iOS Foundations II
Day 4

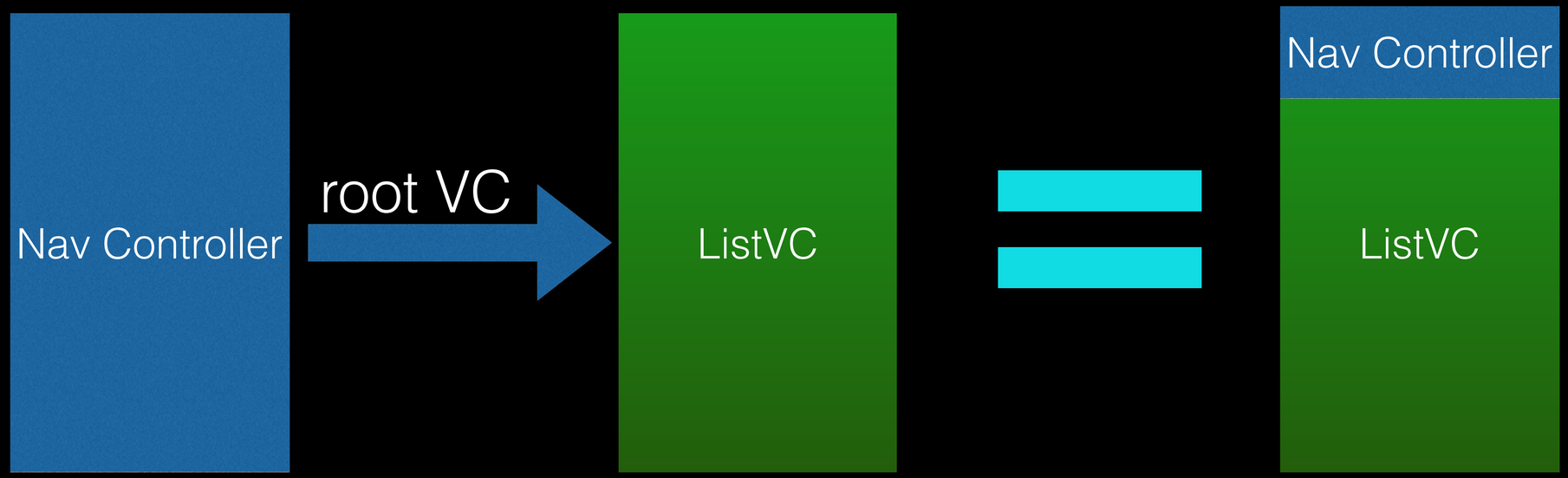
- Table View Review
- Navigation Controller
- Segues
- ‘Passing’ objects through segues
- Maybe Autolayout if we have time

NavigationControllers

- “A navigation controller manages a stack of view controllers to provide a drill-down interface for hierarchical content.”

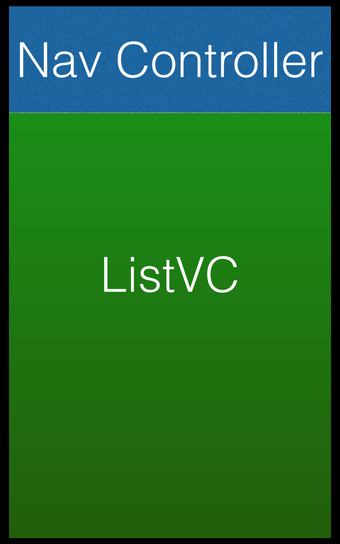
NavigationControllers

- “The navigation controller’s primary responsibility is to respond to user actions by pushing new content view controllers onto the stack or popping content view controllers off of the stack”
- The first ViewController you push onto the stack becomes the rootViewController and is never popped off because then no view would be on screen, which would look really bad.
- Nav Controllers have a property to the topViewController and an array property for all its viewControllers currently on the stack.



