

Adaptive Layout

Gene Whitaker



@gene_whitaker

@bombingbrain

gene@bombingbrain.com

What is Adaptive Layout?

- iOS 8's new default layout system
- De-couples UI and device screen dimensions
- Dynamically Adapts UI to varying screen sizes
- Tightly coupled with Auto-Layout
 - Defines layout in terms of UI objects and constraints...
 - on the objects themselves (eg. size, ratio)
 - relative to other objects (eg. spacing)
 - relative to parent (eg. pinning, scaling)
- Uses *Size Classes* to generically describe view layouts
- Views should be designed to conform to size class combinations, rather than specific pixel dimensions

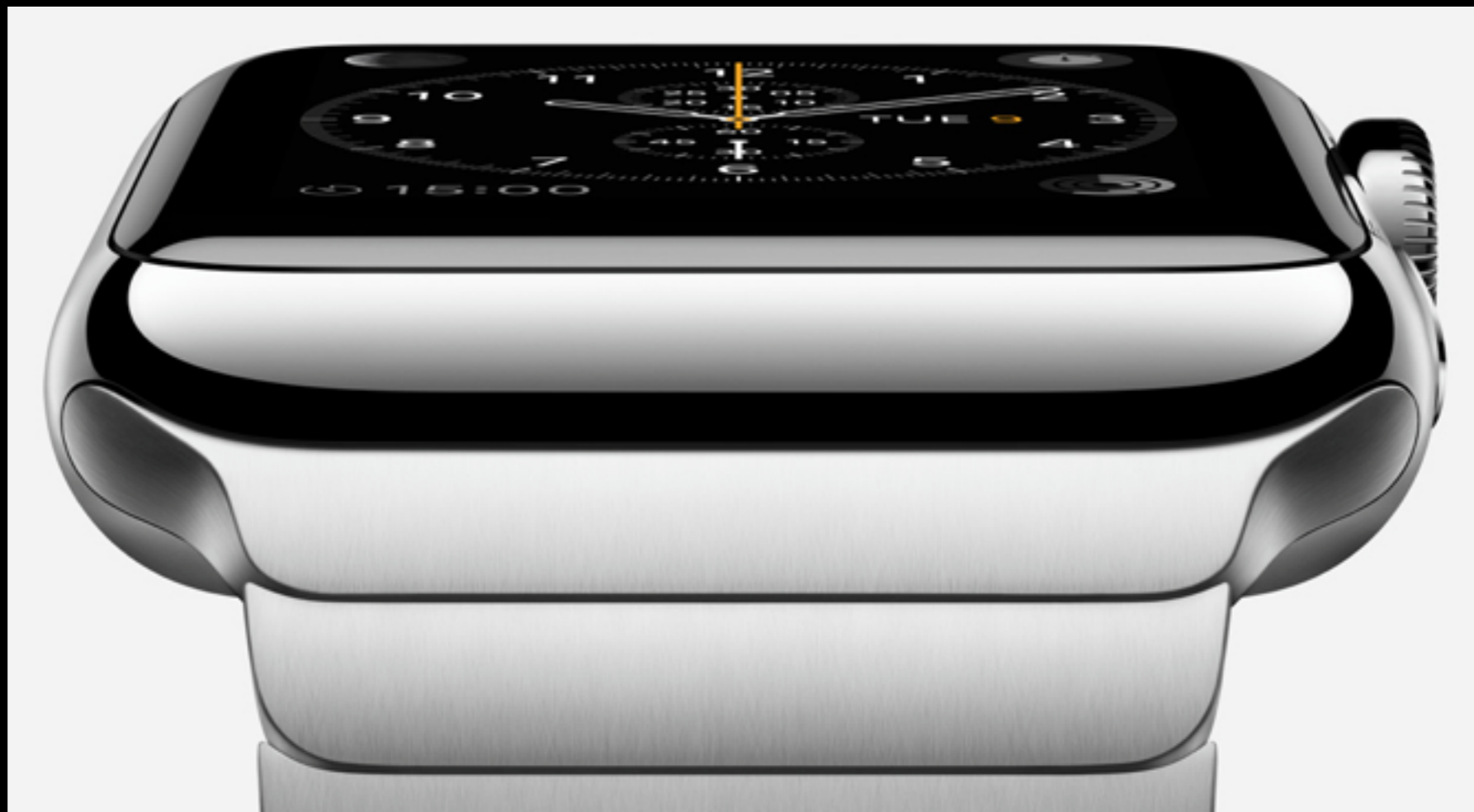
Why Adaptive Layout?

