UINavigationControllers In Storyboard

Mobile Computing - iOS

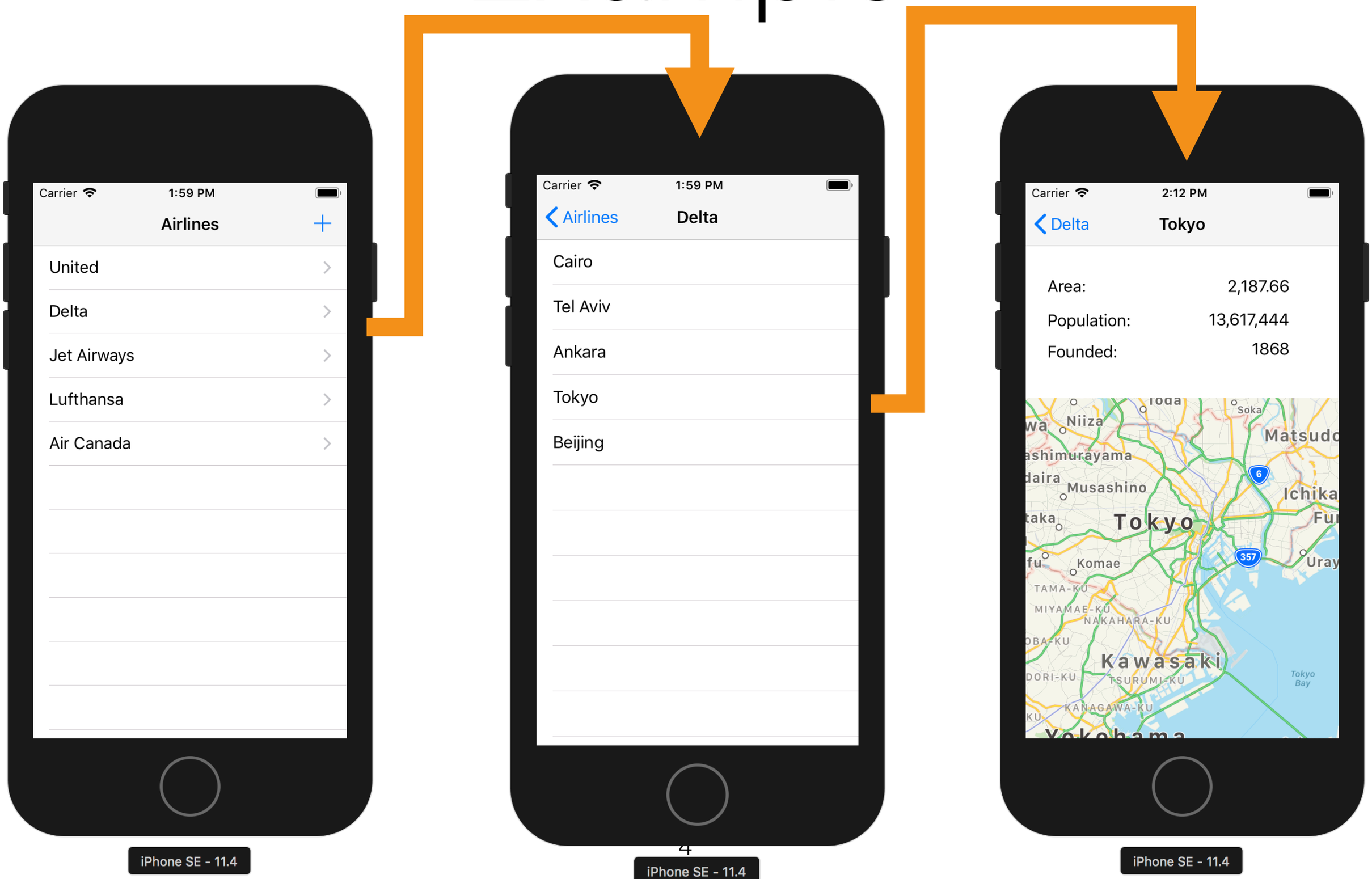
Objectives

- Students will be able to:
  - explain the purpose of navigation controllers
  - create a nav-based app in Storyboard
  - explain the advantages of Storyboard over code

Overview

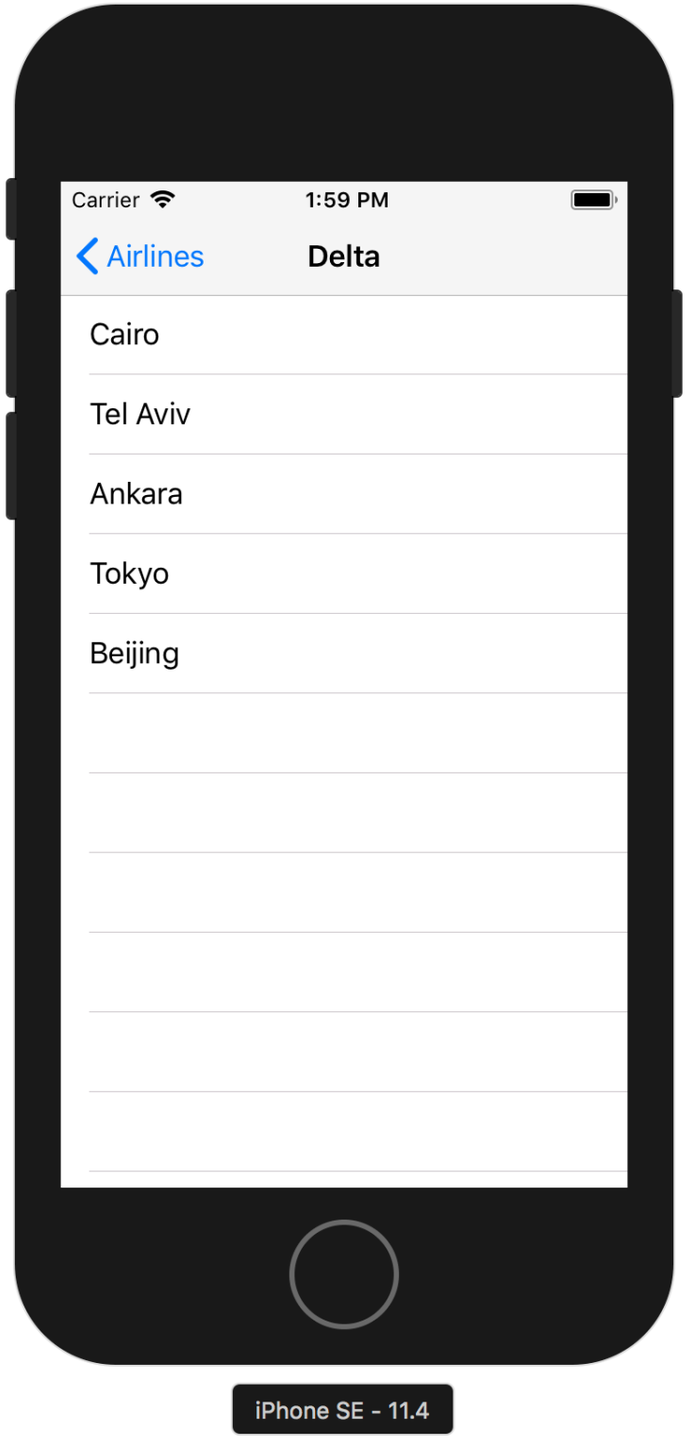
- A **navigation controller** is a container view controller (i.e., it contains other view controllers) used primarily to provide an easy way to explore hierarchically related data.
- Navigation controllers and table views are like peanut butter and jelly — perfect together!

Example


Carrier
1:59 PM
Airlines
United
Delta
Jet Airways
Lufthansa
Air Canada
iPhone SE - 11.4
Carrier
1:59 PM
Airlines
Delta
Cairo
Tel Aviv
Ankara
Tokyo
Beijing
iPhone SE - 11.4
Carrier
2:12 PM
Delta
Tokyo
Area:
2,187.66
Population:
13,617,444
Founded:
1868
Niiza
Soka
Matsudo
Musashino
Tokyo
Ichika
Komae
TAMA-KU
MIYAMAE-KU
NAKAHARA-KU
Kawasaki
TSURUMI-KU
KANAGAWA-KU
Tokyo Bay
iPhone SE - 11.4

Overview

- The **root view controller** is the view controller that first appears when a navigation controller is presented
- Each level in the data hierarchy is represented by its own view controller, stored in a **stack**
- Initially, the stack contains just one view controller - the root view controller
- Each time you navigate one step deeper into the data, another view controller is pushed onto the stack. Pressing the back button has the effect of popping the stack.