A Navigation controller is always instantiated with a root view controller. This will be the first view controller you see. A view controller contained within a navigation controller has a navigation bar at the top

Sharing the screen



We have seen that the navigation controller can manage multiple view controllers. But you will only ever see the ‘top’ view controller. The top view controller is the latest view controller to pushed onto the navigation controller’s ‘stack’. It basically acts like a stack of cards, you only see the top card.

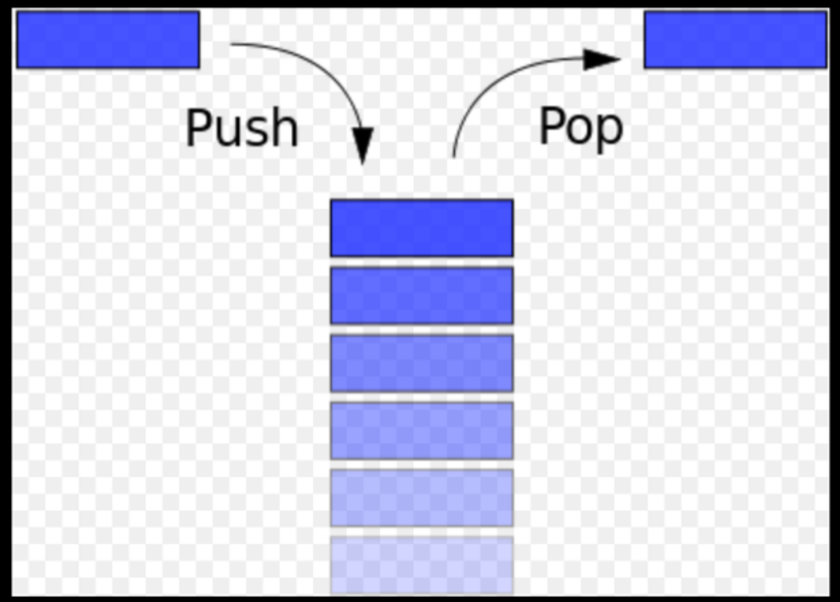
NavigationControllers

- 2 ways to get navigation controllers into your app:
  - Via storyboard, selected the view controller you want to set as the root view controller and Go up to the top menu, Editor> Embed In> Navigation Controller
  - In Code, you can instantiate an instance of UINavigationController and give it a root view controller

Demo

Stack Data Structure

- “A stack is particular kind of abstract data type or collection in which the only operations on the collection is adding (push) or removal (pop).” - Wikipedia
- LIFO : Last-In-First-Out. The last item added is the first to be removed.



NavigationControllers

- Storyboards make navigation controllers extremely easy to install into your app. Here's all the methods you need to do it in code without the storyboard:
- `init(rootViewController:)` UINavigationController is initialized with a rootViewController.
- `pushViewController(animated:)` To add or 'push' a view controller onto the stack.
- `popViewController(animated:)` To remove or 'pop' a view controller from the stack.

Demo

Segues

- Segues are **provided by the storyboard** to help you easily transition from one view controller to another.
- They are pronounced “SAYG-WAY”, not “SEEG”
- There are 2 primary segues that are used : Show and Present. (There are others that we will talk about eventually)
- Show refers to pushing a view controller onto the navigation stack, it will slide in from the right to the left. **A Show only works if the view controller triggering the segue is inside of a navigation controller.**
- Present refers to modally presenting a view controller, it usually slides up from the bottom. Modal presentations are usually ‘one-off’ screens, like settings or quick uploads.
- You can also create your own custom segue to customize the behavior to match your needs.

Creating Segues

- Segues are always created via storyboard.
- You cannot create a segue via code, but you can trigger a segue via code (more on this in a bit).
- **A segue in code is just called a transition.**
- If you do an app without storyboards, there will be no segues.
- Creating segues is just like creating outlets and actions, lots of dragging and hoping you dragged to the right thing

Segues always create new view controllers for the destination

Demo

Segues in Code

- 2 primary methods for dealing with segues in code:
    1. performSegueWithIdentifier: triggers a segue manually in code
    2. prepareForSegue:sender: allows you to run some code before the segue actually fires. You don’t call this method yourself, you just implement it in your view controller, it is called by the system at the appropriate time.

