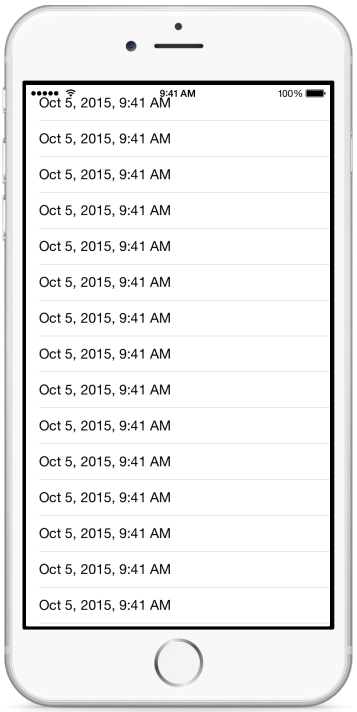
Journal
Lesson 6

Description

Add a `Journal` model to the project, and use it to populate the table view.



Learning Outcomes

- Practice analyzing model requirements and implementing a model class.
- Practice declaring stored properties, computed properties, initializers, and method implementations.
- Practice implementing `UITableViewDelegate` and `UITableVideDataSource` methods, and displaying rows of data in a table view.
- Recognize opportunities for abstraction and greater safety in existing code, and apply access control to enforce abstraction.
- Relate the use of loops to the `map` function and closure expressions.

Vocabulary

array	mutability	property
computed property	initialization	default property value
array subscripting	abstraction	safety

Materials

- **Journal Lesson 6** Xcode project

Opening

What other model is missing from our app?

Agenda

- Discuss the need for a `Journal` model to represent a collection of `JournalEntry` objects.
- Add a new (⌘N) `Journal` class to the project.

```
import Foundation

class Journal {

   var entries = [JournalEntry]()

}
```

- Discuss the mutability of the `entries` array, and the use of a default value to initialize the `entries` property with an empty `[JournalEntry]` array.
- In the `JournalTableViewController` class, replace the `sampleData` property with a `Journal` property.

```
...
let journal = Journal()
...
```

- Discuss how the controller assigns a `Journal` object as the default value of the `journal` property.
- Update the implementation of `viewDidLoad` to naively append a large sample of `JournalEntry` objects to the `Journal`.

```
override func viewDidLoad() {
   super.viewDidLoad()
   for index in 0..<1000 {
      journal.entries.append(JournalEntry(date: NSDate(),
         contents: "Contents for entry \(index)"))
   }
}
```

- Update the implementation of `tableView:numberOfRowsInSection:` to use the `journal` property.

```
override func tableView(tableView: UITableView,
   numberOfRowsInSection section: Int) -> Int {
   return journal.entries.count
}
```

- Update the implementation of `tableView:cellForRowAtIndexPath:` to retrieve a `JournalEntry` from the `journal` property.

```
override func tableView(tableView: UITableView,
   cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
   let cell =
      tableView.dequeueReusableCellWithIdentifier(cellReuseIdentifier,
         forIndexPath: indexPath) as! UITableViewCell
   if let label = cell.textLabel {
      label.text = "\(journal.entries[indexPath.row])"
   }
   return cell
}
```

- Explain how the `indexPath.row` is used to subscript the `entries` array, and the `JournalEntry` `description` property is automatically used when interpolated in the string.
- Run the app (⌘R), and observe the rows of table cells that display the date and time of each `JournalEntry`.
- Discuss how the implementation of the `Journal` class exposes how `JournalEntry` objects are stored in an array, and how this can be improved by abstracting the details of the `Journal` class.
- Update the implementation of `tableView:numberOfRowsInSection:` to illustrate the needed improvements to the `Journal` class.

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

- Discuss how the controller now interacts directly with the `journal` to determine the number of contained `JournalEntry` objects, rather than accessing the `entries` array directly.
- Add a `count` computed property to the `Journal` class.

```
var count: Int {
   return entries.count
}
```

- Run the app (⌘R), and observe that the functionality remains the same.

