In-Class Homework #6 – Custom iOS Table View Cell Layouts

CUS 357 – Winter 2017
Due Date: End of class, Mar 17, 2017

Learning Objectives

- Become familiar with custom styling table views via custom `UITableViewCell` prototypes.
- Learn how to use `UITableViewDelegate` to further customize a `UITableView`.

Please work in pairs to get as much of this assignment done as possible by the end of today's class session.

Overview

Begin this exercise using the finished product of your In-Class Homework #5 solution. If you did not complete that assignment – you should be close, finish it up and use it as your starting point.

Using what you have learned in the two screencasts you viewed this week, you will be further stylizing your table view of historic geo calculations. You will break the data up into sections according to the date on each entry (e.g. one section per day), and you will also customize the styling on each cell as shown in Figure 1 below.

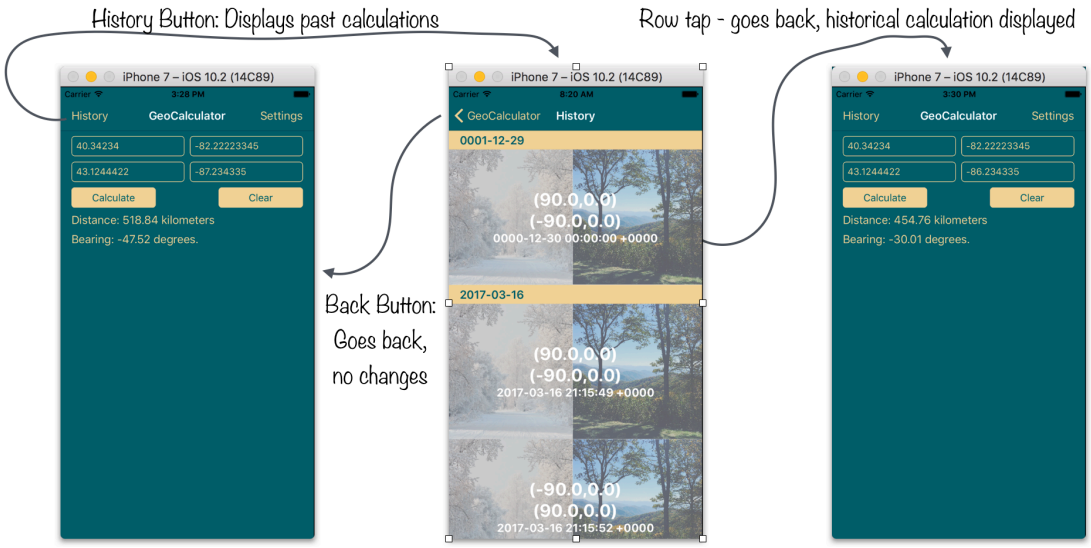

Figure 1. Adding a "history" feature via UITableView.

Introducing Sections in the `HistoryTableViewController`

Eventually, when we figure out how to persist our calculation history to a local database or cloud store, a significant number of historical calculations may be accumulated. The usability of our table view could be improved if we sectioned the entries off by date. Let's go ahead and make the changes necessary to accomplish this.

Step 1: In order to have some entries scattered across several days, let's pre-populate our entries data structure in our calculator view with a couple of entries – one from the past and one from the future:

```
var entries : [LocationLookup] = [
        LocationLookup(origLat: 90.0, origLng: 0.0, destLat: -90.0, destLng: 0.0,
timestamp: Date.distantPast),
        LocationLookup(origLat: -90.0, origLng: 0.0, destLat: 90.0, destLng: 0.0,
timestamp: Date.distantFuture)]
```