Triggering a Segue

- There are 2 ways a segue is triggered:
    1. The user presses an interface object that is hooked up to a segue
    2. Code is run that triggers the segue ‘manually’
- To fire the segue manually, you call the method performSegueWithIdentifier

A Segue and its Identifier

- The Identifier of the Segue that just saw me fill out, has 2 important roles:

    1. Allows us to trigger the segue completely in code, as long as we know the identifier (rare)
    2. Allows us to prepare for the segue to happen (super important, we will cover this in depth today)

prepareForSegue:sender:

- This method is called on the source view controller of the segue, right before the segue is actually performed.
- The first paramater is the segue itself, which is an instance of the UISegue class.
- The most important property of a UISegue instance is the destinationViewController property, which gives you a reference to the view controller you are about to segue to.
- This is a great spot to pass information to the next screen!

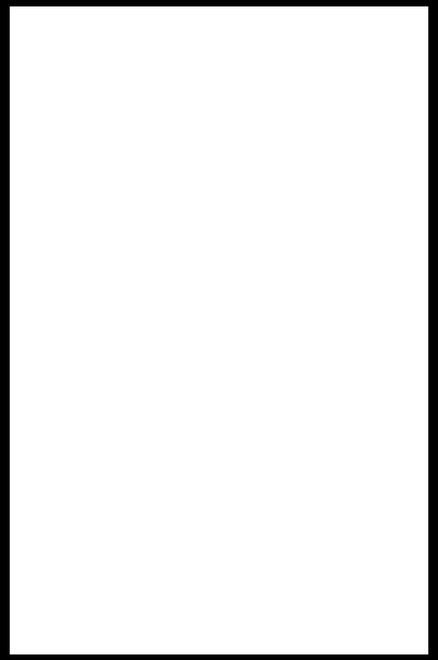
Demo

Passing data to View Controllers

- There are many patterns we can use to pass data around our app:
  - Delegation
  - Notification Center
  - Singletons
  - Persistence
- But for now, we can do something as simple as passing a reference directly to the new view controller.

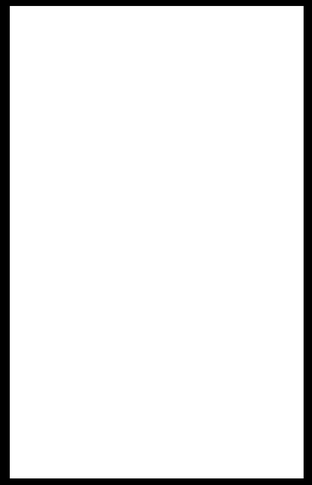
Passing data to View Controllers

ListViewController

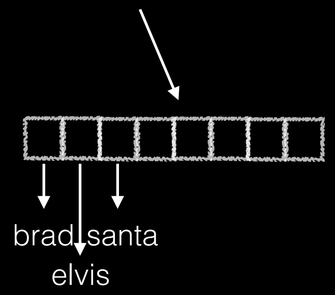


ListViewController creates an array of Person objects and sets it people property ’s value to that array

