iOS Foundations II
Day 2

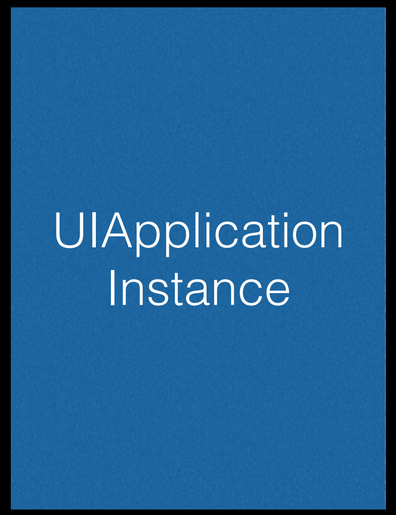
- Homework Review
- The story of an App
- View Controllers
- Design Patterns/MVC
- IBOutlet & IBAction
- Arrays

Homework review

The story of an app

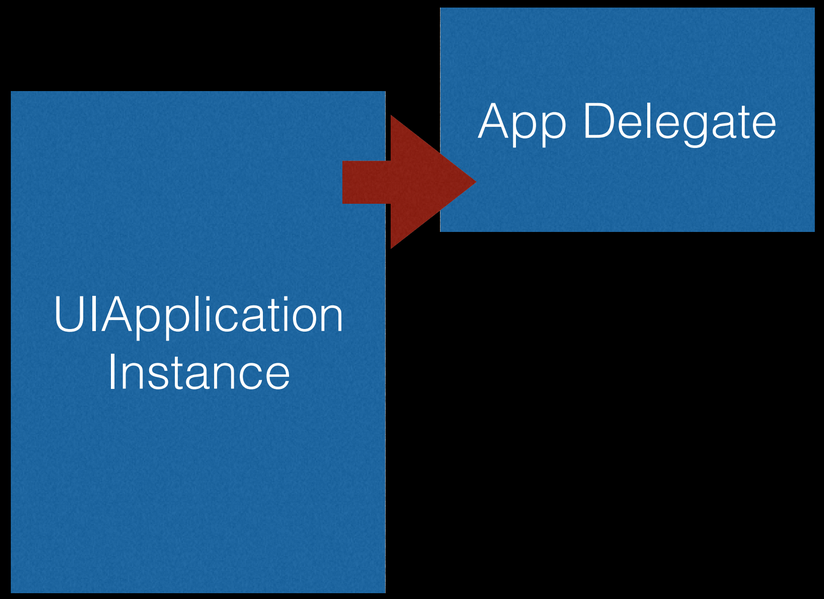
Step 1

- When your app is first launched, either from the home menu screen of the device or by hitting Play in Xcode:



