Variable shadowing

```
let points = 100

for index in 1...3 {
  let points = 200
  print("Loop \(index): \(points+index)")
}
print(points)
```

```
Loop 1: 201
Loop 2: 202
Loop 3: 203
100
```

Variable shadowing

```
var name: String? = "Robert"

if let name = name {
 print("My name is \(name)")
}
```

Variable shadowing

```
func exclaim(name: String?) {
  if let name = name {
    print("Exclaim function was passed: \(name)")
  }
}
```

```
func exclaim(name: String?) {
  guard let name = name else { return }
  print("Exclaim function was passed: \(name)")
}
```

Shadowing and initializers

```
struct Person {
  var name: String
  var age: Int
}

let todd = Person(name: "Todd", age: 50)
print(todd.name)
print(todd.age)
```

```
Todd
50
```

Shadowing and initializers

```
struct Person {
  var name: String
  var age: Int

  init(name: String, age: Int) {
    self.name = name
    self.age = age
  }
}
```

Unit 3, Lesson 1
Lab: Optionals.playground

Open and complete the exercises in `Lab - Optionals.playground`

Unit 3—Lesson 3
Lab: Guard

Open and complete the exercises in `Lab - Guard.playground`

Unit 3—Lesson 4
Lab: Constant and Variable Scope

Open and complete the exercises in `Lab - Scope.playground`

Session 10: Advanced navigation

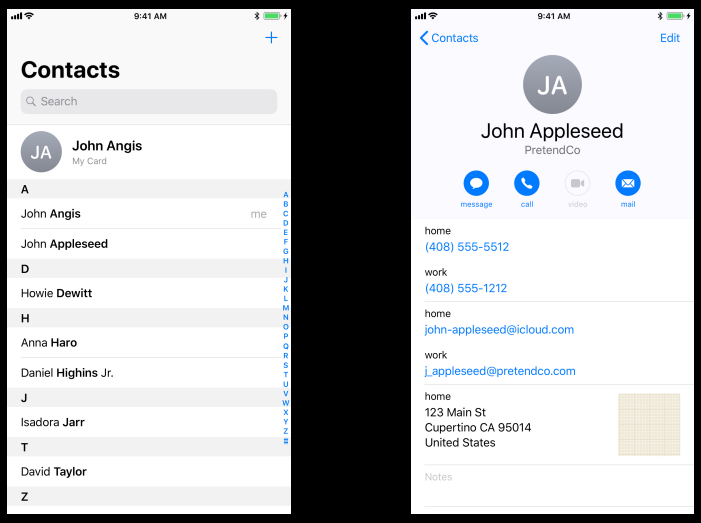
- Segues
- Navigation
- Tab bars
- Lifetime of an app
- *Lab 10: Making complex apps*

- This covers lessons 3.6, 3.7, 3.8 of Development in Swift Fundamentals

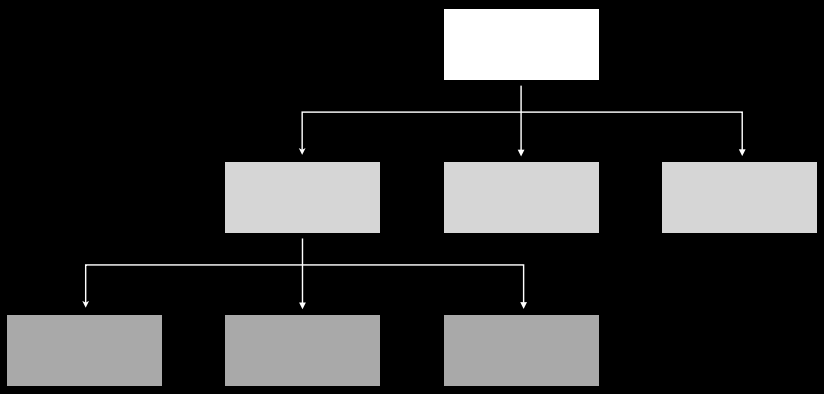
1

Unit 3—Lesson 6: Segues and Navigation Controllers

Segues and navigation controllers



Navigation hierarchy
Hierarchical



Segues (UIStoryboardSegue)

A `UIStoryboardSegue` object performs the visual transition between two view controllers

It is also used to prepare for the transition from one view controller to another

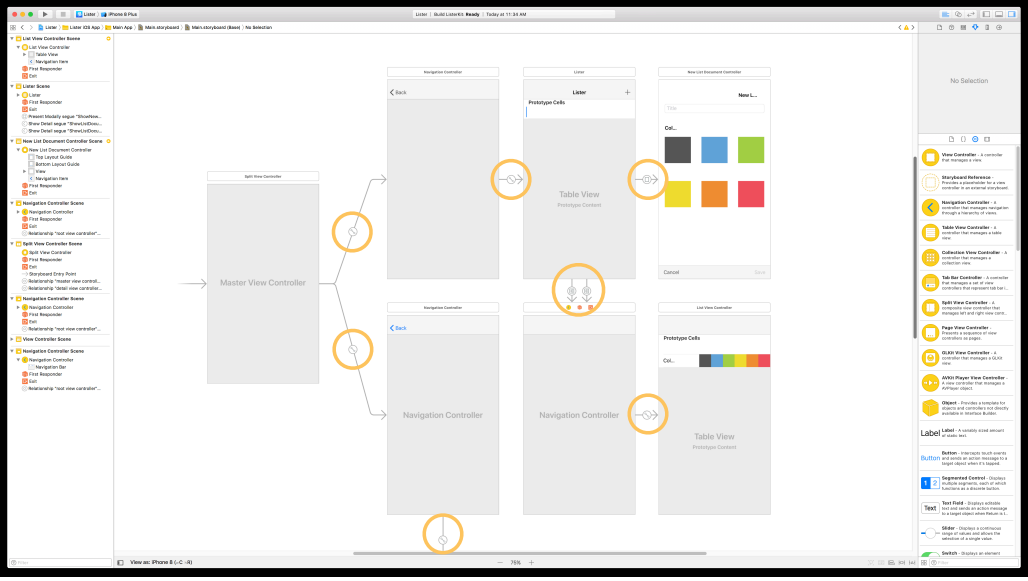
Segue objects contain information about the view controllers that are involved in a transition

When a segue is triggered, before the visual transition occurs, the storyboard runtime can call certain methods in the current view controller

Useful if you need to pass information forward

Segues (UIStoryboardSegue)

Segues between scenes



Segues (UIStoryboardSegue)
Unwind

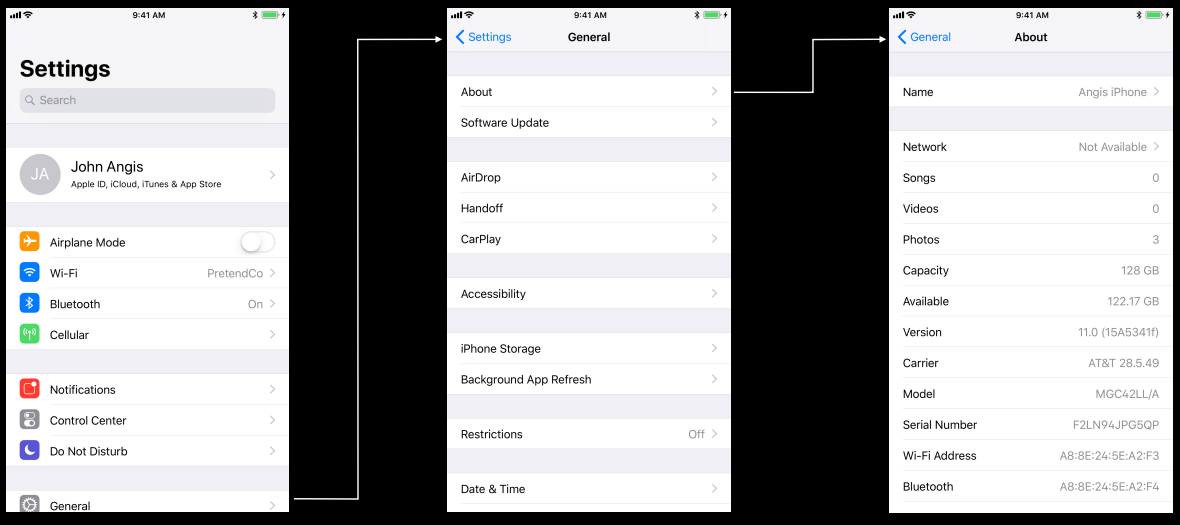
```
@IBAction func myUnwindFunction(unwindSegue: UIStoryboardSegue) {
}
```

Implement the returned method in the view controller you wish to return to

Doesn't need to do anything apart from being implemented

Connect this to the view controller returning from

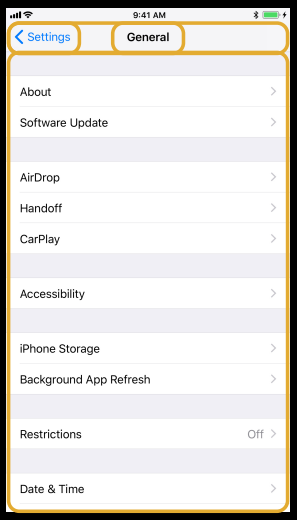
Navigation controller (UINavigationController)



Navigation controller

The top view controller's title

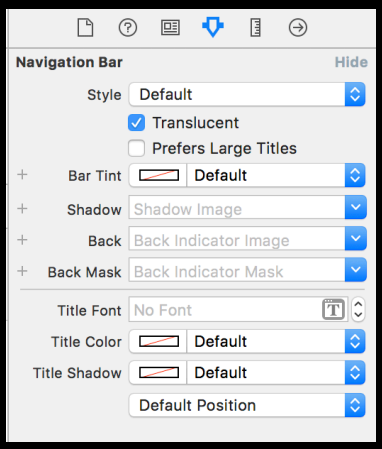
Back button

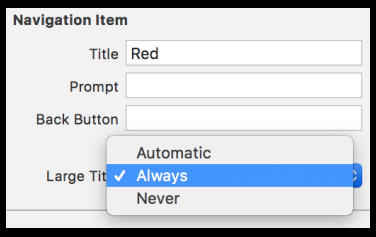


Navigation bar

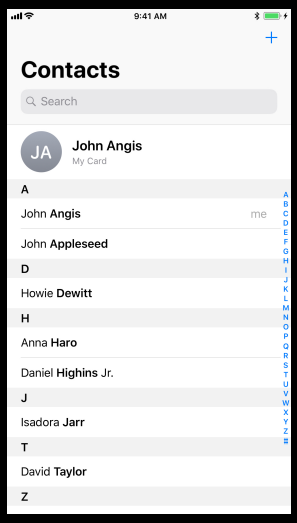
The top view controller's view

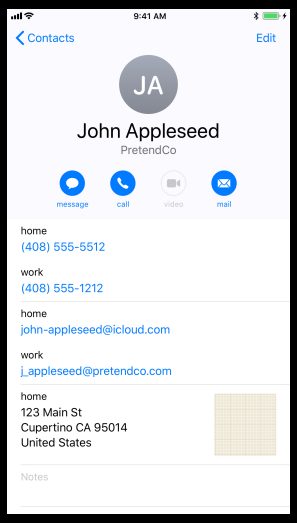
Navigation controller
Large titles





Pass information





Pass information

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

Segue properties

- `identifier`
- `destination`

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
   segue.destination.navigationItem.title = textField.text
}
```

Create programmatic segues

