GENERAL ASSEMBLY

MOBILE DEVELOPMENT

TABLE VIEWS IN DEPTH

William Martin
Head of Product, Floored

Angel X. Moreno
EIR, Developer

LEARNING OBJECTIVES

- Know what it means to "meet a protocol," first look at the "delegate" pattern.
- Deploy table views without a UITableViewController.
- Customize table view cells.
- Implement passing data from one scene (View Controller) to another.

REVIEW OF DATA STRUCTURES

ARRAYS

SYNTAX: ARRAYS

```
var arr = [1, 2, 3]    // Create implicitly typed
var arr: [Int] = []    // Create empty explicitly typed
var arr = [Int]()      // Create empty explicitly typed

arr.append(4)          // Add element to the end
arr[0] = 12            // Change an existing element at an index
arr.insert(7, atIndex:0)    // Insert at the beginning
arr.removeAtIndex(0)        // Remove an element at an index
countElements(arr) or count(arr)   // Return the number of elements

arr + [4, 5, 6]        // Concatenate two arrays
```

SYNTAX: ARRAYS

```
let firstElement = arr[0]    // Access an element by index

for el in [2, 4, 6] {
    // This loops three times. el is first 2, then 4, then 6.
}

for (index, element) in enumerate([“hi”, “there”, “class!”]) {
    // Loops three times...
    // index is 0, 1 then 2. element is “hi”, “there” then “class!”
}
```

DICTIONARIES

SYNTAX: DICTIONARIES

```
var constants : [String: Double] = [:]   // Create empty dictionary
var constants = [“e”: 2.71828]           // Create implicitly typed
var constants = [String: Double]()    // Create empty instantiating the type

constants[“pi”] = 3.14159   // Add another key-value pair.
constants[“e”]              // Look up a value in the dictionary.
constants[“c”]              // OOPS! The key’s not there. You get nil.

constants.removeValueForKey(“e”)  // Remove a key-value pair given a key.
```

SYNTAX: DICTIONARIES

```
// Iterate over the key-value pairs in the Dictionary.
for (constant, value) in constants {
    // constant will be “e”, “pi”, etc.
    // value will be 2.71828, 3.14159, etc.
}

if let c = constants[“c”] {
    // Does this look familiar? Check for nil this way!
} else {

}
```

TABLE VIEWS

TABLE VIEWS

- Table views display a one-dimensional list of cells.
  - Section: All table views contain multiple sections.
  - Row: Every section has a number of rows, which are entries in that section.
  - NSIndexPath: The combination of a section and row that is a unique entry in a table view.
  - Cell: The view that is displayed for an NSIndexPath (class UITableViewCell, which is a subclass of UIView).

