Week 5: Storyboards continued and Autolayout

Topics covered:

- Storyboards (continued)
- Autolayout (Demo and BNR Ch. 15, 16, 17)
- Universal Applications
- A nod to Objective-C

In-class:

- I will demonstrate to students my experiences using Autolayout and demonstrate the material from Ch. 15 – 17 in the BNR book in Objective-C but the focus will be on Autolayout.
- Implementation notes:
  - Need to handle orientation changes

Homework:

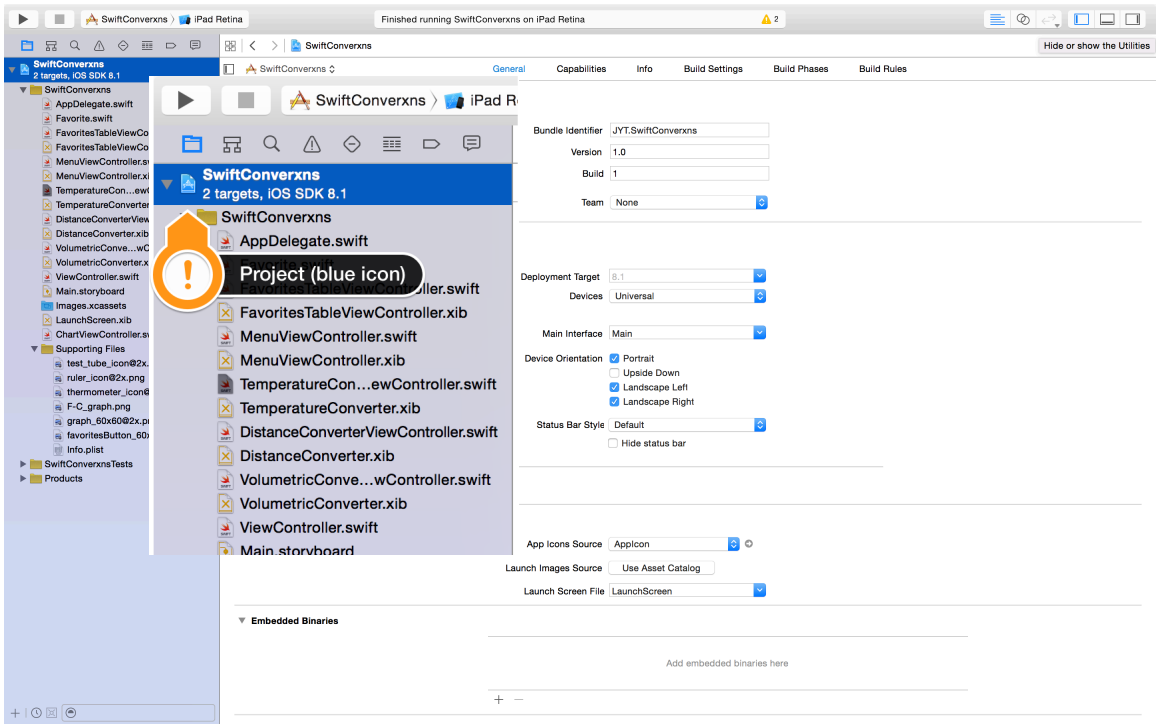
Reading:

Coding:

- I grant this week as a catch-up week for students and therefore I will not collect homework for Week 5 but in order to give students an opportunity to practice the new material I am providing the following exercise.
- Students will make their application run on both iPhone and iPad. We will place the camera inside a UIPopoverController if we are on an iPad

How to accomplish this:

1. **Setup the project.** Create a copy of your project folder from Week 4 and rename your folder for Week 5.
2. **Make your application run on iPad.** From the 'File Navigator' in Xcode, select your project.



In the main area that appears to the right, set the 'Devices' property to 'Universal' if it is not already. This will allow your application to be compiled to run natively on iPad.