```
performSegue(withIdentifier:, sender:)
```

```
performSegue(withIdentifier: "ShowDetail", sender: nil)
```

Unit 3—Lesson 6
Segues and Navigation Controllers

Learn how to use segues to transition from one view controller to another

How to define relationships between view controllers

How navigation controllers can help you manage scenes that display related or hierarchical content

Unit 3—Lesson 6
Lab: Login

Create a login screen that will pass a user name between view controllers

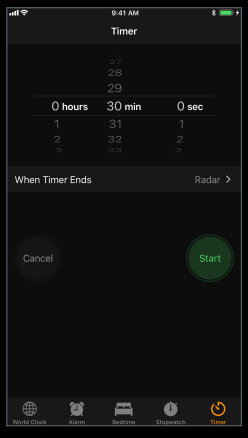
Use view controllers, a navigation controller, and segues to create both the login screen and a simple landing screen that will display in its title either the user name or text related to a forgotten user name or password

Unit 3—Lesson 7: Tab Bar Controllers

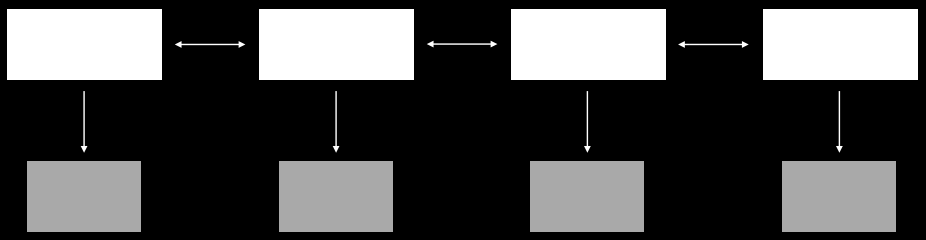
UITabBarController

A specialized view controller that manages a radio-style selection interface

A tab bar is displayed at the bottom of the view



Navigation hierarchy
Flat



UITabBarController
Configuration

Add a tab bar controller

Using a storyboard

Drag in a UITabBarController from the object library

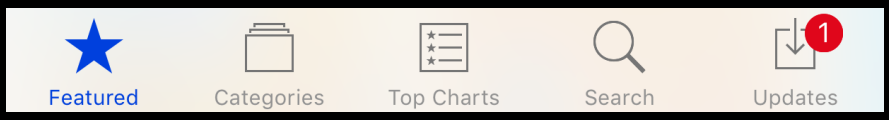
Add tabs

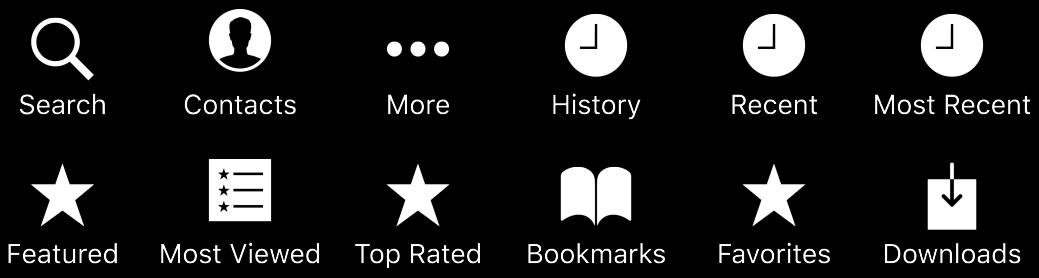
Drag a new view controller object onto the canvas

To create a segue, control-drag from the `UITabBarController` to the view controller

Select "view controllers" under Relationship Segue

UITabBarItem
Glyphish





Programmatic customization

```
tabBarItem.badgeValue = "!"
```

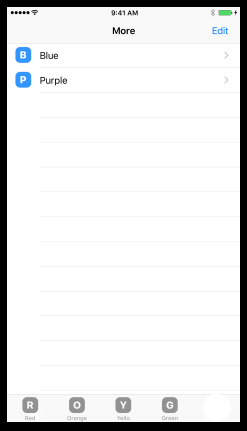

```
tabBarItem.badgeValue = nil
```

Even more tab items
More ! = Better

More view controller:

- Appears when needed
- Can’t be customized

If possible, plan app to avoid More



Unit 3—Lesson 7
Tab Bar Controllers

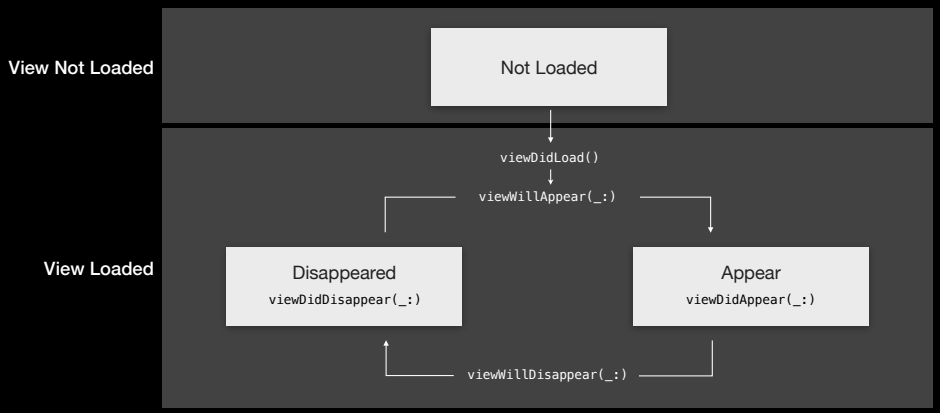
Learn how to use tab bar controllers to organize different kinds of information or different modes of operation.

Unit 3—Lesson 7
Lab: About Me

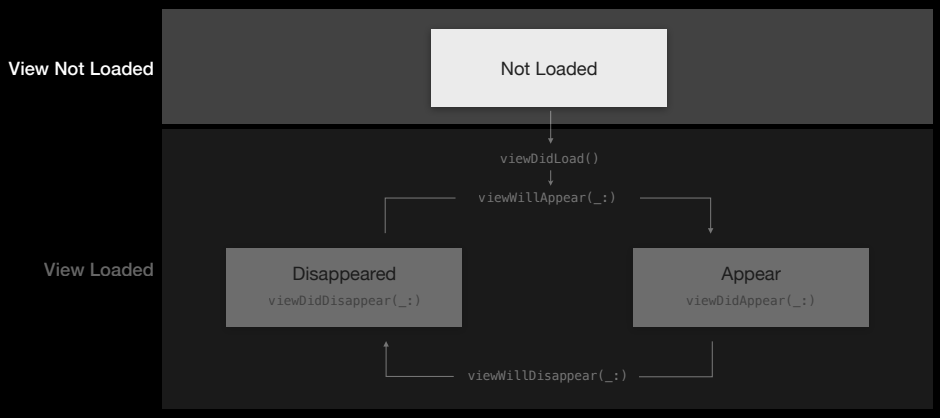
Create an app that displays distinct types of information about yourself in separate tabs.

Unit 3—Lesson 8:
View Controller Life Cycle

View controller life cycle



# View controller life cycle
viewDidLoad( )

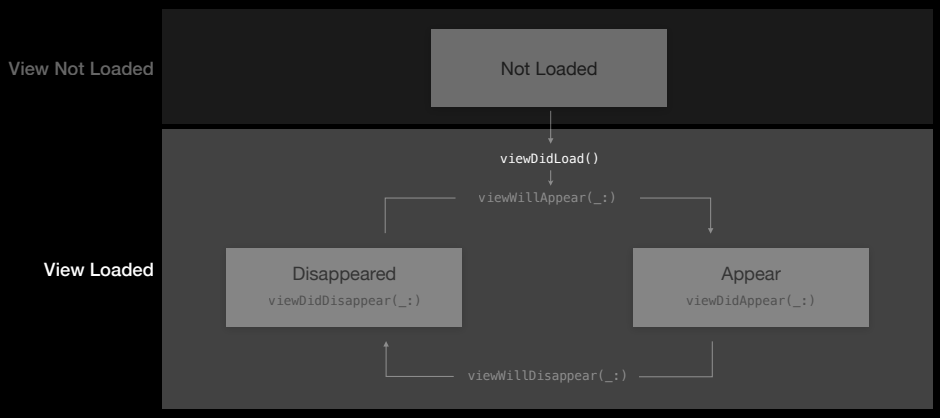


View event management

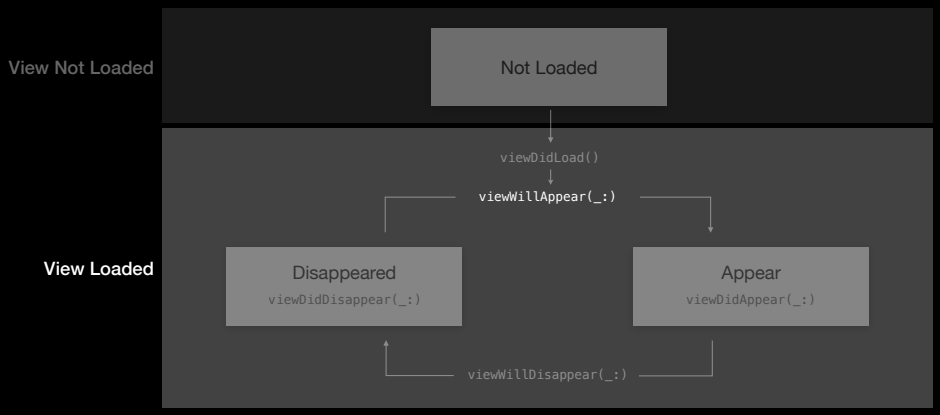
```
viewWillAppear(_:)
viewDidAppear(_:)
viewWillDisappear(_:)
viewDidDisappear(_:)
```

