EE 443 / EE 593

Mobile App Development

Anthony Gallante

Lecture 13

Chapter 16: Lists

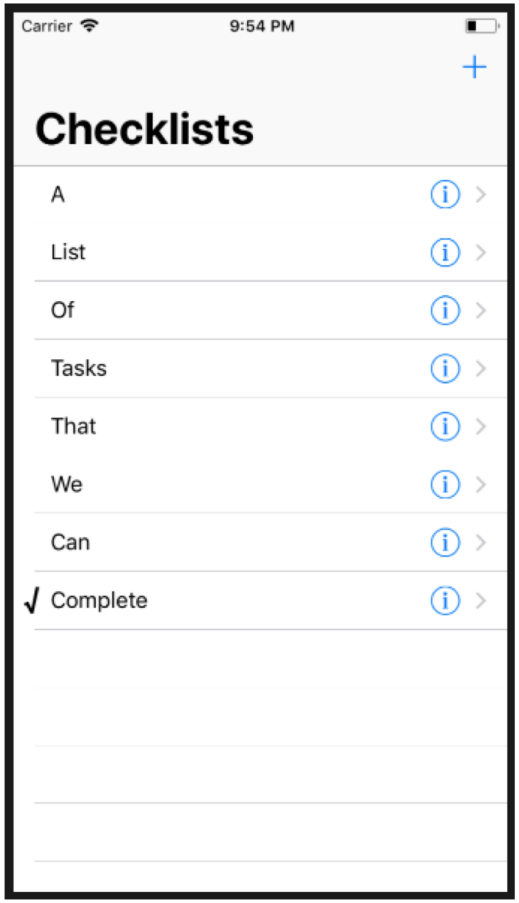
Anthony Gallante

Outline

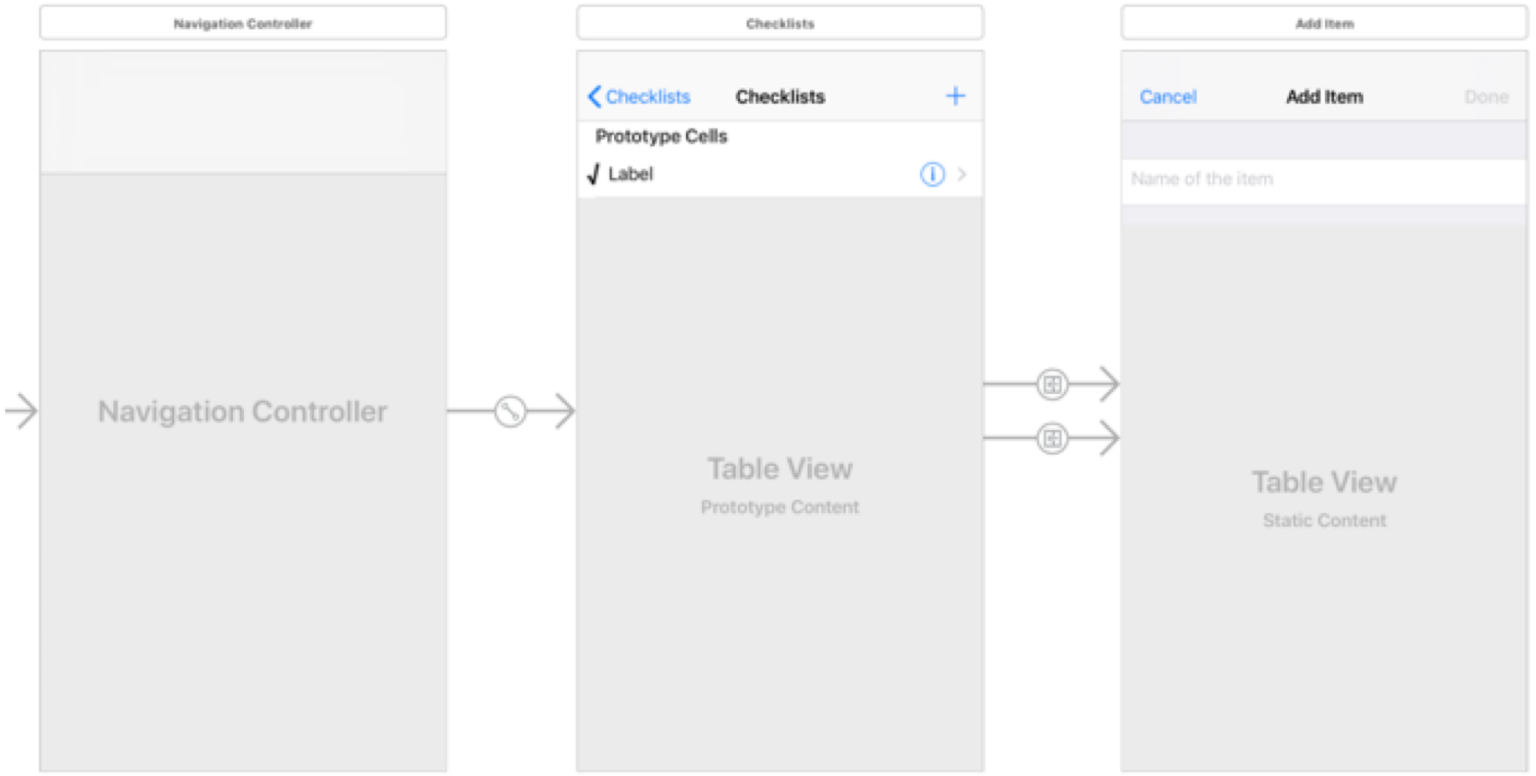
- Chapter 16 Overview
- Programming To-Do List
- Start coding!

