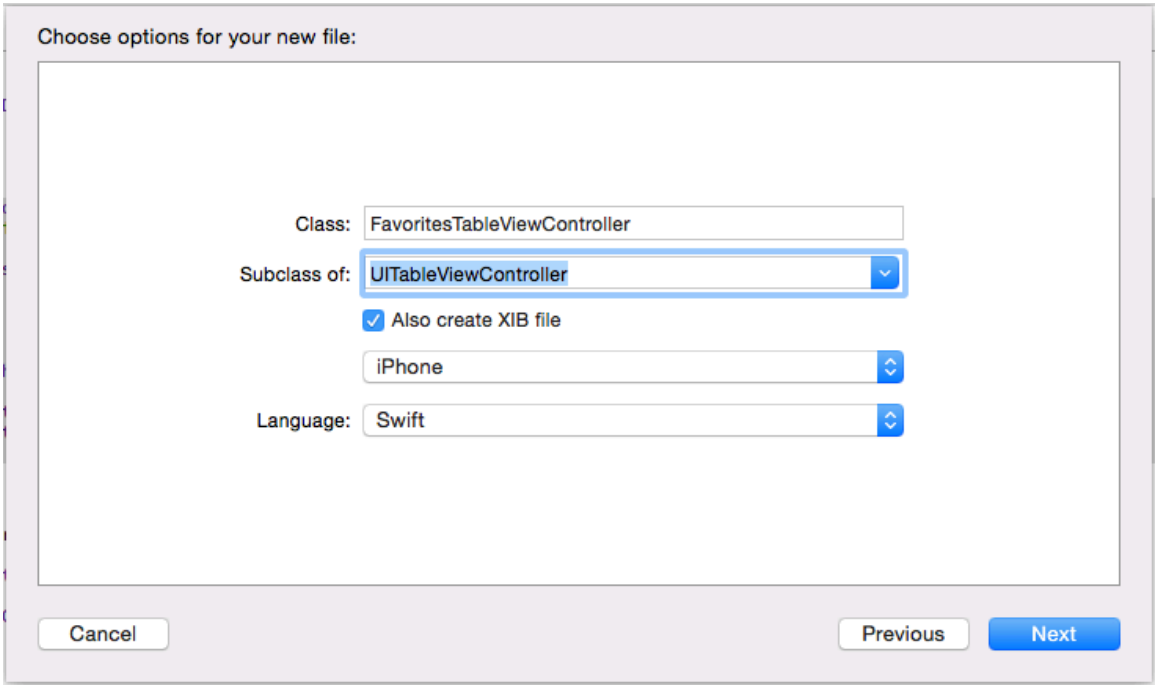
```
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.

        let favoriteBarButton = UIBarButtonItem(image: UIImage(named: "favoritesButton_60x60"),
style:
            UIBarButtonItemStyle.Plain, target: self, action: "favoriteButtonPressed:")

        self.navigationItem.rightBarButtonItem = favoriteBarButton

    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    func createConversionObject(inputValue: Double, inputUnits: String, outputValue: Double,
outputUnits: String)
    {

        let favoriteConversion = Favorite(conversionType: "Temperature", inputValue: inputValue,
inputUnits: inputUnits, outputValue: outputValue, outputUnits: outputUnits)

        conversions += [favoriteConversion]

    }

    func calculateCelsius(fahrenheitTemp: Double) -> String
    {

        var celsiusTemp = (fahrenheitTemp - 32) * 5 / 9

        self.createConversionObject(fahrenheitTemp, inputUnits: "ºF", outputValue: celsiusTemp,
outputUnits: "ºC")

        return "\(celsiusTemp)"

    }

    func calculateFahrenheit(celsiusTemp: Double) -> String
    {

        var fahrenheitTemp = celsiusTemp * 9 / 5 + 32

        self.createConversionObject(celsiusTemp, inputUnits: "ºC", outputValue: fahrenheitTemp,
outputUnits: "ºF")

        return "\(fahrenheitTemp)"

    }

    func textFieldShouldReturn(textField: UITextField) -> Bool
    {

        if textField == fahrenheitTextField
        {
```

CONTINUED ON NEXT PAGE

4. **View the Favorite Conversions.**

   Now that we have created a way for users to save their Temperature conversions, let's enable them to view the list of Favorites. We will need to create table view as the UI for this (we will user UITableViewController this time). We also need to add the Favorites screen to the Menu so that users can access it.

   a. **Create the UI of the Favorites Table.** Create a new Cocoa Touch class and configure it as follows.



   b. **Remember how UITableViewController works.** Open the FavoritesTableViewController.xib and right click on the table view. Notice that the delegate and dataSource outlets have been set for us already.

   Now look at FavoritesTableViewController.swift and notice that we inherit from UITableViewController. Remember that

UITableViewController conforms to the UITableViewDelegate and UITableViewDataSource protocols for us, and Xcode provided stubs for us for the required protocol methods, so now we are all set to start implementing those methods.