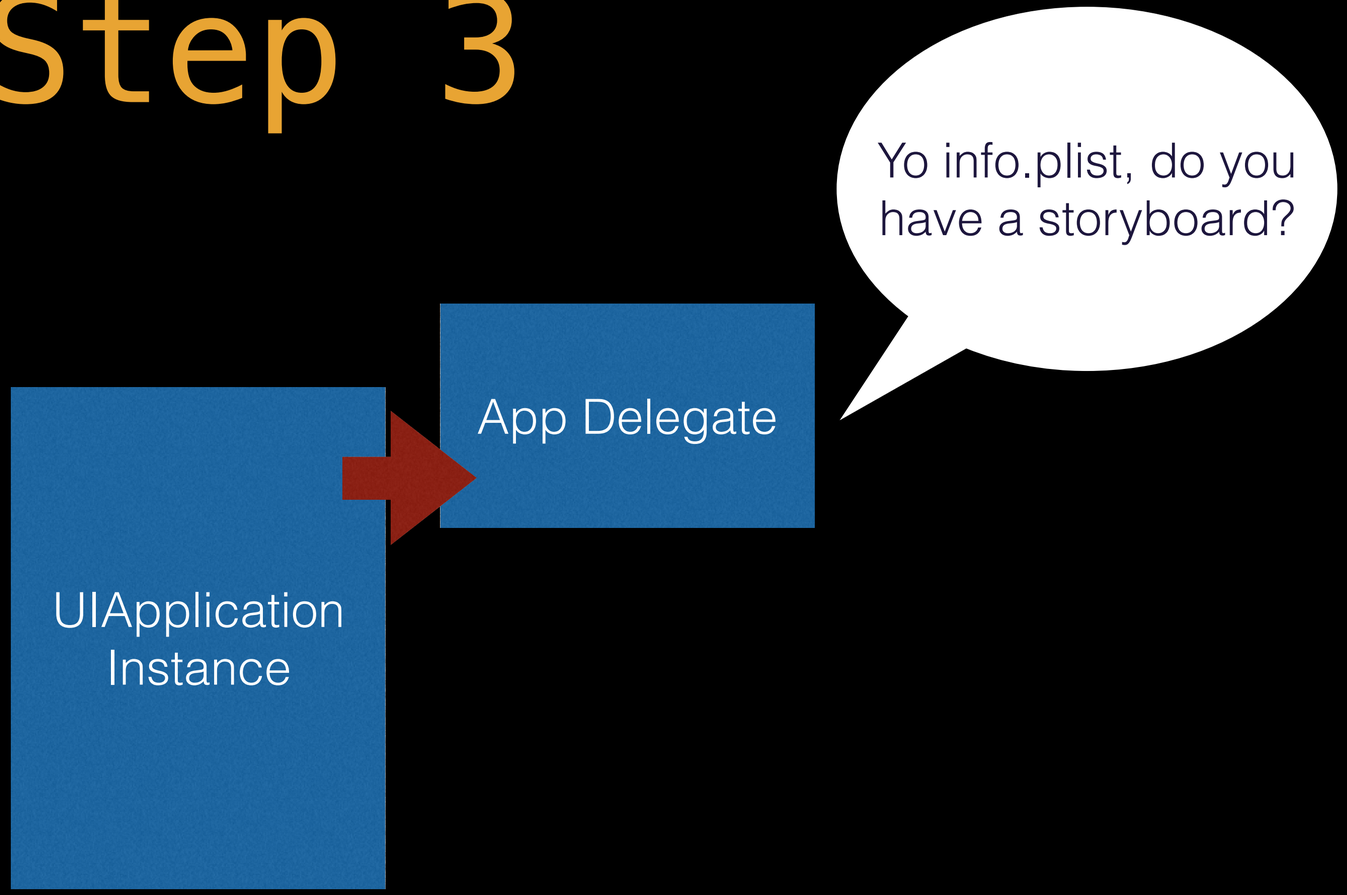
An instance of the UIApplication class is created, representing your app