What we're starting with:

- Checklists View Controller
- Add new Checklist Items
- Check those items off
- Edit those items
- Delete those items
- Store the data

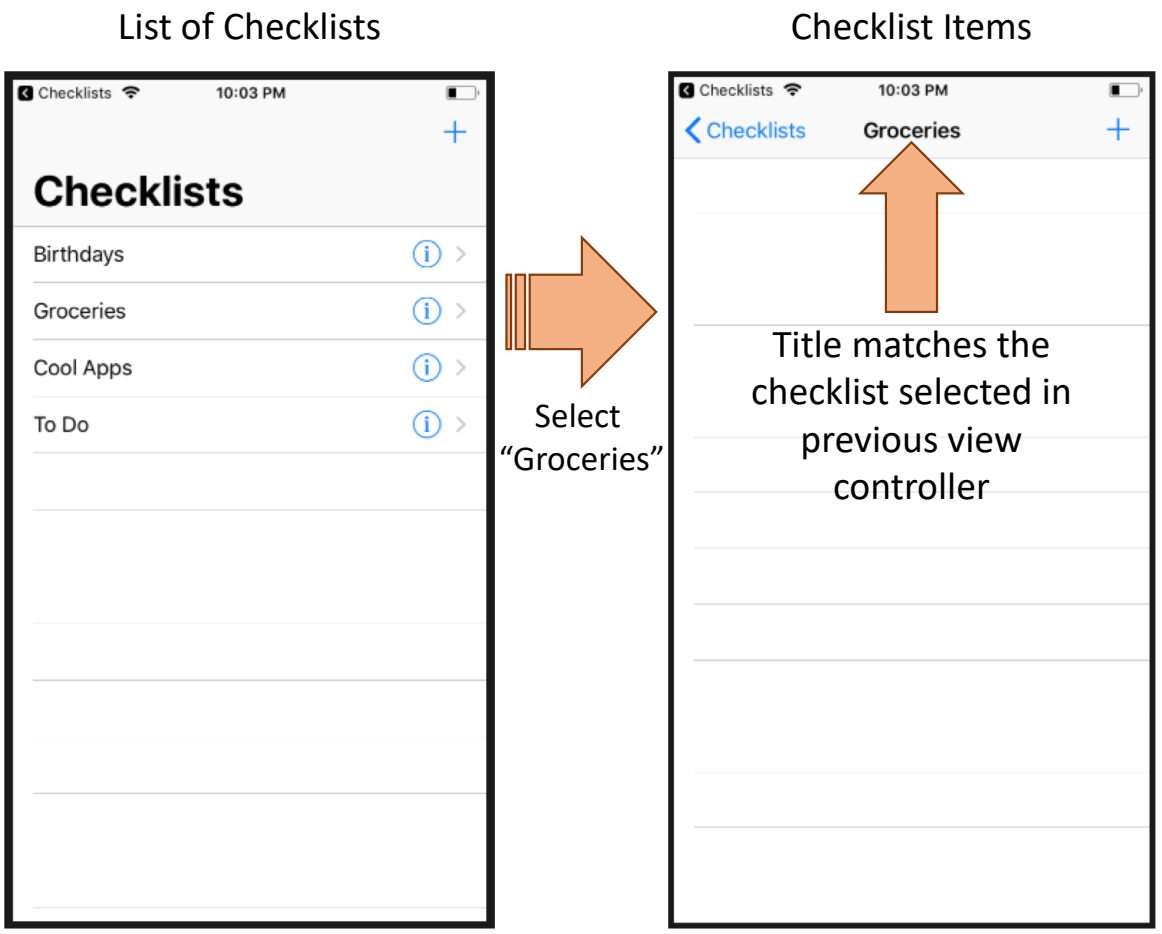


What we’re starting with:

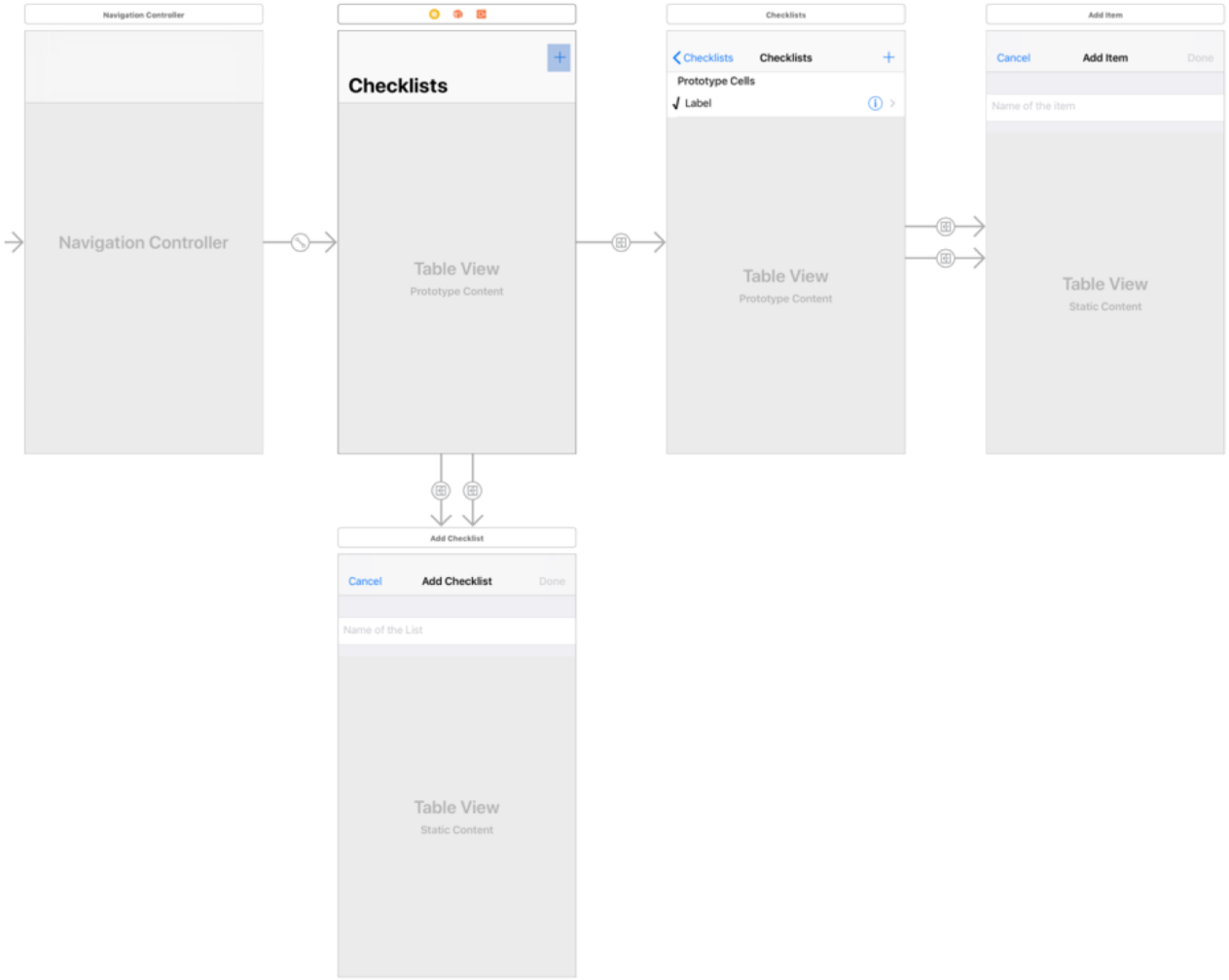


Finished Product:

- Add a new Table View Controller to display all of our lists
- Add new checklists
- Edit Checklist names
- Change Checklist view controller's title to match the selected list



Finished Product:



Programming To-Do List
Chapter 16

- Add a new table view controller to display all of the checklists
- Use markers to organize code
- Learn about Implicitly Unwrapped Optionals
- Review Type Casting and Downcasting
- Add a new table view controller to add/edit new checklists
- Use delegates to pass information between view controllers

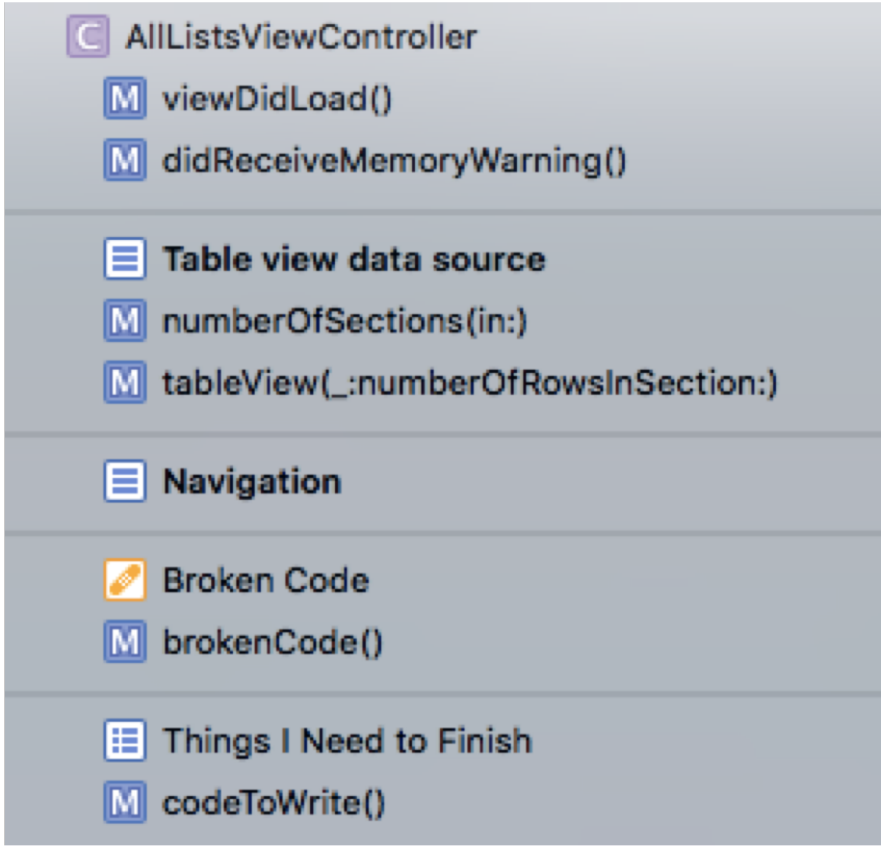
//MARKERS: - Tools to Organize Code

Markers are special comments that allow one to quickly separate and search for sections of code using the Xcode Jump Bar.

Types of Markers

- // MARK :
- // FIXME :
- // TODO :

Using "// MARK: -" (with a hyphen) includes a horizontal line break in the jump bar.