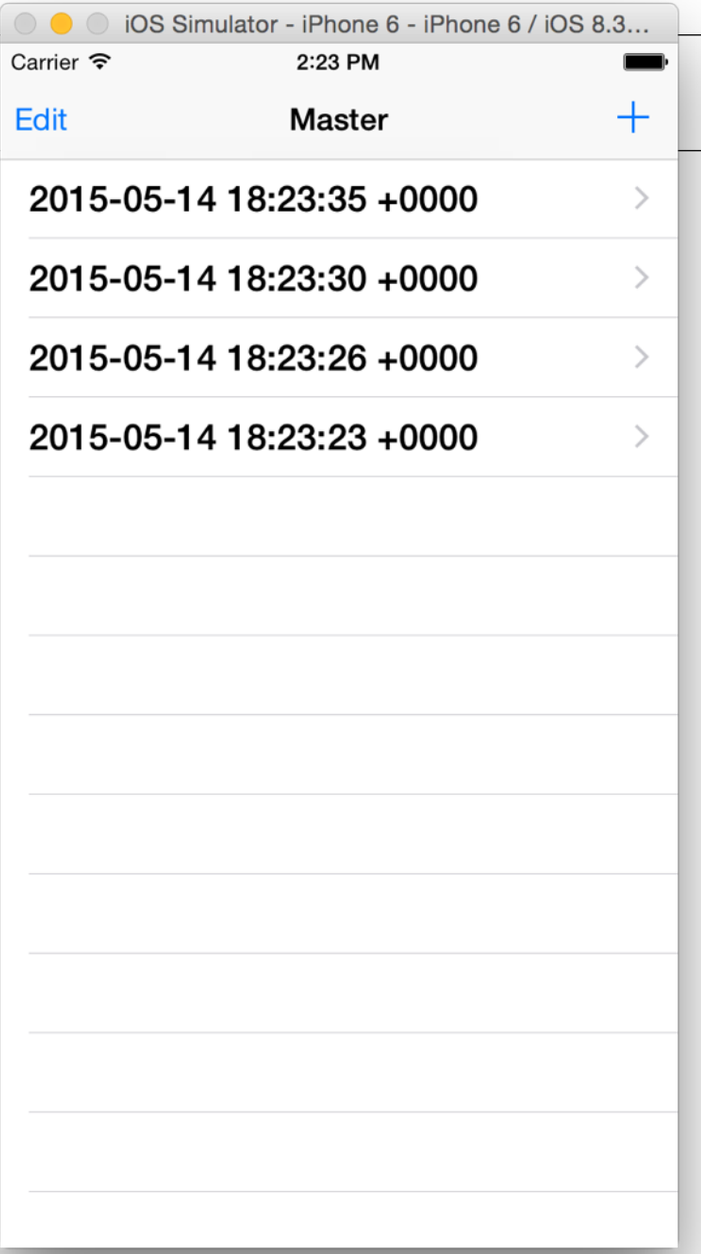
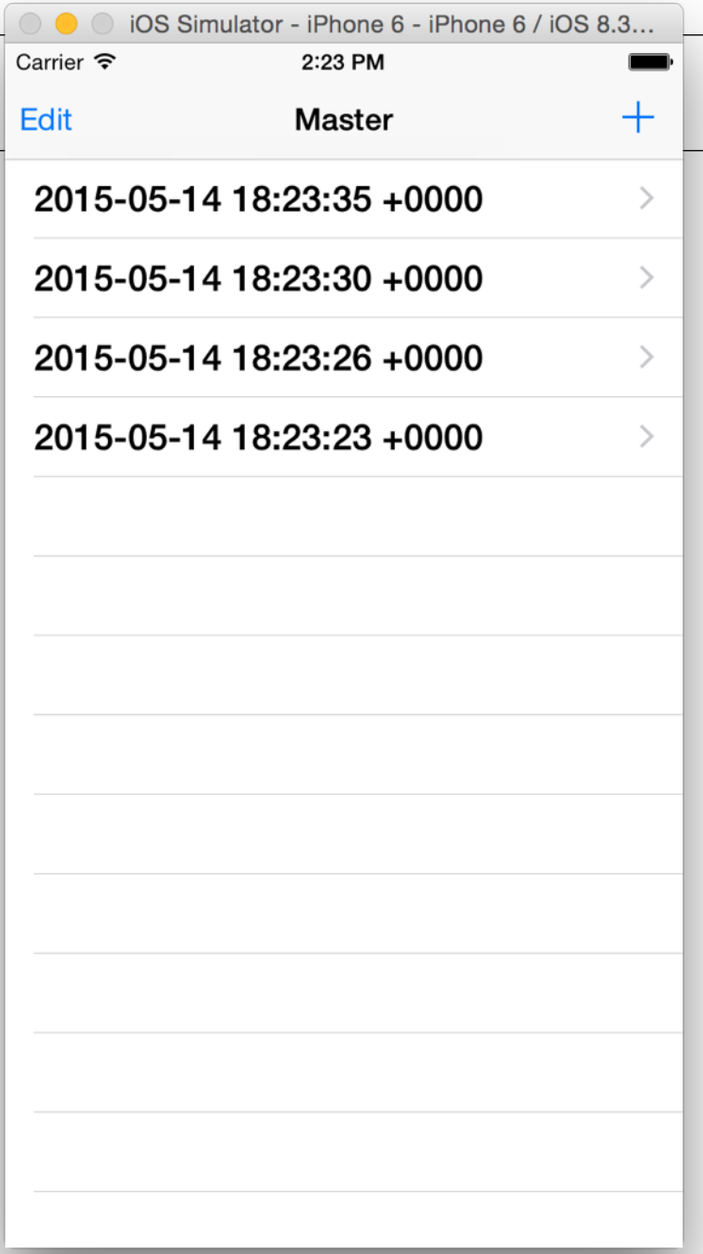


TABLE VIEWS

- Table views should have a data source and a delegate.
  - Data source: Provides cells, number of cells, and sections.
  - Delegate: Gets called when things happen to the table view, provides some views (e.g. header and footer).



EXAMPLE: DOG CATALOG

TABLE VIEW DOG CATALOG

- Uses a UITableViewController. Not recommended for anything more complex than what you see (i.e. you can't add a toolbar to this controller easily).
- What happens when we add a Dog instance?
  - User taps "Add" and a modal dialog shows with some UI elements. The segue sets up the modal's view controller with a reference to the main view controller.
  - User populates the form with data and taps Done.
  - The IBAction associated with the Done button is called.
    - Grab data from the UI elements.
    - Make a new Dog() instance.
    - Call the insertNewObject method on the main view controller reference with the new Dog.
  - The view controller adds the Dog instance to its Array.
  - The view controller updates the table.