Step 2



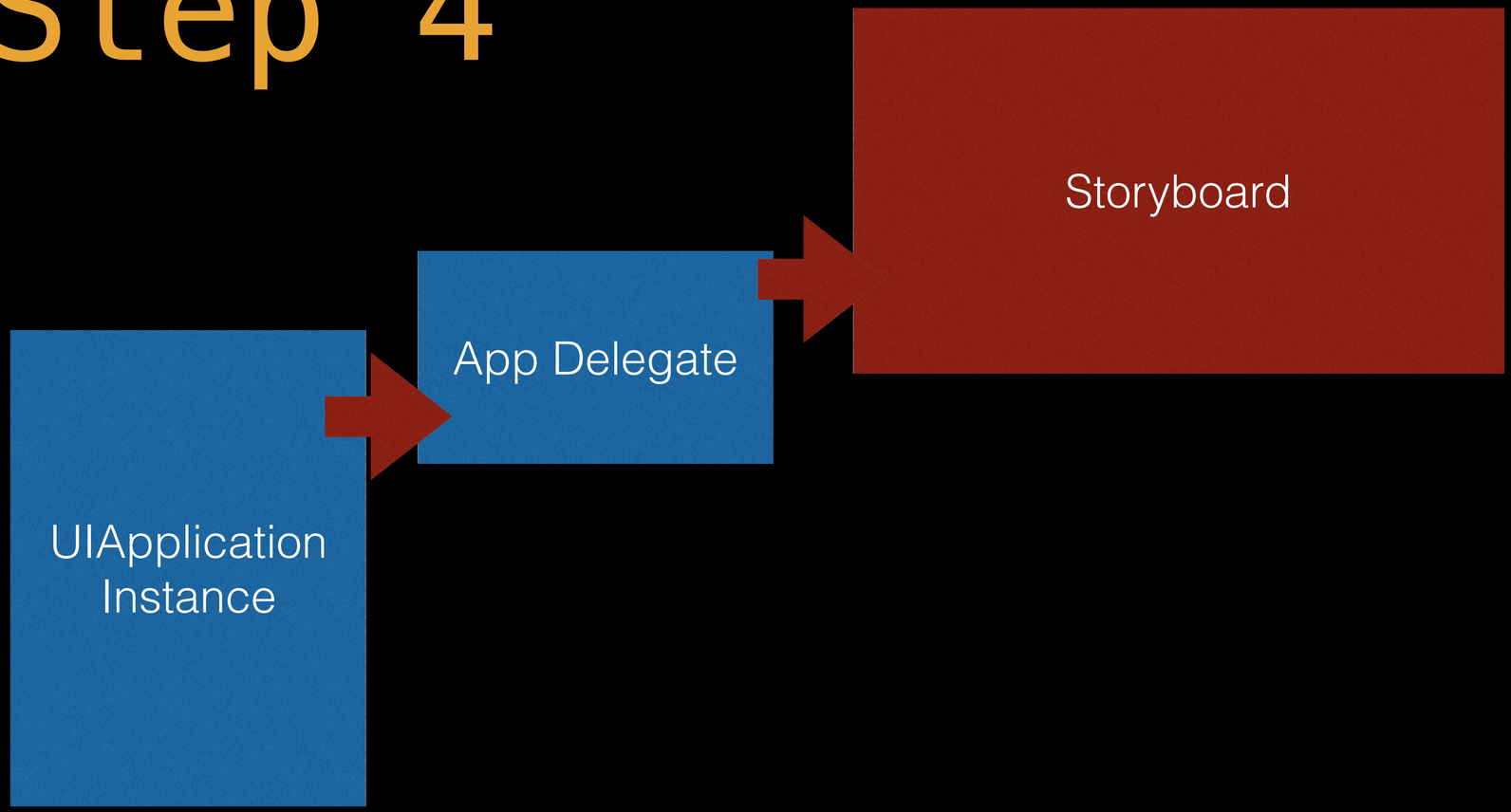
An instance of the UIApplicationDelegate class is created by the system, and connected to the UIApplication Instance

