**CSCI E65g: Mobile Application Development Using Swift and iOS**

**Fall 2018**

**Assignment 4**

**GRAD STUDENTS ONLY**

**Issued: 09/25/2018 Due: 10/02/2018 TOGETHER with Assignment 3**

**Purpose:** Enhance the Restaurant Bill App to be a little more realistic. Put more demands on the data model to support more functionality in the UI including deletion and menu item categorization.

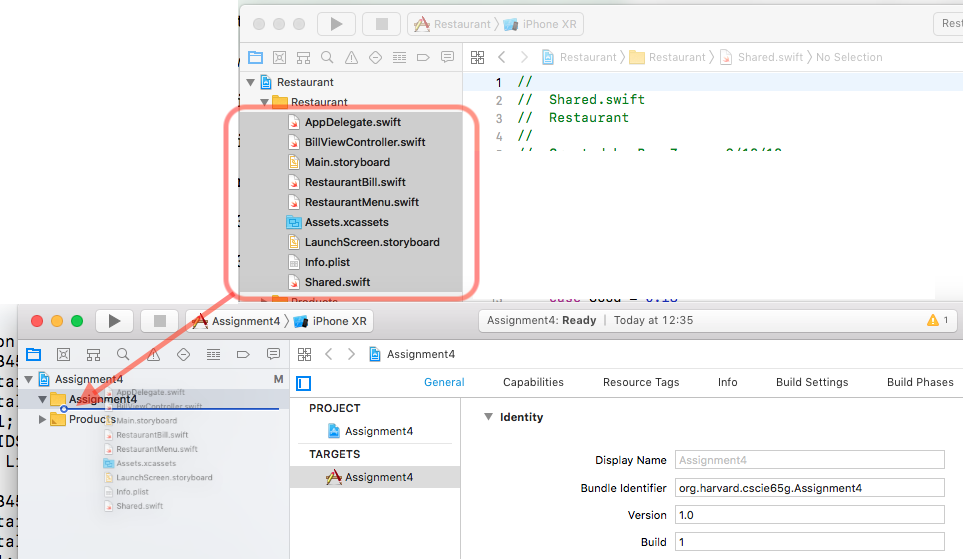
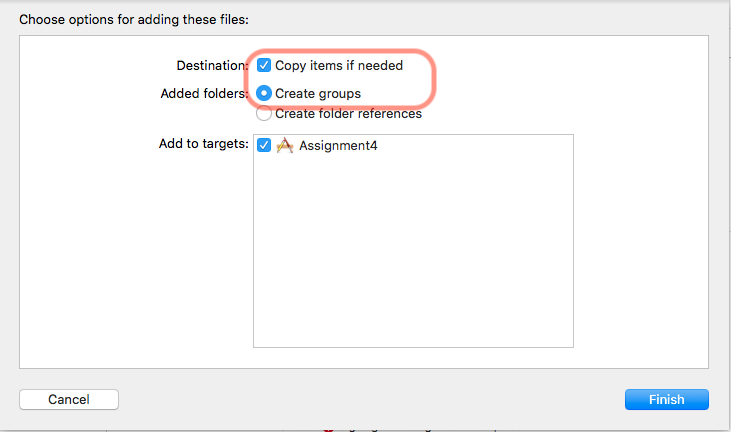
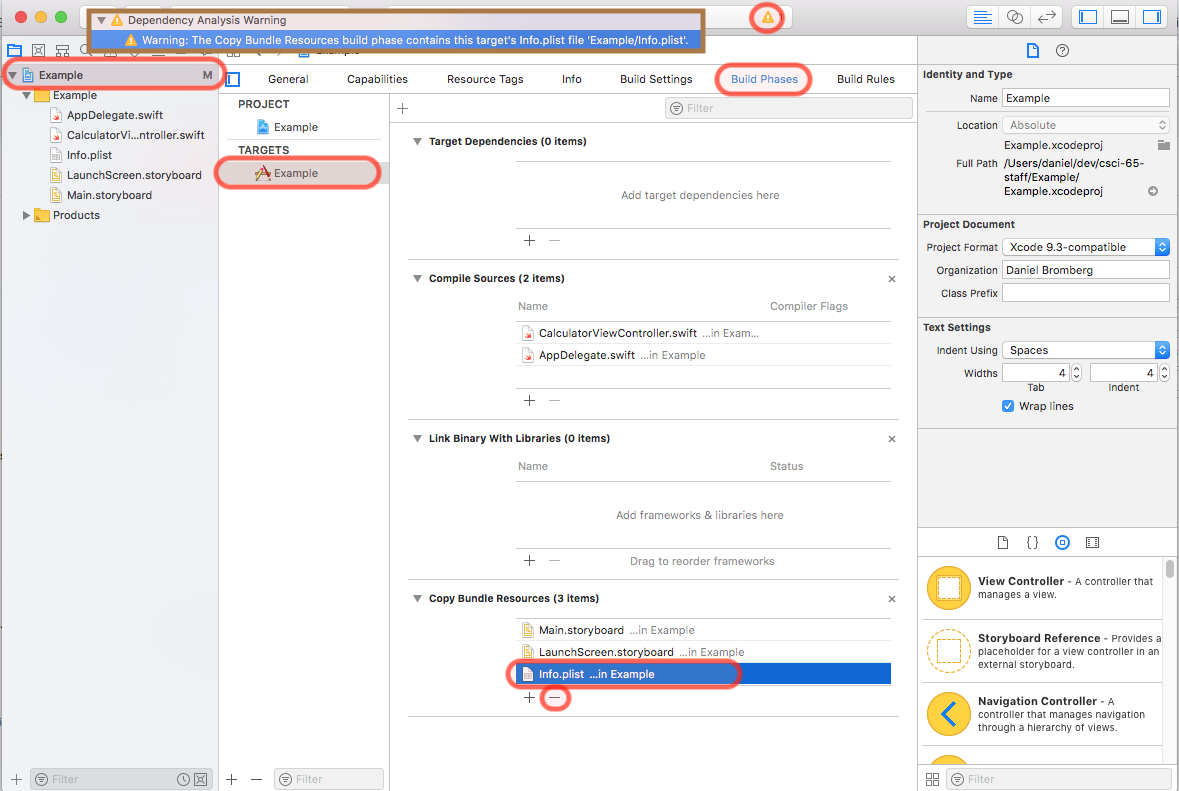
**Scoring:** Out of 75 points (Graduate only; Undergrads need only complete Assignment 3 for Oct 02)

