In-Class Homework #9 – Working with CocoaPods

CUS 357 – Winter 2017
Due Date: End of class, Mar 29, 2017

Learning Objectives

- Learn how to manage third party components and libraries using CocoaPods, the de facto Xcode dependency management tool.

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 #6 solution. If you did not finish that homework, you are welcome to start with the instructor's solution, available on Blackboard.

A vibrant open source community exists around iOS and Android development. You can save a lot of time and in many cases build a much better app by standing on the shoulders of others. Xcode does not have a built-in native dependency management solution for non-Apple components and libraries, but the open source tool CocoaPods has become the de facto standard solution.

You can learn about CocoaPods and how to install it on your computer by visiting its website (https://cocoapods.org). For the remainder of this tutorial, we are going to assume you've got CocoaPods installed on your Mac and ready to go. Let's look at how you can manage dependencies in Xcode using CocoaPods.

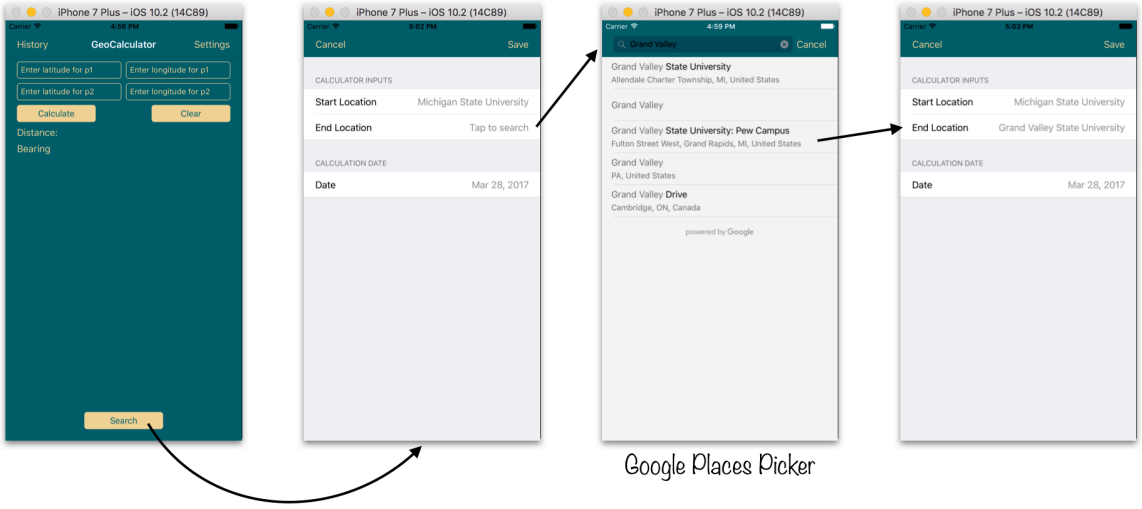

Figure 1. Searching for locations using Google Places Picker.