Step 2: Add the code needed in `HistoryTableViewController` to sort the entries into sections by day. This code is provided for you below – cut/paste but please do read it and make sure you understand it[1].

```
    var tableViewData: [(sectionHeader: String, entries: [LocationLookup])]? {
        didSet {
            DispatchQueue.main.async {
                self.tableView.reloadData()
            }
        }
    }

    func sortIntoSections(entries: [LocationLookup]) {

        var tmpEntries : Dictionary<String,[LocationLookup]> = [:]
        var tmpData: [(sectionHeader: String, entries: [LocationLookup])] = []

        // partition into sections
        for entry in entries {
            let shortDate = entry.timestamp.short
            if var bucket = tmpEntries[shortDate] {
                bucket.append(entry)
                tmpEntries[shortDate] = bucket
            } else {
                tmpEntries[shortDate] = [entry]
            }
        }

        // breakout into our preferred array format
        let keys = tmpEntries.keys
        for key in keys {
            if let val = tmpEntries[key] {
                tmpData.append((sectionHeader: key, entries: val))
            }
        }

        // sort by increasing date.
        tmpData.sort { (v1, v2) -> Bool in
            if v1.sectionHeader < v2.sectionHeader {
                return true
            } else {
                return false
            }
        }

        self.tableViewData = tmpData
```

[1] Please note that the code snippets provided in this assignment reference two simple extensions, one for the Date class and one for the Double class. You can find them listed at the end of the assignment. Feel free to paste them into the your HistoryTableViewController source file if you choose to use the code as is.

```
}
```

Step 3: Once you've pasted the code in the previous step, make sure you call the new helper method from `viewDidLoad`:

```
override func viewDidLoad() {
    super.viewDidLoad()
    self.sortIntoSections(entries: self.entries)
}
```

Please note that at this point, we've prepared the data to display it in sections, but we still have to make some code changes in how the controller implements the data source and delegate methods of the tableview it is managing. We will cover those changes momentarily, but let's first create a custom cell in the storyboard.

Updating the Storyboard

In your main.storyboard file, a number of changes are in order:

Step 4: Drag a new **Table View Cell** prototype from the object library into the table view of the `HistoryTableViewController`. Set its identifier to "FancyCell" and its custom height to 200. You can leave the original "subtitle" styled cell from the previous assignment there, or you can simply delete it as it won't be need anymore.

Step 5: Drag two **Image View** objects into the cell and place them side by side, and then click the Stack View button in Interface Builder (towards the bottom of the screen) to place them within a horizontal `UIStackView`. Constraint the widths of the two images to be equal, and make sure their content mode is "Aspect Fill". Find a couple of pretty pictures somewhere to act as filler images for now, drag them into your image asset catalog and then set them to these images views. Eventually, we'll figure out how to automatically download a photo from a web services that characterizes the lat/long pairs we enter.

Step 6: Pin the `UIStackView` created in the previous step to the edges of the cell with the pin tool, making sure the **Constrain to Margins** option is not checked.

Step 7: Add a translucent empty **View** to the cell and pin it to the edges of the cell with the pin tool, exactly like you did the `UIStackView` in the previous step. Set the background color of the view to Light Gray, and dial its alpha setting from 1.0 down to 0.5.

Step 8: Now drag three **Labels** out onto the translucent view, and stack them up vertically in a `UIStackView`. You can adjust the color, style, and size of the label fonts as you see fit, but generally, the first two labels should have a larger font size than the third. Lighter colors will be more visible.

Step 9. At this point, the `UIStackView` may be a child of the View added in Step 7. We don't want that! We want the `UIStackView` to be superimposed *on top* of the translucent view. You can do this with a couple of clever select/drags in the structure view of your layout. First, select and drag the `UIStackView` out of the child position in the translucent view and position it right above the translucent view, then release. Next drag the translucent view and drop it right above the `UIStackView`. Sorry for the tedium / awkwardness of this step, but this is the best way I can describe what you need to do! See the order of the items within the cell in Figure 2 below to check your work.

Step 10. Pin the `UIStackView` containing the labels to the leading and trailing edges of the cell, and center it in the cell horizontally. Once again, at this point, if you did all these steps correctly, you will have an arrangement something like that in Figure 2 below.

