Stanford CS193p

Developing Applications for iOS
Spring 2016

Today

- Multiple MVCs
  - Segues
  - Demo: Emotions in FaceIt
- View Controller Lifecycle
  - Demo: VCL in FaceIt

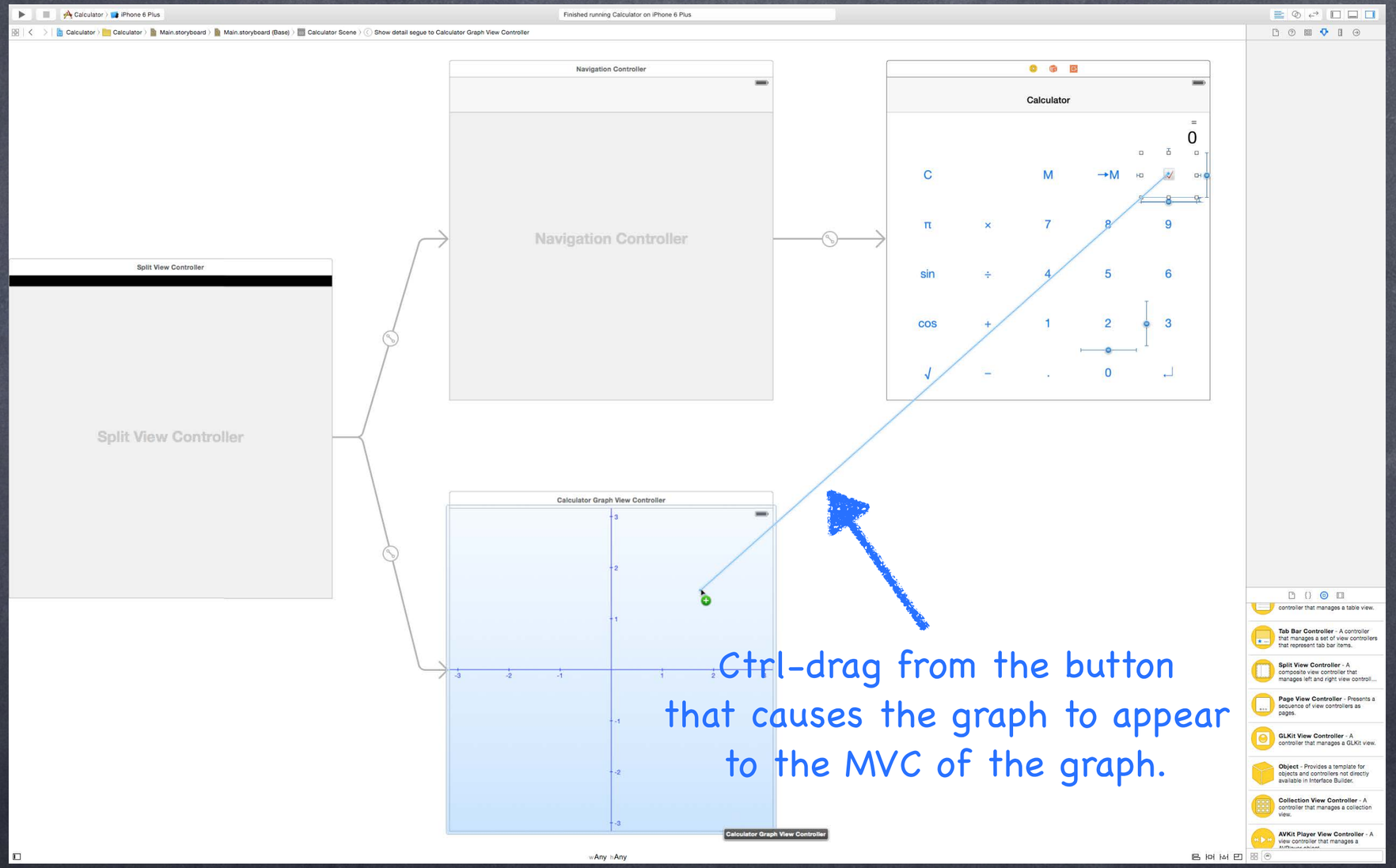
Segues

- We've built up our Controllers of Controllers, now what?
  - Now we need to make it so that one MVC can cause another to appear
  - We call that a "segue"
- Kinds of segues (they will adapt to their environment)
  - Show Segue (will push in a Navigation Controller, else Modal)
  - Show Detail Segue (will show in Detail of a Split View or will push in a Navigation Controller)
  - Modal Segue (take over the entire screen while the MVC is up)
  - Popover Segue (make the MVC appear in a little popover window)
- Segues always create a new instance of an MVC
  - This is important to understand
  - The Detail of a Split View will get replaced with a new instance of that MVC
  - When you segue in a Navigation Controller it will not segue to some old instance, it'll be new