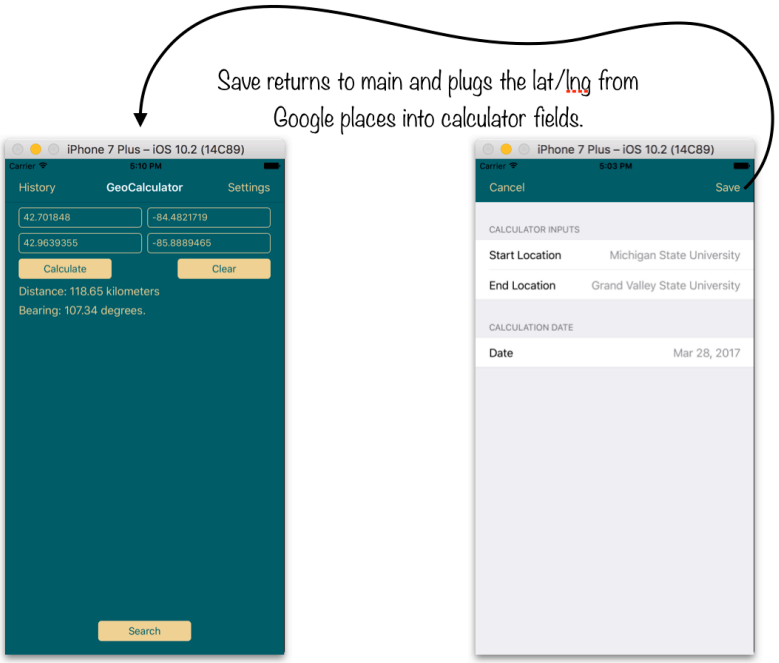

Figure 2. Saving the selected places stuffs the corresponding lat/longs into the calculator fields.

We're going to use the Google Place Picker pod and the Eureka Forms pod to eliminate some of the tedium in entering latitude / longitude values in our geo calculator. Take a look at the finished product in Figures 1 and 2 above.

Setting Up the Pods

Step 1. To bring your Xcode project under cocoapods/, simply cd to the top level directory of your project and type this command:

```
pod init
```

Step 2. If you look at the contents of your directory after this command, you'll notice that some new files have been added. In particular, you will see a file named Podfile which is where you will specify our third party dependencies. Let's edit that file with your favorite text editor and add a couple of pods, as we did in the screencast:

```
pod 'Eureka', '~> 2.0.0-beta.1'
pod 'GooglePlacePicker'
```

If you don't include a version, it will simply take the last released version. If you need a specific version you can specify it. So in the above entries, we are specifying a specific version of the Eureka pod (e.g. the only version I could get to work with current Swift 3 syntax!) and latest-greatest version of the GooglePlacePicker pod. Generally, you can learn about and discover more pods by searching cocoapods.org. Once you find a Pod of interest, the pod's github repo will often give you more info about usage (For example, take a peek at the Eureka Forms github readme.md).

Step 3. Next, you need to issue the command that will fetch and configured the specified pods in your project. Cd into the directory holding your Podfile (top level directory of your project) and type the following command from a shell prompt:

```
pod install
```

Step 4. If you look in the directory you will notice a new workspace file with the *.xcworkspace extension. You can open the project by double clicking the filename in Finder, or typing in this command:

```
open GeoCalculator.xcworkspace
```

It is important to note that you cannot open the project using the old project file anymore - you must use this xcworkspace file from now on or you will have build problems!

Setting up the Google Place Picker Pod

Step 5. In order to use Google place picker you need an API key from Google. You can accomplish this at the google developer console:

https://console.developers.google.com

If you aren't that familiar with Google's developer console, you can follow the instructions in the Google Places for iOS getting started guide to create your key.

https://developers.google.com/places/ios-api/start

Step 6. In your AppDelegate you need to import GooglePlaces, and define a constant for your key.

```
import GooglePlaces
let GOOGLE_PLACES_API_KEY = "YOUR-KEY-VALUE-GOES-HERE"
```

Replace the string literal with the actual key value from Step 5.

Step 7: In the AppDelegate's application:didFinishLaunchingWithOptions method, provide Google Places with your API key by adding this line of code right before the method returns.

```
GMSPlacesClient.provideAPIKey(GOOGLE_PLACES_API_KEY)
```

Updating the Storyboard

Step 8. Add a GeoCalcButton (e.g. your stylized version of UIButton) labeled "Search" to the bottom center of your main calculator scene as shown in the figures above (width is 1/3 the horizontal space in containing view).

Step 9. Drag a new View Controller scene out to the IB canvas.

Step 10. Add a segue from the button to the view controller and give it the id searchSegue.

Step 11. Add a new class named LocationSearchViewController that extends FormViewController (a class defined by the Eureka pod).

```
import UIKit
import Eureka
import GooglePlacePicker

class LocationSearchViewController: FormViewController {

    var startPoint:GMSPlace?
    var endPoint:GMSPlace?
    var selectedPoint: Int = 0

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

We will use the startPoint/endpoint attributes later to capture the values we are searching for.

Step 12. Update the storyboard custom class id on the new View Controller scene to be LocationSearchViewController.

Implementing the Location Search Form

Step 13. In the LocationSearchViewController, implement the form by pasting the following code to the viewDidLoad() method. Be sure to read through the code and understand what it is doing. You can consult the Eureka documentation online as well as the lecture if anything there puzzles you.

