Assignment 3

BeverageList: Swift Types, Undo, and Unit Tests

Due: November 16th, 2015. 9:00 AM **Group size:** 2

Description

In this assignment, you will learn how to structure your code using swift types. You will also learn about the undo architecture in iOS and how to test your data.

Task

Part 1: Create BeverageList app

In the kitchen of our chair we have a fridge filled with beverages. Anyone working at the chair can buy a beverage. Each person has to write down which beverage he/she took and pay for the them at the end of the month.

1. Create an app that provides an interface for people to record what beverage they took and the price of that beverage. Provide a way for the person to see the total he must pay at the end of the month (people pay monthly for their purchases).
2. Each month the fridge should contain 8 different types of beverages, 15 bottles of each. Your system should keep track of that, and notify the secretary (the app's admin) what the fridge is missing.
3. Implement the following entities in Swift:
    - 3.1. Storage: Containing a list of beverages and the number of bottles which are currently in the fridge.
    - 3.2. User: Containing the name of the user and a total amount he has to pay.
    - 3.3. Beverage: Containing the name of the beverage and the price.

   For each of these entities choose if it should be a struct, a class, or a enum and explain your choice in a additional text file named: `SwiftStructures.txt` in your submission.
4. When creating a new project, enable `Include Unit Tests` (see Part 4).
5. Disable `Use Size Classes.` But make sure that your app can be simulated on iPhones 4S - 6S Plus and in portrait orientations.
6. Challenge yourself: Make the UI adjust for landscape orientation.

Part 2: Undo

Implement undo mechanism that allows the user to undo his selection.

1. Add a button to the app's interface to undo the last selection

2. Challenge yourself: Replace the undo button with a shake gesture (`UIEventSubtypeMotionShake`). Check out Event Handling Guide for iOS.
3. Challenge yourself: Implement a redo (button/gesture)
4. Challenge yourself: Enable multiple undos

Part 3: Storing results

Save and restore the beverage counts after restarts.

Store the beverage counts in a property list file at the `Application Support` directory. (See: "Accessing Files and Directories" and "OS X Library Directory Details".) Restore counts when the application restarts.

Part 4: Testing data

Implement unit tests (see example. For more details: About Testing with Xcode) to test your model functions.

Submission
Create a zip archive including the following items

- ☐ `BeverageList Xcode project`
- ☐ `SwiftStructures.txt`
- ☐ `Members.txt` — (Only for new teams)
- ☐ (optional) `addendum.pdf` 1-page of anything further than the required submission

Email your submission to hamdan@cs.rwth-aachen.de with subject [iPhone 2015] A03 submission

Grading
We will grade this assignments using the following questions.

- The app is working as expected in all simulations (without warning or errors)?
- The UI does not require a lot/any text input?
- The UI design follows the iOS Human Interface Guidelines?
- The structure of the app follows MVC correctly?
- Answers in `SwiftStructures.txt` are convincing?
- All projects provide modular implementation (use function for concrete tasks instead of a code jam)?

★ 1.0 — Accomplish all "challenge yourself" tasks and clearly went above and beyond what was given in the assignment sheet by improving usability, features, or performance of the implementation.

Incomplete submission will receive at maximum 2.3.
Late submissions *will not be graded*.

Looking forward
For advanced students, the following pointers will shape your mindset for the topic we will discuss in the next lab and beyond this class.

- Should your user uses multiple devices, you may want to use iCloud to store user preferences; see “Designing for Key-Value Data in iCloud”.

- In an advanced applications, you may want to store the current screen that the user is working on together with the data. UIKit provide a more elaborate mechanism to support application state restoration; see “State Preservation and Restoration”.

- If you have a big file to load, you might want to load files asynchronously; see “Techniques for Reading and Writing Files Without File Coordinators”