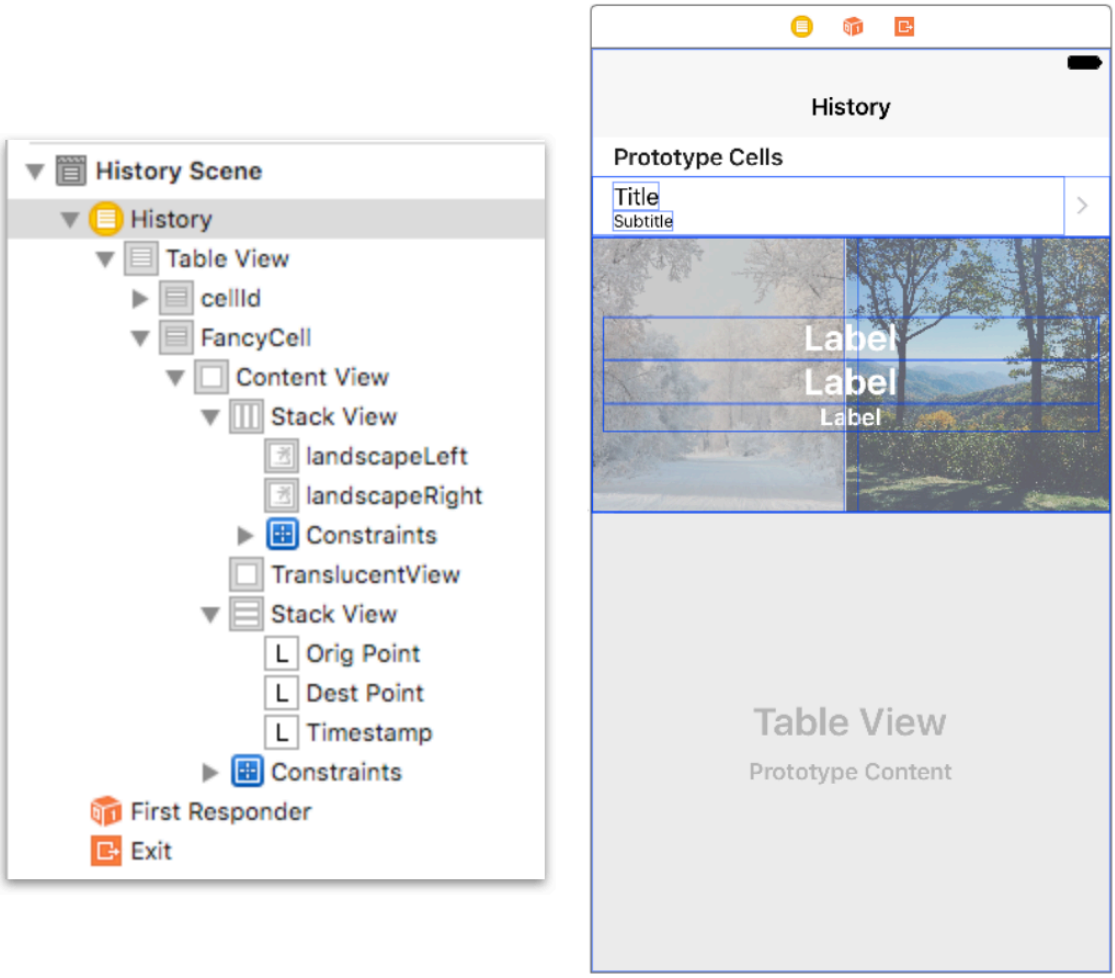

Figure 2. How your cell should be laid out - pay attention to the ordering and parent/child relationships of the items in the cell as shown on the left.

Adding a Custom Table View Cell Class

Step 11: Introduce a new Swift class `HistoryTableViewCell` that extends `UITableViewCell`.