Bonus: Creating custom method descriptions

```
/// You can add a quick description like this
func theMethodYouWantToDescribe(){
}
```

option + click

```
155    /// You can add a quick description like this
156    func theMethodYouWantToDescribe(){
```

Declaration: `func theMethodYouWantToDescribe()`

Description: You can add a quick description like this

Declared In: AllListsViewController.swift

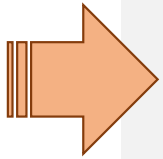
Bonus: Creating custom method descriptions

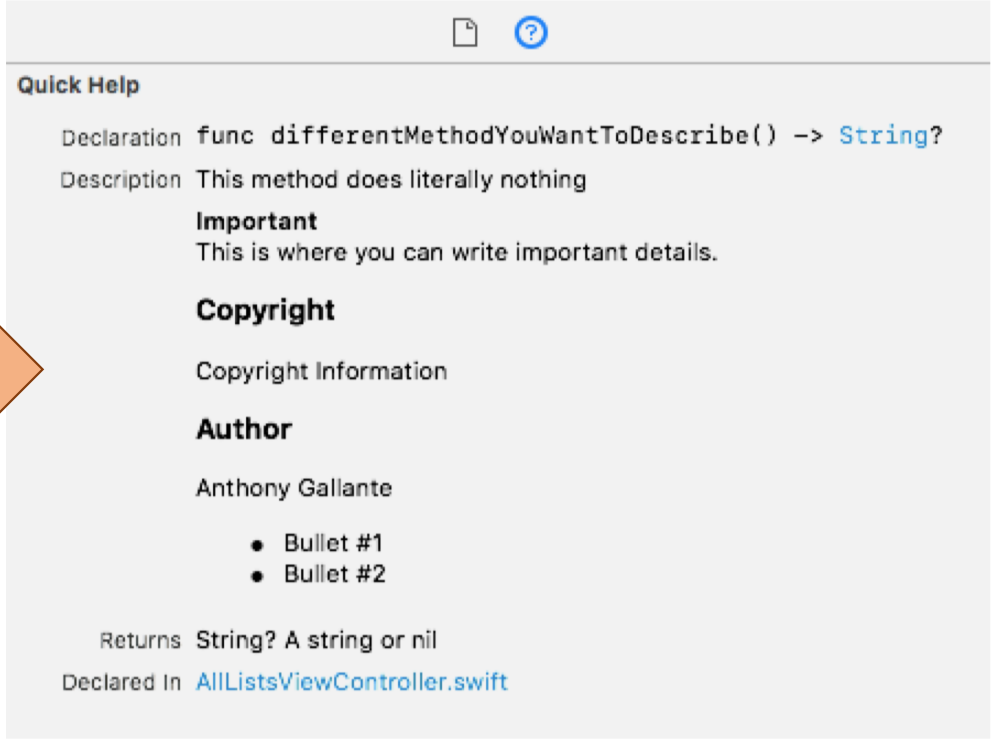
```
/**
 This method does literally nothing

 - important: This is where you can write important
     details.
 - copyright: Copyright Information
 - author: Anthony Gallante

 - Bullet #1
 - Bullet #2

 - returns: String? A string or nil
 */
func differentMethodYouWantToDescribe() -> String?{
  return nil
}
```





More markup tips and syntax can be found [Here]

Programming To-Do List
Chapter 16

- ~~Add a new table view controller to display all of the checklists~~
- ~~Use markers to organize code~~
- Learn about Implicitly Unwrapped Optionals
- Review Type Casting and Downcasting
- Add a new table view controller to add/edit new checklists
- Use delegates to pass information between view controllers