Step 3



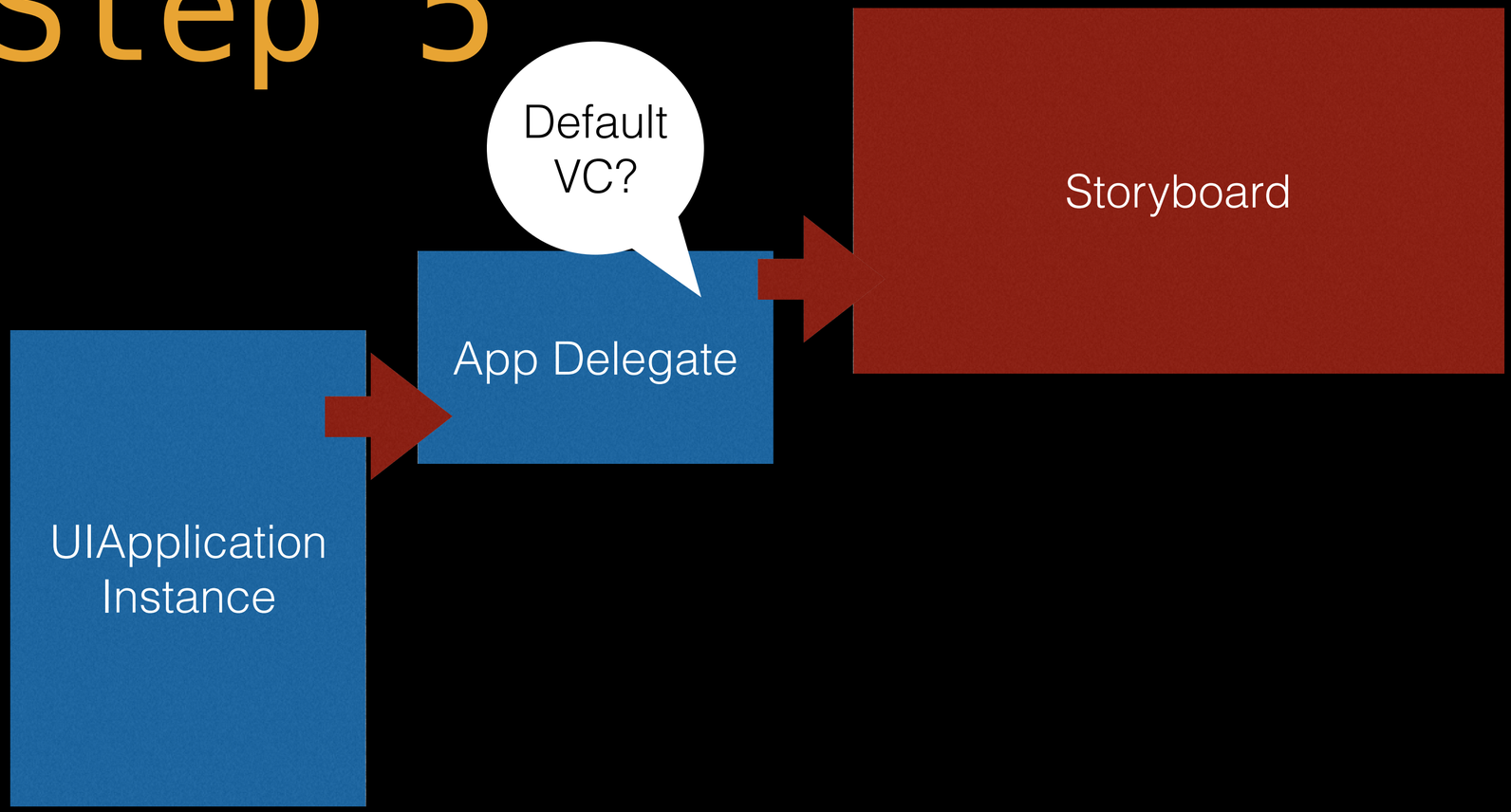
App Delegate looks at your app's project settings (info.plist), and sees if you have a storyboard or not

Step 4



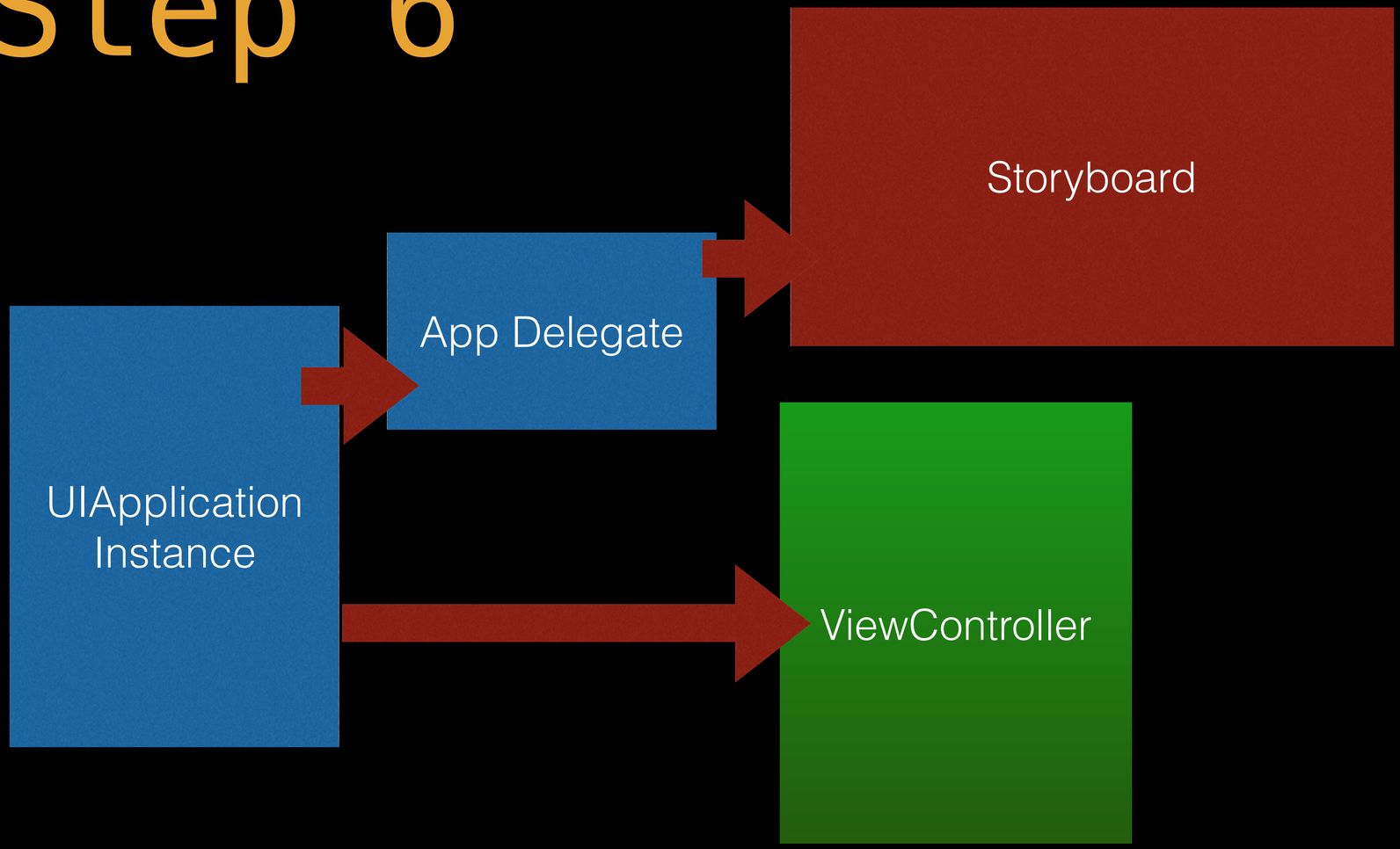
The answer was yes, so an instance of UIStoryboard is created from the storyboard you setup in Xcode