ListViewController

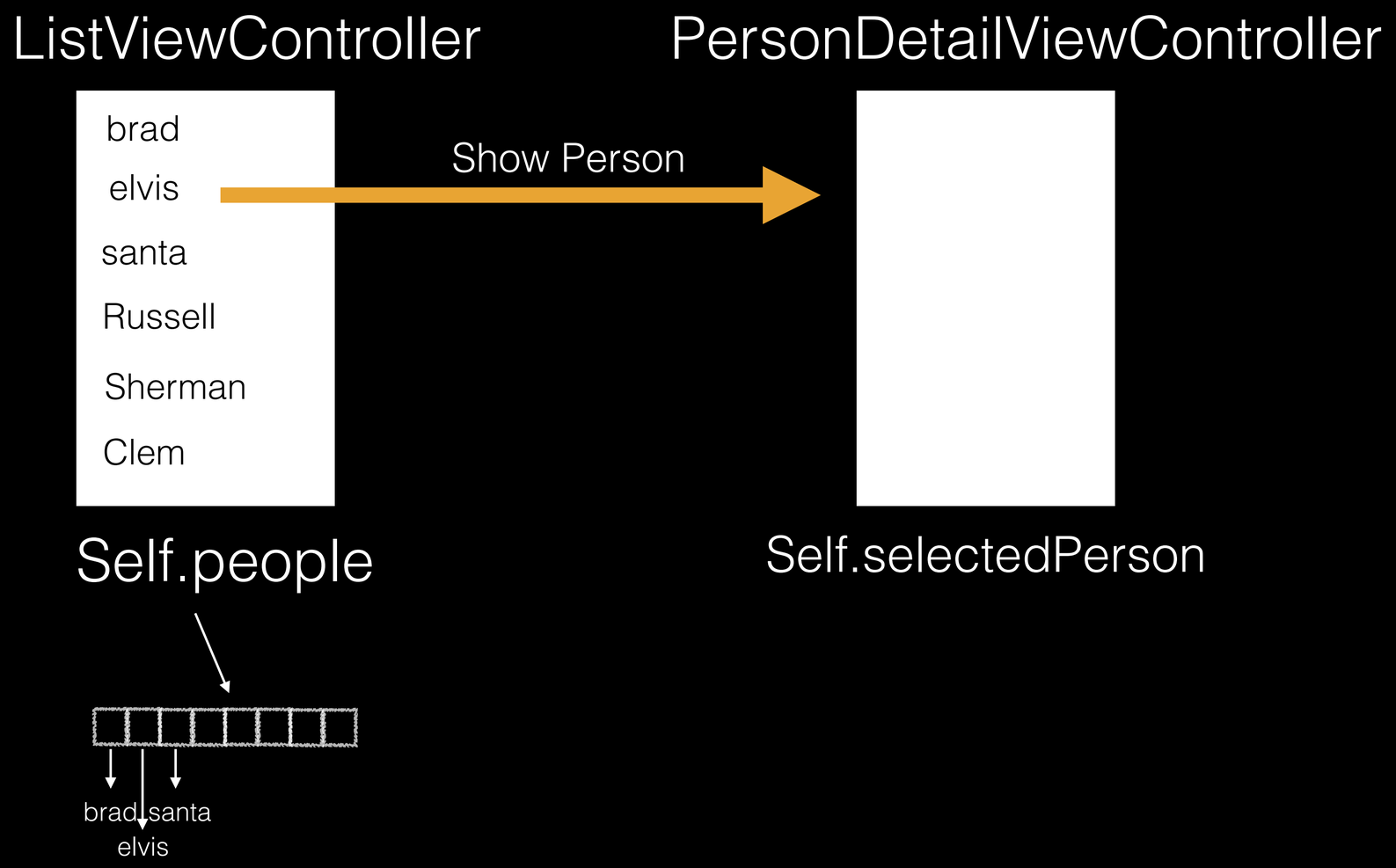


Self.people



ListViewController, as the datasource of the tableview, uses the self.people array as the backing array for the tableview