```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    // Add your code here
}
```

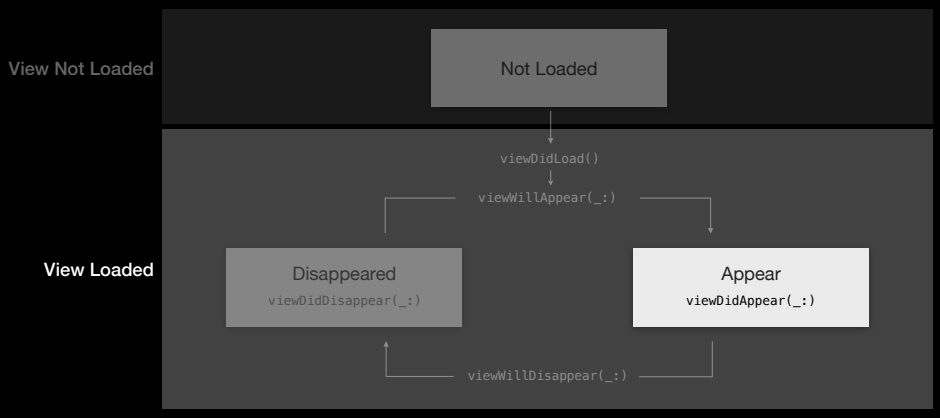
View event management
viewWillAppear(_:)



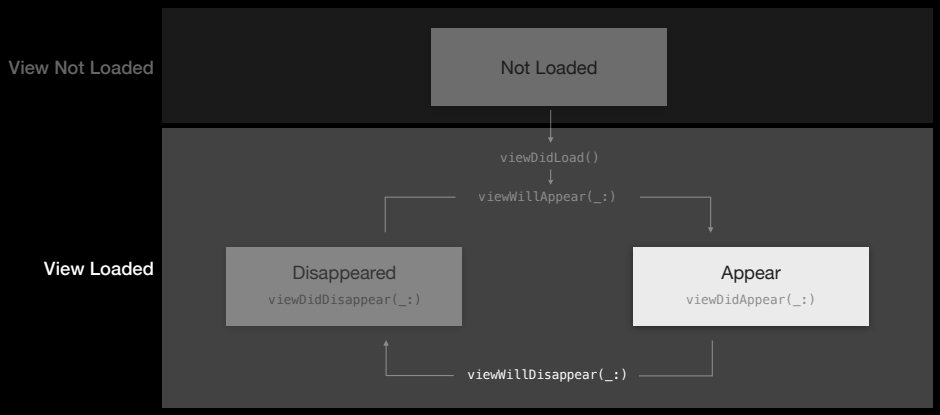
View event management
viewDidAppear(_:)



View event management
viewWillDisappear(_:)



View event management
viewDidDisappear(_:)



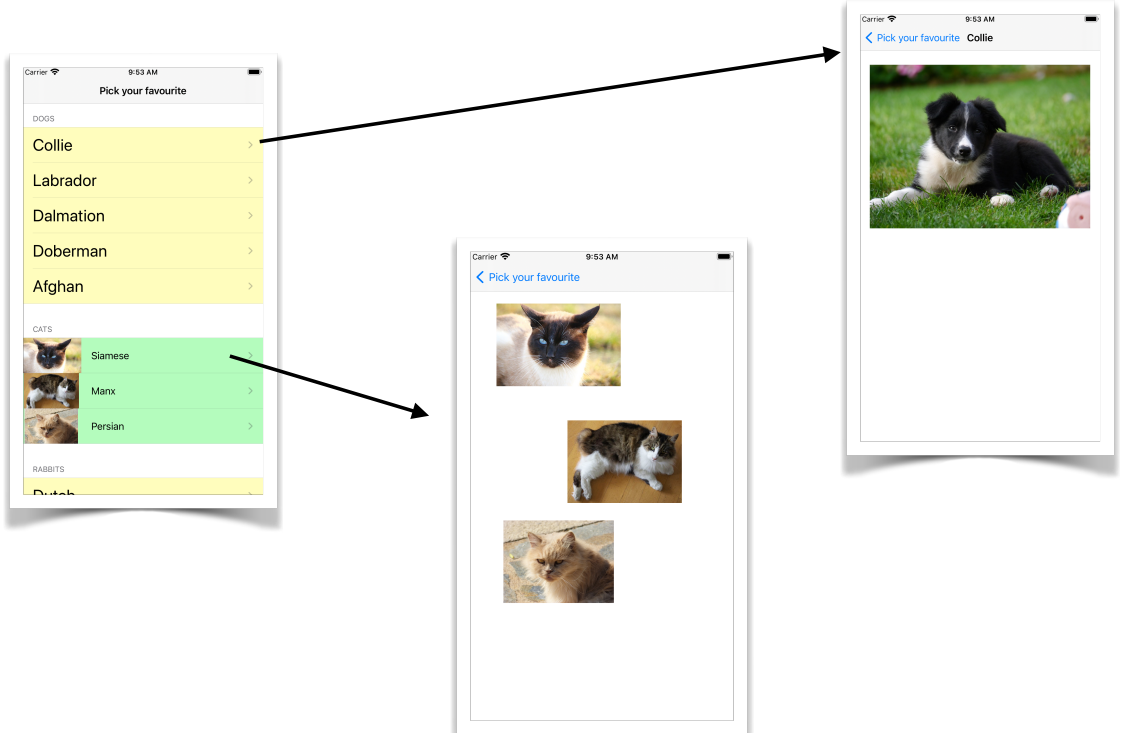
Unit 3—Lesson 8
View Controller Life Cycle

This lesson will explain more about the view controller life cycle so you can understand the infinite potential of this important class

Unit 3—Lesson 8
Lab: Order of Events

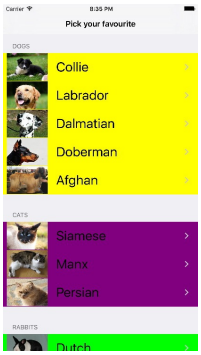
Further your understanding of the view's life cycle by creating an app that adds to a label's text based on the events in the view controller life cycle

Session 12: Making Dynamic Tables



Delegates - allow you to supply info

UITableViewController uses *delegates* to specify what to show in a table:



- How many SECTIONS are in the table?
- What are the headers/footers for each section?
- How tall is each header/footer?
- How many cells are in each section?
- What does each cell in each section look like?

There are default answers for each of these. If you don't want the default, you must provide a delegate.

There are two sets of delegates associated with a table:

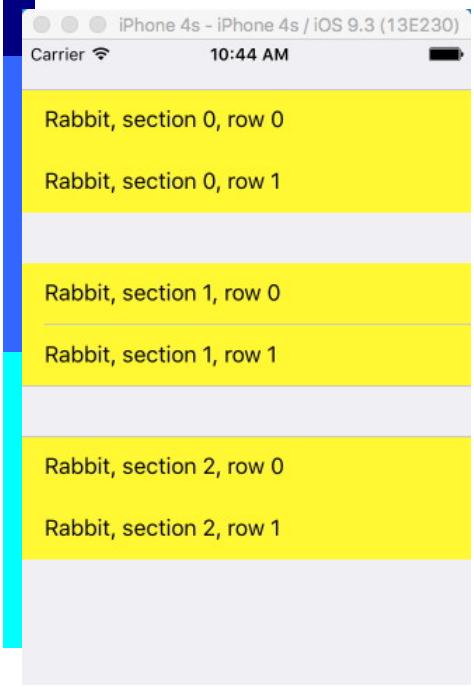
- UITableViewDataSource has delegates that provide details of what data to use in the table
- UITableViewDelegate has other delegates that you customise to say what the table looks like

Minimum that you need to define is:

- How many sections there are
- How many rows are in each section
- What text is in each row in each section

3

First Animal App - Minimum answers for a table



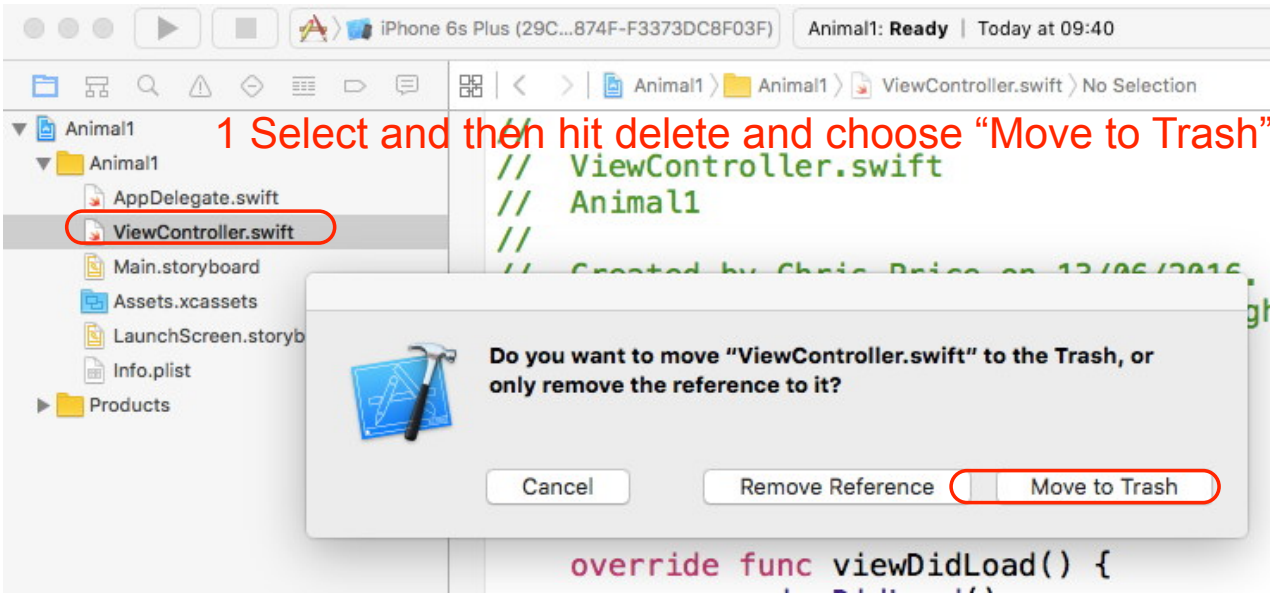
Step 1

•Make a new Project called Animal1
•Make it a Single View project

4

Step 2

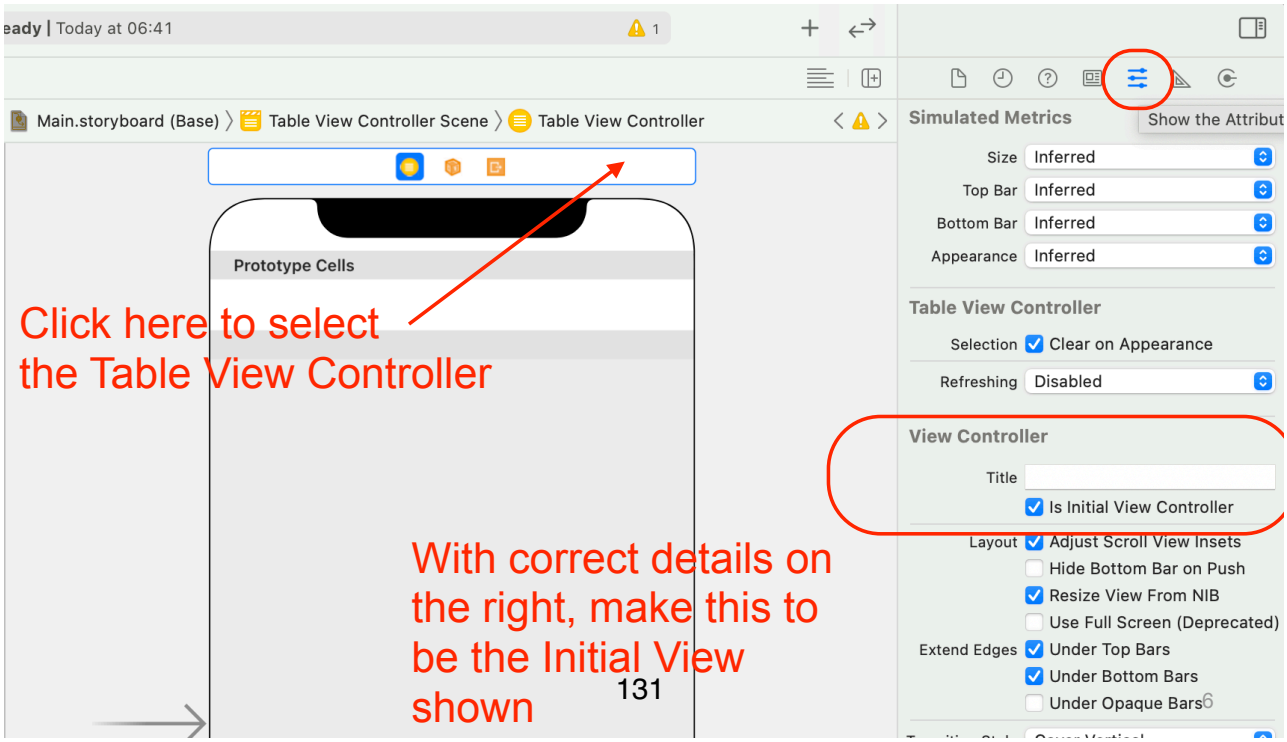
- Delete ViewController.swift as we won't be using it.
- Go to Main.storyboard, select the window and delete it (as we did when making static tables).



2 Select storyboard and get rid of the ViewController view

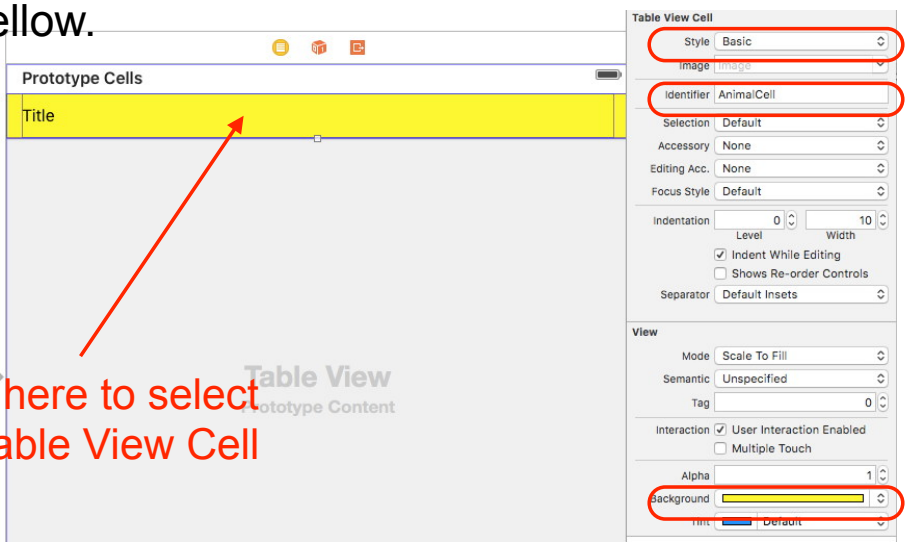
Step 3

- Add a new TableViewController screen to the story board.
- Select it and choose to make it initial view controller.



Step 4

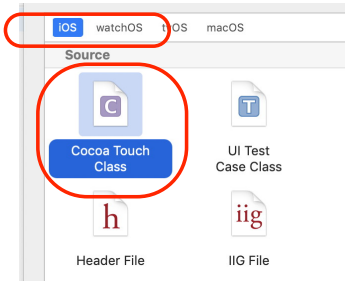
- Select the Table View Cell. Make its style Basic, make its identifier AnimalCell, and make its background colour yellow.



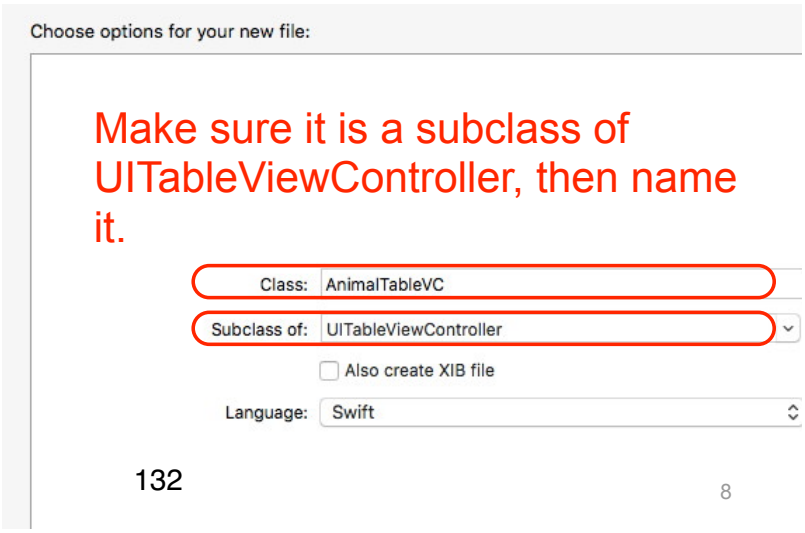
- You can run your app now, but it will look pretty dull as you don't have any code associated with the View.
- You now need to create a UITableViewController file and write some code to define what goes in the table.

Step 5

•Select File/New/File to be asked for a template for the file.
•Choose iOS/Source/Cocoa Touch Class, and click Next to be asked "Choose options for your file".
•We are going to call our class AnimalTableVC and base ir on UITableViewController, then hit Next a couple of times to create it.



Choose the right kind of file



Step 6

The template for a UITableViewController contains outline definitions for several functions:

- viewDidLoad - called when the window starts the first time. Good place to set things up. *Leave for now.*
- numberOfSectionsInTableView - This function returns how many sections there are in the table. *Make it return 3 instead of zero.*
- tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int
- This function tells you for which section it wants to know how many rows it has. *Make it 2 for each section.*

Step 7

UITableViewController also has some commented out functions. Uncomment the first of those

- tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: IndexPath) -> UITableViewCell
- This delegate is told the section and row of the cell that is wanted (indexPath.section and indexPath.row) and has to return a UITableCell that defines what the cell should look like.

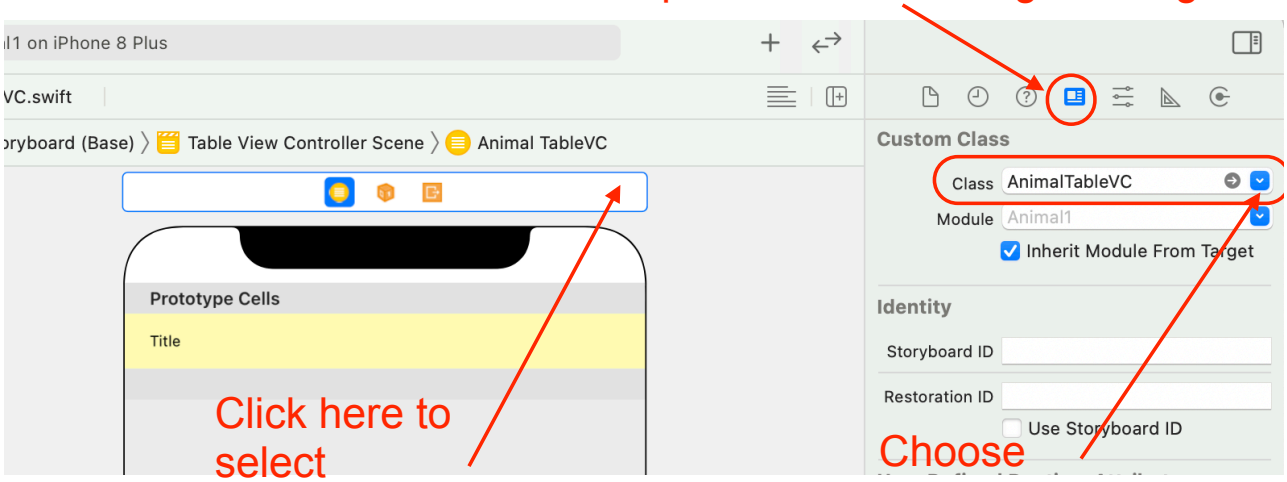
Make cellForRowAtIndexPath look like this:

```
override func tableView(_ tableView: UITableView,
                        cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "AnimalCell",
                                             for: indexPath)
    // Configure the cell...
    cell.textLabel?.text = "Rabbit, section \(indexPath.section), row \(indexPath.row)"
    return cell
}
```

Step 8

•We still need to tell the View on the storyboard that this new class AnimalTableVC tells it how to prepare its content.

Need the Identity Inspector instead of the Attribute Inspector to see the right dialog.



Click here to select the TableView Controller screen

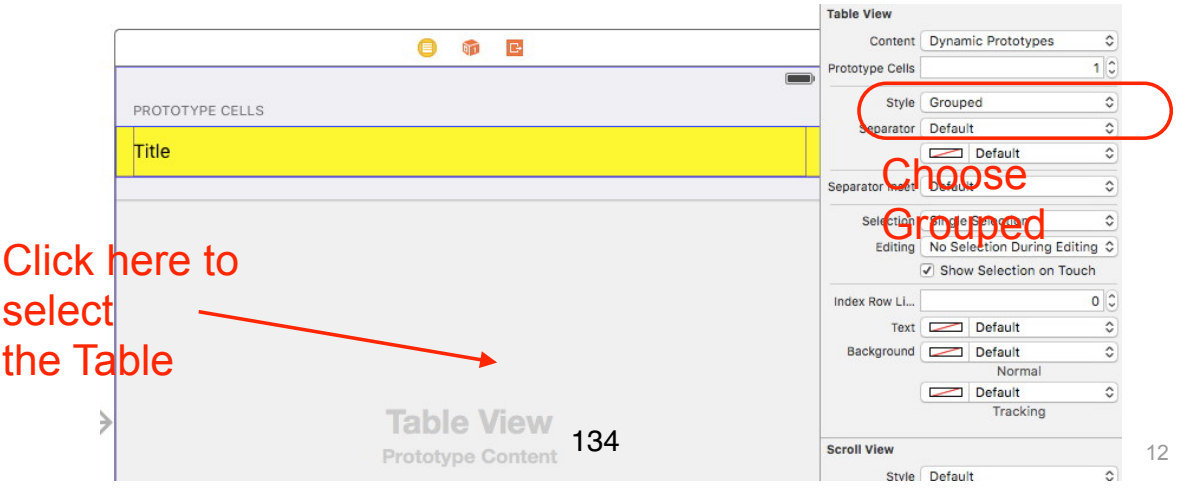
Choose AnimalTableVC from drop-down menu

11

Finished But!

•If you run it now, it should generate content for 2 cells in each of 3 sections (note that sections and cells are numbered from zero)

•The section split can be made more obvious by selecting the Table in the story board, and then changing the style to Grouped instead of Plain.



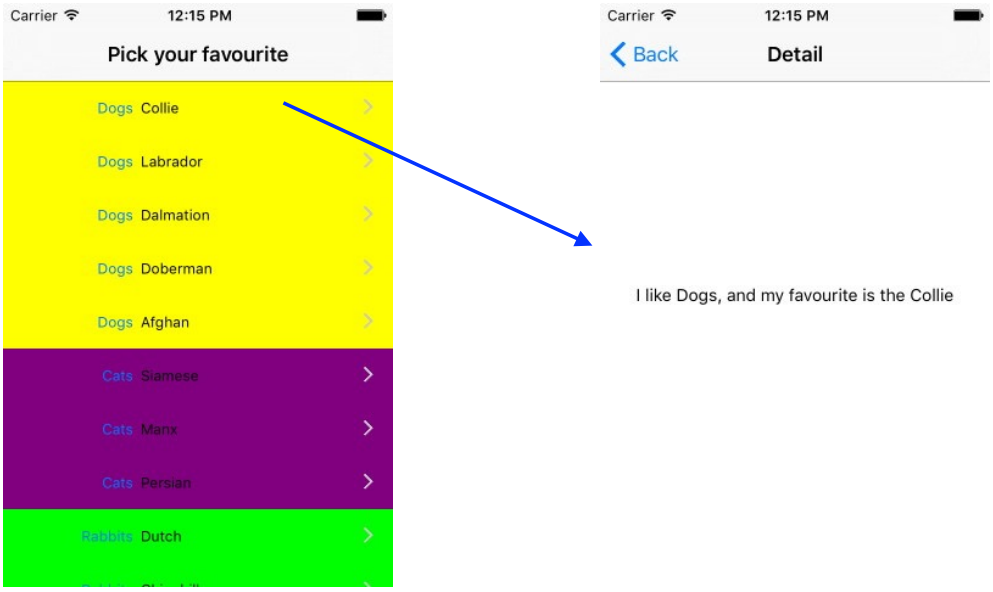
Click here to select the Table

Choose Grouped

12

Second Animal App
Use structured data and add a detail screen

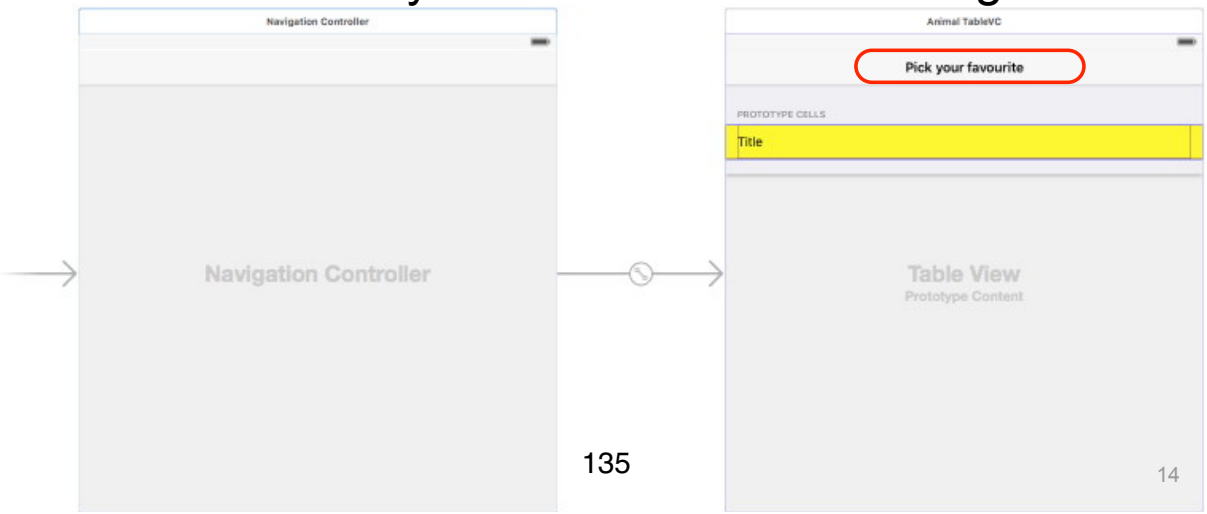
Either use your completed Animal1 App, or use the provided “Animal1 Finished” if you hit problems.



13

Step 1 - Get the structure right

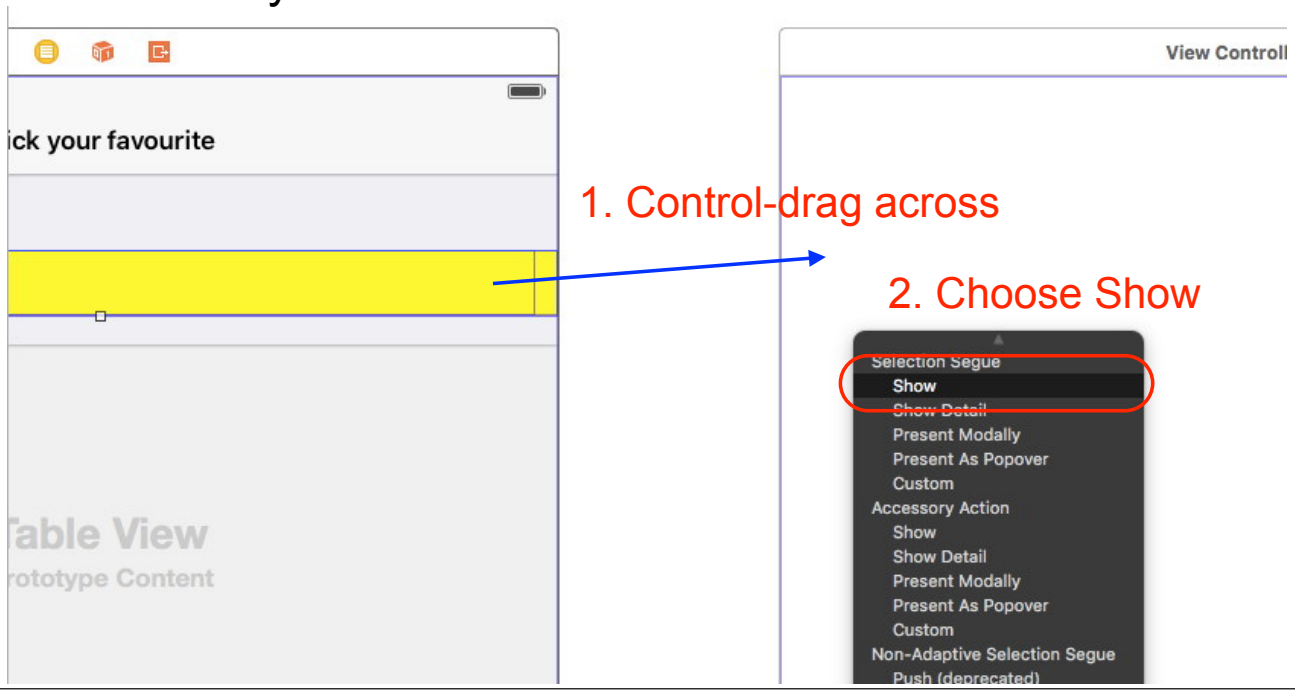
- If we want to push screens then we need a NavigationController - select AnimalTableVC in storyboard, and choose Editor/Embed in/ NavigationController
- Click in the new area at the top of AnimalTableVC, and add the title “Pick your favourite” to the Navigation Item



14

Step 2 - Add the detail screen

- Add another ViewController to the storyboard
- CONTROL-DRAG from the table cell to new controller
- Run and you can move between screens



Step 3 - Add the data

- Add animal data at the top of the class

First line below already exists… add rest underneath it