Step 5



App delegate asks Storyboard, do you have an initial view controller? If so, what is it?

Step 6



Storyboard finds the view controller/scene you marked as the initial view controller, instantiates an instance of it WITH ALL THE COOL STUFF YOU ADDED VIA STORYBOARD ADDED TO IT.

Moral of the story

- So the moral of the story is….your app is just a bunch of objects.
- And objects are just instances of classes.
- So to make an app, you need to not only make your own custom classes, but also use classes provided by Apple (and maybe even other third parties)
- But how or why did viewDidLoad get called in our view controller, which set our background color to red???
- Lets talk about View controllers.

ViewControllers

- Basic Visual Component of iOS.
- For the most part, every ‘screen’ in an app is a separate view controller, although it is possible to have more than one ViewController on the screen at once.
- iOS provides many built-in ViewControllers to support standard interfaces.
- The most important property of a ViewController is it’s view property. Every ViewController has a root view so it can display your interface objects. You can access this view inside of a view controller with self.view, just like you access any other property.

ViewControllers

- Really only 4 ways to get a ViewController's view on screen:
    1. Make the ViewController a window's rootViewController (Use that arrow in storyboard to denote the rootViewController!)
    2. Make the ViewController a child of a container View Controller
    3. Show the ViewController in a popover
    4. Present it from another View Controller (The most common scenario)

ViewController Life Cycle

- **The concept of ViewController's lifecycle is important to understand. So important I bolded this.**
- There are 4 methods to know:
    1. ViewDidLoad - Called when the ViewController's view is loaded into memory
    2. ViewWillAppear - Called before the ViewController's view appears onscreen
    3. ViewDidAppear - Called immediately after the ViewController's view has appeared onscreen
    4. ViewWillDisappear - Called when the ViewController's view is about to be removed from the view hierarchy
    5. ViewDidDisappear - Called when the ViewController's view is now removed from the view hierarchy.

