CSCI S-65 Summer 2015

Assignment 4

Due: July 22th just before CLASS at 6:30PM
(Note: this is a change: No longer Midnight)
Total: pts. 140 Undergrad /160 grad

For this assignment, you'll be making a big start on your final projects, and get practice integrating previous assignments into navigation.

1. (60 pts.) (Estimated time: 4-6 hrs.) Create a Storyboard for your project. It should include all scenes that you plan to implement, and the navigation that will connect them. It's OK if your segues aren't the correct type or aren't labeled yet, as long as they're all there.

   GRAD CREDIT: You must have some form of meaningful navigation (and hence segues) in your App. [By meaningful, each screen must do something useful, not just a trivial segue from an Intro screen.]

   Hard-code mock data so we can see what a realistic runtime snapshot will look like.

   For `UITableViews`, click on the table view in the Scene Hierarchy, go to the Attributes Inspector, and increase the number of table cells so you can hardcode several examples of the data your table will contain.

   For `ImageViews`, lay them out in StoryBoard even if you plan to allocate them dynamically in actual code. Use the 'Image' attribute in the Inspector to actually put an demo image in place. This will vastly improve the preview quality if images are important to your project.

   Color & contrast are also important for some Apps. If you plan to use color to convey information (such as priority, or errors, or differentiate sections of the screen), set up those colors, at least roughly, again using the Inspector. This will give is a chance to comment on effective use.

Your submission should be an Xcode project only. It should have no code*, and instead just use StoryBoard. Be sure to include any image and other media assets directly in the submission. For any images, photos, music, audio clips, or other media that you did not personally create, include the source and full copyright information in a separate '***LastName-FirstName*-Prob1-Copyright.pdf**' document at top level.

If you need to annotate your UI with questions or comments, toss on a UITextView and prefix the content with "@comment". Naturally don't bother with constraints. It may be helpful to set the Background attribute to Default (transparent). It's OK if the comment content doesn't fit visually because we can look at full content in the Inspector.

If there's information you feel you must convey that just won't fit in StoryBoard, such as animation, or discussion of any non-Apple library iOS code you plan to integrate, or Internet servers you're relying on, include it in an '***LastName-FirstName*-Prob1-AppNotes.pdf**' document at top level.

[*If you are feeling adventurous, and if you're writing a custom subclass of `UIView`, you may write *a little bit of code* that can work with the `@IBInspectable` and `@IBDesignable` attributes to enhance the preview that StoryBoard is capable of generating. See: http://nshipster.com/ibinspectable-ibdesignable/ and http://www.thinkandbuild.it/building-custom-ui-element-with-ibdesignable/]

The grading criteria will be fully constrained layout and clear labeling / titles.

Turn in a separate project called *LastName-FirstName-Prob1-StoryBoard.xcodeproj*

2. (40 pts) *Estimated time: 4 hrs.* Tab navigation integration

   Take 3 apps written thus far in class: The Restaurant Bill, The Game of Life simulation, and the ColoredSquareDemo. Use the best version

available: either your own latest working code, or the most recent demo given in class if you never got it to work. Prefer your code even if there are minor bugs such as rounding and formatting and layout; we want you to integrate your own code if at all possible.

Integrate all 3 into a UITabBarController.

Design requirements and hints:

a. Each tab button along the bottom should be labeled appropriately.

b. You will need to sub-class the parent controller and equip it with a 'master model'.

c. The master model class will track the 3 models.

    - The master model is just a "shell" that holds child references to model classes you have already written and ensures that only one of each ever gets created. The parent controller must instantiate the master model early on.

    - The master model is in charge of instantiation of child models, and prevents overwriting of references to child models by outside classes
        - It should use `private` properties to contain child references and computed properties to return read-only versions in the '`get`' clause

    - Child controllers may be the only place where required model initialization data is available, such as screen dimensions. In this case the master model must supply a method that can take parameters, rather than a computed property, which cannot. An example might be:

```
// returns reference to a ColoredSquareModel
masterModel.coloredSquareModelWithParams(maxXDim: ###, …)
```

    - The parent controller needs to grant access to the master model as a whole, so that the child can do its own lookups

(do you trust your children?) or if you're feeling ambitious about O-O design, use protocols and specialized access methods so that child controllers can only access the one child model they really need (no extra credit).

- Of course, before child controllers ask something of their parent controller, they must get a reference to it. See the example code in the Lecture 8 NavDemo project.

Turn in a separate project called *LastName-FirstName-Prob2-Tab.xcodeproj*

3. (40 pts) *Estimated time: 3 hrs* Navigation Controller integration

   Do the same as #2, but this time use a Navigation Controller with one level of detail. The master model should hold an *array* of child models all of the same type. Pick any **one** of the 3 aforementioned projects. There is only *one detail view controller class,* and at most only one instance of it – the one currently being displayed. But, it will be fed by its appropriate model just in time by `prepareForSeque`, which will look up the right model using the selected cell's row index. Remember that every time you navigate into a child view controller, it is built from scratch as if the App were starting up for the first time. So as long as it is fed from the appropriate model, it will look correct. Because the array of child models are *persistent* (never torn down by the ARC memory manager during the App's lifetime), the different detail views should maintain their state even when the user navigates back to the root table view, perhaps changes some other models, and returns to the original child view.

   Note the key difference in object lifetimes of models versus views and view controllers. This is extremely important to appreciate as an App developer.

   Turn in a separate project called *LastName-FirstName-Prob3-Nav.xcodeproj*

4. (20 pts) **GRAD CREDIT ONLY** *Estimated time: 2 hrs*

   Take the `UITabBarController` from problem 2, and integrate the entire Navigation Controller from problem 3 in place of the original view controller. Do this just for the one project you chose for problem 3 and leave the other 2 tabs unchanged. The layering of the models should follow the layering of the controllers, so that nearly all of the code from problem 3 can be re-used.

   Turn in a separate project called *LastName-FirstName-Prob4-TabNav.xcodeproj*