...and by extension, why auto layout?

- Greatly reduces the need for complex, error-prone layout code
- Define layout visually in Interface Builder
- Simplified (in many cases automatic) rotation layout handling
- Universal Storyboards (eliminates need for device-specific XIBs/Storyboards)
- Flexibility to support a wider range of screen sizes
- iOS evolving as a platform...







...a wider range of devices



Size Classes

- Generically describe layout in terms of horizontal and vertical size
- *Three enumerations - Regular, Compact, Unspecified*
- Replaces *UIInterfaceOrientation* and *UIInterfaceIdiom*
- Design for *size class combinations* instead

Mapping Size Classes to Current Device Lineup

HORIZONTAL	VERTICAL	DEVICE	
regular	regular	iPad (both orientations)	
compact	regular	iPhone (portrait)	
regular	compact	iPhone 6+ (landscape)	
compact	compact	iPhone 4, 5, 6 (4.7-inch) (landscape)	

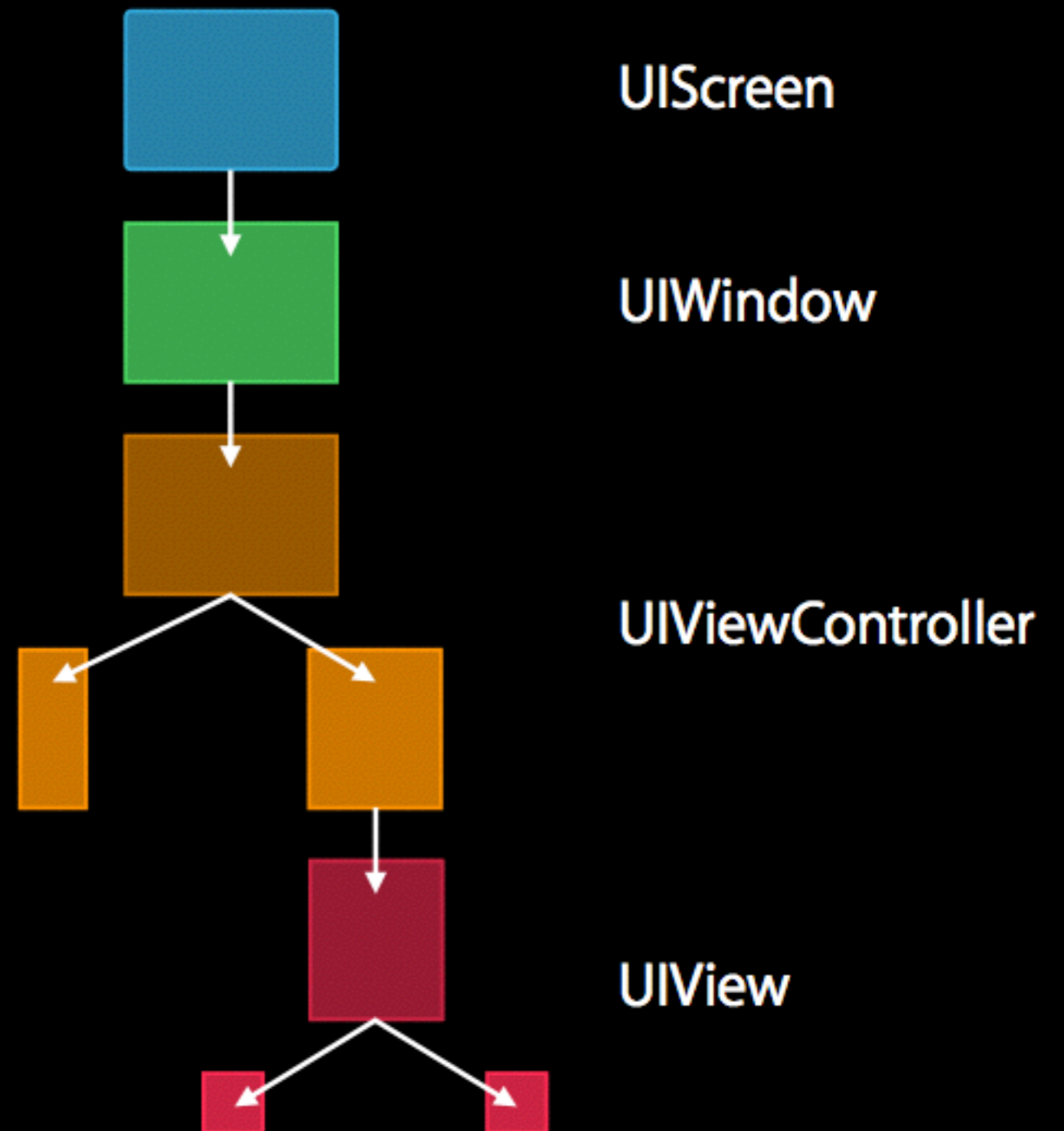
Traits and TraitCollections

- Horizontal and vertical size class are identified as *traits*
- Traits can be accessed in a *UITraitCollection*, which is implemented in a *Trait Environment*
 - New protocol implemented in UIWindow, UIScreen, UIViewController, UIView, and UIPresentationController
- *UITraitCollection* Structure:
 - Horizontal size class - Regular / Compact
 - Vertical size class - Regular / Compact
 - User interface idiom - iPhone / iPad / iPod Touch
 - Display scale - 1.0 / 2.0
- Can be accessed to determine current size class (self.traitCollection)
Example:

```
UIUserInterfaceSizeClass horizontalSizeClass =  
self.traitCollection.horizontalSizeClass;  
if (horizontalSizeClass == UIUserInterfaceSizeClassCompact) {  
    //Do some specialized work  
}
```