Step 12: Set the class id of your freshly created custom cell prototype to this class, and add the following outlets to the labels (make sure you hook them up in the storyboard scene!):

```
@IBOutlet weak var destPoint: UILabel!
@IBOutlet weak var origPoint: UILabel!
@IBOutlet weak var timestamp: UILabel!
```

Finish up the `HistoryTableViewController`

In the initial section of the assignment, you re-organized your entries into a data structure that would lend itself nicely to displaying the data in sections, by day. In this section, we'll finish up that change by adjusting the implementation of the data source and delegate of the table view.

Step 13: Update the data source delegate calls to use our newly sectioned / sorted data structure. Once again, the code is provided below for cut/paste, but read through it and convince yourself you know what it is doing!

```
// MARK: - Table view data source
    override func numberOfSections(in tableView: UITableView) -> Int {
        // #warning Incomplete implementation, return the number of sections
        if let data = self.tableViewData {
            return data.count
        } else {
            return 0
        }
    }

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
        // #warning Incomplete implementation, return the number of rows
        if let sectionInfo = self.tableViewData?[section] {
            return sectionInfo.entries.count
        } else {
            return 0
        }
    }
```

Step 14: Adjust the `tableView:cellForRowAt` method to use the new custom cell:

```
    override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "FancyCell", for:
indexPath) as! HistoryTableViewCell

        //let ll = entries[indexPath.row]
        if let ll = self.tableViewData?[indexPath.section].entries[indexPath.row] {
            cell.origPoint.text =
"(\(ll.origLat.roundTo(places:4)),\(ll.origLng.roundTo(places:4)))"
            cell.destPoint.text = "(\(ll.destLat.roundTo(places:
4)),\(ll.destLng.roundTo(places: 4)))"
            cell.timestamp.text = ll.timestamp.description
        }
        return cell
    }
```

Step 15: Finish out the prettiness by overriding the table view delegate protocol:

```
    // MARK: - UITableViewDelegate
    override func tableView(_ tableView: UITableView, titleForHeaderInSection
section: Int) ->
        String? {
            return self.tableViewData?[section].sectionHeader
    }

    override func tableView(_ tableView: UITableView, heightForRowAt indexPath:
IndexPath) ->
        CGFloat {
            return 200.0
    }
```

```
    override func tableView(_ tableView: UITableView, willDisplayFooterView view:
UIView, forSection
        section: Int) {
        let header = view as! UITableViewHeaderFooterView
        header.textLabel?.textColor = THEME_COLOR2
        header.contentView.backgroundColor = THEME_COLOR3
    }


    override func tableView(_ tableView: UITableView, willDisplayHeaderView view:
UIView,
                  forSection section: Int) {
        let header = view as! UITableViewHeaderFooterView
        header.textLabel?.textColor = THEME_COLOR2
        header.contentView.backgroundColor = THEME_COLOR3
    }

    override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
        // use the historyDelegate to report back entry selected to the calculator
scene
        if let del = self.delegate {
            if let ll =
self.tableViewData?[indexPath.section].entries[indexPath.row] {
                del.selectEntry(entry: ll)
            }
        }

        // this pops to the calculator
        _ = self.navigationController?.popViewController(animated: true)
    }
```

If you did everything correctly, you should now be done! Verify your work by running the app. Perform a quick demo for the instructor, and then have one of the pair post a zipped solution (or github url) to Blackboard. If you do not finish, post the project to Blackboard with a sentence or two description of what you finished, and what remains to be done.

Appendix

As noted previously, the above code snippets refer to the following two extensions. You are welcome to use them as you see fit. It should be obvious what they do.

```
extension Double {
    /// Rounds the double to decimal places value
    func roundTo(places:Int) -> Double {
        let divisor = pow(10.0, Double(places))
        return (self * divisor).rounded() / divisor
    }
}

extension Date {
    struct Formatter {
        static let short: DateFormatter = {
            let formatter = DateFormatter()
            formatter.dateFormat = "yyyy-MM-dd"
            return formatter
        }()
    }
```

```
    var short: String {
        return Formatter.short.string(from: self)
    }
}
```