Example

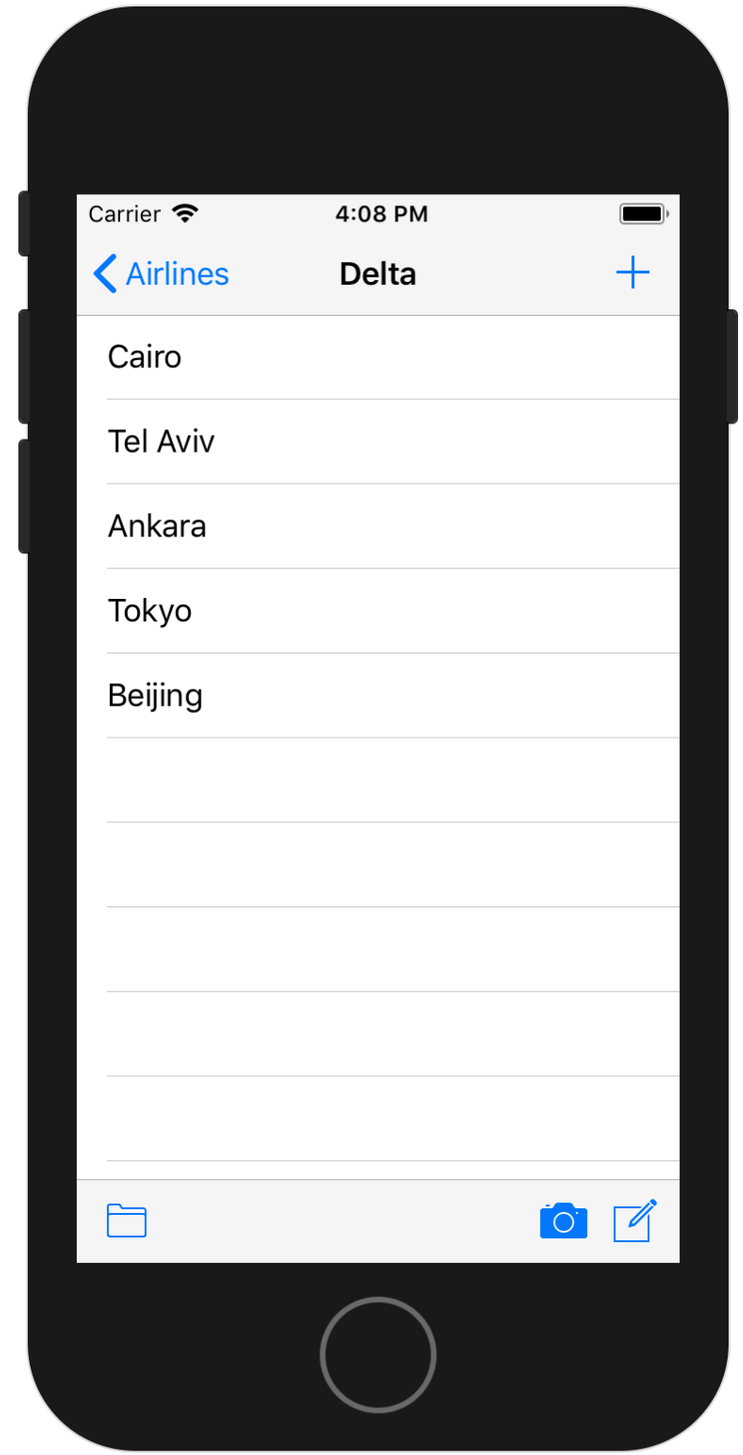
What the user sees

State of the stack

Navigation Bar

- A navigation controller's view consists of 3 parts — a **navigation bar** (always visible), **content** (from the view controller on the top of the stack), and, optionally, a **tool bar** at the bottom of the screen

- The navigation bar tells the user where they are in the hierarchy. The information comes from the top view controller on the stack and

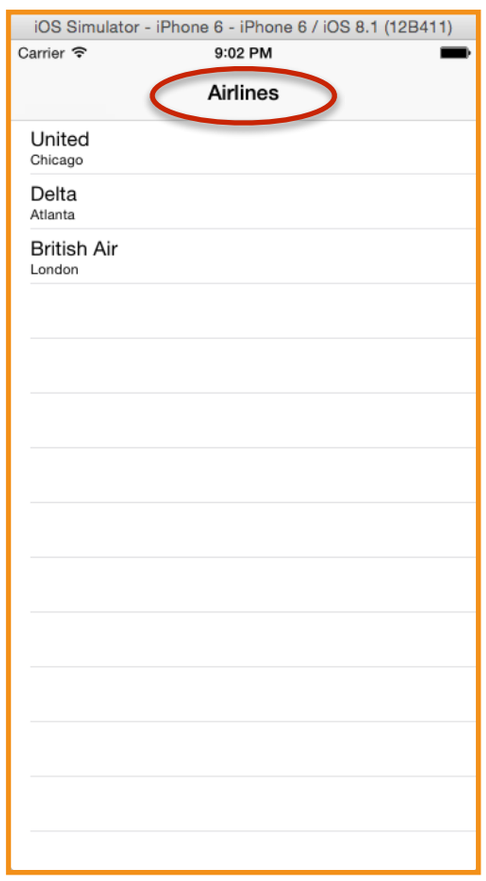


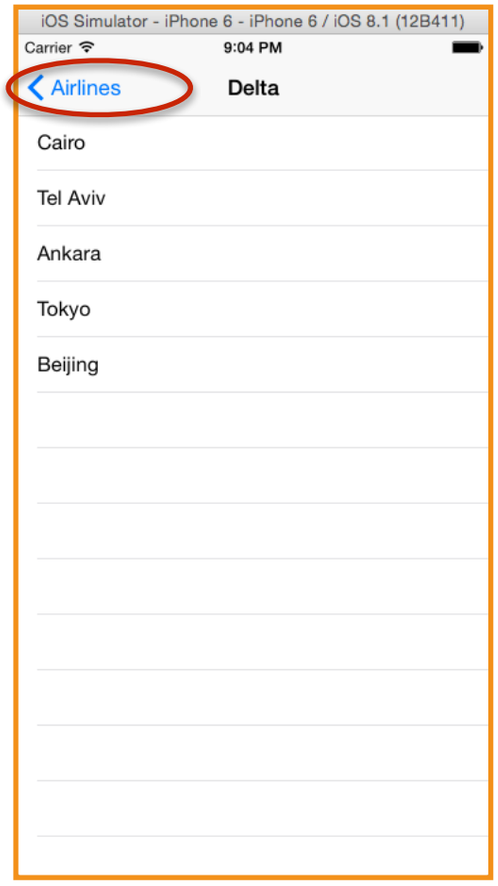
Navigation Bar Details

- Each view controller comes with a <u>navigationItem</u> property, consisting of a **title**, **leftBarButtonItem** and **rightBarButtonItem.** It is used when a view controller is contained in a navigation controller, to populate the navigation bar

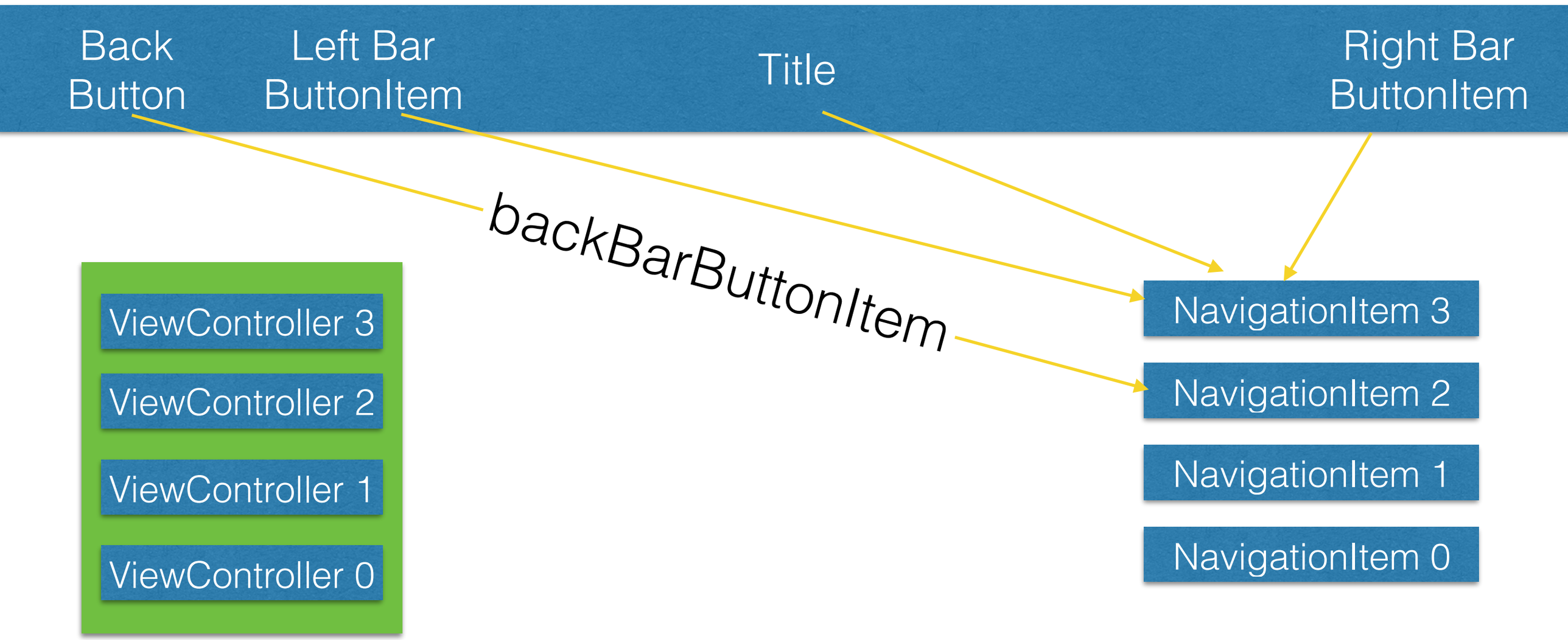
Getting Back ...