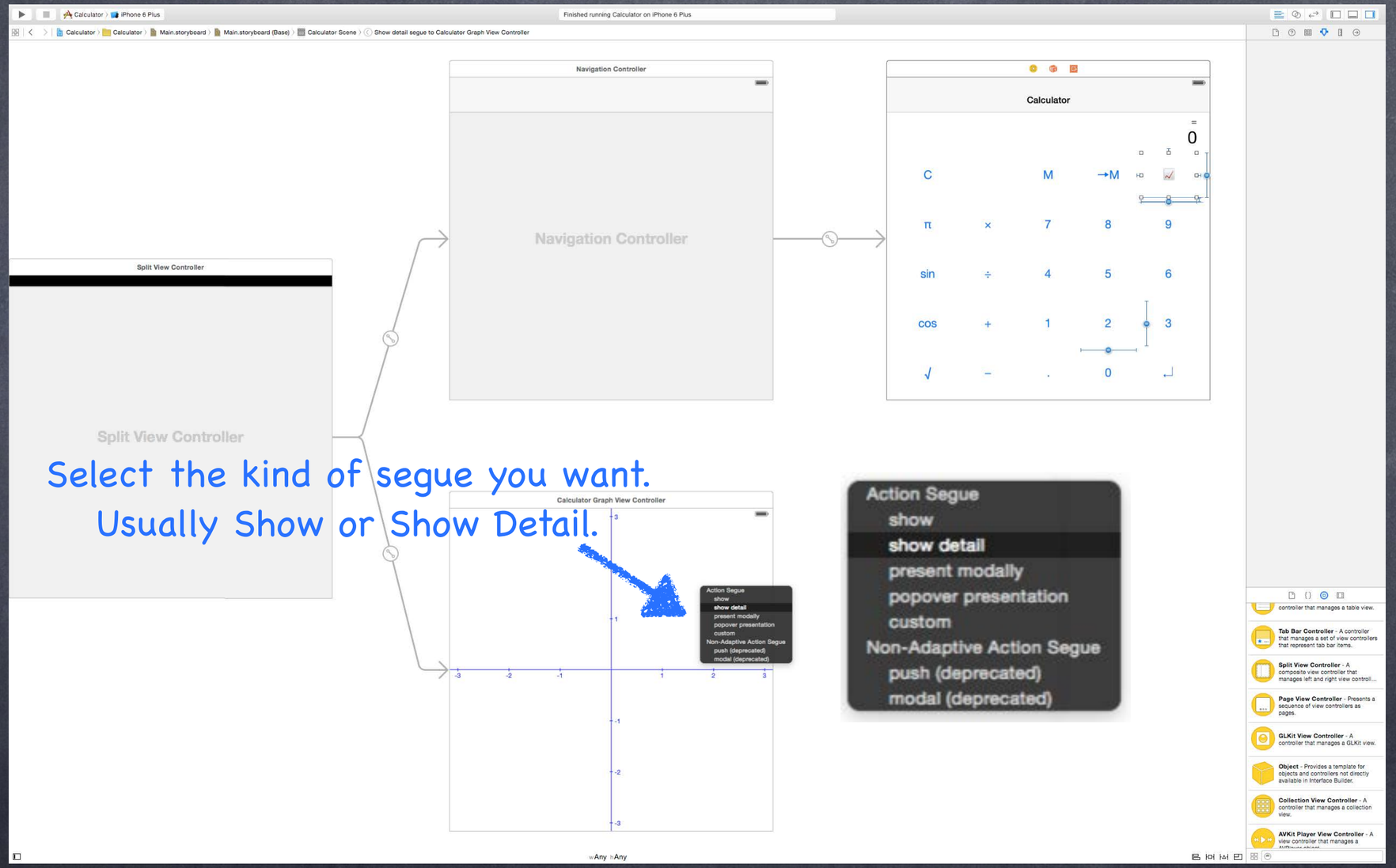
Segues

- How do we make these segues happen?
  - Ctrl-drag in a storyboard from an instigator (like a button) to the MVC to segue to
  - Can be done in code as well