**Readings**

* The [IndexPath](https://developer.apple.com/documentation/foundation/indexpath) data structure used to represent an item an a table. It’s a very simple data structure, and boils down even more simply to the section and rowfields when used with table views. Nevertheless de-mystify it now.
* [Apple Tutorial for App Development](https://developer.apple.com/library/archive/referencelibrary/GettingStarted/DevelopiOSAppsSwift/index.html#//apple_ref/doc/uid/TP40015214-CH2-SW1), as written review for pulling together a Project, Storyboard, a small data model, Outlets, Actions, constraints, and customizing text-entries.
* [Pretty good tutorial](https://www.ralfebert.de/ios-examples/uikit/uitableviewcontroller/) for a non-trivial table view with some images (Remember E65g requires your data to be separated; don’t follow Internet-wide tutorial pattern of jamming raw data into view controller.)
* [Straightforward tutorial for adding sections](https://medium.com/swift-programming/swift-enums-and-uitableview-sections-1806b74b8138) to a table. Take note of:
  + Bad naming practices
    - Container type: ViewController
    - Helper type: enum TableSection: Uses implementation detail rather than *domain* description.
    - Bad property naming: generic name data
  + Non-optimal syntax
    - Failure to use automatic conversion of enum to its own string label in sortData  
      { $0["genre"] == "action" } vs. { $0["genre"] == "\(TableSection.action)" }
    - Failure to use trailing closure syntax
  + Bad data model practices: [TableSection: [[String: String]]]()
  + viewForHeaderInSection is not necessary; [titleForHeaderInSection](https://developer.apple.com/documentation/uikit/uitableviewdatasource/1614850-tableview) is much simpler. Use Storyboard to design custom table cells; don’t worry about constructing views in code.
  + Good use of anonymous closures for sorting
  + Good use of if let to step carefully
  + Good use of enums to define categories
  + Good avoidance of major magic constants, e.g. MovieData, but not always
* [commit editingStyle: Handle edit events in UITableViewDataSource](https://developer.apple.com/documentation/uikit/uitableviewdatasource/1614871-tableview)

**Project Structure:** Enhance assignment 3 by leveraging more existing features of the UIPickerView and UITableView data sources.