- When a UIViewController is pushed onto the navigation controller's stack, the navigation bar will display, in the left hand side, a button (the back button) that will, by default, automatically pop the stack
- Hence t*here is no need to write any code* to navigate back to the previous view controller
- By default, the name of the button is the navigationItem.title of the View Controller beneath the current view controller
- If that name is too long, the word "Back" is substituted.





The NavigationBar, Revealed!



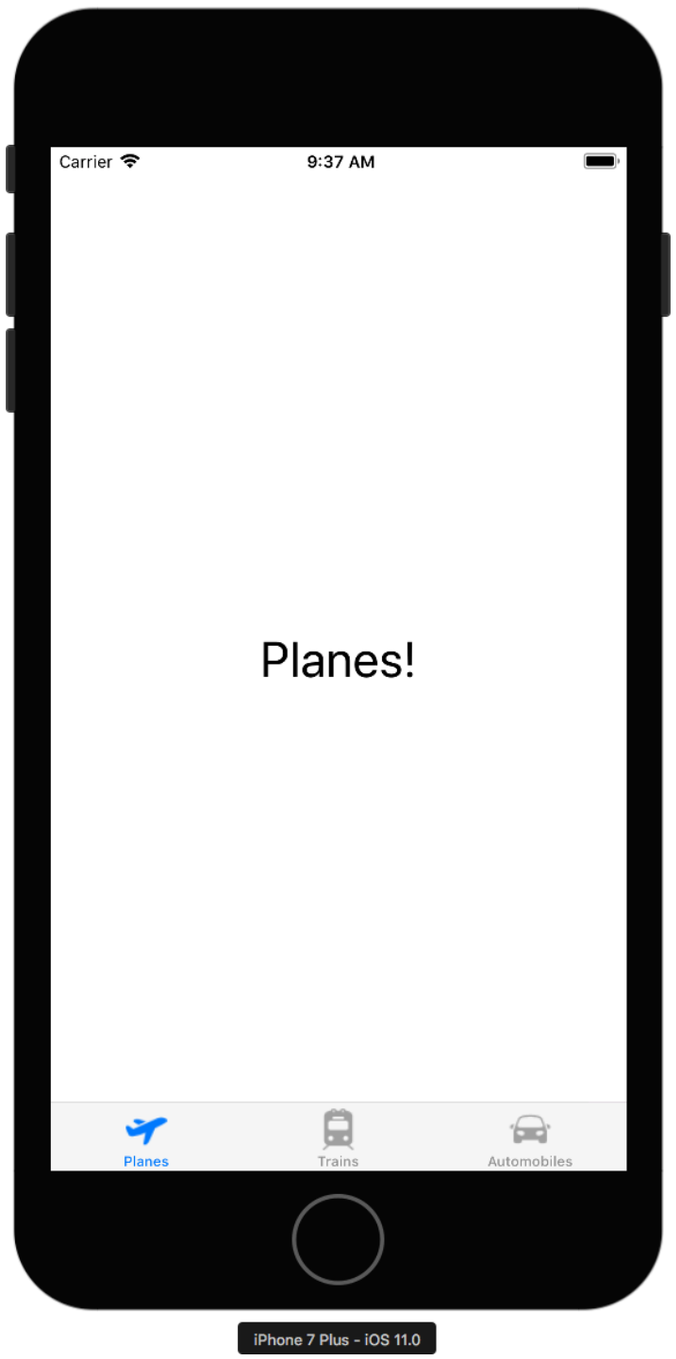
The UINavigationController's stack of UIViewControllers

1. If a navigation item has a leftBarButtonItem, it appears in the navigation bar on the left side.
2. If the leftBBI is nil, the backBarButtonItem of the item *beneath* the top one appears there.
3. If the backBBI is also nil, the items' title appears (the default case).
4. If the title is too long to show, the word "Back" appears instead. Whew!

Memory Aide

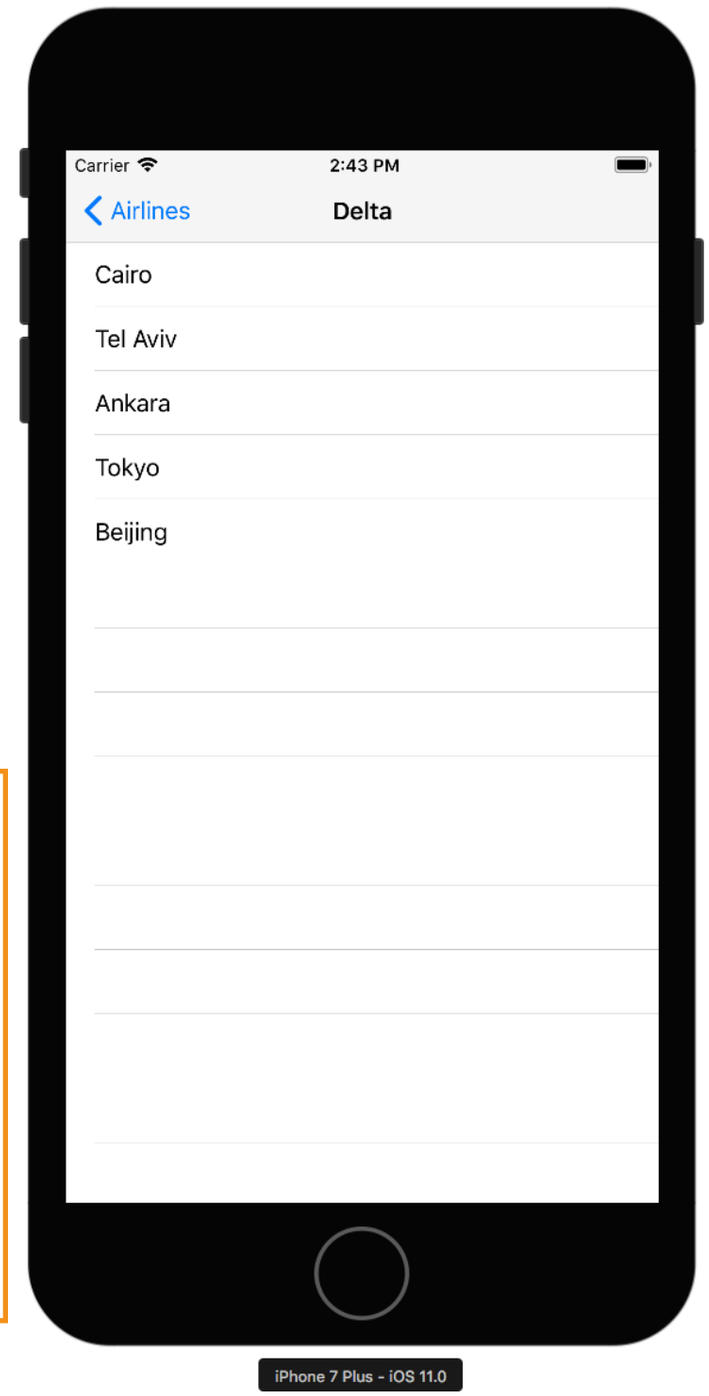
- leftBarButtonItem & backBarButtonItem sound suspiciously similar, so remember ...
- A view controller atop the stack sets its own rightBarButtonItem. Thus it makes sense (and is true) that it also sets its leftBarButtonItem -- backBarButtonItem is supplied by the view controller *beneath* the one at the top of the stack.

TabBarControllers v. NavigationControllers



- A UIViewController in a UITabBarController is displayed with a **UITabBar** beneath it
- A UITabBarController gets its information - title, image, badge - from a UIViewController's **tabBarItem**

- A UIViewController in a UINavigationController is displayed with a **UINavigationBar** above it
- A UINavigationController get its information - title, back bar item - from a UIViewController's **navigationItem**



ICE: Creating a Navigation-Based App

- Storyboard makes creating a navigation based app easy. Since it works well with table views, we will describe the process using a modified version of our Best Restaurants app (download it here).