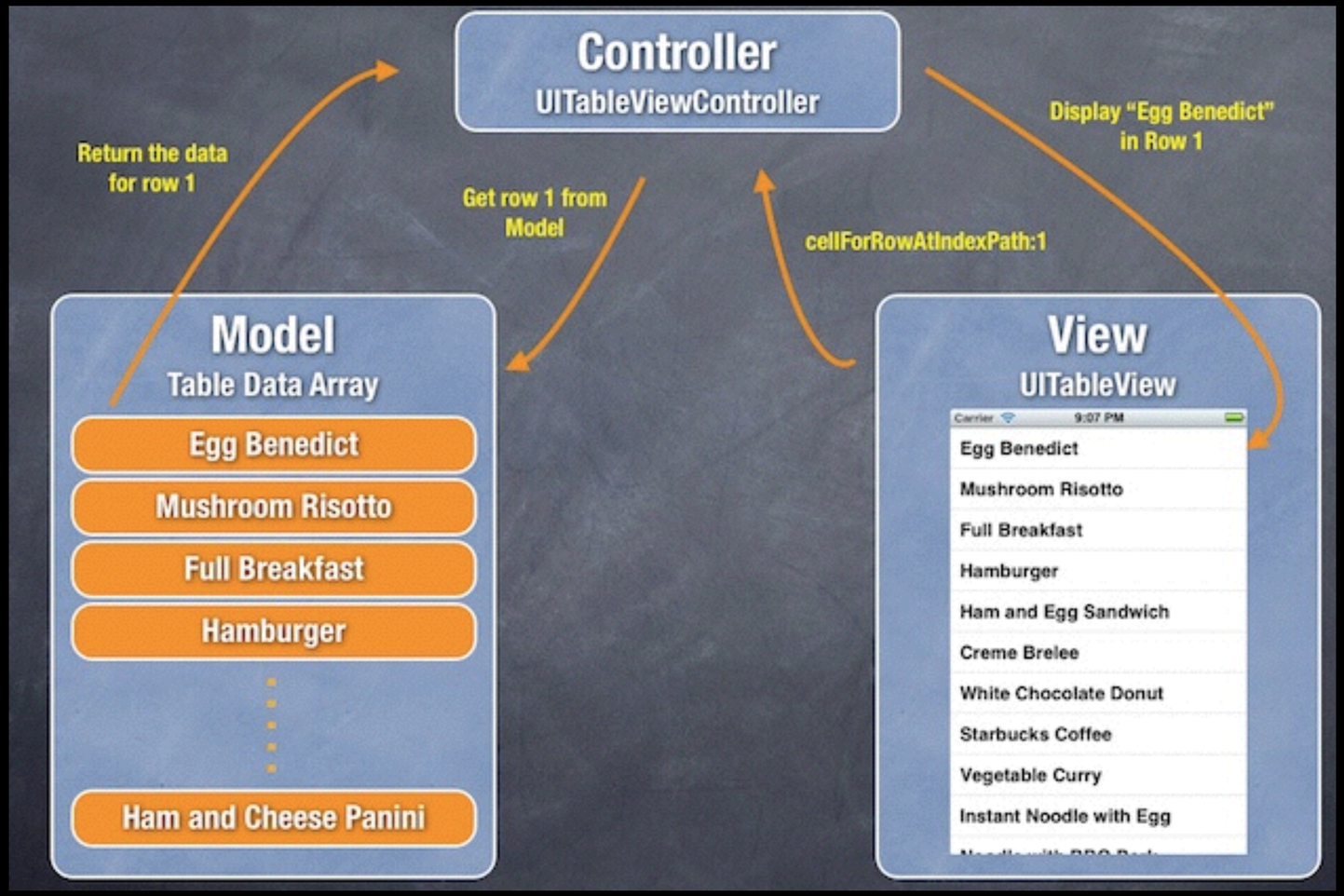
Demo

iOS Design Patterns

- “A software design pattern is a general reusable solution to commonly occurring problems within a given context in software design”
- No matter what kind of app it is, there are a few fundamental design pattern techniques that all apps use.
- The most important design patterns to know for iOS development:
  - **Model View Controller - MVC - Governs the overall structure of your app.**
  - Delegation - Facilitates the transfer of data and responsibilities from one object to another.
  - Target-Action - Translates user interactions into code your app can execute.
  - Closures/Blocks - Use these for callbacks and asynchronous code.

Design Patterns Are Cool

- A benefit of the universal usage of design patterns is commonality between all apps besides just the language they are programmed in.
- If someone is showing you the source code of their app, you can ask things like “What kind of model classes are you using?” or “How many View Controllers do you have?” and sound like you know what you are talking about.