**Problems**

1. 0 points Pick up the [Assignment 4 repository](https://classroom.github.com/a/ot8B98Ff).
2. 0 points Make a complete copy of Assignment 3’s files. The command-line method is a zoo; unless you have an hour to practice your perl or Python scripting and learning internal formats, do it with Xcode support:
   1. Open the project for Assignment 3. **Don’t change or commit anything in the Assignment 3 repository when doing Assignment 4.** Leave it as a snapshot representing that work. It will also be a useful point of reference.
   2. Start a new Single-View iOS App Project inside Assignment 4. Use the same bundle identifier convention as for Assignment 3.
   3. Delete **all** the starting boilerplate in Assignment 4 from the Project Browser (left panel). Group-select and Delete is easiest. Confirm **Move to Trash**.
   4. Copy all Assignment 3 resources by dragging as follows:
   5. Confirm the copy choices. Create copies, and not include anything by reference. (Note how this concept applies everywhere!)
   6. (New, non-essential step discovered 2018-09-29): Fix a minor build warning, where Xcode doesn’t understand quite how to import the Info.plist file. It needs to be removed from the **Copy Bundle Resources** Build phase.
   7. Do a test run immediately to ensure a solid starting point. Immediately post to the group if issues. *If you attempt to be brave by dragging an Xcode 9 project into Xcode 10, there will be some conflicts. Some errors will not be obvious. If the error is familiar we can help.*
3. (35 points) Your menu is growing to have many items. To make it navigable, they need to be broken up by category and presented as such.
   1. UI perspective: The picker view needs two components. The left will select **Food** or **Drink**. This will cause the right component to immediately update with items only from that category.
   2. Helper type perspective: Food categories map neatly to enums, so be sure to use one.
   3. Data model perspective: Items need to know their category so this struct will need updating, as will the test data initialization. Retrieving an item and getting an item count needs to be specialized by category now. You will have to decide whether to 1) continue to implement the storage as a single Array or Dictionary and use dynamic filtering, or 2) explicitly separate them out with two properties, or 3) whether to get fancy and have a super-dictionary that contains a series of sub-containers, each by the category (in anticipation of more categories in the future). Any of the 3 approaches is fine; there are tradeoffs no matter what you do. If you feel inexperienced, the second option is most straightforward. Try to use filter rather than an explicit for loop, although the latter will only receive a 1 point deduction.
   4. Abstract data perspective: The protocols will need updating to match these new methods. Stay compatible with code that uses them in the old way. It’s a good chance to practice using *default value* parameters. (How do you ensure backward-compatibility? What does this imply about your enum?)
   5. DataSource and delegate perspective: You’ll need to adjust the call to UIPickerViewDataSource.pickerView:numberOfComponents:. Somewhat harder part: you’ll need to pay attention to the previously unused forComponent parameter in UIPickerViewDelegate.pickerView:titleForRow:.
   6. @IBAction perspective: Now we really care about UI events in the picker. You will need to react to a UIPickerViewDelegate.didSelectRow event in the first component by forcing the other component to [reload](https://developer.apple.com/documentation/uikit/uipickerview/1614385-reloadcomponent). Since we do not (may not) call titleForRow *here* but rely on Cocoa Touch to *call back* our code, how do we ensure this future callback knows which category to use? It is allowable but not necessary to introduce additional storage for this.
   7. Back to the UI perspective: the picker change-up will immediately propagate elsewhere in the UI. Make sure the line item views remain consistent and that adding to the bill still works as usual with the new data.
   8. Refinement perspective: if you switch back and forth between food and drink quickly, would you expect the picker UI to remember the most recently selected item in each, and restore it every time? Not necessary, but bits for thought or extra credit.
4. (30 points) Waiters are growing frustrated that when they make a mistake, they have to force-quit the App and start over. Allow them to delete items.
   1. UI: The idiomatic swipe left and tap Delete should work, and all totals should update immediately. No confirmation step (e.g. pop-up alert) is necessary.
   2. Data Model and protocols: These will need updating. We need to ensure that we can identify what should be deleted based on how Cocoa Touch reports a swipe-delete. A deletion event is [reported to the data source](https://developer.apple.com/documentation/uikit/uitableviewdatasource/1614871-tableview) with only the section and row number. So the data model must be able to take a category and a row number and convert it into an appropriate deletion. Note your choice of representation above will change how much work this is. If you find switching to Arrays is preferable, that’s fine; just be aware you’ll need to re-implement the item-name based functions as an Array search rather than a simple dictionary lookup.
   3. Delegates: All the low level details are taken care of, such as animation and rendering of the button. You just need to mark cells as delete-able by implementing the commitEditingStyle method referenced in the reading above. Once the model has completed its work (and *only* after!), you also need to [tell the table view that a (group of) cell(s) was deleted.](https://developer.apple.com/documentation/uikit/uitableview/1614960-deleterows) Finally, the rest of the UI will need to be synced to the just-mutated RestaurantBill model.
5. (10 points) Customers demand seeing the same sections that Waiters do, in the bill view. However with all your hard work above, this is a welcome change where you can show off the power of your enhanced data models and leveraging an easy part of the UITableView API.
   1. UI: The table view should also report two sections. Each section should be titled correctly. Items should show up in the proper section with no regression bugs. Make sure the waiter can delete the items down to nothing in both sections without crashing.
   2. Data model: You’ve broken all the ground with the picker view. This is easy now. All set!
   3. Delegate perspective: You’ll need to use titleForHeaderInSection as discussed in the above reading. Hopefully this part is a breeze.
   4. Extra credit idea: If you’re moving fast and want an extra challenge: Use the above tutorials to incorporate images in the menu display. Rough outline: Implement a custom UITableViewCell subclass. Tell Storyboard to use this custom type in the cell prototype. Use Storyboard to design it with a UIImageView and put the needed outlet in the subclass code. You’ll need to add images as resources to the project (simple drag), keep track of their names, model those names in the MenuItem structure, and dynamically create and store UIImages in the UITableViewDataSource.tableView:cell:forRowAt: method.