- Our current Restaurants model was really just an array of Strings: to illustrate navigation, we need something to show on our details screen.

- Consequently, we have modified the model slightly: it is already in the app you just downloaded (although not being used, yet)

A Restaurant

```
struct Restaurant : CustomStringConvertible, Equatable {
    var name:String
    var genre:Genre
    var rating:Int {
        willSet {
            if newValue < 0 || newValue > maxNumStars {
                print("DEBUG: Rating is supposed to
                      always be between 0-\(maxNumStars), inclusive")
            }
        }
    }

   let maxNumStars:Int = 5

    // Equatable protocol, lets us compare 2 restaurants using ==
    static func == (lhs: Restaurant, rhs: Restaurant) -> Bool {
        return lhs.name == rhs.name && lhs.genre == rhs.genre &&
               lhs.rating == rhs.rating
    }

    // Printing a Restaurant instance, it will show whatever this returns
    var description: String {
        return "\(name) -- \(genre)"
    }
}

 enum Genre {case chinese, greek, italian, mexican, pizza, popular,
 sandwiches, vegetarian }
```

Restaurants

```
struct Restaurants {

    static var shared = Restaurants()

    private var restaurants:[Restaurant] = [
        Restaurant(name: "Planet Sub",  genre: .sandwiches, rating: 5),
        Restaurant(name: "A&G's", genre: .greek, rating: 3),
        Restaurant(name: "Jimmy Johns", genre: .sandwiches, rating: 3),
        ]

    private init(){}

    func numRestaurants()->Int {
        return restaurants.count
    }

    func restaurant(_ index:Int) -> Restaurant {
        return restaurants[index]
    }

    // this lets us use [ ] notation instead of the above
    subscript(index:Int) -> Restaurant {
        return restaurants[index]
    }

    mutating func add(restaurant:Restaurant){
        restaurants.append(restaurant)
    }

    mutating func delete(restaurant:Restaurant){
        for i in 0 ..< restaurants.count {
            if restaurants[i] == restaurant {
                restaurants.remove(at:i)
                break
            }
    }
}
```

ICE: Creating a Navigation-Based App in 4 Easy Steps (and 2 Hard Ones)

1. Modify the RestaurantsTableViewController class so that it uses the Restaurants model

2. Embed RestaurantsTableViewController in a Navigation Controller

3. Create a ViewController subclass, **RestaurantDetailsViewController**, with outlets for 2 UILabels — genreLBL and ratingLBL — and add an analogous file in Storyboard (drag in a ViewController, change its identity to RestaurantDetailsViewController, then make and connect the labels)

4. Control-drag from the prototype cell in RestaurantsVC to the RestaurantDetailsVC: when prompted choose Selection Segue >> Show

   - Ta-dah! You have created a **segue**, part of a navigation-based app 😍, albeit a trivial one. Run the app and tap on a cell to verify that it works

5. Define a stored property, **restaurant**, in the RestaurantDetailsViewController, and implement viewWillAppear(_:animated:) to display genre and rating in the labels

6. In RestaurantsViewController, implement **prepare(for segue:sender:)** to initialize restaurant

   - Bravo! You now have a working nav-based app, pat yourself on the back!

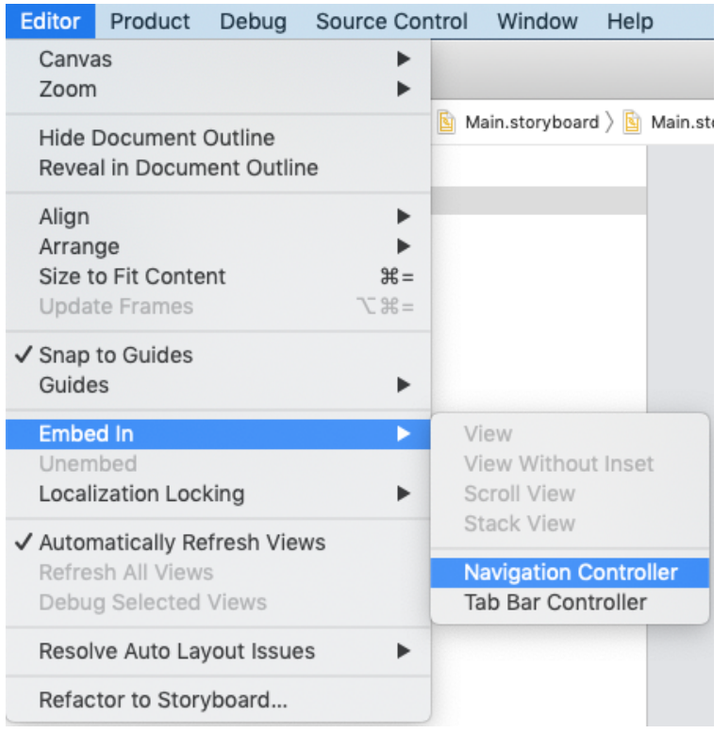
1. Modify RestaurantsTableViewController to work with the Restaurants model

1. Delete the restaurants property

2. Change the stub files as follows:

```
 override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int)
-> Int {
      return Restaurants.shared.numRestauraants()
}

 override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
                                                                  -> UITableViewCell {
  let cell = tableView.dequeueReusableCell(withIdentifier: "restaurantCell")
   cell?.textLabel?.text = Restaurants.shared[indexPath.row].name
   cell?.detailTextLabel?.text = Restaurants.shared[indexPath.row].lastName
   return cell!
 }
```

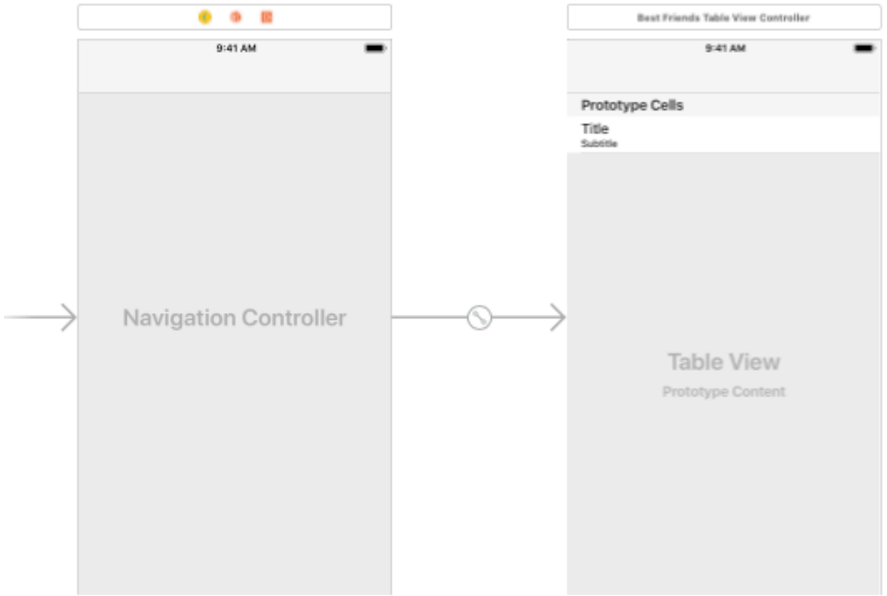
2. Embed RestaurantsViewController in a Navigation Controller



1. In storyboard, select RestaurantsTableViewController

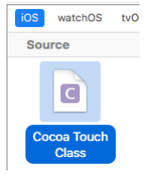
2. Choose Editor >> Embed in >> Navigation Controller

   - Now the initial controller for the project is a navigation controller, its root view controller is RestaurantsTVC



3. Create a ViewController subclass, **RestaurantDetailsViewController,** and add an analogous file in Storyboard

1. Create a ViewController subclass



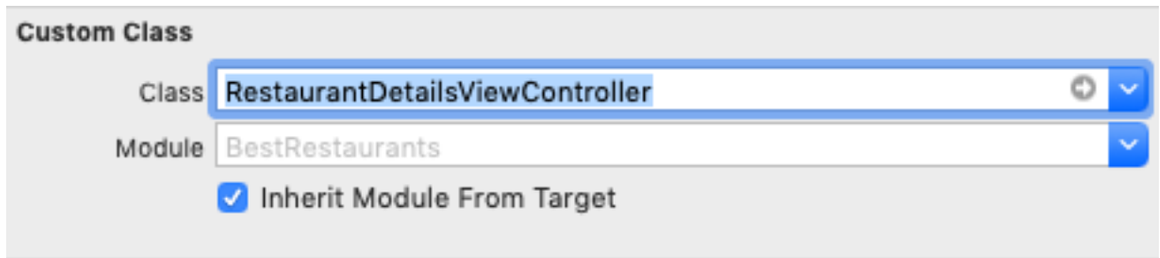
2. Define outlets for 2 UILabels — genreLBL and ratingLBL

```
@IBOutlet weak var genreLBL:UILabel!
@IBOutlet weak var ratingLBL:UILabel!
```