MVC
(Model-View-Controller)

MVC Facts

- Introduced in the 70's with the Smalltalk programming language.
- Didn't become a popular concept until the late 80's
- The MVC pattern has spawned many evolutions of itself, like MVVM (Model-View-ViewModel)
- MVC is very popular with web design and applications. It's not just for mobile or desktop.

So what is MVC?

- MVC is simply the separation of **M**odel, **V**iew, and **C**ontroller.
- It is a separation of concerns for your code. Being able to separate out these components makes your code easier to read, re-use, test, think about, and discuss.
- The **Model layer** is the data of your app, the **View layer** is anything the user sees and interacts with, and the **Controller layer** mediates between the two.
- Examples of bad MVC practices: Your model classes directly communicating with your views, your view classes making network calls, etc

The MVC layout of our App

Model Layer

Person

Controller Layer

PersonTable ViewController

PersonDetail ViewController

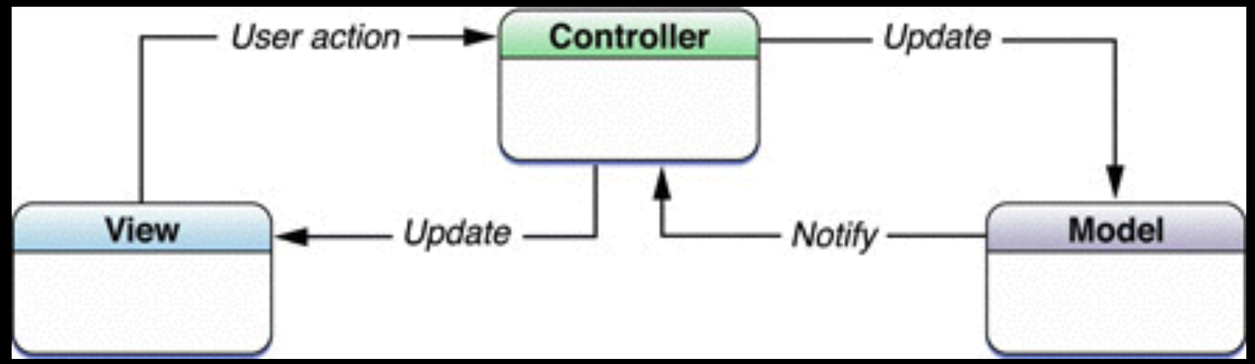
View Layer

Storyboard

TableView

PersonCell

MVC or MCV LOL?



Some people joke its more like MCV, because the controller is the middle man so the C should go in the middle*

*Classic programming joke

Model Layer

- Model objects encapsulate data and logic that are going to be used by your application.
- The Twitter App has a Tweet model class, a User model class, a Favorite model class, etc.

View Layer

- A View object is an object the user can see and possibly interact with.
- Enables displaying and editing of the app's data.
- Communication between the View and Model layers is made possible by…..

Controller Layer

- Act as the intermediary between the model layer and view layer.
- The most common form of a controller in iOS is a view controller.
- Another common controller is a network controller.
- At first your view controllers will have a lot of code. Eventually you should strive to make them lighter so its easier to understand what they are doing at a glance.

Demo

IBOutlets & IBActions

IBAction

- InterfaceBuilderAction, or IBAction for short, are special methods triggered by a user interface object that was laid on storyboard.

```
18  @IBAction func buttonPressed(sender: AnyObject) {
19
20    //do something cool
21
22  }
```

- Multiple interface objects can be hooked up to the same IBAction method
- Whenever an IBAction has a parameter, like you see above, it is simply the interface object that triggered this action. Best practice is to name this parameter ‘sender’.
- If you have multiple buttons hooked up to the same IBAction, you can inspect sender to see which button actually triggered the action.

Demo

IBOutlet

- InterfaceBuilderOutlet, or IBOutlet for short, is a special type of property that creates a link between your code and your interface objects on your storyboard.
- So if you drag a UIImageView or UIButton onto your storyboard, you can create an outlet for these interface objects:

```
13    @IBOutlet weak var imageView: UIImageView!
14    @IBOutlet weak var button: UIButton!
15
```