```
class AnimalTableVC: UITableViewController {

    var categories = ["Dogs", "Cats", "Rabbits" ]

    var categoryItems = [ [ "Collie", "Labrador", "Dalmation",
                            "Doberman", "Afghan"],
                          [ "Siamese", "Manx", "Persian"],
                          [  "Dutch", "Chinchilla", "Lionhead"]]
```

Step 4a - Add basic table code

- Need to say how many sections, and how many in each section - replace previous versions of functions below with these versions

```
override func numberOfSectionsInTableView(tableView:
UITableView) -> Int {
    return categories.count
}

override func tableView(tableView: UITableView,
numberOfRowsInSection section: Int) -> Int {
    return categoryItems[section].count
}
```

17

Step 4b - Add basic table code

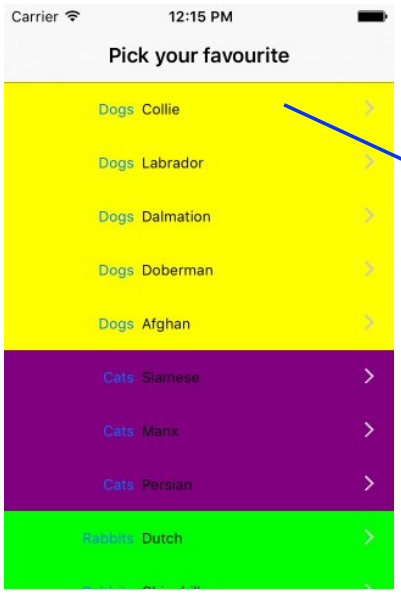
- Need to say what each cell has in it
- Replace existing cellForRowAtIndexPath as below - also need to change cell type in storyboard to be “Left Detail”

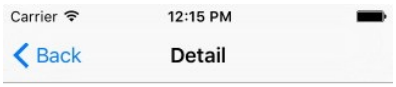
```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "AnimalCell",
                for: indexPath)
    cell.textLabel!.text = categories[indexPath.section]
    cell.detailTextLabel!.text = categoryItems[indexPath.section][indexPath.row]

    switch indexPath.section {
    case 0: cell.backgroundColor = UIColor.yellow
    case 1: cell.backgroundColor = UIColor.purple
    default: cell.backgroundColor = UIColor.green
    }
    return cell
}
```


18

Where are we with Second Animal App?

This screen all works





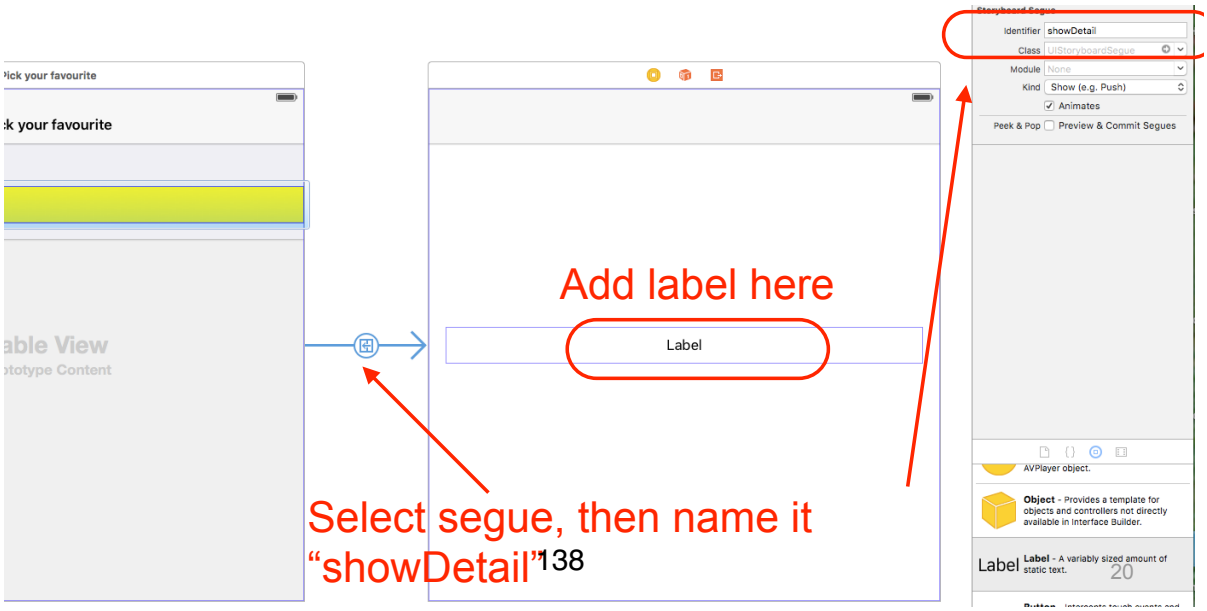
I like Dogs, and my favourite is the Collie

Need to pass the chosen info to the other screen

This happens in "prepareToSegue"

Step 5 - Get the storyboard ready

- Add a label to the detail screen, and name the segue (the transition from one screen to the other)



Step 6a - Need code to set up detail screen

- Create new controller file called AnimalDetailVC.swift as you did for AnimalTableVC.swift (but make a new UIViewController class not a UITableViewController class)
- Associate AnimalDetailVC with the Detail screen by selecting screen in storyboard and choosing it as before in the Identity Inspector
- Use Assistant Editor to drag drop from label on screen to code of AnimalDetailVC to make an @IBOutlet called favourite (as we did in very first app example)
- Run at this point - it should all work, but the label on the detail screen will be blank. We need to add code to pass selected info to the detail screen

Step 7 - Add the code to execute on segue

- In AnimalTableVC.swift, un-comment the declaration of prepareToSegue that is at the end of AnimalTableVC.swift, and add code below

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if segue.identifier == "showDetail" {
        if let indexPath = self.tableView.indexPathForSelectedRow {
            let detailVC = segue.destination as! AnimalDetailVC
            detailVC.valueForLabel = "I like " + categories[indexPath.section]
                + ", and my favourite is the "
                + categoryItems[indexPath.section][indexPath.row]
        }
    }
}
```

Header exists - add code outlined above

Step 8 - Use the value in AnimalDetailVC

- Could not set the label directly from prepareForSegue as it did not exist at that point. Once the new view controller has been created, can initialise label from passed value

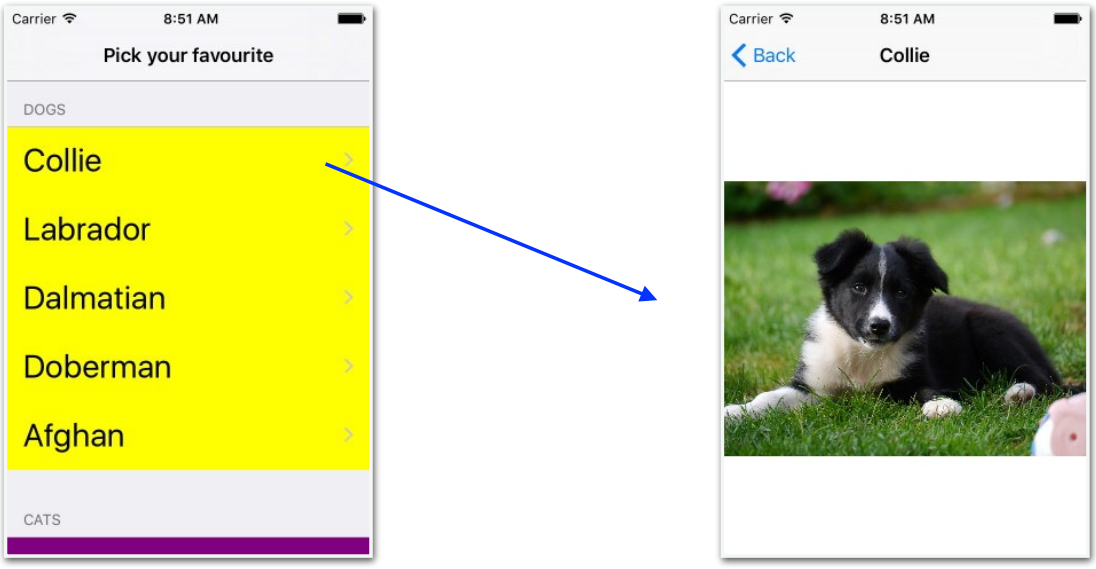
```
class AnimalDetailVC: UIViewController {

    @IBOutlet weak var favourite: UILabel!

    var valueForLabel = ""

    override func viewDidLoad() {
        super.viewDidLoad()

        favourite.text = valueForLabel
    }
```

Add code in red circle instead of existing viewDidLoad

23

Third Animal App
Put in headers and show photos

Either use your completed Animal2 App, or use the provided completed version if you hit problems.

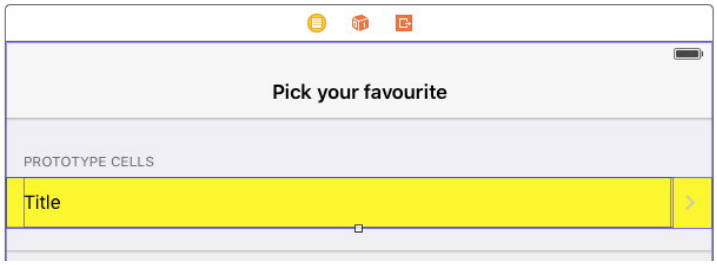


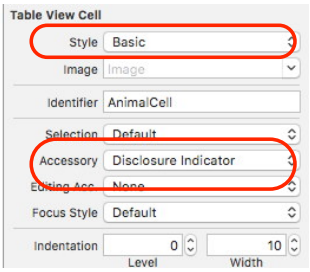
24

Step 1 - Rearrange AnimalTable

•Change style of AnimalCell to Basic - this will only show a single text. Increase the size of the label text to 28.
•Change Accessory to Disclosure Indicator (this implies there is a lower level of detail available)
•Delete the following line of code from rowAtIndexPath in AnimalTableVC or code will crash:

```
cell.detailTextLabel!.text =
categoryItems[indexPath.section][indexPath.row]
```





25

Step 2 - Add headers and resize

•Add the following routines to AnimalTableVC.swift - the first adds titles for each section, the others resize the height of the section title and the height taken by each cell
•If you run after this, you will see we need another change

```
override func tableView(_  tableView: UITableView, titleForHeaderInSection
section: Int) -> String? {
        return categories[section]
}

override func tableView(_ tableView: UITableView, heightForHeaderInSection
section: Int) -> CGFloat {
        return 40
}

override func tableView(_ tableView: UITableView, heightForRowAt indexPath:
IndexPath) -> CGFloat {
        return 60
}
```


26

Step 3 - Correct each cell

•Change the line that sets the text in each cell to be what used to be in the subcell

```
cell.textLabel!.text = categoryItems[indexPath.section][indexPath.row]
```

•In prepareForSegue, also change the item that is passed to just be the breed name instead of the long message, and add a title for the screen

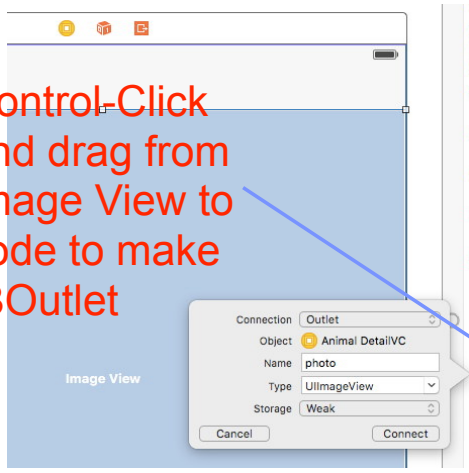
```
detailVC.valueForLabel = categoryItems[indexPath.section][indexPath.row]
detailVC.title = categoryItems[indexPath.section][indexPath.row]
```