3. Add an analogous file in SB

   1. Drag in a ViewController

   2. Change its identity to RDVC



   3. Drag in labels

   4. Connect the labels to their respective outlets


```
11 class RestaurantDetailsViewController: UIViewController {
12
    @IBOutlet weak var genreLBL:UILabel!
    @IBOutlet weak var ratingLBL:UILabel!
```

4. Control-drag from the prototype cell in RestaurantsTVC to the RestaurantDetailsVC

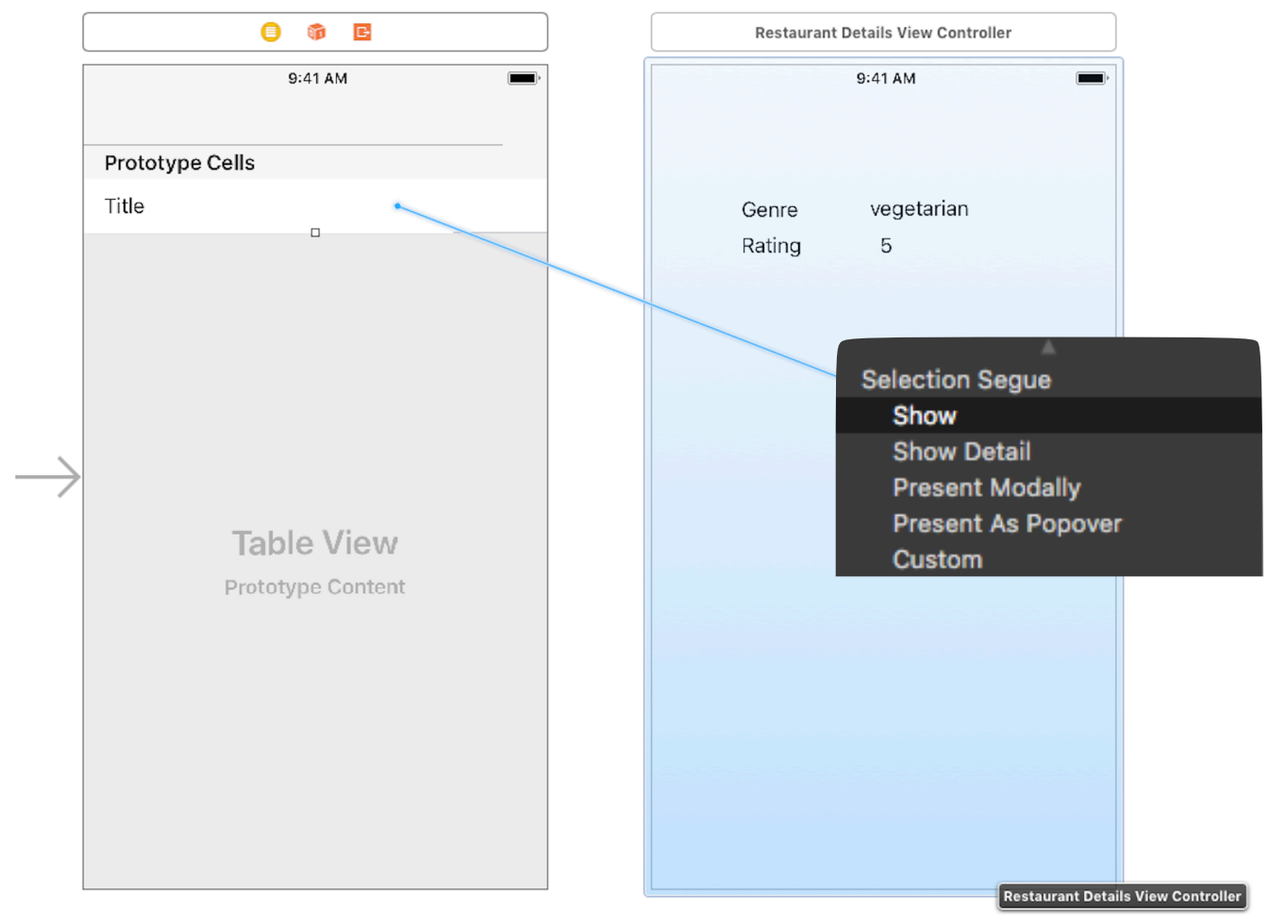
1. Control-drag from the prototype cell in RestaurantsTVC to the RestaurantDetailsVC

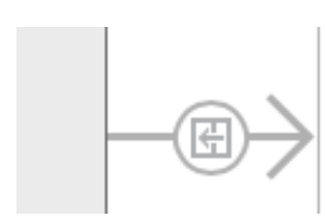
2. Choose Selection Segue >> Show

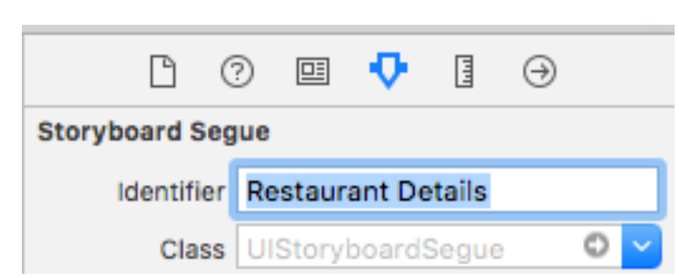
   - This creates a **segue** between the 2 view controllers, triggered when the user taps a cell

   - Notice that RestaurantDetailsVC now is also sporting a navigation bar -- it will be pushed onto the navigation controller's stack when the cell is tapped

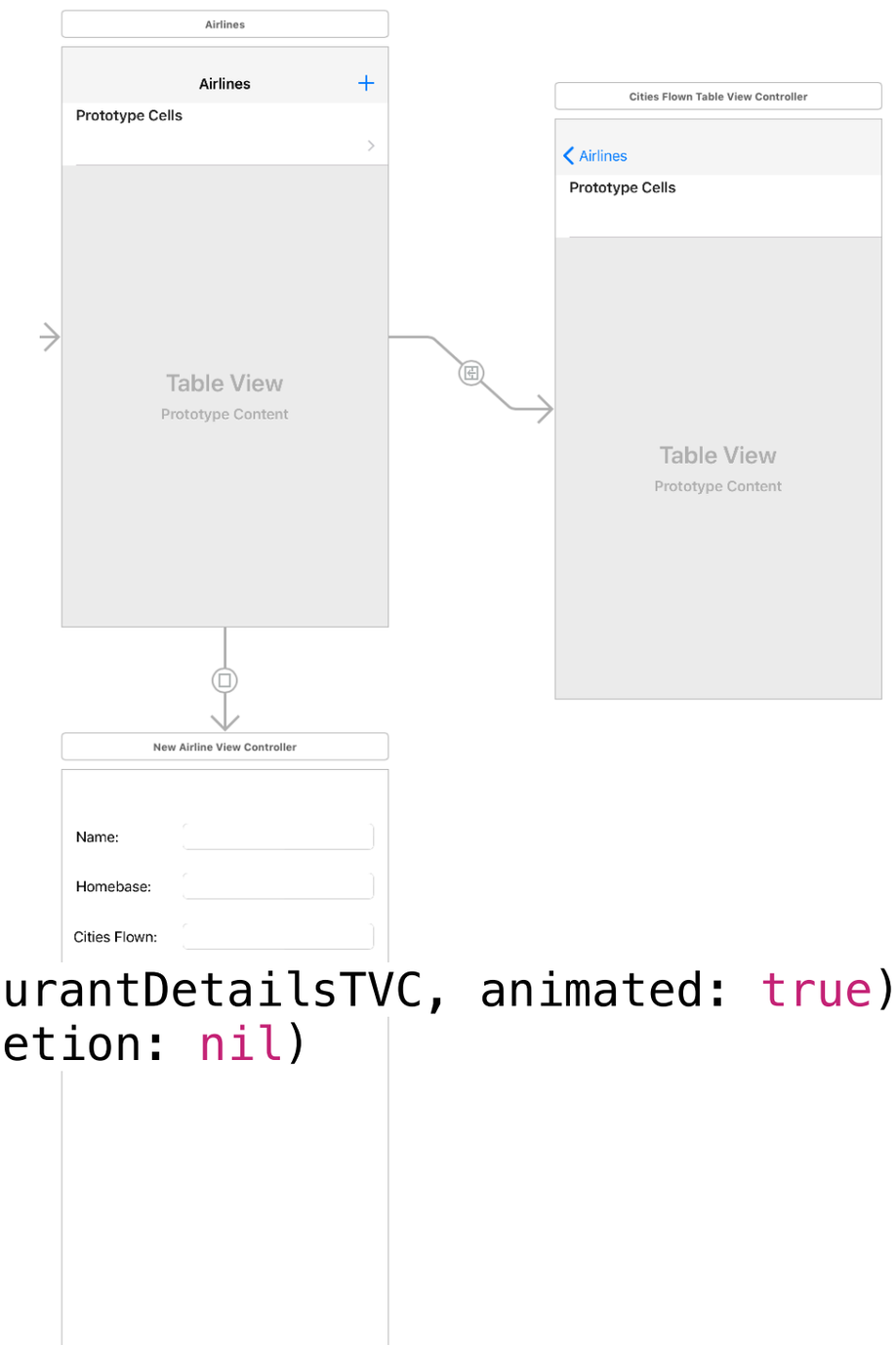
3. Give the segue you just selected an identifier, **Restaurant Details**







What's a Segue?