SKILLZ YOU'LL NEED

REFERENCES, PROTOCOLS, SEGUES, CASTING

REFERENCES

- Sometimes a class needs to hold an instance of another class.
- Instances like this are “held by reference,” meaning it’s like holding a shortcut or alias to that instance.
- For example, we can create a Dog and a Person class, and a Dog can hold a reference to its owner:

```
class Dog {
    var name : String
    var owner : Person!
    init(name:String) {
        self.name = name
    }
}
```

```
class Person {
    var name : String
    init(name:String) {
        self.name = name
    }
}
```

```
var dog = Dog(name:”Layla”)
var em = Person(name:”Emily”)
dog.owner = em
```

PROTOCOLS

- Describes a group of related methods that a class *should* have if it's to be used properly by another class.
  - Looks like a class (syntax-wise), but it isn't.
  - Its methods can be required or optional.
- Classes can 'meet' as many protocols (i.e. interfaces) as they'd like.
- In our examples, we'll be using two protocols:
  - UITableViewDelegate - tells our class to provide methods to handle particular table events
  - UITableViewDataSource - tells our class to have methods that provide data to display
- Defining and deploying a protocol:
  - `protocol MyProtocol { /* protocol definition */ }`
  - `class MyClass: SuperClass, MyProtocol { /* class definition */ }`

SEGUES

- A segue isn't just a fancy line between View Controllers in IB, it's an instance of a class.
- A segue is the only object that really "knows" (i.e. has references to) both View Controllers on either end of a transition from one View Controller to another.
- To pass information from one scene to another, we'll need to use the Segue instance just at the right moment. We'll be writing code inside the prepareForSegue method provided to us for use in View Controllers.
- Why do we need to pass information?
  - We'll create a data entry (e.g. a Dog or a Task) in one View Controller, but add it to a data structure (i.e. an Array) in *another* View Controller.
  - Sometimes, we'll need to edit an existing piece of data. But to do so, we'll need to tell the edit form, another View Controller, which piece of data to edit.

CASTING

- Casting is a process of telling Swift to treat an instance of a class as an instance of another class.
- *It can sometimes be dangerous!*
- Use the “as” (sometimes “as!”) keyword to “cast” a value to a different type.
- For example:

```
var controllerInMyType = segue.destinationViewController as MyViewController
```

`ControllerInMyType` will be of type `MyViewController`, even though `segue.destinationViewController` is of type `AnyObject`

- Why do we need to do this?
    - Some iOS code is written generically by Apple for our use, but it can’t ever be written for our specific types that we’re creating as we write an app.

EXERCISE: TODO APP

TABLE VIEW TODO APP - WATCH

- Starting with a single view application, create an app that manages a list of Tasks to perform (e.g. “Buy an Apple Watch”, “Purchase AppleCare”, “Sell my iPad”)
- Add a UITableView to the Main.storyboard file.
  - Add a prototype cell. Set its identifier as “Cell”.
  - Reference the table view with an IBOutlet.
- Add UITableViewDelegate and UITableViewDataSource protocols to the ViewController class.
- Add the appropriate methods for those protocols to populate the table.
  - Use the ESC autocomplete feature in Xcode to help you write these methods.
  - Look up these protocols and methods in the iOS online documentation.
- Add a modal to add a new task as a String.
  - Remember to provide a reference using prepareForSegue to the modal’s view controller.
  - Provide a method that adds a new task to the Array of tasks and update the table.

TABLE VIEW TODO APP – DO

- Create a Task class with properties title and description (String).
- Redefine ViewController using an Array of Tasks (instead of an Array of Strings).
- Add a UITextView to set the description of the Task in the modal for new tasks.
- Add a detail view controller to the table to show all the task's information.
  - Create a new view controller: TaskDetailViewController
  - Link the "selection" triggered segue of a table cell (right-click in IB) to that new view controller.
- Add a view to edit the Task. Use a Segue to pass the instance of a Task. Use the next slide as a hint.

TABLE VIEW TODO APP - DO

When you need a specific Task in prepareForSegue, you'll have to get it using an indexPath associated with its cell. This assumes your class name is TaskDetailViewController and that you have a segue from the "selection" action of your UITableViewCell (right-click a cell in Interface Builder):

```
let dest = segue.destinationViewController as! TaskDetailViewController
let cell = sender as! UITableViewCell
let indexPath = self.tableView.indexPathForCell(cell)
if let _indexPath = indexPath {
    let task = self.tasks[_indexPath.row]
    // Use the task variable to pass it to "dest"
}
```

TABLE VIEW TODO APP - DO

If you decide to use a Push segue, you can go back (say, in an IBAction) like this:

```
if let _navigationController = self.navigationController {
    _navigationController.popViewControllerAnimated(true)
}
```