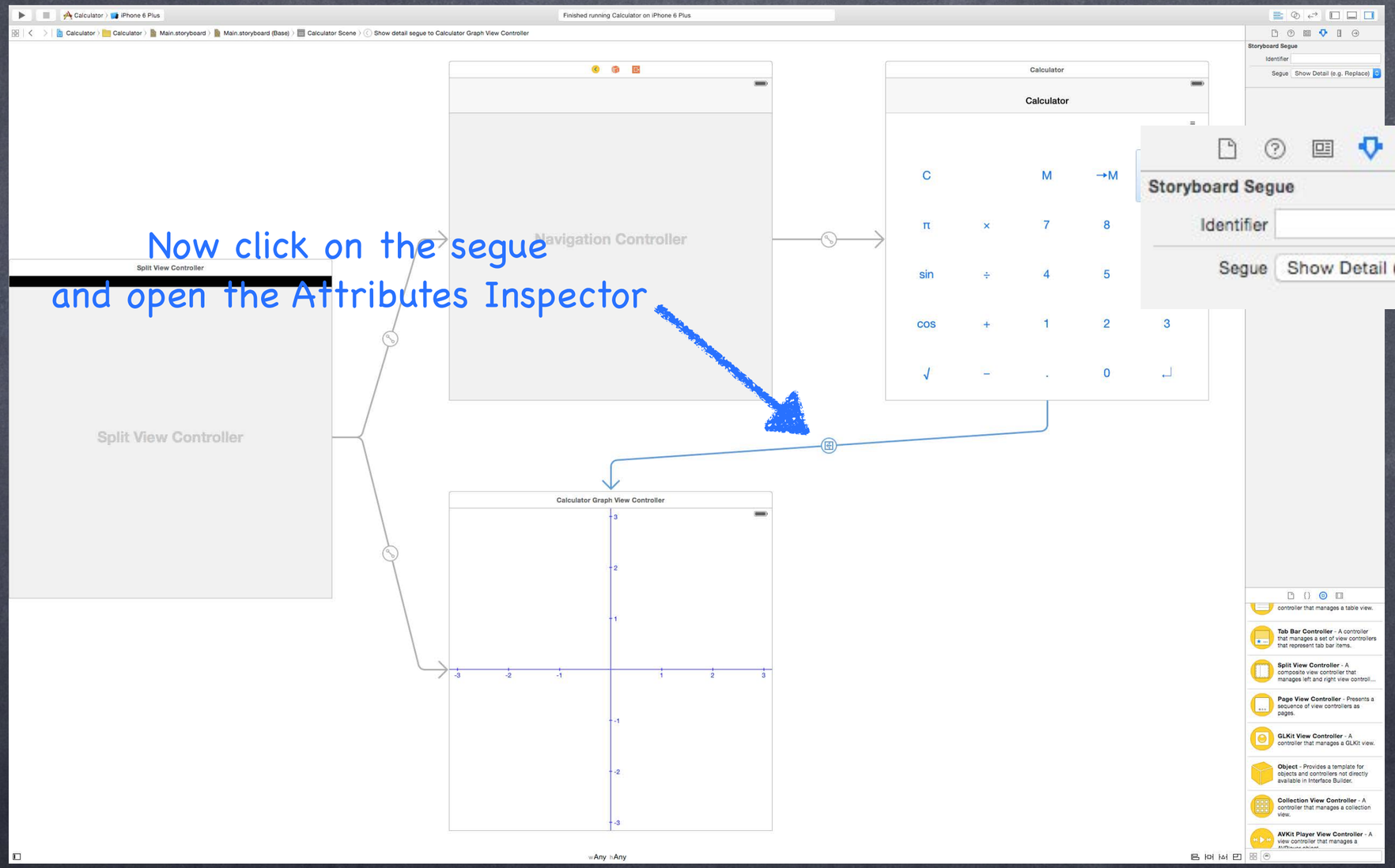
Segues


Navigation Controller
Navigation Controller
Calculator
Split View Controller
Split View Controller
Calculator Graph View Controller
Calculator Graph View Controller
Ctrl-drag from the button that causes the graph to appear to the MVC of the graph.