•If you run now, the table should list breeds correctly, and the detail screen should just say the breed

27

Step 4 - Make detail screen show a picture

•Delete the label from the detail screen in storyboard
•Add an Image View instead
•Add IBOutlet for the Image View to AnimalDetailVC.swift and call it photo

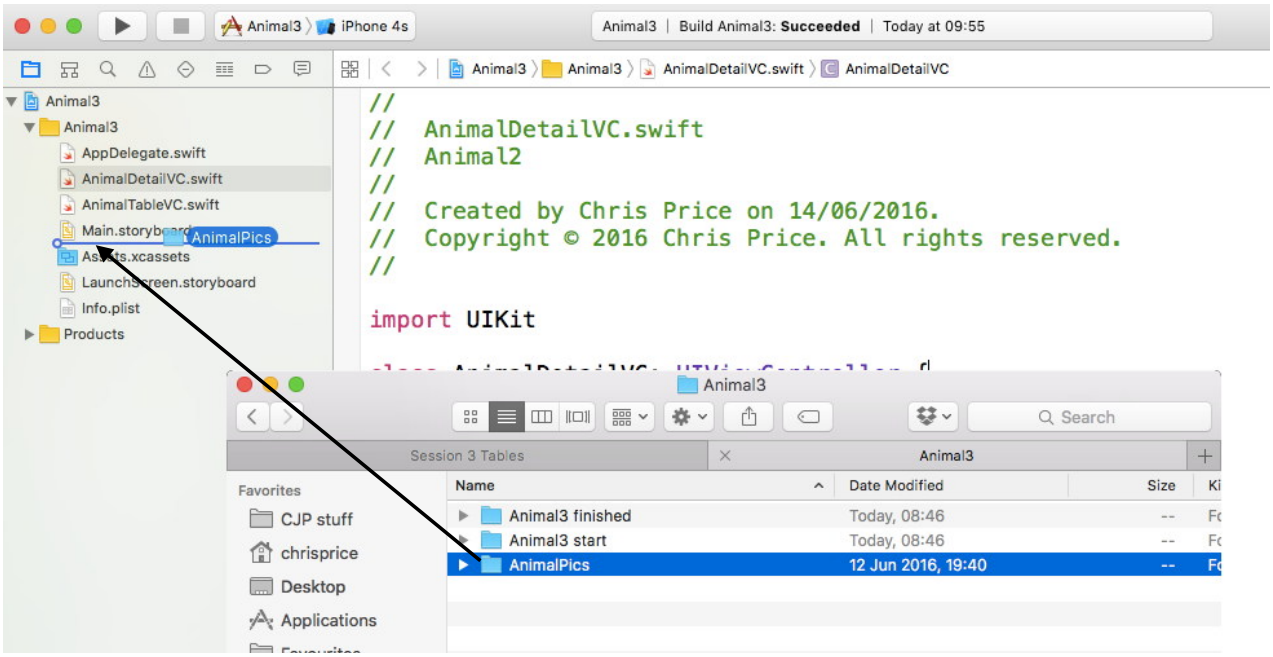



```
//
//  AnimalDetailVC.swift
//  Animal2
//
//  Created by Chris Price on
    14/06/2016.
//  Copyright © 2016 Chris Price.
    All rights reserved.
//

import UIKit

class AnimalDetailVC:
    UIViewController {

    @IBOutlet weak var favourite:
        UILabel!

    var valueForLabel = ""

    override func viewDidLoad() {
        super.viewDidLoad()
```


28

Step 5 - Add all the photos we need

•There is a folder of animal pictures in Animal3 called AnimalPics -drag the folder to the Assets in Xcode, and drop it in - the folder will then be included in the project



Step 6 - Change set up for screen

•Change viewDidLoad in AnimalDetailVC as follows (this will now load the correct photo in the Image View when the screen loads):

```
override func viewDidLoad() {
   super.viewDidLoad()
   self.title = valueForLabel
   photo.image = UIImage(named: "\(valueForLabel).jpg")
}
```

• Tidying up - also delete the now unused outlet below

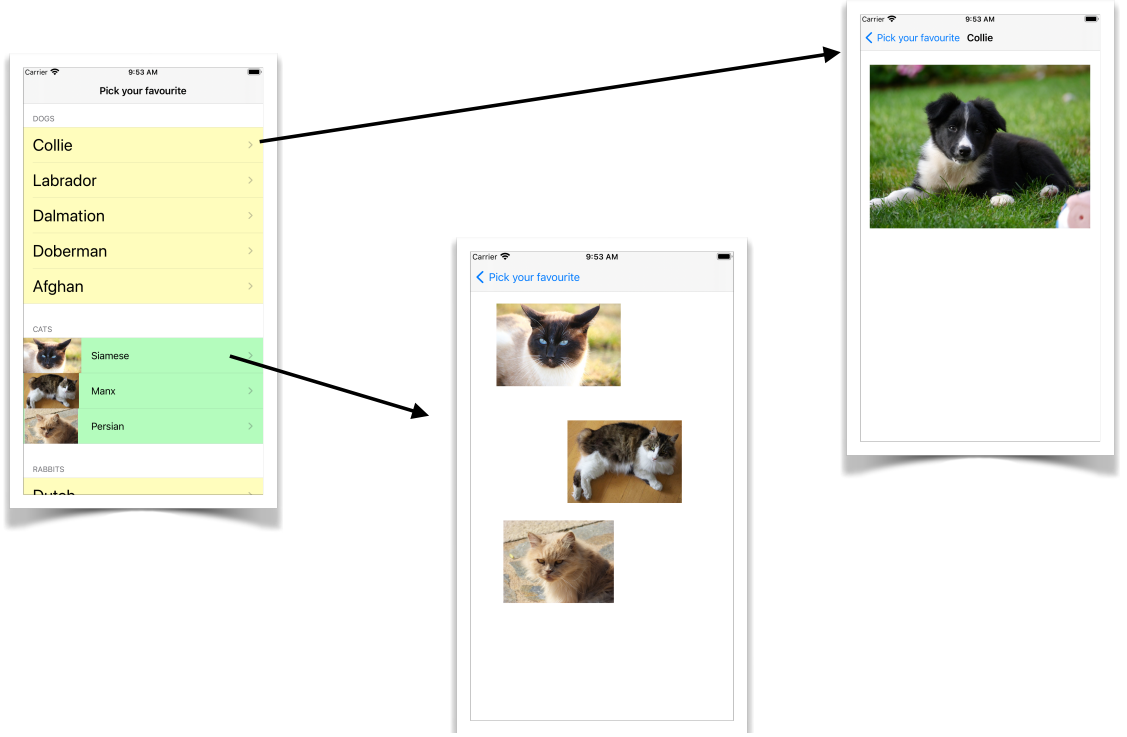
```
@IBOutlet weak var favourite: UILabel!
```

• Finally change the View Mode for the Image View from Scale To Fill to Aspect Fill

Finished!

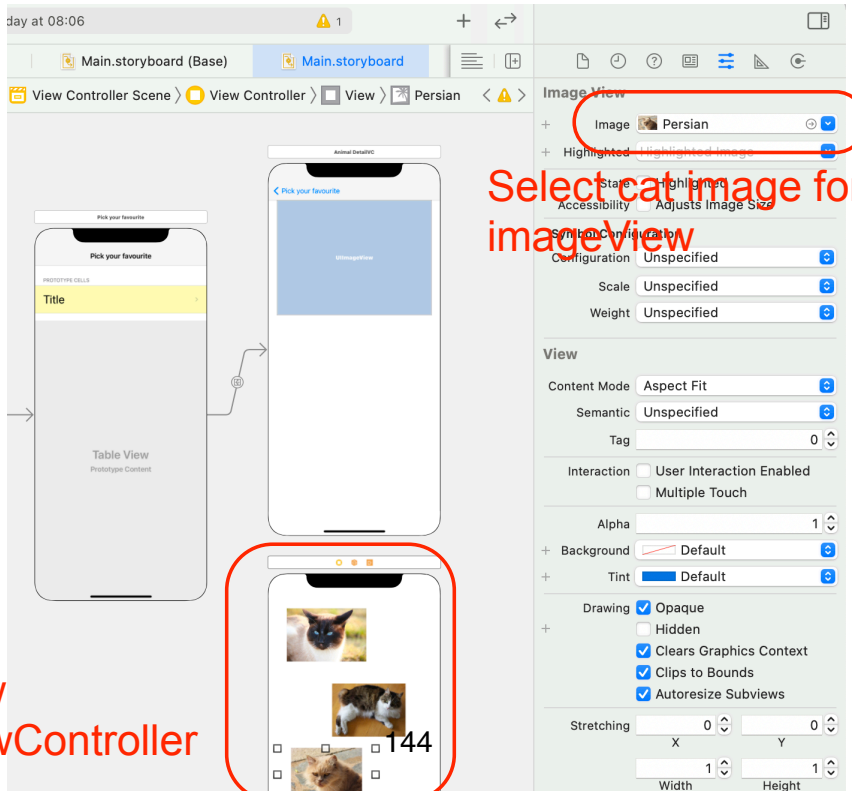
Fourth Animal App
Add custom cells and multiple destinations

Either use your completed Animal3 App, or use the provided completed Animal3 app if you hit problems.



Step 1 - Add a second destination

- Add another ViewController screen to the story board



Select cat image for each imageView

New ViewController

Step 2 - Make a second cell type

•Select the Tableview in the TableViewController in the storyboard
•Change the number of prototype cells from 1 to 2
•Make the reuse identifier for the second cell “PictureCell”
•Control drag from the new cell to the new ViewController and choose Show
•Colour the new cell green to be able to tell the difference

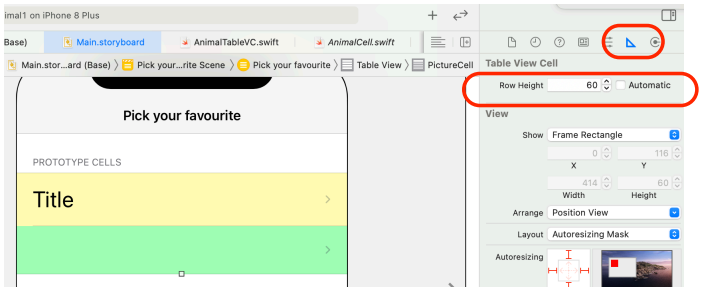
Step 3 - Make cats use new cell

•Select the Tableview in the TableViewController in the storyboard
•Change the number of prototype cells from 1 to 2
•Make the reuse identifier for the second cell “PictureCell”
•Control drag from the new cell to the new ViewController and choose Show
•Colour the new cell green to be able to tell the difference
•Update cellForRowAt indexPath as follows

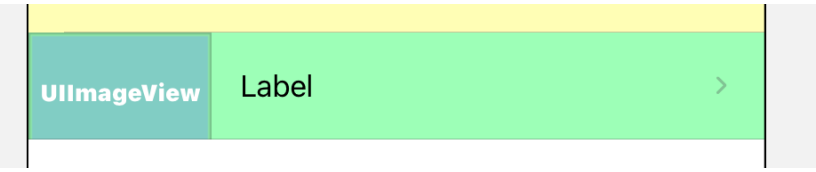
```
override func tableView(_ tableView: UITableView,
                        cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "AnimalCell",
                                             for: indexPath)
    // Configure the cell...
    cell.textLabel?.text = "Rabbit, section \(indexPath.section), row \(indexPath.row)"
    return cell
}
```

Step 4 - Make PictureCell a custom cell

•Select the PictureCell in the TableViewController screen in the storyboard
•Change its style to custom and it will go blank
•In the size inspector, make its height to be 60



•Add an image view and a label to the cell



35

Step 4 - Make code for custom PictureCell

•To have a custom Cell, you need to have some definitions and possibly code to go with it. You do this by creating a UITableCell class associated with the cell, and then making IBOutlets in that class for the custom labels and views

•Create a new file (like when you were making view controller code, but this time make it a UITableViewCell class, and call it PictureCell.swift.

•Associate it with the PictureCell table cell in the table by selecting the cell, and then clicking on the identity inspector, and associating the PictureCell class with the cell.