c. **Load the array of Favorites.** We need to load the Favorites array into memory from NSUserDefaults so that we can use it to drive the table view. In FavoritesTableViewController.swift, add a property to hold this array and override viewWillAppear as follows so that we are assured the table view's data will be current every time the view comes on screen.

```
import UIKit

class FavoritesTableViewController: UITableViewController {

    var favorites: Array<AnyObject> = []
```

```
override func viewWillAppear(animated: Bool) {

        if let data: NSData =
NSUserDefaults.standardUserDefaults().objectForKey("FavoriteConversions") as? NSData
        {

            if var savedFavorites: Array<AnyObject> =
NSKeyedUnarchiver.unarchiveObjectWithData(data) as? Array
            {

                self.favorites = savedFavorites

            }

        }
    }
```

d. **Prepare a cell for reuse.** In FavoritesTableViewController.swift, add the following initializers to prepare a cell class for reuse and configure the title of our view controller so that the word “Favorites” appears in the Menu table view and in the title label in the navigation bar. (We will add the Favorites screen to the Menu in Step 5.)

```
override init(nibName nibNameOrNil: String?, bundle nibBundleOrNil: NSBundle?) {

    super.init(nibName: nibNameOrNil, bundle: nibBundleOrNil)

     self.title = "Favorites"

    self.tableView.registerClass(UITableViewCell.self, forCellReuseIdentifier: "Cell")

}

required init(coder aDecoder: NSCoder) {
    super.init(coder: aDecoder)
}
```

e. **Provide the datasource and configure the cells to display the user's favorite conversions.** Implement the following functions in FavoritesTableViewController.swift.

```
override func numberOfSectionsInTableView(tableView: UITableView) -> Int {
        // #warning Potentially incomplete method implementation.
        // Return the number of sections.
        return 1
    }

    override func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        // #warning Incomplete method implementation.
        // Return the number of rows in the section.
        return favorites.count
    }

    override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -
> UITableViewCell {

        let cell = tableView.dequeueReusableCellWithIdentifier("Cell", forIndexPath: indexPath) as
UITableViewCell

        // Configure the cell...

        let favorite: Favorite = favorites[indexPath.row] as Favorite

        cell.textLabel.text = NSString(format: "%f%@ = %f%@", favorite.inputValue,
favorite.inputUnits, favorite.outputValue, favorite.outputUnits)

        return cell
    }
```

AT THIS POINT, FavoritesTableViewController.swift SHOULD APPEAR AS FOLLOWS

(NEXT PAGE)

```
import UIKit

class FavoritesTableViewController: UITableViewController {

    var favorites: Array<AnyObject> = []

    override init(nibName nibNameOrNil: String?, bundle nibBundleOrNil: NSBundle?) {

        super.init(nibName: nibNameOrNil, bundle: nibBundleOrNil)

         self.title = "Favorites"

        self.tableView.registerClass(UITableViewCell.self, forCellReuseIdentifier: "Cell")

    }

    required init(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
    }

    override func viewWillAppear(animated: Bool) {
        if let data: NSData =
NSUserDefaults.standardUserDefaults().objectForKey("FavoriteConversions") as? NSData
        {

            if var savedFavorites: Array<AnyObject> =
NSKeyedUnarchiver.unarchiveObjectWithData(data) as? Array
            {

                self.favorites = savedFavorites

            }

        }
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        // Uncomment the following line to preserve selection between presentations
        // self.clearsSelectionOnViewWillAppear = false

        // Uncomment the following line to display an Edit button in the navigation bar for this
view controller.
        // self.navigationItem.rightBarButtonItem = self.editButtonItem()

    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    // MARK: - Table view data source

    override func numberOfSectionsInTableView(tableView: UITableView) -> Int {
        // #warning Potentially incomplete method implementation.
        // Return the number of sections.
        return 1
    }
```

CONTINUED ON NEXT PAGE

```
override func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        // #warning Incomplete method implementation.
        // Return the number of rows in the section.
        return favorites.count
    }

    override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -
> UITableViewCell {

        let cell = tableView.dequeueReusableCellWithIdentifier("Cell", forIndexPath: indexPath) as
UITableViewCell

        // Configure the cell...

        let favorite: Favorite = favorites[indexPath.row] as Favorite

        cell.textLabel.text = NSString(format: "%f%@ = %f%@", favorite.inputValue,
favorite.inputUnits, favorite.outputValue, favorite.outputUnits)

        return cell
    }

}
```

5. **The Home Stretch – Add 'Favorites' to the "Menu"**

   Now, we have everything we need to allow users to save and view their favorites. All that is left is to add an option for 'Favorites' to the "Menu" table view. Add the following code to the bottom of the application:didFinishLaunchingWithOptions: function in AppDelegate.swift.

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject: AnyObject]?) -> Bool {

.
.
.
     let favoritesViewController = FavoritesTableViewController(nibName:
"FavoritesTableViewController", bundle: NSBundle.mainBundle())

     let menuVC = MenuViewController(viewControllers: [temperatureConverterViewController,
distanceConverterViewController, volumetricConverterViewController, favoritesViewController])

     menuVC.title = "Menu"

     let mainNavigationController = UINavigationController(rootViewController: menuVC)

     self.window?.rootViewController = mainNavigationController

     self.window?.addSubview(tabBarController.view)

     self.window?.makeKeyAndVisible()

     return true

    }
}
```

Build and Run the application. If you have no errors, you should be able to do a temperature conversion, tap the Favorite (Heart) button, then go back to the Menu, select 'Favorites' and see the conversion show up in a user-friendly string.

CHALLENGES ON NEXT PAGE

- **Challenge 1:**
    - Enable users to save the other types conversions you allow them to perform (i.e. Distance and Volumetric)
- **Challenge 2:**
    - Create a custom table view cell that shows an icon for the type of conversion – Temperature, Distance, or Volumetric as well as the conversion itself.
- **Challenge 3:**
    - Modify the Favorites table. Break the favorite conversions into sections based on the conversion type. (Hint: You will need to somehow provide an array to drive each section of the table view.)