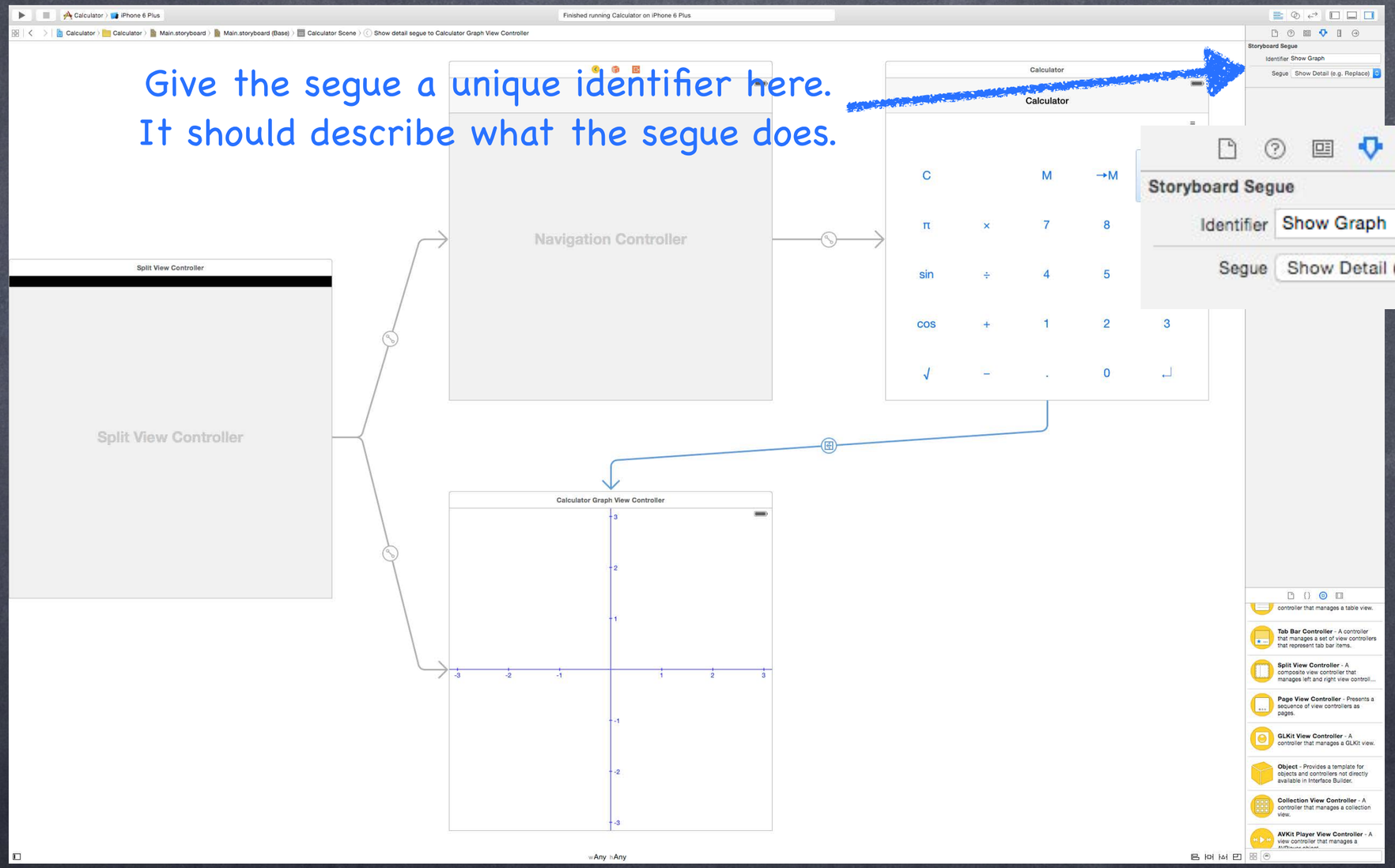
Segues


Navigation Controller
Navigation Controller
Calculator
Split View Controller
Split View Controller
Select the kind of segue you want.
Usually Show or Show Detail.
Calculator Graph View Controller
Action Segue
show
show detail
present modally
popover presentation
custom
Non-Adaptive Action Segue
push (deprecated)
modal (deprecated)

Segues


Now click on the segue
and open the Attributes Inspector
Split View Controller
Navigation Controller
Calculator
Calculator Graph View Controller
Storyboard Segue
Identifier
Segue
Show Detail (e.g. Replace)

Segues


Give the segue a unique identifier here.
It should describe what the segue does.
Split View Controller
Navigation Controller
Calculator
Calculator Graph View Controller
Storyboard Segue
Identifier Show Graph
Segue Show Detail (e.g. Replace)

Segues

- What's that identifier all about?
  - You would need it to invoke this segue from code using this `UIViewController` method
  - `func performSegueWithIdentifier(identifier: String, sender: AnyObject?)`
  - (but we almost never do this because we set usually ctrl-drag from the instigator)
  - The `sender` can be whatever you want (you'll see where it shows up in a moment)
  - You can ctrl-drag from the Controller itself to another Controller if you're segueing via code
  - (because in that case, you'll be specifying the `sender` above)
- More important use of the identifier: preparing for a segue
  - When a segue happens, the View Controller containing the instigator gets a chance to prepare the destination View Controller to be segued to
  - Usually this means setting up the segued-to MVC's Model and display characteristics
  - Remember that the MVC segued to is always a fresh instance (never a reused one)

Preparing for a Segue

- The method that is called in the instigator's Controller