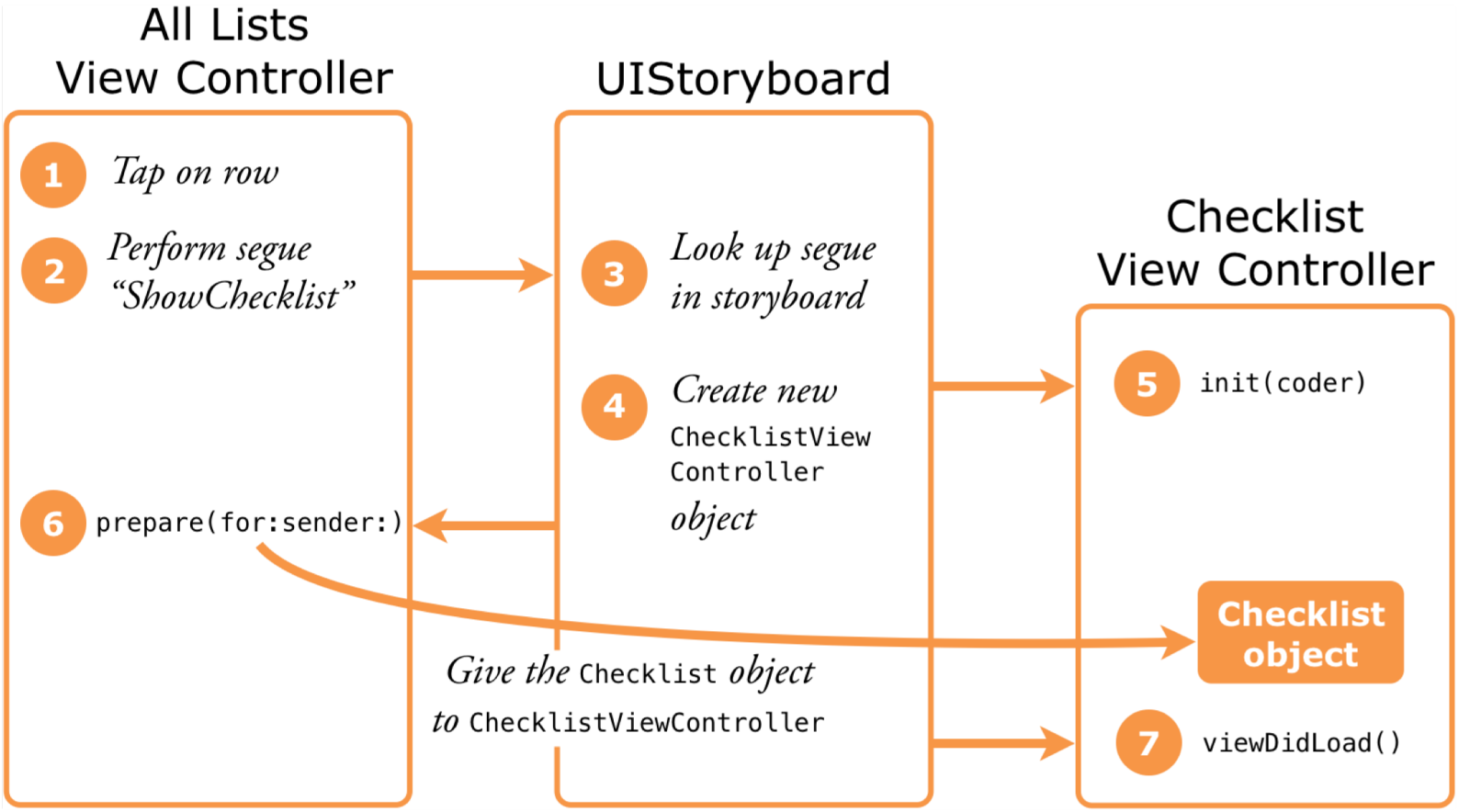
Implicitly Unwrapped Optionals

```
1  let possibleString: String? = "An optional string."
2  let forcedString: String = possibleString! // requires an exclamation mark
3
4  let assumedString: String! = "An implicitly unwrapped optional string."
5  let implicitString: String = assumedString // no need for an exclamation mark
```

An implicitly unwrapped optional is like a normal optional, except we need to *promise* Xcode that there's a value stored in it.

We use it for our Checklist property when loading a new view controller, because for a split second, there is no Checklist Object in the Checklist View Controller.

How Data is Passed Between View Controllers



Downcasting

Downcasting:

A special type cast that uses "as?" or "as!" in order to interpret a value as having a different data type.

Conditional Downcast – as?

Variable optionalString will be type String?, but it will be set to nil if given anything other than a String.

```
1.  var optionalString = dict.objectForKey("SomeKey") as? String
```

Forced Downcast – as!