- A **segue** defines a transition between two view controllers in a storyboard file
- The starting point is a button, table view cell, or gesture associated with the originating ViewController
- The end point is the UIViewController to be displayed
  - The ViewController associated with the starting point is the **presenting** ViewController; the destination ViewController is the **presented** ViewController. [This terminology applies whether a segue is used or the ViewController is presented programmatically.]
  - Segues are not triggered programmatically. We need not write, in tableView(tableView:didSelectRowAtIndexPath:), or in a button's IBAction, respectively:

- `self.navigationController?.pushViewController(restaurantDetailsTVC, animated: true)`
- `self.present(newRestaurantVC, animated: true, completion: nil)`

  - Instead, that code (as well as the code to instantiate the ending ViewController) happens behind the scenes
  - We will have more to say about segues in another presentation

prepare(for segue:sender:)

- Often we need to configure the destination ViewController, supplying it information that it can display once it appears.
- To do so, in the presenting (originating) ViewController, override
  - **prepare(for segue:UIStoryboardSegue, sender:Any?)**
  - In that method, we can access the ending ViewController using **segue.destination**, typecasting accordingly
  - Since a ViewController can be the originator for multiple segues, we can assign each segue an identifier, to execute code only when the appropriate segue has been triggered.

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
      // Now we instantiate a CitiesFlownTableViewController, where we will display a list of cities flown

      if segue.identifier == "friend_details" {
          let citiesFlownTVC:CitiesFlownTableViewController = segue.destination as! CitiesFlownTableViewController
          citiesFlownTVC.airline = FAA.airlineNum(tableView.indexPathForSelectedRow!.row)
      }
  }
```

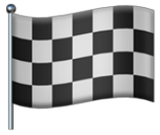
5. Define a stored property, **restaurant**, in the RestaurantDetailsViewController, and implement viewWillAppear(_:animated:)

```
class RestaurantDetailsViewController: UIViewController {

    @IBOutlet weak var genreLBL:UILabel!
    @IBOutlet weak var ratingLBL:UILabel!

    var restaurant:Restaurant!

    override func viewWillAppear(_ animated: Bool) {
        genreLBL.text = "\(restaurant.genre)"
        ratingLBL.text = "\(restaurant.rating)"
        navigationItem.title = "\(restaurant.name)"
    }
```

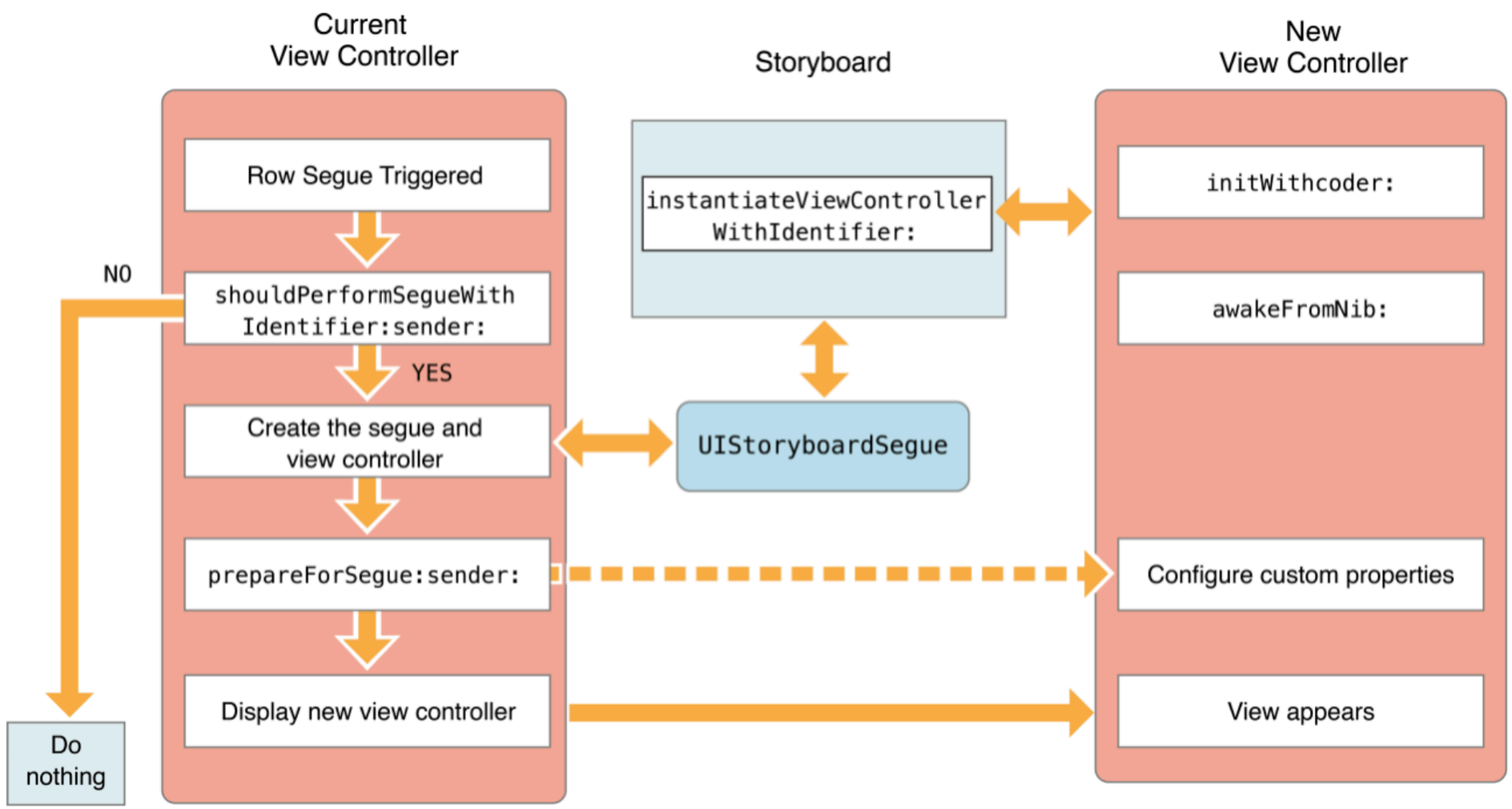
6. In RestaurantsTVC, implement **prepare(for segue:sender:)** to initialize restaurant

```
// In a storyboard-based app, you may wish to do a little prep before navigation

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

 // Get the new view controller using segue.destinationViewController.
 let restaurantDVC = segue.destination as! RestaurantDetailsViewController

 // Pass the selected object to the new view controller.
 restaurantDVC.restaurant = Restaurants.shared[tableView.indexPathForSelectedRow!.row]

}
```

Every table view controller can access its table view via the tableView property, and every tableView knows the index path of the most recently selected row



What Happens when a Segue is Triggered



Returning from a Segue

- There are at least 3 ways to return from a segue:

1. If a navigation controller is involved, tapping the back button handles it automatically

2. You can programmatically call the ViewController's dismiss(animated:completion:) method, eg.,