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case “Show Graph”:
                if let vc = segue.destinationViewController as? GraphController {
                    vc.property1 = …
                    vc.callMethodToSetItUp(…)
                }
            default: break
        }
    }
}
```

Preparing for a Segue

- The method that is called in the instigator's Controller

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case “Show Graph”:
                if let vc = segue.destinationViewController as? GraphController {
                    vc.property1 = …
                    vc.callMethodToSetItUp(…)
                }
            default: break
        }
    }
}
```

The segue passed in contains important information about this segue:

1. the identifier from the storyboard
2. the Controller of the MVC you are segueing to (which was just created for you)

Preparing for a Segue

- The method that is called in the instigator's Controller

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case “Show Graph”:
                if let vc = segue.destinationViewController as? GraphController {
                    vc.property1 = …
                    vc.callMethodToSetItUp(…)
                }
            default: break
        }
    }
}
```

The sender is either the instigating object from a storyboard (e.g. a UIButton)
or the sender you provided (see last slide) if you invoked the segue manually in code

Preparing for a Segue

- The method that is called in the instigator's Controller

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case “Show Graph”:
                if let vc = segue.destinationViewController as? GraphController {
                    vc.property1 = …
                    vc.callMethodToSetItUp(…)
                }
            default: break
        }
    }
}
```

Here is the identifier from the storyboard (it can be `nil`, so be sure to check for that case)