•Add IBOutlets for the Image View (photo) and the Label (animalName) in the table cell in PictureCell.swift

•We can now refer to these outlets in cellForRowAtIndexPath

36

Step 5 - Add code to initialise cells

•Make your cellForRowAtIndexPath look like this:

```
override func tableView(_ tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell {
    if indexPath.section == 1 { //Cats
        let cell = tableView.dequeueReusableCell(
            withIdentifier: "PictureCell", for: indexPath) as! PictureCell
        let wantedName =
            categoryItems[indexPath.section][indexPath.row]
        cell.photo.image = UIImage(named: "\(wantedName).jpg")
        cell.animalName.text = wantedName
        return cell
    } else {
        let cell = tableView.dequeueReusableCell(
            withIdentifier: "AnimalCell", for: indexPath)
        cell.textLabel!.text =
            categoryItems[indexPath.section][indexPath.row]
        return cell
    }
}
```

Finished!

What we've learned

- How to make dynamic tables
- How to link several views and pass data
- The different kinds of tables and cells that are available and how to use them
- Some of the wide variety of ways in which tables can be used and cells can be configured, but there are many more

Trying this for ourselves

- You are ready to start filling in some of the Conference app
- Folder Conference in this session has a set of files with data about the conference and how it can be used
- Try building the Speaker list and individual speaker page from our prototype a few days ago

39

Session 12: Protocols and closures

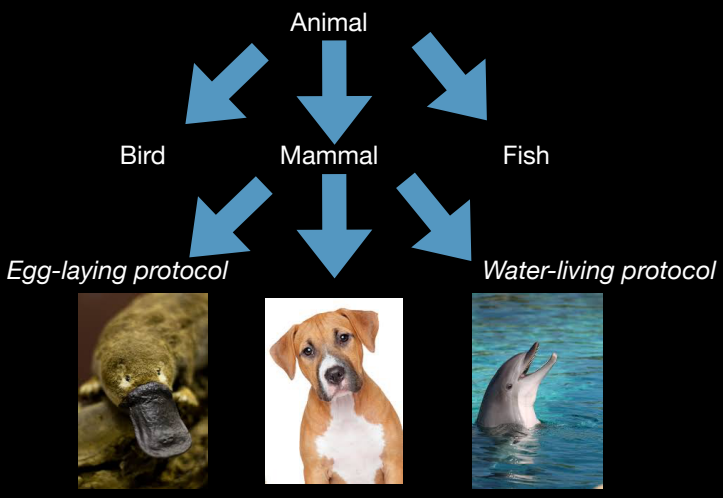
- Built in protocols – CustomStringConvertible, Equatable, Comparable, Codable
- Defining and implementing your own protocols
- Using closures
- Shortening closures
- Lab 12: Protocols and Closures
- This covers lessons 1.1 and 2.1 of Development in Swift Data Collections (the second Apple book)

1

Protocols

What are protocols for?

Shared behaviour that is not necessarily inherited
(And inheritance is not a choice for Structs)



Protocols

Define a blueprint of methods, properties, and other requirements that suit a particular task or piece of functionality

Swift standard library defines many protocols, including these:

`CustomStringConvertible`

`Equatable`

`Comparable`

`Codable`

When you adopt a protocol, you must implement all required methods.

Printing with CustomStringConvertible

```
let string = "Hello, world!"
print(string)

let number = 42
print(number)

let boolean = false
print(boolean)
```

```
Hello, world!
42
false
```

Printing with CustomStringConvertible

```
class Shoe {
  let color: String
  let size: Int
  let hasLaces: Bool

  init(color: String, size: Int, hasLaces: Bool) {
        self.color = color
        self.size = size
        self.hasLaces = hasLaces
    }
}

let myShoe = Shoe(color: "Black", size: 12, hasLaces: true)
print(myShoe)
```

```
__lldb_expr_1.Shoe
```

```
class Shoe: CustomStringConvertible {
    let color: String
    let size: Int
    let hasLaces: Bool

    init(color: String, size: Int, hasLaces: Bool) {
        self.color = color
        self.size = size
        self.hasLaces = hasLaces
    }









}
```

```
class Shoe: CustomStringConvertible {
    let color: String
    let size: Int
    let hasLaces: Bool

    init(color: String, size: Int, hasLaces: Bool) {
        self.color = color
        self.size = size
        self.hasLaces = hasLaces
    }

    var description: String {
        return "Shoe(color: \(color), size: \(size), hasLaces: \(hasLaces))"

    }
}
let myShoe = Shoe(color: "Black", size: 12, hasLaces: true)
print(myShoe)

Shoe(color: Black, size: 12, hasLaces: true)
```

Comparing information with Equatable

```
struct Employee {
  let firstName: String
  let lastName: String
  let jobTitle: String
  let phoneNumber: String
}

struct Company {
  let name: String
  let employees: [Employee]
}
```

Comparing information with Equatable

```
let currentEmployee = Session.currentEmployee
let selectedEmployee = Employee(firstName: "Jacob", lastName: "Edwards",
                                jobTitle: "Marketing Director", phoneNumber: "415-555-9293")

if currentEmployee == selectedEmployee {
  // Enable "Edit" button
}
```

Comparing information with Equatable

```
struct Employee: Equatable {
  let firstName: String
  let lastName: String
  let jobTitle: String
  let phoneNumber: String

  static func ==(lhs: Employee, rhs: Employee) -> Bool {
    // Logic that determines if the value on the left hand side and right hand side are equal
  }

}
```

Comparing information with Equatable

```
struct Employee: Equatable {
  let firstName: String
  let lastName: String
  let jobTitle: String
  let phoneNumber: String

  static func ==(lhs: Employee, rhs: Employee) -> Bool {
    return lhs.firstName == rhs.firstName && lhs.lastName == rhs.lastName
  }
}
```

Comparing information with Equatable

```
let currentEmployee = Employee(firstName: "Jacob", lastName: "Edwards",
    jobTitle: "Industrial Designer", phoneNumber: "415-555-7766")
let selectedEmployee = Employee(firstName: "Jacob", lastName: "Edwards",
    jobTitle: "Marketing Director", phoneNumber: "415-555-9293")

if currentEmployee == selectedEmployee {
  // Enable "Edit" button
}
```

Comparing information with Equatable

```
struct Employee: Equatable {
  let firstName: String
  let lastName: String
  let jobTitle: String
  let phoneNumber: String

  static func ==(lhs: Employee, rhs: Employee) -> Bool {
      return lhs.firstName == rhs.firstName && lhs.lastName == rhs.lastName
             && lhs.jobTitle == rhs.jobTitle && lhs.phoneNumber == rhs.phoneNumber
  }
}
```

Sorting information with Comparable

```
let employee1 = Employee(firstName: "Ben", lastName: "Atkins", jobTitle: "Front Desk",
phoneNumber: "415-555-7767")
let employee2 = Employee(firstName: "Vera", lastName: "Carr", jobTitle: "CEO", phoneNumber:
"415-555-7768")
let employee3 = Employee(firstName: "Grant", lastName: "Phelps", jobTitle: "Senior Manager",
phoneNumber: "415-555-7770")
let employee4 = Employee(firstName: "Sang", lastName: "Han", jobTitle: "Accountant",
phoneNumber: "415-555-7771")
let employee5 = Employee(firstName: "Daren", lastName: "Estrada", jobTitle: "Sales Lead",
phoneNumber: "415-555-7772")

let employees = [employee1, employee2, employee3, employee4, employee5]
```

```
struct Employee: Equatable, Comparable {
  let firstName: String
  let lastName: String
  let jobTitle: String
  let phoneNumber: String

  static func ==(lhs: Employee, rhs: Employee) -> Bool {
      return lhs.firstName == rhs.firstName && lhs.lastName == rhs.lastName
             && lhs.jobTitle == rhs.jobTitle && lhs.phoneNumber == rhs.phoneNumber
  }

  static func < (lhs: Employee, rhs: Employee) -> Bool {
    return lhs.lastName < rhs.lastName
  }
}
```

```
let employees = [employee1, employee2, employee3, employee4, employee5]

let sortedEmployees = employees.sorted(by:<)

for employee in sortedEmployees {
  print(employee)
}
```

```
Employee(firstName: "Ben", lastName: "Atkins", jobTitle: "Front Desk", phoneNumber: "415-555-7767")
Employee(firstName: "Vera", lastName: "Carr", jobTitle: "CEO", phoneNumber: "415-555-7768")
Employee(firstName: "Daren", lastName: "Estrada", jobTitle: "Sales Lead", phoneNumber: "415-555-7772")
Employee(firstName: "Sang", lastName: "Han", jobTitle: "Accountant", phoneNumber: "415-555-7771")
Employee(firstName: "Grant", lastName: "Phelps", jobTitle: "Senior Manager", phoneNumber: "415-555-7770")
```

```
let employees = [employee1, employee2, employee3, employee4, employee5]

let sortedEmployees = employees.sorted(by:>)

for employee in sortedEmployees {
  print(employee)
}
```

```
Employee(firstName: "Grant", lastName: "Phelps", jobTitle: "Senior Manager", phoneNumber: "415-555-7770")
Employee(firstName: "Sang", lastName: "Han", jobTitle: "Accountant", phoneNumber: "415-555-7771")
Employee(firstName: "Daren", lastName: "Estrada", jobTitle: "Sales Lead", phoneNumber: "415-555-7772")
Employee(firstName: "Vera", lastName: "Carr", jobTitle: "CEO", phoneNumber: "415-555-7768")
Employee(firstName: "Ben", lastName: "Atkins", jobTitle: "Front Desk", phoneNumber: "415-555-7767")
```

Encoding and decoding objects with Codable

```
struct Employee: Equatable, Comparable, Codable {
    var firstName: String
    var lastName: String
    var jobTitle: String
    var phoneNumber: String

    static func ==(lhs: Employee, rhs: Employee) -> Bool {
        return lhs.firstName == rhs.firstName && lhs.lastName ==
            rhs.lastName && lhs.jobTitle == rhs.jobTitle &&
            lhs.phoneNumber == rhs.phoneNumber
    }

    static func < (lhs: Employee, rhs: Employee) -> Bool {
        return lhs.lastName < rhs.lastName
    }
}
```

Encoding and decoding objects with Codable

```
let ben = Employee(firstName: "Ben", lastName: "Atkins", jobTitle: "Front Desk",
                   phoneNumber: "415-555-7767")

let jsonEncoder = JSONEncoder()
if let jsonData = try? jsonEncoder.encode(ben),
    let jsonString = String(data: jsonData, encoding: .utf8) {
    print(jsonString)
}
```

```
{"firstName":"Ben","lastName":"Atkins","jobTitle":"Front Desk","phoneNumber":"415-555-7767"}
```

Creating a protocol

```
protocol FullyNamed {
  var fullName: String { get }

  func sayFullName()
}

struct Person: FullyNamed {
  var firstName: String
  var lastName: String
}
```

Creating a protocol

```
struct Person: FullyNamed {
  var firstName: String
  var lastName: String

  var fullName: String {
    return "\(firstName) \(lastName)"
  }

  func sayFullName() {
    print(fullName)
  }
}
```

Lab: Protocols

Open and complete the first page of exercises in `Lab - Protocols.playground`

Closures

Closures

```
(firstTrack: Track, secondTrack: Track) -> Bool in
 return firstTrack.trackNumber < secondTrack.trackNumber
```

```
let sortedTracks = tracks.sorted (   )
```

Syntax

```
func sum(numbers: [Int]) -> Int {
  // Code that adds together the numbers array
  return total
}
```

```
let sumClosure = { (numbers: [Int]) -> Int in
  // Code that adds together the numbers array
  return total
}
```

```
// A closure with no parameters and no return value
let printClosure = { () -> Void in
  print("This closure does not take any parameters and does not return a value.")
}

// A closure with parameters and no return value
let printClosure = { (string: String) -> Void in
  print(string)
}

// A closure with no parameters and a return value
let randomNumberClosure = { () -> Int in
  // Code that returns a random number
}

// A closure with parameters and a return value
let randomNumberClosure = { (minValue: Int, maxValue: Int) -> Int in
  // Code that returns a random number between `minValue` and `maxValue`
}
```

Passing closures as arguments

```
let sortedTracks = tracks.sorted { (firstTrack: Track, secondTrack: Track) -> Bool in
  return firstTrack.trackNumber < secondTrack.trackNumber
}
```

```
 let sortedTracks = tracks.sorted { (firstTrack: Track, secondTrack: Track) -> Bool in
   return firstTrack.starRating < secondTrack.starRating
 }
```

Syntactic sugar

```
let sortedTracks = tracks.sorted { (firstTrack: Track, secondTrack: Track) -> Bool in
  return firstTrack.starRating < secondTrack.starRating
}
```

Syntactic sugar

```
let sortedTracks = tracks.sorted { (firstTrack, secondTrack) -> Bool in
  return firstTrack.starRating < secondTrack.starRating
}
```