3. **Make use of iPad design paradigms.** Since our iPad application has the same UI as our iPhone application and since we used Autolayout to dynamically position and size our views (mainly our text fields, labels, and the panHandle object), this is all we need to do to make our application run on iPad. But we can still improve things a bit.

a. First, let's make sure that everything is laid-out correctly in both portrait and landscape. Use what you learned in class in Week 5 to ensure your toolbars and text fields (and everything else) look neat and take up space on the screen properly for an iPad.

b. Now, let's get some practice using UIPopoverController to present the UIImagePicker in a popover—only if we are on an iPad. We will start with the temperature converter (later, you can repeat these steps on your own to show the camera for the other converters in a popover).

In TemperatureConverterViewController.swift, modify your cameraButtonPressed() function so that it appears as follows.

```
@IBAction func cameraButtonPressed(sender: AnyObject) {

        let imagePicker = UIImagePickerController()

        imagePicker.delegate = self

        if (UIImagePickerController.isSourceTypeAvailable(.Camera))
        {
            imagePicker.sourceType = UIImagePickerControllerSourceType.Camera
        }
        else
        {
            imagePicker.sourceType = .SavedPhotosAlbum
        }

        if (UIDevice.currentDevice().userInterfaceIdiom == .Phone)
        {

            presentViewController(imagePicker, animated: true, completion: nil)

        }
        else
        {

            var popover = UIPopoverController(contentViewController: imagePicker)

            popover.delegate = self

            popover.presentPopoverFromBarButtonItem(cameraBarButton,
permittedArrowDirections: .Any, animated: true)

        }

    }
```

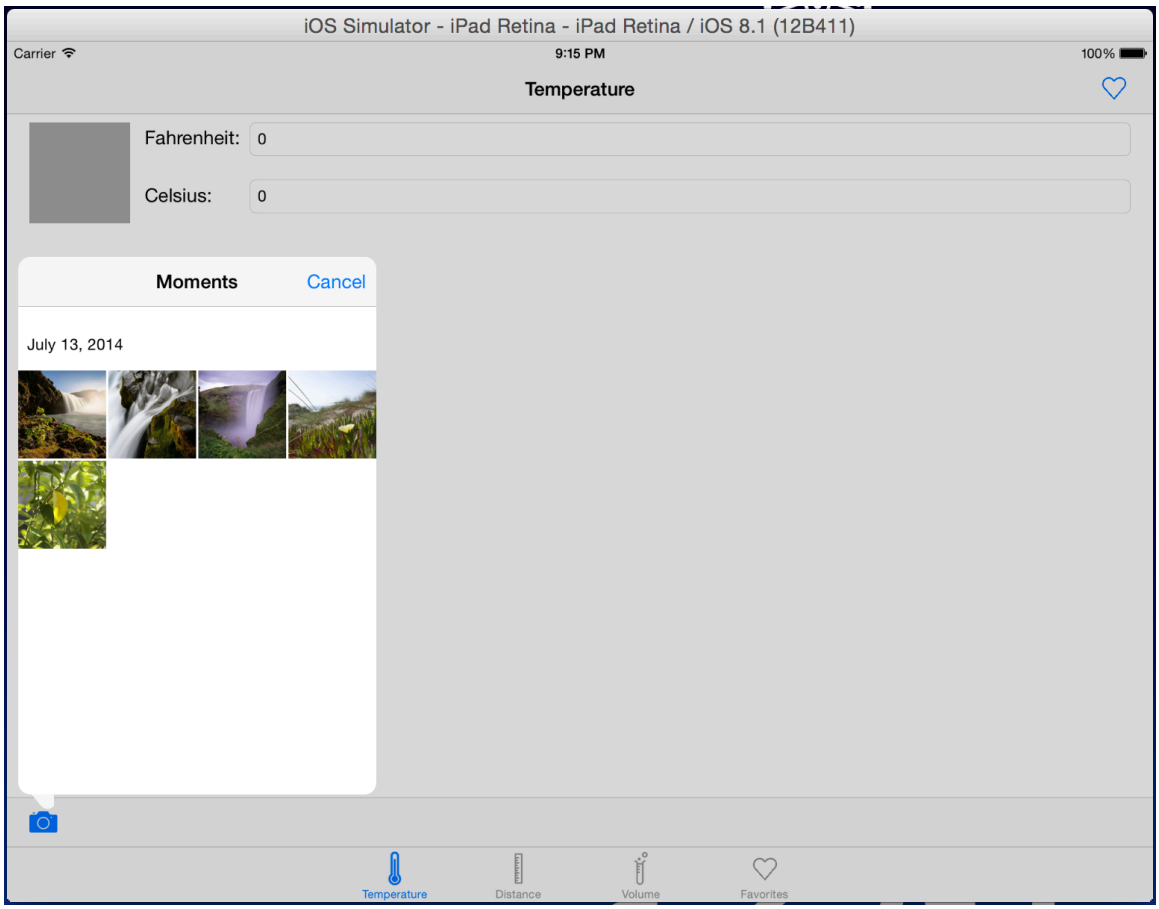
First, we inspect the type of device being used via the `userInterfaceIdiom` property of the current device. If we are on a phone, we simply present the `imagePicker` modally the same as we did in Week 4. Otherwise, we must be on an iPad (based on the devices on which we are currently allowed to run iOS applications), so we instantiate the popover, set the delegate, and finally, tell the popover to present itself from the cameraBarButton (remember, popovers have an arrow to indicate "where the popover is coming from", so that is why the method has to present it "from" the bar button item.)