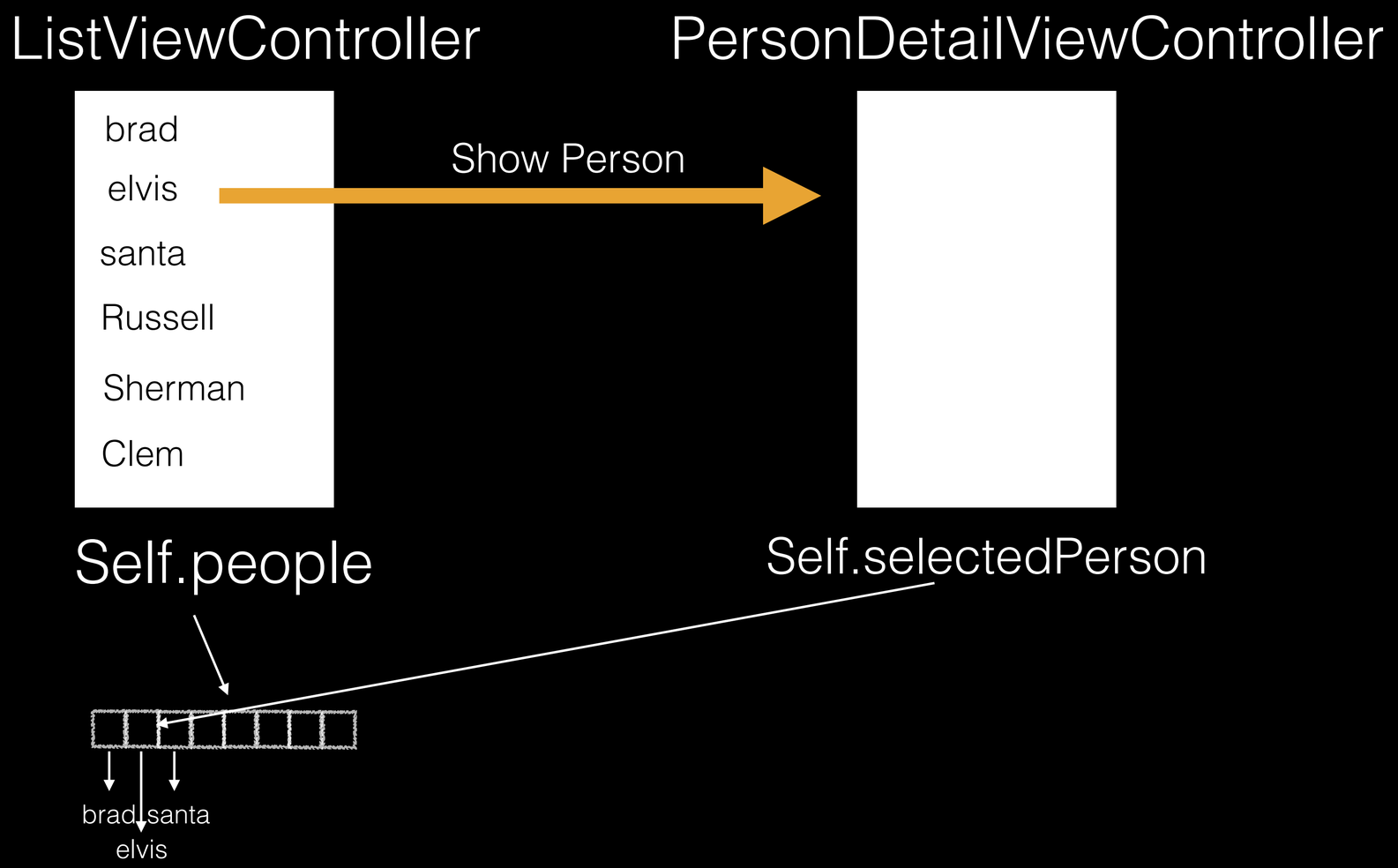
The User clicks on a cell, triggering our ShowPerson Segue. The PersonDetailViewController is initialized.



In Prepare for Segue, the ListViewController intercepts the segue and grabs a reference to the destination view controller, which is the PersonDetailViewController.

```
override func prepareForSegue(segue: UIStoryboardSegue!, sender: AnyObject!) {

    if segue.identifier == "ShowPerson" {

        var personDetailViewController = segue.destinationViewController as
  PersonDetailViewController

        //now we can prepare this view controller to be displayed with what
  ever data we need to pass to it.
    }
}
```