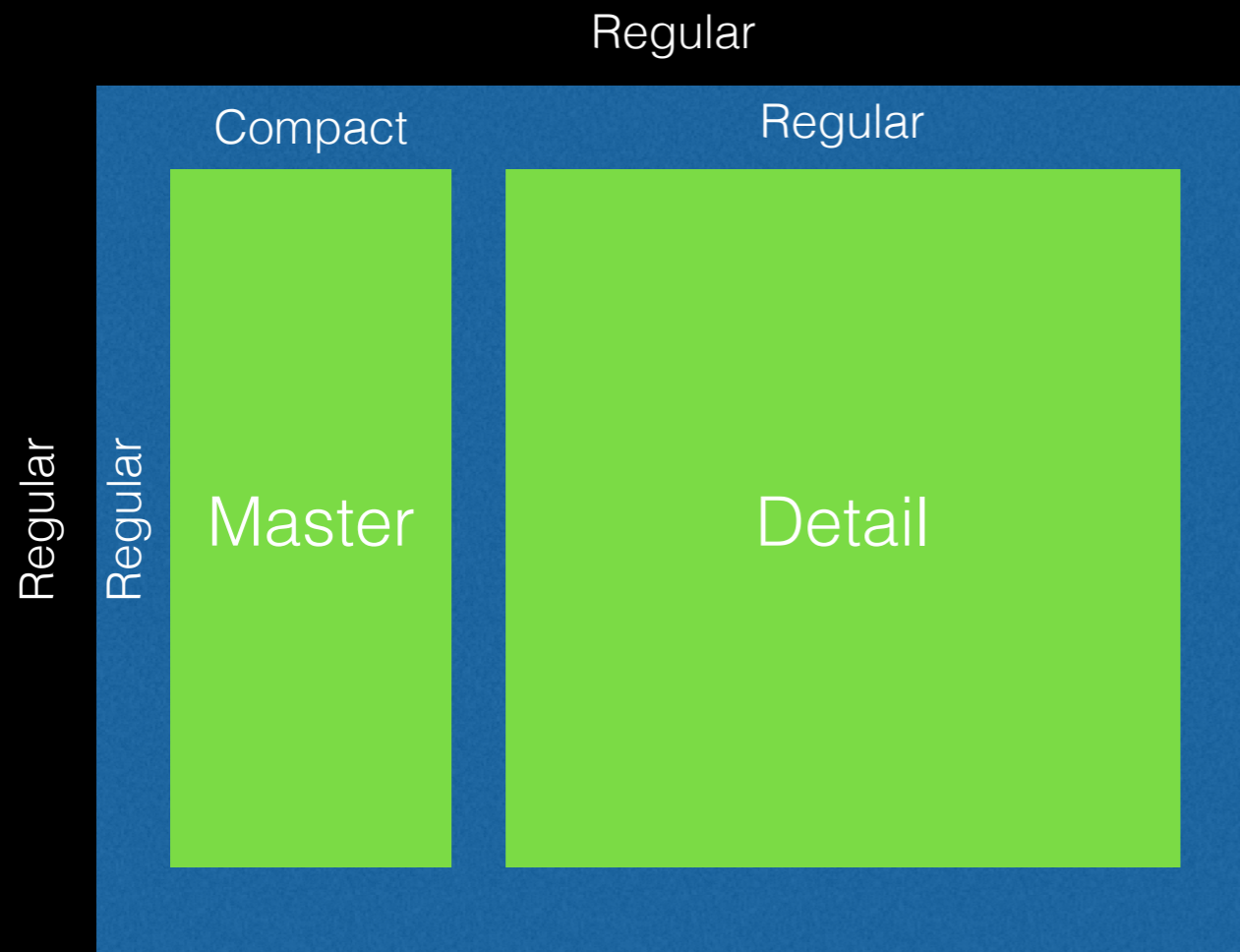
Trait Environment

- Subviews inherit trait collections of the parent
- Parent View Controllers can override the trait collections of child view controller
- Allows custom layout in certain scenarios

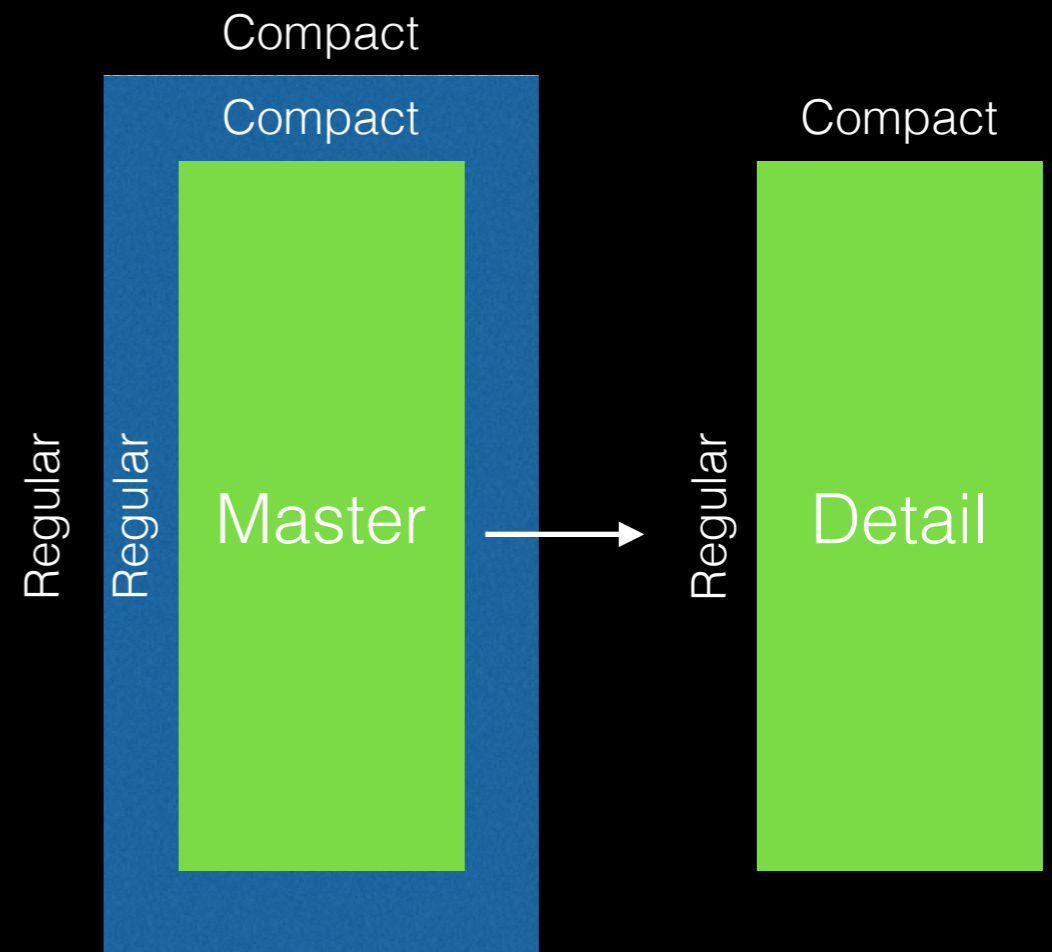


Overriding Trait Collection Example - UISplitViewController

iPad



iPhone



Overriding Trait Collection

```
UITraitCollection *customTraits = [UITraitCollection  
traitCollectionWithVerticalSizeClass:UIUserInterfaceSizeClassCompact];
```

```
UITraitCollection *combinedTraits = [UITraitCollection  
traitCollectionWithTraitsFromCollections:@[self.traitCollection, customTraits]];
```

```
[self setOverrideTraitCollection:combinedTraits  
forChildViewController:self.childViewController];
```

self.traitCollection

customCollection

combinedTraits

Horizontal	Regular	+	Horizontal	(unspecified)	→	Horizontal	Regular
Vertical	Regular		Vertical	Compact		Vertical	Compact
Vertical	iPad		Vertical	(unspecified)		Vertical	iPad
Scale	2.0		Scale	(unspecified)		Scale	2.0