```
@IBAction func cancel(_ sender: Any) {
        self.dismiss(animated: true, completion: nil)
}
```

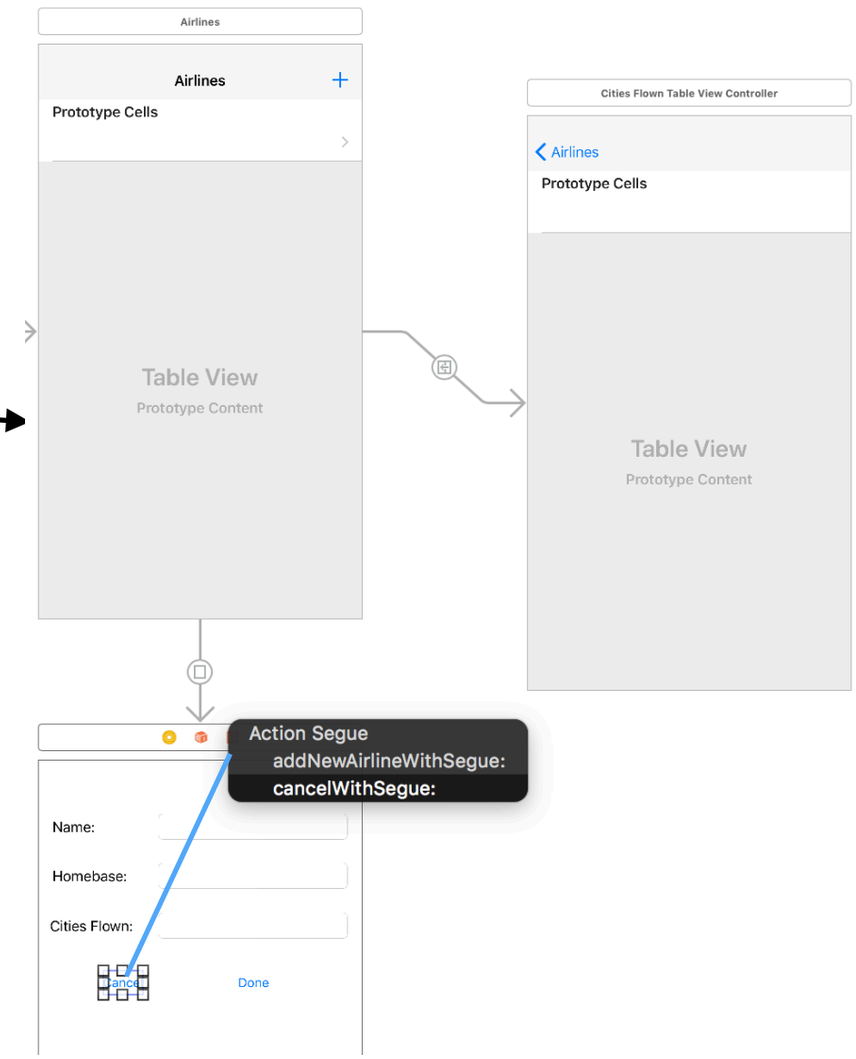
3. You can create an **unwind segue**

Unwind Segues

- Unwind segues let you dismiss a presented view controller
- To create an unwind segue, identify the ViewController that should appear when the unwind segue completes
- In that ViewController, define an unwind action method, with this signature*

```
@IBAction func cancel(unwindSegue: UIStoryboardSegue)
```

- In the presented View controller, control-drag from the button that initiates the unwind segue to the exit icon
- Release it and select the unwind action
- An unwind segue will trigger the prepare(for segue:sender:) method in the presented view controller before the segue happens



*the method and parameter names can vary: **the rest must be as written here***

ICE: Care to Look at a Menu?

- Currently, Restaurants is a *very* simple model. What if we wanted to display a Restaurant's menu: a collection of **menu items**, each with nutritional information?
  - what additional elements will we need in our model?
  - how will we redesign our UI to show Restaurants, each Restaurant's details (including its menu items), and then for each item in the menu, nutritional information?
- In class or as an OOCE* we will explore these issues

*Out Of Class Exercise

Resources

- https://cocoacasts.com/how-do-unwind-segues-work

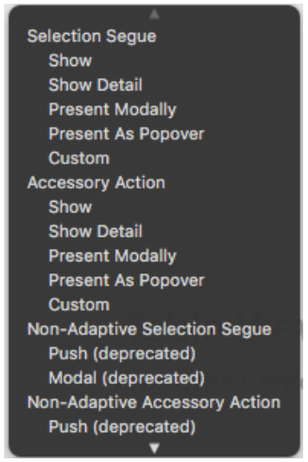
Exercises

UINavigationControllers in Storyboard: A Tutorial

1. Create a new single view project, **Nav Cons in SB**
    1. Drag in 2 UITableViewControllers. Position them horizontally, with the View Controller scene beneath the UITableViewController on the left
    2. For the leftmost scene,
        1. tap on the yellow dot in the scene dock (to select the entire UITableViewController)
        2. change its identity to **AirlinesTableViewController**
        3. change its prototype cell reuse identifier to **airline_cell**
    3. Repeat steps 2.3 for the rightmost scene, using the identity **CitiesFlownTableViewController** and reuse identifier **city_cell**
    4. Change the lower UIViewController's identity from ViewController to **NewAirlineViewController**

UINavigation Controllers in Storyboard: A Tutorial

4. Embed the Airlines Table View Controller in a UINavigationController
    1. Select the Airlines Table View Controller
    2. Choose menu item Editor >> Embed in Navigation Controller
        1. AirlinesTVC now sports a NavigationBar 😍
    3. Select the Navigation Controller Scene
    4. Change the title to **Airlines**
    5. Make the NavigationController the initial view controller for the app -- it will now sport a sleek looking grey arrow
5. Select the Airlines Table View Controller's prototype cell
6. Drag from it to the Cities Flown Table View Controller
7. Choose **Show** from **Selection Segue**
8. Select the resulting segue
    - Set its identity to **cities_flown**



UINavigation Controllers in Storyboard: A Tutorial

4. Create a table view controller for Airlines Table
    1. From the menu do File - New - File and select Cocoa Touch Class
    2. Name it **AirlinesTableViewController**
    3. Make it a subclass of UITableViewController
    4. In the identity inspector for the Airlines Table, set the class property to be AirlinesTableViewController.
5. Create a table view controller for Cities Table

UINavigation Controllers in Storyboard: A Tutorial

9. Time to add in the data source and delegate methods for the table
    1. But first, we need the model. Create a new swift file named Airline and copy in the model code on the next slide. You will need to fix the invisible characters embedded in the slide text.
    2. Let's work on the data source. In the AirlinesTableView controller modify numberOfSections(in tableView:) to return 1.
    3. Modify tableView(_:numberOfRowsInSection:) to use the model, i.e., `return FAA.numAirlines()`

Our Model

```
struct FAA {

    static var faa:FAA = FAA()

    private init(){}

    var airlines:[Airline] = [
        Airline(name: "United", profits: 25.0, homebase: "Chicago",
           numEmployees: 2500,citiesFlown:["New York", "Toronto", "London", "Paris"]),
        Airline(name: "Delta", profits: 50.0, homebase: "Atlanta",
           numEmployees: 3500,citiesFlown:["Cairo", "Tel Aviv", "Ankara", "Tokyo", "Beijing"]),
        Airline(name: "Jet Airways", profits: 87.50, homebase: "Mumbai",
           numEmployees: 3000,citiesFlown: ["Bangalore", "Colombo", "Chennai", "Budapest",
                                            "London"]),
        Airline(name: "Lufthansa", profits:75.0,homebase: "London",
           numEmployees:1000,citiesFlown:["Rome", "Munich", "Johannesburg"]),
        Airline(name: "Air Canada", profits: 315.0, homebase: "Ottawa",
           numEmployees: 500,citiesFlown: ["Toronto", "Vancouver", "Tokyo"])
    ]

    // returns # of airlines
     func numAirlines()->Int {
        return airlines.count
    }

    // returns a particular airline
     func airlineNum(_ index:Int) -> Airline {
        return airlines[index]
    }

    // adds a new airline to the mix
     mutating func addNewAirline(_ airline:Airline){
        airlines.append(airline)
    }
}
```

```
struct Airline {

    var name:String
    var profits:Double
    var homebase:String
    var numEmployees:Int
    var citiesFlown:[String]

    init(name:String, profits:Double,
        homebase:String, numEmployees:Int,
        citiesFlown:[String]){
            self.name = name
            self.profits = profits
            self.homebase = homebase
            self.numEmployees = numEmployees
            self.citiesFlown = citiesFlown
    }
}
```

UINavigation Controllers in Storyboard: A Tutorial

9. Time to add in the data source methods for the table
    1. We need to configure and return the cell… Uncomment the method tableView(_:cellForRowAt:)
    2. Change the reuse identifier to airline_cell
    3. Configure the cell thusly:

```
let airline = FAA.airlineNum(indexPath.row)
cell.textLabel?.text = airline.name
cell.detailTextLabel?.text = airline.homebase
```

10. We can now run the application!

UINavigation Controllers in Storyboard: A Tutorial

11. Now we can implement the delegate methods for the table view.
    1. Because we are going to use a navigation controller, delegate behavior that was specified in tableView(_:didSelectRowAtIndexPath:) should now happen in the **prepare(for segue:sender:)** method.
    2. Uncomment that method and add the following 2 lines of:

Every segue has a source and a destination.