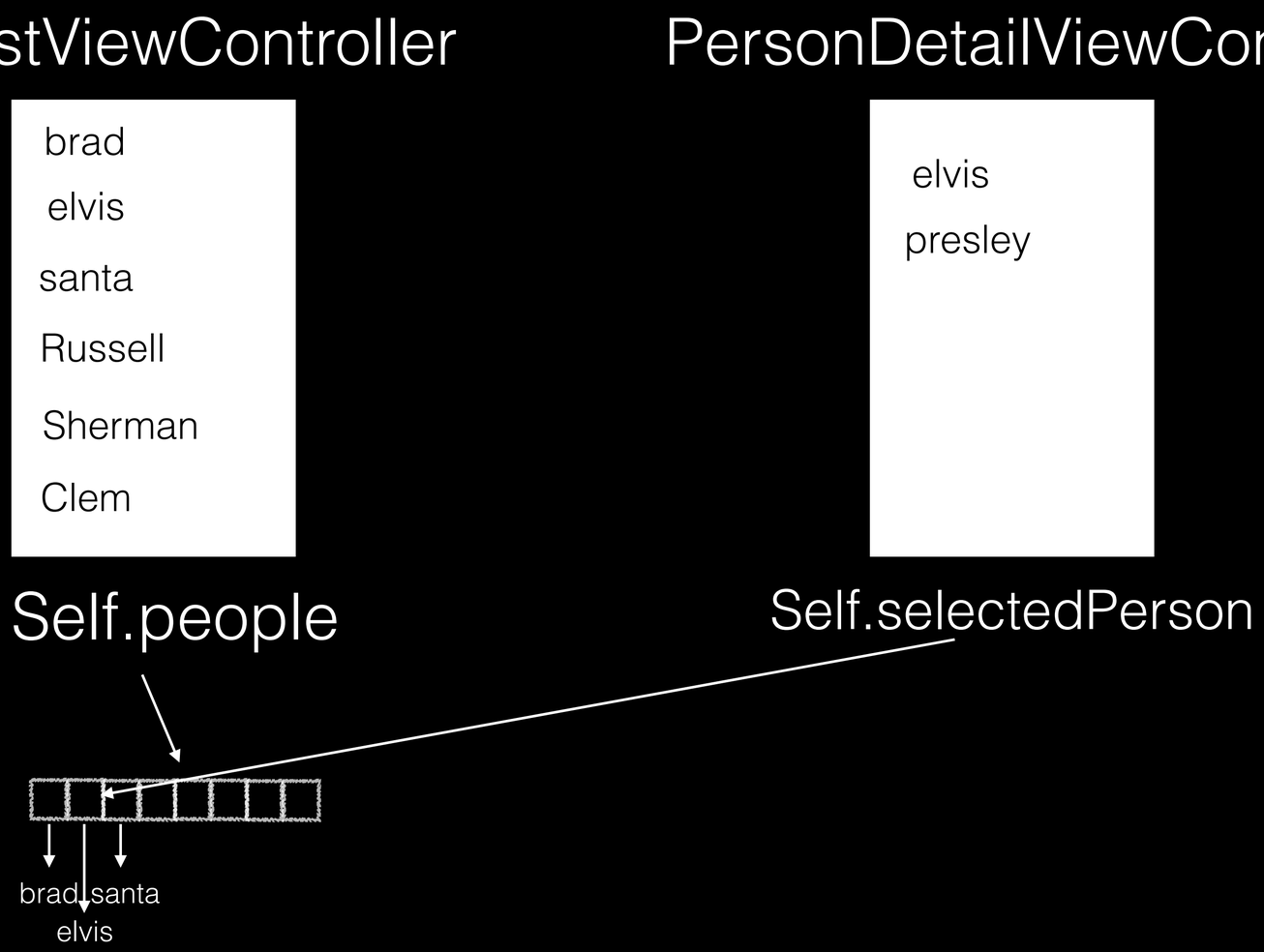
ListViewController sets PersonDetailViewController's selectedPerson property to reference the person that was clicked from the list.



ListViewController sets PersonDetailViewController's selectedPerson property to reference the person that was clicked from the list.

```
    if segue.identifier == "ShowPerson" {

        var personDetailViewController = segue.destinationViewController as
PersonDetailViewController

        //now we can prepare this view controller to be displayed with what
ever data we need to pass to it.

        //grab the selected index path from our tableview
        var selectedIndexPath = self.tableView.indexPathForSelectedRow()

        //grab the selected person using the indexPath as the index in our
people array
        var selectedPerson = self.people[selectedIndexPath.row]

        //set destinationViewController's person property to reference the
selectedPerson
        personDetailViewController.person = selectedPerson


    }
```

Now PersonDetailViewController can use it's selectedPerson property to fill out his interface with that Person's name properties. Any changes he makes to this Person object apply to the person object inside the original array!