Don't forget to conform to the UIPopoverControllerDelegate protocol in your class declaration.

```
class TemperatureConverterViewController: UIViewController, UITextFieldDelegate,
UIGestureRecognizerDelegate, UIImagePickerControllerDelegate,
UINavigationControllerDelegate, UIPopoverControllerDelegate {
```

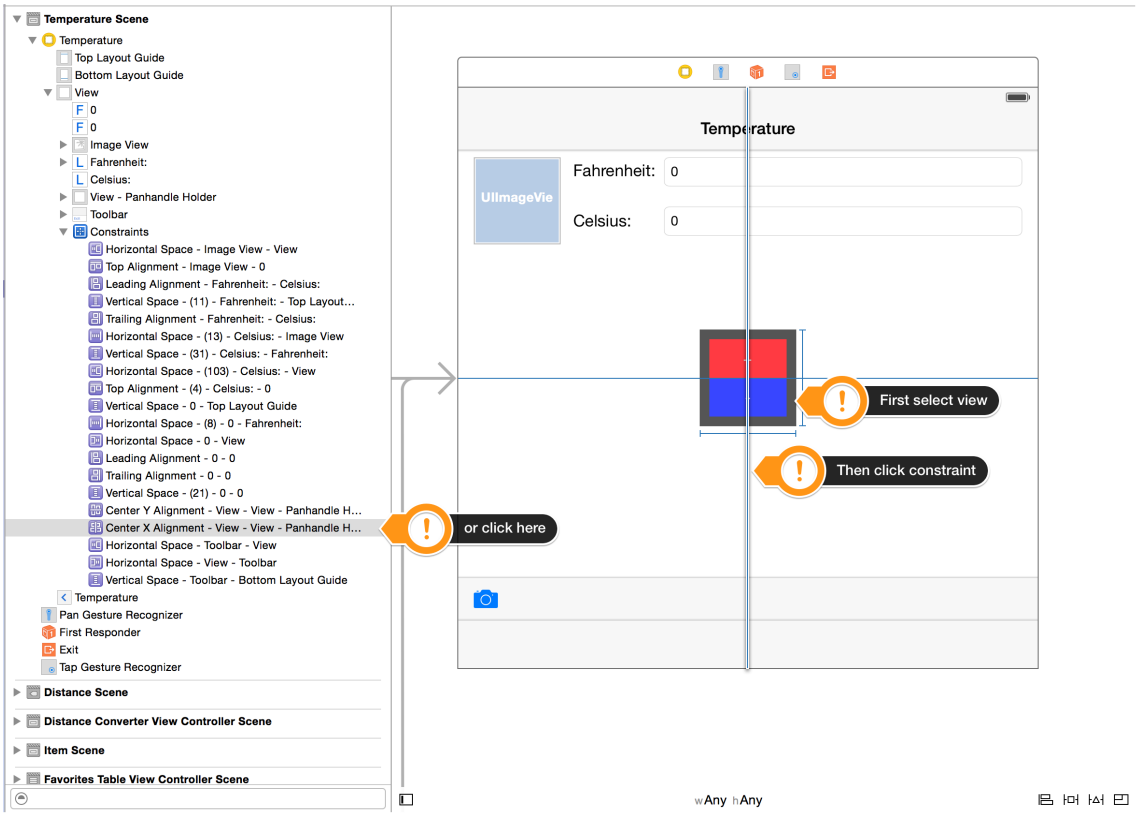
That's it! We don't have to dismiss the popover because the popover contains the imagePicker and we preserved the code to dismisss imagePicker. Look at the documentation for UIViewController and read the discussion on the dismissViewControllerAnimated: method (you can get there quickly by Option-clicking the function in your code then selecting the UIViewController Class Reference link from the window that appears.) You'll see that calling this method on a presented view controller also traverses *up* the hierarchy of presented view controllers and removes the top level object from screen in an animated fashion.