```
            // Describe our beautiful eureka form
        form = Section("Calculator Inputs")
            <<< LabelRow () { row in
                row.title = "Start Location"
                row.value = "Tap to search"
                row.tag = "StartLocTag"
                var rules = RuleSet<String>()
                rules.add(rule: RuleClosure(closure: { (loc) -> ValidationError? in
                    if loc == "Tap to search" {
                        return ValidationError(msg: "You must select a location")
                    } else {
                        return nil
                    }
                }))
                row.add(ruleSet:rules)
                }.onCellSelection { cell, row in
                    // crank up Google's place picker when row is selected.
                    let autocompleteController = GMSAutocompleteViewController()
                    autocompleteController.delegate = self
                    self.selectedPoint = 0
                    self.present(autocompleteController, animated: true,
                            completion: nil)
            }
            <<< LabelRow () { row in
                row.title = "End Location"
                row.value = "Tap to search"
                row.tag = "EndLocTag"
                var rules = RuleSet<String>()
                rules.add(rule: RuleClosure(closure: { (loc) -> ValidationError? in
                    if loc == "Tap to search" {
```

```
                    return ValidationError(msg: "You must select a location")
                } else {
                    return nil
                }
            }))
            row.add(ruleSet:rules)
            }.onCellSelection { cell, row in
                // crank up Google's place picker when row is selected.
                let autocompleteController = GMSAutocompleteViewController()
                autocompleteController.delegate = self
                self.selectedPoint = 1
                self.present(autocompleteController, animated: true,
                        completion: nil)
        }
        +++ Section("Calculation Date")
        <<< DateRow(){ row in
            row.title = "Date"
            row.value = Date()
            row.tag = "StartDateTag"
            row.add(rule: RuleRequired())
        }

    let labelRowValidationUpdate : (LabelRow.Cell, LabelRow) -> () =
                { cell, row in
        if !row.isValid {
            cell.textLabel?.textColor = .red
        } else {
            cell.textLabel?.textColor = .black
        }
    }
    LabelRow.defaultCellUpdate = labelRowValidationUpdate
    LabelRow.defaultOnRowValidationChanged = labelRowValidationUpdate
```

Step 14. Next, implement the GMSAutocompleteViewControllerDelegate protocol as an extension of LocationSearchViewController. This is the code that will get called by the Google Place Picker when a place is picked, canceled or error occurs. Notice how we store the startPoint or endPoint when a place is chosen. We also turn on and off the network activity in the status bar, as appropriate.

```
extension LocationSearchViewController: GMSAutocompleteViewControllerDelegate {

    public func viewController(_ viewController: GMSAutocompleteViewController,
                               didFailAutocompleteWithError error: Error)
    {
        print(error.localizedDescription)
    }

    // Handle the user's selection.
    func viewController(_ viewController: GMSAutocompleteViewController,
                        didAutocompleteWith place: GMSPlace)
    {
        if self.selectedPoint == 0 {
            if let row = form.rowBy(tag: "StartLocTag") as? LabelRow {
                row.value = place.name
                row.validate()
                self.startPoint = place
            }
        } else {
            if let row = form.rowBy(tag: "EndLocTag") as? LabelRow {
                row.value = place.name
                row.validate()
                self.endPoint = place
```

```
            }
        }
        self.dismiss(animated: true, completion: nil)
    }

    func viewController(viewController: GMSAutocompleteViewController,
                        didFailAutocompleteWithError error: NSError)
    {
        // TODO: handle the error.
        print("Error: ", error.description)
    }

    // User canceled the operation.
    func wasCancelled(_ viewController: GMSAutocompleteViewController)
    {
        self.dismiss(animated: true, completion: nil)
        let row: LabelRow? = form.rowBy(tag: "LocTag")
        row?.validate()
    }