Variable optionalString will be type String?, but the program will crash if given anything other than a String.

```
2.  var optionalString = dict.objectForKey("SomeKey") as! String
```

Downcasting

```
override func prepare(for segue: UIStoryboardSegue,
                                 sender: Any?) {
  if segue.identifier == "ShowChecklist" {
    let controller = segue.destination
                          as! ChecklistViewController
    controller.checklist = sender as! Checklist
  }
}
```

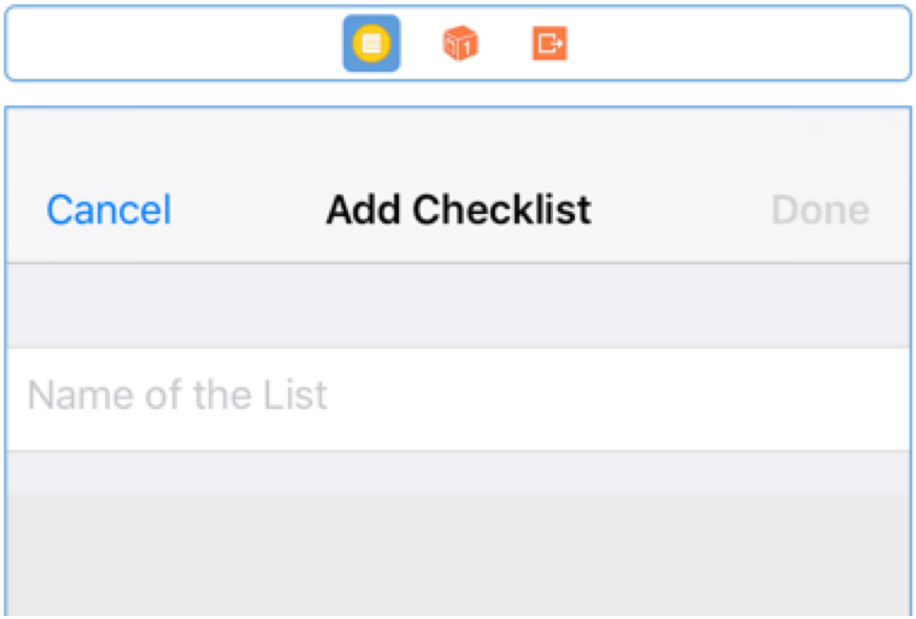
1. `controller = segue.destination as! ChecklistViewController`
   - segue.destination refers to a general UIViewController. We want to specifically segue to ChecklistViewController, so we cast segue.destination to be ChecklistViewController
2. `controller.checklist = sender as! Checklist`
   - sender can be type Any? But controller.checklist expects to be given a Checklist object. Checklist is the string contained within a UITableViewCell.
   - It's as if our final property is `ChecklistViewController.SelectedChecklist`

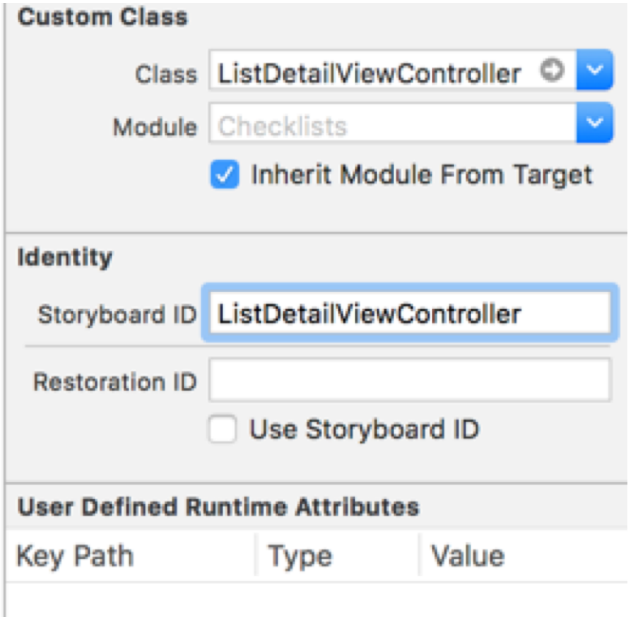
Programming To-Do List
Chapter 16

- ~~Add a new table view controller to display all of the checklists~~
- ~~Use markers to organize code~~
- ~~Learn about Implicitly Unwrapped Optionals~~
- ~~Review Type Casting and Downcasting~~
- Add a new table view controller to add/edit new checklists
- Use delegates to pass information between view controllers

Loading a View Controller Using Code

- Each View Controller has a storyboard property that refers to the storyboard the View Controller was loaded from
- We can use that storyboard property to instantiate other View Controllers!
- `storyboard!.instantiateViewController(withIdentifier:)`





Thank you!

Questions?