```
let citiesFlownTVC:CitiesFlownTableViewController = segue.destination as!
                                                    CitiesFlownTableViewController
citiesFlownTVC.airline = FAA.airlineNum(tableView.indexPathForSelectedRow!.row))
```

Ask the tableView for the indexPath of the selected cell.

Note: We did not have to create the table view controller. We did not have to push. Those happened automatically as part of the segue.

UINavigation Controllers in Storyboard: A Tutorial

12. We need to finish up the Cities Flown Table View Controller.
    1. We need to create the variable that the segue will touch.

```
var airline:Airline!
```

    2. Modify to return the number of sections (1)
    3. Modify to return the number of rows in the section.

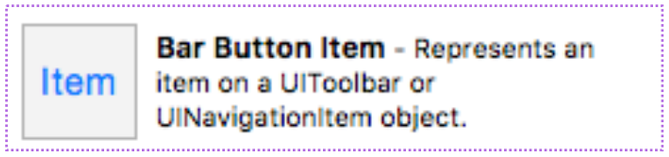
```
airline.citiesFlown.count
```

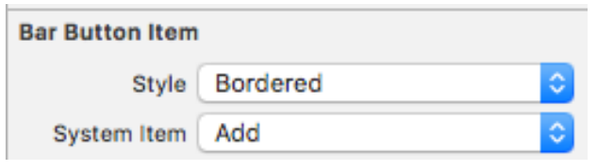
    4. Configure and return the cell for reuse id “city_cell”

```
cell.textLabel?.text = airline.citiesFlown[indexPath.row]
```

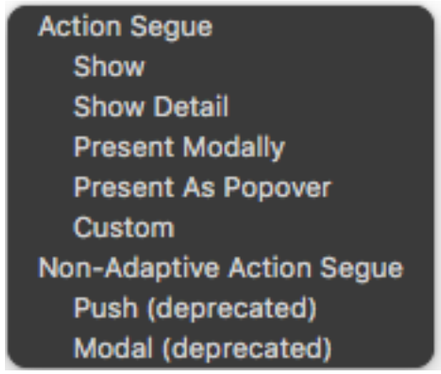
Final Details: The + Button

1. Finally, we would like to install an **+ (add)** button in the navigation bar that will take us to a view where we can add a new airline. The basic idea is that we need a bar button to the airlines view, that when pressed will trigger a segue.
    1. In Storyboard, drag a BarButtonItem into the NavigationBar of the AirlinesTableViewController
    2. With the BBI selected, change its System item to Add
    3. Control drag from that button to NewAirlineViewController and create a Present Modally segue
        - Make the segue identifier **new_airline**
    4. Add three textfields to the NewAirlineTableViewController, then add **@IBOutlet weak** before nameTF, homebaseTF, and citiesFlownTF
    - This exposes the properties so we can connect to them in Storyboard

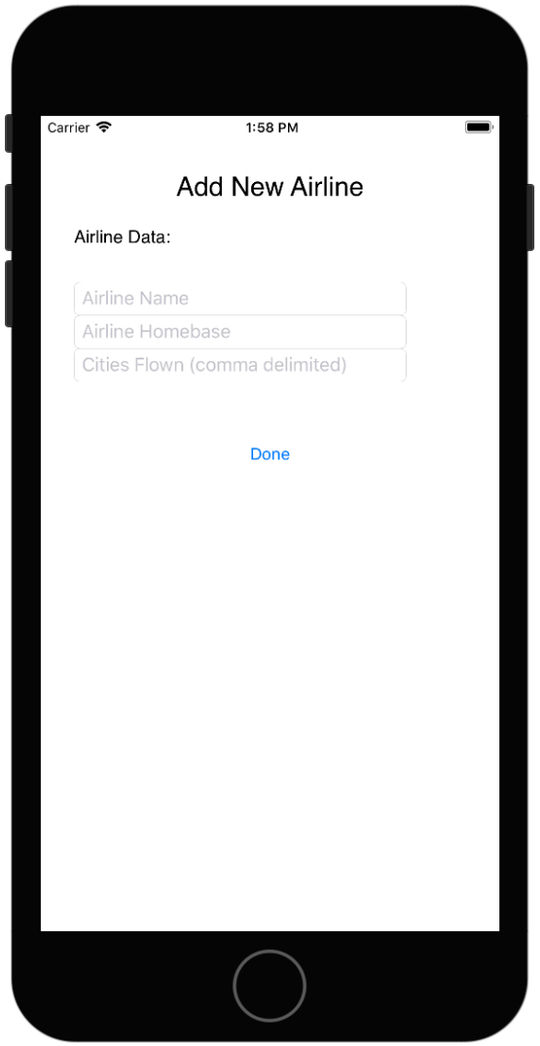





```
@IBOutlet weak var nameTF:UITextField!
@IBOutlet weak var homebaseTF:UITextField!
@IBOutlet weak var citiesFlownTF:UITextField!
```



Final Details: The + Button



4. In Storyboard, create the New Airline UI as shown at right.
    - Connect the outlets
5. Create an unwind segue method in AirlinesTableViewController:

```
@IBAction func addNewAirline(segue:UIStoryboardSegue){ }
```

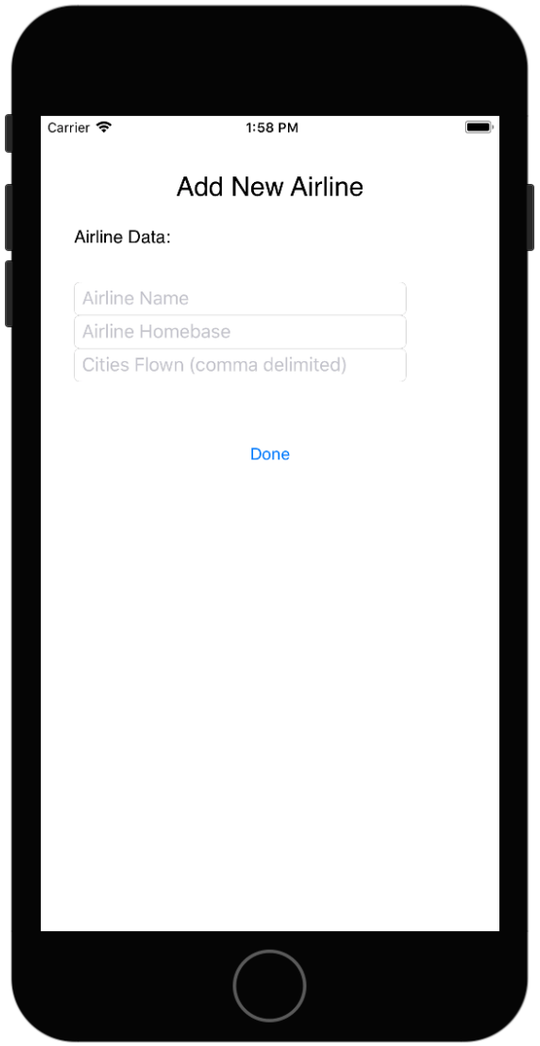
6. Configure the Done button to trigger that unwind segue
    1. Control drag from the Done button to the Exit segue icon in the scene doc, and choose addNewAirline
7. In NewAirlineViewController, instead of having an action method associated with our button, we will intercept the segue… See next slide!

Final Details: The + Button

7. Create the following method

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?)
{
  let airlineToAdd = Airline(name: nameTF.text!,
                 profits: 0.0,
                 homebase: homebaseTF.text!,
                 numEmployees: 0,
                 citiesFlown:
                 (citiesFlownTF.text?.components(separatedBy: ",")!)

     FAA.addNewAirline(airlineToAdd)
}
```

- Everything is *almost* ready to go ... run the app and enjoy the results ... until you click on the + button and get a crash 😢



Final Details: The + Button

- Our AirlineTableViewController now has **two** segues — one to CitiesFlownTableViewController, and the other to NewAirline. Unfortunately, our code in prepare(for segue:sender:) is designed for the former, so when we tapped on the + button, the downcast to CitiesFlownTableViewController failed.

- The code in prepareForSegue(_:sender) should be specialized based on the segue that we are using. Make sure both segues have an identifier and then use an if statement. The additional code is highlighted:

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
      // Now we instantiate a CitiesFlownTableViewController, where we will display a list of cities flown

      if segue.identifier == "cities_flown" {
          let citiesFlownTVC:CitiesFlownTableViewController = segue.destination as! CitiesFlownTableViewController
          citiesFlownTVC.airline = FAA.airlineNum(tableView.indexPathForSelectedRow!.row)
      }
  }
```

Final Details: The + Button

- At least it didn't crash this time… but the new airline does not show up. At some point we need to let iOS know that the data has changed. We need to use the table view method `reloadData()`.

- We can call it either in viewWillAppear or in the unwind segue we created earlier.

```
@IBOutlet weak var myTV:UITableView!

@IBAction func addNewAirline(segue:UIStoryboardSegue){
    myTV.reloadData()
}
```