    // Turn the network activity indicator on and off again.
    func didRequestAutocompletePredictions(_ viewController:
        GMSAutocompleteViewController)
    {
        UIApplication.shared.isNetworkActivityIndicatorVisible = true
    }

    func didUpdateAutocompletePredictions(_ viewController:
        GMSAutocompleteViewController)
    {
        UIApplication.shared.isNetworkActivityIndicatorVisible = false
    }
}
```

Step 15. Now we need to pass the filled location search form data back to the main calculator controller. Define a delegation protocol for LocationSearchViewController:

```
protocol LocationSearchDelegate {
    func set(calculationData: LocationLookup)
}
```

Provide a property on the LocationSearchViewController to hold a reference to the delegate:

```
var delegate : LocationSearchDelegate?
```

Step 16. Have your main calculator view controller implement the delegate in the form of an extension (the instructor's main view is ViewController – you can substitute your controller name if different).

```
extension ViewController: LocationSearchDelegate {
    func set(calculationData: LocationLookup)
    {
        self.p1Lat.text = "\(calculationData.origLat)"
        self.p1Lng.text = "\(calculationData.origLng)"
        self.p2Lat.text = "\(calculationData.destLat)"
        self.p2Lng.text = "\(calculationData.destLng)"
        self.doCalculatations()
    }
}
```

Notice that ultimately, we aren't using the date you entered on the form. We just threw it in there so you could see how cool the DateRow() user interface was! There are a lot of other row types in Eureka forms as well – very cool stuff.

Step 17. Once again, we need to update the prepareForSegue in our main calculator view controller to make itself the delegate of the LocationSearchViewController upon segue.

```
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if segue.identifier == "settingsSegue" {
            if let dest = segue.destination as? SettingsViewController {
                dest.dUnits = self.distanceUnits
                dest.bUnits = self.bearingUnits
                dest.delegate = self
            }
        } else if segue.identifier == "historySegue" {
            if let dest = segue.destination as? HistoryTableViewController {
                dest.entries = self.entries
                dest.delegate = self
            }
        } else if segue.identifier == "searchSegue" {
            if let dest = segue.destination as? LocationSearchViewController {
                dest.delegate = self
            }
        }
    }
```

Step 18. Almost done. The last thing we will do is programmatically add Cancel/Save buttons to our LocationSearchViewController, and their respective actions. These should be self explanatory, but do make sure you understand what is going on. In the controller's viewDidLoad() method add this code to create the buttons:

```
let cancelButton : UIBarButtonItem = UIBarButtonItem(title: "Cancel",
       style: .plain,
       target: self,
       action: #selector(LocationSearchViewController.cancelPressed))
self.navigationItem.leftBarButtonItem = cancelButton

let saveButton : UIBarButtonItem = UIBarButtonItem(title: "Save",
       style: .plain,
       target: self,
       action: #selector(LocationSearchViewController.savePressed))
self.navigationItem.rightBarButtonItem = saveButton
```

And then implement the two actions:

```
func cancelPressed()
    {
        _ = self.navigationController?.popViewController(animated: true)
    }

    func savePressed()
    {
        let errors = self.form.validate()
        if errors.count > 0 {
            print("fix ur errors!")
        } else {

            let endDateRow : DateRow! = form.rowBy(tag: "DateTag")
            let endDate = endDateRow.value! as Date
```

```
            let p1Lat = (self.startPoint?.coordinate.latitude)!
            let p1Lng = (self.startPoint?.coordinate.longitude)!
            let p2Lat = (self.endPoint?.coordinate.latitude)!
            let p2Lng = (self.endPoint?.coordinate.longitude)!

            // return the newly created Journal instance via the delegate
            self.delegate?.set(calculationData: LocationLookup(origLat: p1Lat,
                    origLng: p1Lng,
                    destLat: p2Lat,
                    destLng: p2Lng, timestamp: endDate))
            _ = self.navigationController?.popViewController(animated: true)
        }
    }
```

Step 19. If you did everything correctly, you should be done! Run the app and test out your handiwork!

If you finish before the end of the session, give a demo to the instructor. If you did not finish, you will have time during our next session.