Demo

Autolayout

Autolayout

- A constraint-based layout system for making user interfaces.
- AutoLayout works by you bossing it around.
- Specifically, you can tell it 2 things about every view in your interface:
    1. Size - How big the view is going to be
    2. Location - Where the object is going to be located in its super view
- Once told ‘the rules’, autolayout will enforce the rules you setup.
- These rules are setup using constraints.

Constraints

- Constraints are the fundamental building block of autolayout.
- Constraints contain rules for the layout of your interface's elements.
- You could give a 50 point height constraint to an imageView, which constrains that view to always have a 50 point height. Or you give it a constraint to always be 20 points from the bottom of its superview.
- Constraints can work together, but sometimes they conflict with other constraints.
- At runtime Autolayout considers all constraints, and then calculates the positions and sizes that bests satisfies all the constraints.

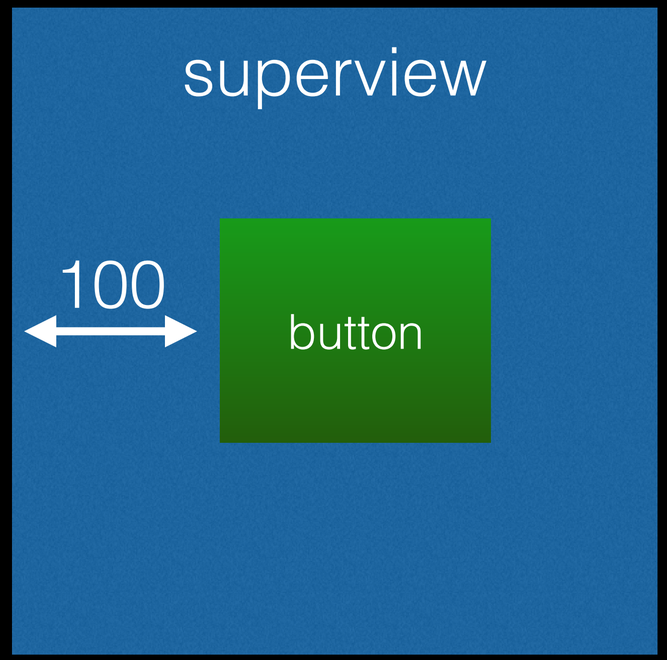
Demo

Attributes

- When you attach constraints to a view, you attach them using attributes.
- The attributes are: **left/leading, right/trailing, top, bottom, width, height, centerX, centerY.**
- Attributes tell the constraint what part of the view(s) they should be manipulating

Attributes

- So if you attach a constraint of 100 points from a button's left attribute to its container's left attribute, thats saying "I want the left side of this button to be 100 points over from its super view's left side"



Demo

Constraints + Attributes = Math time

- “You can think of a constraint as a mathematical representation of a human-expressable statement”
- So if you say “the left edge should be 20 points from the left edge of its containing view”
- This translates to button.left = container.left x 1.0 + 20
- which is in the form of y = mx+b
- **first attribute = second attribute * multiplier + constant**
- In the case of an absolute value, like pinning height, width, centerX, or centerY, the second attribute is nil.
- You can change the constants in code as an easy way to programmatically adjust your interface.

Demo

Storyboard and Autolayout

- Storyboard makes setting up autolayout pretty intuitive and painless, and even though you can setup autolayout completely in code, Apple strongly recommends doing it in storyboard.
- Xcode will let you build your app even if you have constraints that are conflicting and incorrect, but Apple says you should never ship an app like that.
- **When you drag a object onto your interface, it starts out with no constraints.**
- **Once you apply one constraint, that view needs to have constraints dictating its size AND location**

Demo