Your Controller might support preparing for lots of different segues from different instigators so this identifier is how you'll know which one you're preparing for

Preparing for a Segue

- The method that is called in the instigator's Controller

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case “Show Graph”:
                if let vc = segue.destinationViewController as? GraphController {
                    vc.property1 = …
                    vc.callMethodToSetItUp(…)
                }
            default: break
        }
    }
}
```

For this example, we'll assume we entered "Show Graph" in the Attributes Inspector when we had the segue selected in the storyboard

Preparing for a Segue

- The method that is called in the instigator's Controller

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case “Show Graph”:
                if let vc = segue.destinationViewController as? GraphController {
                    vc.property1 = …
                    vc.callMethodToSetItUp(…)
                }
            default: break
        }
    }
}
```

Here we are looking at the Controller of the MVC we're segueing to
It is `AnyObject`, so we must cast it to the Controller we (should) know it to be

Preparing for a Segue

- The method that is called in the instigator's Controller

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case “Show Graph”:
                if let vc = segue.destinationViewController as? GraphController {
                    vc.property1 = …
                    vc.callMethodToSetItUp(…)
                }
            default: break
        }
    }
}
```

This is where the actual preparation of the segued-to MVC occurs
Hopefully the MVC has a clear public API that it wants you to use to prepare it
Once the MVC is prepared, it should run on its own power (only using delegation to talk back)

Preparing for a Segue

- The method that is called in the instigator's Controller

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case “Show Graph”:
                if let vc = segue.destinationViewController as? GraphController {
                    vc.property1 = …
                    vc.callMethodToSetItUp(…)
                }
            default: break
        }
    }
}
```

It is crucial to understand that this preparation is happening BEFORE outlets get set!

It is a very common bug to prepare an MVC thinking its outlets are set.

Preparing for a Segue

- The method that is called in the instigator's Controller

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case “Show Graph”:
                if let vc = segue.destinationViewController as? GraphController {
                    vc.property1 = …
                    vc.callMethodToSetItUp(…)
                }
            default: break
        }
    }
}
```

Preventing Segues

- You can prevent a segue from happening too
  - Just implement this in your `UIViewController` …

```
func shouldPerformSegueWithIdentifier(identifier: String?, sender: AnyObject?) -> Bool
```

  - The `identifier` is the one in the storyboard.
  - The `sender` is the instigating object (e.g. the button that is causing the segue).

Demo

- Emotions in FaceIt
  - This is all best understood via demonstration
  - We will create a new Emotions MVC
  - The Emotions will be displayed segueing to the Face MVC
  - We'll put the MVCs into navigation controllers inside split view controllers
  - That way, it will work on both iPad and iPhone devices

View Controller Lifecycle

- View Controllers have a "Lifecycle"
  - A sequence of messages is sent to a View Controller as it progresses through its "lifetime".
- Why does this matter?
  - You very commonly override these methods to do certain work.
- The start of the lifecycle ...
  - Creation.
  - MVCs are most often instantiated out of a storyboard (as you've seen).
  - There are ways to do it in code (rare) as well which we may cover later in the quarter.
- What then?
  - Preparation if being segued to.
  - Outlet setting.
  - Appearing and disappearing.
  - Geometry changes.
  - Low-memory situations.

View Controller Lifecycle

- After instantiation and outlet-setting, `viewDidLoad` is called

  This is an exceptionally good place to put a lot of setup code.
  It's better than an `init` because your outlets are all set up by the time this is called.