- Discuss how the controller `viewDidLoad` method also interacts directly with the `Journal` `entries` property.
- Add a `private` access control modifier to the `Journal` `entries` property

```
private var entries = [JournalEntry]()
```

- Discuss how the `JournalTableViewController` `viewDidLoad` method no longer has the ability to modify the `Journal` `entries` array, and must create an array of `JournalEntry` objects that will be passed to an initializer.
- Modify the `Journal` class by removing the default value for `entries`, and adding an initializer.

```
class Journal {

   private var entries: [JournalEntry]
   ...
   init(entries: [JournalEntry]) {
      self.entries = entries
   }

}
```

- Discuss the challenge in the `JournalTableViewController`, which uses a default value for the `journal` property, but must also construct an array of `JournalEntry` objects to pass to the initializer.
- Discuss how the implementation of `viewDidLoad` uses a `for-in` loop to append numerous `JournalEntry` objects to the `entries` array, and the potential for an alternative approach with `map`.
- Remove the `for-in` loop from `viewDidLoad` and update the `journal` property declaration with a call of `map`.

```
let journal = Journal(entries: (0..<1000).map {
   JournalEntry(date: NSDate(), contents: "Contents for entry \($0)") })

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

- Discuss how the range of numbers is transformed into an array of `JournalEntry` objects, the use of the trailing closure syntax, the implicit return value of the closure expression, and how the resulting `[JournalEntry]` array is passed to the `Journal` initializer.
- Discuss how the implementation of `tableView:cellForRowAtIndexPath:` relies on the `entries` array, and what might happen if `indexPath.row` is out of the bounds of the `entries` array.

- Update the `Journal` class with methods for adding and retrieving a single `JournalEntry`.

```
func addEntry(entry: JournalEntry) {
   entries.append(entry)
}

func entry(index: Int) -> JournalEntry? {
   if index >= 0 && index < entries.count {
      return entries[index]
   } else {
      return nil
   }
}
```

- Discuss how the `addEntry:` method abstracts appending a `JournalEntry` to the `entries` array.
- Discuss how the `entry` method returns an optional `JournalEntry?`, which will wrap a retrievable `JournalEntry` object or `nil`, when the `index` is invalid.
- Update the implementation of `tableView:cellForRowAtIndexPath:` with the safer use of the `entry` method.

```
override func tableView(tableView: UITableView,
   cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
   let cell =
      tableView.dequeueReusableCellWithIdentifier(cellReuseIdentifier,
         forIndexPath: indexPath) as! UITableViewCell
   if let label = cell.textLabel,
      entry = journal.entry(indexPath.row) {
      label.text = "\(entry)"
   }
   return cell
}
```

- Discuss how multiple optional binding is used to avoid nesting `if let` statements, and how the retrieval of the `JournalEntry` is now safer.
- Run the app (⌘R), and observe that the rows of `JournalEntry` descriptions appear in the table view.

Closing

What happens when you tap on a table cell?

Modifications and Extensions

- Add a convenience initializer to the `Journal` class that causes an empty `[JournalEntry]` array to be assigned to the `entries` property.
- Add a `rating` property to each `JournalEntry`, and use the `rating` property to display different background colors for the table cells.

Resources

Cocoa Core Competencies: Model Object https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/ModelObject.html

The Swift Programming Language: Classes and Structures https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/ClassesAndStructures.html

The Swift Programming Language: Properties https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Properties.html

The Swift Programming Language: Collection Types https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/CollectionTypes.html

The Swift Programming Language: Control Flow https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/ControlFlow.html

The Swift Programming Language: Access Control https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/AccessControl.html

The Swift Programming Language: Initialization https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Initialization.html

The Swift Programming Language: Methods https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Methods.html

NSIndexPath Class Reference https://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSIndexPath_Class/index.html

UITableViewDelegate Protocol Reference https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/index.html

UITableViewDataSource Protocol Reference https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDataSource_Protocol/index.html