Syntactic sugar

```
let sortedTracks = tracks.sorted { (firstTrack, secondTrack) in
  return firstTrack.starRating < secondTrack.starRating
}
```

Syntactic sugar

```
let sortedTracks = tracks.sorted { return $0.starRating < $1.starRating }
```

Syntactic sugar

```
let sortedTracks = tracks.sorted { $0.starRating < $1.starRating }
```

Syntactic sugar

```
let sortedTracks = tracks.sorted(by: <)
```

Collection functions using closures

Map

Filter

Reduce

Collection functions using closures

```
// Initial array
let firstNames = ["Johnny", "Nellie", "Aaron", "Rachel"]

// Creates an empty array that will be used
// to store the full names
var fullNames: [String] = []

for name in firstNames {
    let fullName = name + " Smith"
    fullNames.append(fullName)
}
```

fullNames

0	"Johnny Smith"
1	"Nellie Smith"
2	"Aaron Smith"
3	"Rachel Smith"

Collection functions using closures

```
// Initial array
let firstNames = ["Johnny", "Nellie", "Aaron", "Rachel"]

// Creates a new array of full names by adding "Smith"
// to each first name
let fullNames = firstNames.map { (name) -> String in
    return name + " Smith"
}
```

fullNames

0	"Johnny Smith"
1	"Nellie Smith"
2	"Aaron Smith"
3	"Rachel Smith"

Collection functions using closures

```
// Initial array
let firstNames = ["Johnny", "Nellie", "Aaron", "Rachel"]

// Creates a new array of full names by adding "Smith"
// to each first name
let fullNames = firstNames.map { $0 + " Smith" }
```

fullNames

0	"Johnny Smith"
1	"Nellie Smith"
2	"Aaron Smith"
3	"Rachel Smith"

Collection functions using closures

```
let numbers = [4, 8, 15, 16, 23, 42]

var numbersLessThan20: [Int] = []

for number in numbers {
    if number < 20 {
        numbersLessThan20.append(number)
    }
}
```

numbersLessThan20

0	4
1	8
2	15
3	16

Collection functions using closures

```
let numbers = [4, 8, 15, 16, 23, 42]

let numbersLessThan20 = numbers.filter { (number) -> Bool in

    return number < 20
}
```

numbersLessThan20

0	4
1	8
2	15
3	16

Collection functions using closures

```
let numbers = [4, 8, 15, 16, 23, 42]

let numbersLessThan20 = numbers.filter { $0 < 20 }
```

numbersLessThan20

0	4
1	8
2	15
3	16

Collection functions using closures

```
let numbers = [8, 6, 7, 5, 3, 0, 9]

var total = 0

for number in numbers {
    total = total + number
}
```

Collection functions using closures

```
let numbers = [8, 6, 7, 5, 3, 0, 9]

let total = numbers.reduce(0) { (currentTotal, newValue) -> Int in
    return currentTotal + newValue
}
```

Collection functions using closures

```
let numbers = [8, 6, 7, 5, 3, 0, 9]

let total = numbers.reduce(0, {$0 + $1})
```

Collection functions using closures

```
let numbers = [8, 6, 7, 5, 3, 0, 9]

let total = numbers.reduce(0, +)
```

Closures capture their environment

```
animate {
  self.view.backgroundColor = .red
}
```

Lab: Closures

Open and complete the exercises in Lab - Closures.playground

Session 13: Mapping and Location

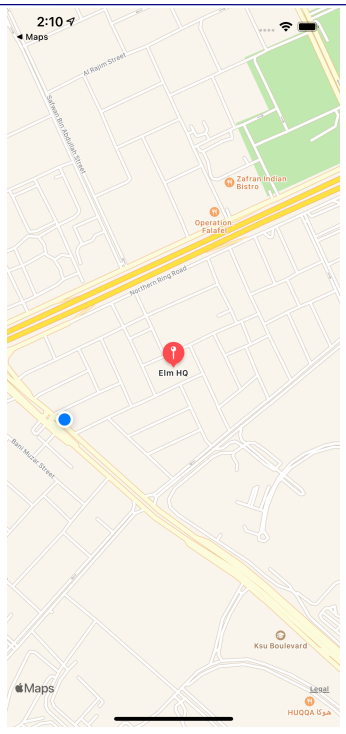
- Setting up location services
- Tracking movement
- Annotating maps
- Lab 13: Adding Maps to an app

- This material is not covered in the Apple Books

1

What we are going to build

- A mapping screen which shows landmarks plus our own position



2

Step 1

•Start a new simple app
•Add a map view to the screen - constrain it to take up the whole window
•Make an outlet for the map View in the associated file (ViewController.swift) and call it *mapView*

The program will fail to compile at this time, as it does not know what an MKMapView is - you can fix this by adding:

Import MapKit

at the top of the screen

If you run it at this point, your app will show a map centred on the country - we need to make it more focused on the place we want to show

Step 2

•We need to define a class that supports the MKAnnotation protocol - this has the right info to define a labelled point on the map

```
class MapNode: NSObject, MKAnnotation {
    let coordinate: CLLocationCoordinate2D
    let title: String?
    let subtitle: String?

    init(latitude: CLLocationDegrees, longitude: CLLocationDegrees, title: String,
subtitle: String) {
        coordinate = CLLocationCoordinate2D(latitude: latitude, longitude: longitude)
        self.title = title
        self.subtitle = subtitle
    }
}
```

Step 3

•In viewDidLoad, we will define an annotation for Elm HQ

```
let elmHQ = MapNode(latitude: 24.747842, longitude: 46.623180, title: "Elm HQ",
subtitle: "Headquarters of Elm Information Security Company")
```

•Also in viewDidLoad, we will set the map to centre on ElmHQ, and add the annotation to the map

```
//Set a coordinate region with ElmHQ as centre and a kilometer span
let regionSpan: CLLocationDistance = 1000
let coordinateRegion = MKCoordinateRegion( center: elmHQ.coordinate,
                    latitudinalMeters: regionSpan,
                    longitudinalMeters: regionSpan)
mapView.setRegion(coordinateRegion, animated: true)
mapView.addAnnotation(elmHQ)
```

•If we run the app now, our annotation is on the map - now we want to know where WE are on the map

5

We want to show OUR location

- Four things needed for that
  - The view controller needs to conform to CLLocationManagerDelegate
  - You need a CLLocationManager set up
  - You need to tell it to show the user location on the map
  - You need to set it so the app asks for permission to monitor user position

6

Step 4

- At the start of the ViewController class, state it uses the CLLocationManagerDelegate, and declare a Manager

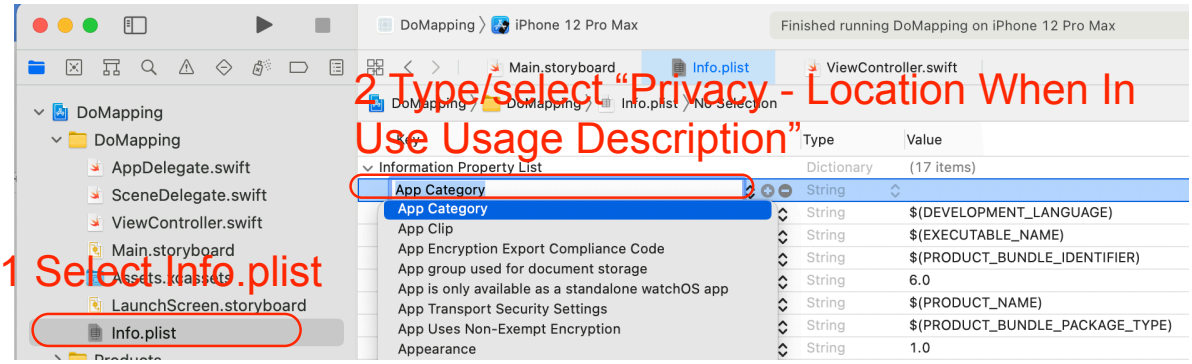
```
class ViewController: UIViewController, CLLocationManagerDelegate {

    let locationManager = CLLocationManager()
```

- At the end of viewDidLoad, add code to ask for permission to use the user location, and give it a reasonable accuracy, then show user location

```
locationManager.requestWhenInUseAuthorization()
locationManager.distanceFilter = kCLDistanceFilterNone
locationManager.desiredAccuracy = kCLLocationAccuracyNearestTenMeters
locationManager.startUpdatingLocation()
mapView.showsUserLocation = true
```


Step 5

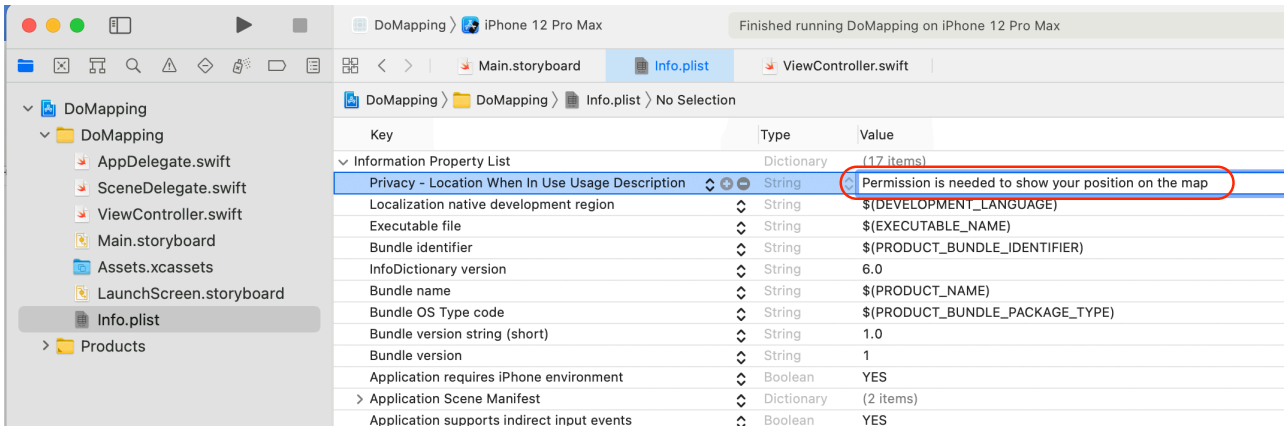
- We now have all the code we need, but when you are doing something you need permission for (using Location, accessing the camera, sending Notifications), you also need to signal that permission in the app's info.plist file



- Click on the + next to Information Property List, and you will get an extra line as shown above. Start typing "Privacy - Location When In Use Usage Description" and it should appear and can be selected

Step 6

•Add a value for the Privacy prompt - this will be used to explain to the user why you want access



Step 7

•The app should now run fine. It will prompt for our position, and on a real device, it will show as a blue dot
•On the simulator, we need to provide a simulated location for where we are on the map
•Lets say we are at the Doka Bakery a few streets away

•In the Simulator, choose Features/Location/ Custom Location
•Put in values 24.746383, 46.619839

Now we should show on the map as a blue dot

Applying to case study

- You should now be able to add the map screen to the Conference app - either to show a specific selected location or to have a map that shows all the locations (you can add a list of landmarks instead of just one).

Session 14: Gestures and accelerometer

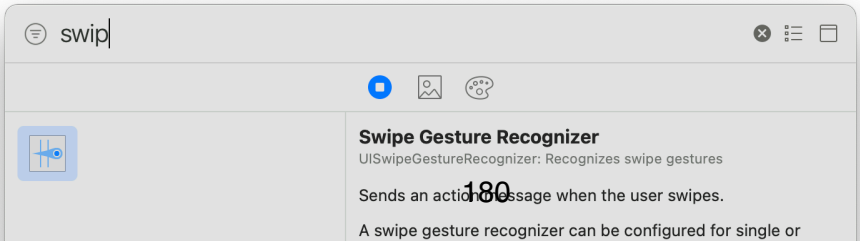
- Implementing taps and swipes
- Using the accelerometer
- Lab 14: Example apps using taps, swipes and the accelerometer

- This material is not well covered in the Apple Books - there is a gesture example on page 280-281 of Develop in Swift Fundamentals

1

Implementing swipes and taps

- **Step 1**
- Add a label to the screen for the ViewController and give it the value zero.
- Add a swipe gesture recognizer from the library to the screen on the main storyboard.



2

Implementing swipes and taps

- **Step 2**
- Make label outlet called *counter*
- Make swipe action outlet for swipe gesture called rightSwipe
- Add second Swipe gesture recognizer to the main screen, and set its attribute to be a left swiper, and add a swipe action outlet called leftSwipe

3

Implementing swipes and taps

- **Step 3**
- Keep a count of the value of counter, and when there is a right swipe increase it if it is less than 9
- When there is a left swipe, decrease it if it is greater than 0
- Update the counter to reflect value change

4