```
override func viewDidLoad() {
    super.viewDidLoad() // always let super have a chance in lifecycle methods
    // do some setup of my MVC
}
```

  One thing you may well want to do here is update your UI from your Model.
  Because now you know all of your outlets are set.

  But be careful because the geometry of your view (its `bounds`) is not set yet!
  At this point, you can't be sure you're on an iPhone 5-sized screen or an iPad or ???.
  So do not initialize things that are geometry-dependent here.

View Controller Lifecycle

- Just before your `view` appears on screen, you get notified

```
func viewWillAppear(animated: Bool) // animated is whether you are appearing over time
```

Your view will only get "loaded" once, but it might appear and disappear a lot.
So don't put something in this method that really wants to be in `viewDidLoad`.
Otherwise, you might be doing something over and over unnecessarily.

Do something here if things your display is changing while your MVC is off-screen.

You could use this to optimize performance by waiting until this method is called
(as opposed to `viewDidLoad`) to kick off an expensive operation (probably in another thread).

Your view's geometry is set here, but there are other places to react to geometry.

- There is a "did" version of this as well

```
func viewDidAppear(animated: Bool)
```

View Controller Lifecycle

- And you get notified when you will disappear off screen too

  This is where you put "remember what's going on" and cleanup code.

```
override func viewWillDisappear(animated: Bool) {
    super.viewWillDisappear(animated)  // call super in all the viewWill/Did... methods
    // do some clean up now that we've been removed from the screen
    // but be careful not to do anything time-consuming here, or app will be sluggish
    // maybe even kick off a thread to do stuff here (again, we'll cover threads later)
}
```

- There is a "did" version of this too

```
func viewDidDisappear(animated: Bool)
```

View Controller Lifecycle

- Geometry changed?

  Most of the time this will be automatically handled with Autolayout.

  But you can get involved in geometry changes directly with these methods …

  ```
  func viewWillLayoutSubviews()
  func viewDidLayoutSubviews()
  ```

  They are called any time a view's `frame` changed and its `subviews` were thus re-layed out.

  For example, autorotation (more on this in a moment).

  You can reset the `frames` of your `subviews` here or set other geometry-related properties.

  Between "`will`" and "`did`", autolayout will happen.

  These methods might be called more often than you'd imagine

  (e.g. for pre- and post- animation arrangement, etc.).

  So don't do anything in here that can't properly (and efficiently) be done repeatedly.

View Controller Lifecycle

- Autorotation

  Usually, the UI changes shape when the user rotates the device between portrait/landscape
  You can control which orientations your app supports in the Settings of your project

  Almost always, your UI just responds naturally to rotation with autolayout

  But if you, for example, want to participate in the rotation animation, you can use this method ...

```
func viewWillTransitionToSize(
    size: CGSize,
    withTransitionCoordinator: UIViewControllerTransitionCoordinator
)
```

  The `coordinator` provides a method to animate alongside the rotation animation
  We are not going to be talking about animation, though, for a couple of weeks
  So this is just something to put in the back of your mind (i.e. that it exists) for now

View Controller Lifecycle

- In low-memory situations, `didReceiveMemoryWarning` gets called …
  - This rarely happens, but well-designed code with big-ticket memory uses might anticipate it.
  - Examples: images and sounds.
  - Anything "big" that is not currently in use and can be recreated relatively easily
    - should probably be released (by setting any pointers to it to `nil`)

View Controller Lifecycle

- `awakeFromNib`

  This method is sent to all objects that come out of a storyboard (including your Controller).
  Happens before outlets are set! (i.e. before the MVC is "loaded")
  Put code somewhere else if at all possible (e.g. `viewDidLoad` or `viewWillAppear`).

View Controller Lifecycle

- Summary
  - Instantiated (from storyboard usually)
  - `awakeFromNib`
  - segue preparation happens
  - outlets get set
  - `viewDidLoad`
  - These pairs will be called each time your Controller's `view` goes on/off screen …
    - `viewWillAppear` and `viewDidAppear`
    - `viewWillDisappear` and `viewDidDisappear`
  - These "geometry changed" methods might be called at any time after `viewDidLoad` …
    - `viewWillLayoutSubviews` (… then autolayout happens, then …) `viewDidLayoutSubviews`
  - If memory gets low, you might get …
    - `didReceiveMemoryWarning`

View Controller Lifecycle

- Demo

  Let's plop some `print` statements into the View Controller Lifecycle methods in FaceIt

  Then we can watch as Face and Emotions MVCs go through their lifecycle