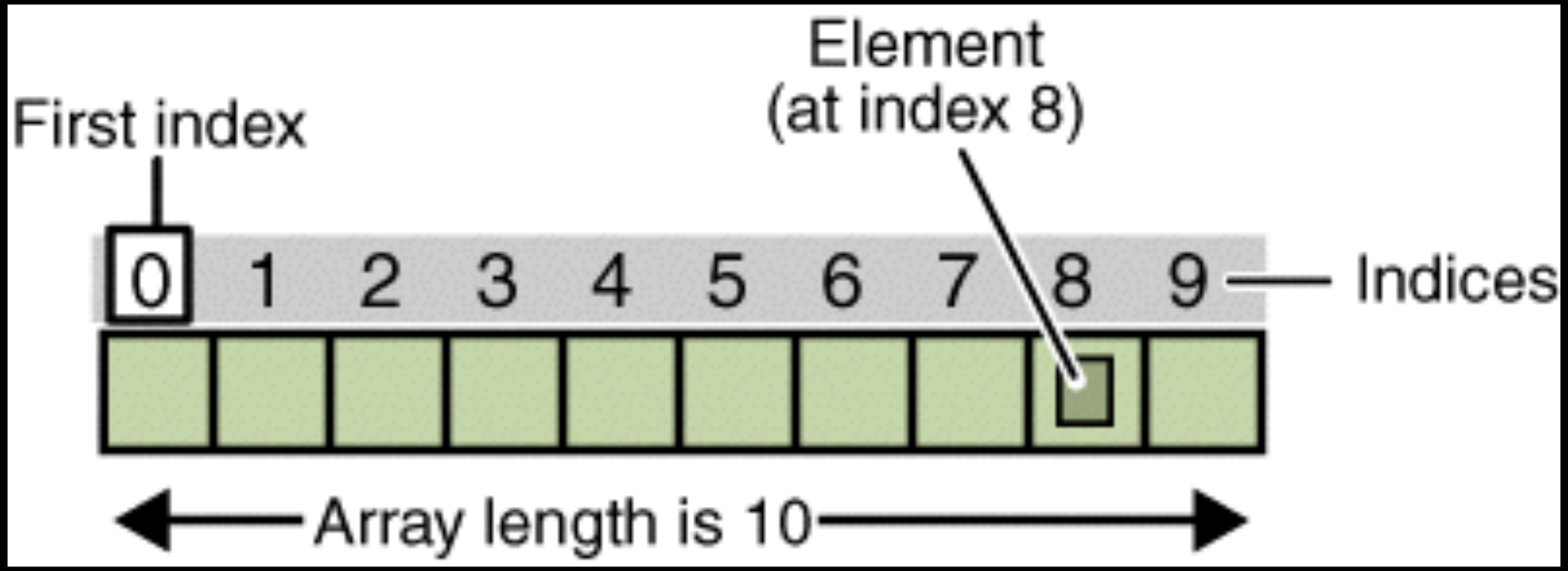
- This allows you to access them in code just like any other regular property.
- Ignore the ‘weak’ keyword for now, memory management will come later.

Demo

Arrays

- Arrays are very important to understand
- Arrays are used in virtually ever app ever made!!!!
- You typically use arrays any time you have a collection of similar objects or data and you want to perform similar operations on them.
- So its considered a collection type.
- Arrays are ordered, which is important.

Arrays


Element
(at index 8)
First index
0
1
2
3
4
5
6
7
8
9
Indices
Array length is 10

Creating an Array in Swift

- An array is considered a type, and the way to signify an array type in Xcode is just []
- But thats not quite complete, because inside the brackets you need to also state the type of objects you are going to be putting inside the array.
- So [String] is the type of an array that holds Strings. and [UIImage] is the type of an array that holds images.
- To actually instantiate an array, you use () after the closing square bracket to create the array:

```
var myNames = [String]()
```

Demo

Adding things to an array

- There are two ways an object can get inside an array in Swift:
  - Arrays have a method called append, which takes in one parameter, the object you want to add to the end of the array:
  - When you initial create an array, you can use a special shorthand syntax where you place all the objects in the brackets of the array you are creating:

```
let names = ["Brad","David","Ryan"]
```

Demo

Retrieving objects from an array

- Retrieving objects from an array uses a special syntax that also involves []
- You retrieve objects from arrays by their index number.
- Remember the index starts at 0, not 1!

```
let names = ["Brad","David","Ryan"]

let brad = names[0]
let david = names[1]
let ryan = names[2]
```