Responding to Changes in Trait Collection

- Use `willTransitionToTraitCollection:withTransitionCoordinator:` and `traitCollectionDidChange:` to add any custom animations/transitions and completions

Adaptive Segues

- Interface Builder now includes *Adaptive Segues*
- Show
- Show Detail
- Present Modally
- Popover

Showing and Presenting View Controllers

- Adaptively shows/presents view controllers appropriately based on containing view controller/parent trait environment
- Examples:
 - Navigation Controller
 - SplitViewController
 - Popovers
 - Alerts
- Child view controllers do not need to be aware of their parent view controller to show or present another view controller
 - Use `[self showViewController:animated:]` instead of `[self.navigationController pushViewController:animated:]`
 - Use `[self presentViewController:animated:]` for popovers and action sheets

Interface Builder in Xcode 6

- Size class support, live previews, layout previews
- Allows most layout to be done visually, minimizing code
- Default generic view defines layouts that should work in in any width, and any height
- Use the size class selector to setup size class-specific objects
- UI Objects, constraints, and some properties can be *installed* (or *not installed*) per size class combination

Interface Builder in Xcode 6

Items that are size class “aware”

- UIView objects
- Constraint objects
- Constraint *constants*
- Font property on many objects (new!)

What is adaptable per size class

- Size/position of views (by altering constraints constants)
- Presence/absence of view
- Presence/absence of constraints
- Font face/size

Limitations/Warnings

NOT size class-aware:

- Most *properties* of UI objects (eg. label color; button text; collection view layout attributes)
- *Properties* of constraints (priority/multiplier)

Solution

- Handle in code
- Use separate objects
- Use size class-specific constraints

Other Warnings

- Be careful which size class is selected when editing
 - When a specific size class is selected - items added are *exclusive* to that size class
- Be careful which version of Xcode is opening the project
 - Xcode 5 will undo all of your size class customizations and save the storyboard!
 - Use source control and commit your storyboard often

Consideration - Orientation-specific Layout

- In iOS 8, drastic differences in layout between orientation are *discouraged*
- Example - iPad, size class is “regular, regular” for both orientations, no distinction
- What about *minor customizations* based on orientation? How to implement?
- Use `viewWillTransitionToSize:withTransitionCoordinator:` to intercept orientation change
 - Handle custom layout in code
 - Override size class of child view controller

Asset Library

- Asset library is size-class aware in Xcode 6
- Allows separate versions of an image to be used per size class

More information...

WWDC 2014 Session Videos:

216 - Building Adaptive Apps with UIKit

228 - A Look Inside Presentation Controllers

411 - What's New in Interface Builder

Apple Developer Pre-Release Documentation:

<http://tinyurl.com/size-classes>

Demo

Demo Notes - Adaptive Popover

- The *Info* button at the top right demonstrates iOS 8's adaptive popover presentation
 - The code to add the “Done” button in the iPhone adaptation is in `RootTableViewController` - see `prepareForSegue` and the `UIPopoverPresentationControllerDelegate` methods

Demo Notes - Adaptive Layout Example

- The *RootTableViewController* is itself an example of adaptive layout - see how the subtitle is only shown when running in a regular horizontal size class (iPad)
- The *adaptive layout example* demonstrates what you can do in IB without even implementing any custom code
 - Play around with the different size classes in IB to see the differences, what's enabled/disabled in each size class combination
 - Special thanks to Joe for being a good sport

Demo Notes - Complex Adaptive Layout

- The **complex adaptive layout example** shows how you can override size classes of child view controllers
 - In *OverrideNavigationController*, the **viewWillTransitionToSize:withTransitionCoordinator:** method overrides the size class of its child view controller depending on the size being transitioned to.
 - The regular width, compact height size class combination is used to define my layout for a large device (iPad) in landscape.
 - Try commenting out **viewWillTransitionToSize:withTransitionCoordinator:** and **setCustomLayoutForSize** methods and run this view on an iPad to see the difference when the override is *not* in place.
 - In *OverrideViewController*, un-comment **willTransitionToTraitCollection:withTransitionCoordinator:** to see an example the child view controller detecting and *responding* to changes in its trait collection. In this case, I'm changing how the setlist/notes view animates during the transition.