When you press the camera button, the imagePicker should be presented as follows if on an iPad.



4. **Take advantage of Autorotation with Autolayout.** Since we have done a nice job using Autolayout, our views should look good in landscape or in portrait. One little issue is that when in landscape, it is a little hard to reach the panHandle with our thumb, so let's move it closer to the edge dynamically if we are in landscape (or rotate the device into a landscape orientation).

In Interface Builder, select the constraint that keeps the panHandleHolder centered horizontally in its container (aka its superview). The simplest way to do this is to first select the panHandleHolder in the dock, then select the constraint that says ‘Center X Alignment…’ (or click the blue line that appears running up and down the center of your view).



Now, create an IBOutlet to this constraint and call it `panHandleHolderHorizontalCenterToSuperviewConstraint`. Again, remember to always name your constraint outlets with the name of the view or views related by the constraint and what is being constrained (here, the panHandleHolder and its superview are related and we are aligning their horizontal centers). Also make sure to always use the word ‘constraint’ in the name of the outlet just to make sure you never get confused.

a. Now, add the following function inside of TemperatureConverterViewController.swift below the viewDidLoad function.

```
override func willAnimateRotationToInterfaceOrientation(toInterfaceOrientation:
UIInterfaceOrientation, duration: NSTimeInterval) {

        if (UIDevice.currentDevice().userInterfaceIdiom == .Pad)
        {
            if toInterfaceOrientation == .LandscapeLeft ||
                toInterfaceOrientation == .LandscapeRight
            {

                panHandleHolderHorizontalCenterToSuperviewConstraint.constant = 300

            }
            else
            {
                panHandleHolderHorizontalCenterToSuperviewConstraint.constant = 0
            }

        }

    }
```

Here, we make sure that we only move the panHandle if we are on an iPad and if we are in a landscape orientation, then if we are, we offset the center of the panHandleHolder by 300 points by modifying the constant of its Center X alignment constraint. (Remember that if we are using autolayout, we cannot just give the view a new frame, we must instead modify the constant of its size constraints.) If we are not on an iPad or we are not landscape, then we make sure the panHandleHolder is aligned with the center of its superview by setting the constant back to zero (0).

With your device or simulator in in Portrait, run the application and then rotate the device left or right into landscape. You should see the panHandle is now closer to the left-hand side of the screen. Stop the application and run it again, leaving the device in landscape. You should notice that even though we are in landscape, the

panHandleHolder is still in the center of the screen. This is because the code to move it to the side only gets triggered *when we rotate the device into landscape*.

To make it also do the same if the view comes onto screen while already in landscape, override viewWillAppear: as follows.

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

  if (UIDevice.currentDevice().userInterfaceIdiom == UIUserInterfaceIdiom.Pad)
  {
        if UIApplication.sharedApplication().statusBarOrientation == .LandscapeLeft ||
                UIApplication.sharedApplication().statusBarOrientation == .LandscapeRight
        {

                panHandleHolderHorizontalCenterToSuperviewConstraint.constant = 300

        }
       else
       {

                panHandleHolderHorizontalCenterToSuperviewConstraint.constant = 0

        }

  }

}
```

Now, when the view comes on screen, it can get the current orientation and adjust the UI accordingly.

Your temperature converter should now appear as follows.