Assignment 1

MVC and Delegation

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

Description

In this assignment, you will apply MVC design pattern and use delegation to develop a temperature converter.

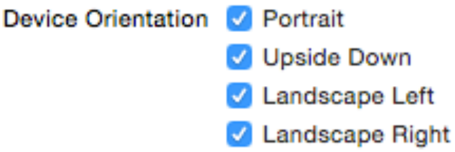
Task

Part 1: Refactoring A01

1. `NSTimer`*:* Use `NSTimer` if you want something to happen at a specific moment in time or to repeat at certain time intervals. `NSTimer` adds tasks to a run loop, and the run loop will execute the task appropriately while keeping the interface responsive.
    a. Remove the notification code from the Clock.app and replace it with an `NSTimer` implementation (Hint: `scheduledTimerWithTimeInterval`). Do not forget to invalidating the scheduled timer in a `deinitializer`.

    b. Move the time updating code to a model class. Declare a property that is continuously updated with the current time. Use key-value observing KOV to update the time label.

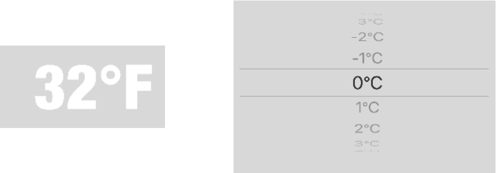
    c. Challenge yourself: Analyze the accuracy of the displayed time and improve it.
2. *Device orientation*: Enable device orientation in all directions. When the device in the landscape, display the time in bigger font (Hint: `supportedInterfaceOrientations` and `UIInterfaceOrientationMask`).
3. *Status bar*: Remove the status bar in your app (Hint: look at the app's info.plist and consider `Status bar is initially hidden`, `View controller-based status bar appearance`). Briefly describe the difference between these two info.plist items. Describe another way (other than manipulating info.plist) to hide the status bar. From a design perspective, provide a 2 line argument why it is best practice not to hide the status bar, and 2 lines to argue why this is justified in the Clock.app.



Device Orientation
☑ Portrait
☑ Upside Down
☑ Landscape Left
☑ Landscape Right

☐ Submit the refactor `Clock Xcode project`.

☐ Use `A01Part1-answers.txt` to 1.c and 3. Submit this file.



Part 2: MVC and delegation

1. *Create an Xcode project "TemperatureConverter1": Use* Single View Application template. The app should allow the user to input a temperature in Celsius and see the converted value in Fahrenheit. Use the Unicode Character 'DEGREE SIGN' (U+00B0) with temperature values.

2. *Build the UI:* Add a `UILabel` and `UIVPickerView` as in the example figure. Disable `Use Size Classes.` But make sure that your app can be simulated on iPhones 4S - 6S Plus and in landscape and portrait orientations. In 2 lines discuss why the picker view is a better choice than a text field + a button. In 2 lines discuss how your ui layout follows iOS Human Interface Guidelines (see S01).

3. *Conforming to protoco*l: Make the view controller adopt UIPickerViewDataSource and UIPickerViewDelegate. Add the picker view as an outlet to the view controller and set the view controller as its data source and delegate. Add a breakpoint to `pickerView:didSelectRow:inComponent:` that generates Log message containing `selected: @row@`.

4. *Ranges*: Use range syntax to initialize a celsiusTemperatureValues array -80 to 80 degrees (Hint: use the `map` function on the array). This array will fill in the picker view.

5. *Model objects:* Your app should have a `TempConverter.swift` class that has a stored property `degreesCelsius` and a computed property `degreesFahrenheit`.

   ☐ Submit `TemperatureConverter1 Xcode project`

6. Duplicate the first project and name it "TemperatureConverter2". Add a ui control to the interface to switch the conversion Fahrenheit <> Celsius. In 2 lines explain your design decision for picking the ui control and its layout in the interface. Choose a suitable range for fahrenheitTemperatureValues.

7. Extend the picker to include one decimal point, user should be able to select 33.3 for example.

8. *Custom operator*: Create 2 unary operators cf and fc that work as follows: cf0 = 32 and fc51 = 11. In the view controller declare a function that takes 2 arguments: user input value from the picker view, and the operator.

9. Change the color of the converted temperature depending on how hot or cold.

10. Challenge yourself: Add an "info" button to the view, and use a UIAlertView to allow the user to select the default temperature (value and unit), save the result using NSUserDefaults. Save the last temperature the user picked in user defaults and restore it when the app is launched.

    ☐ Submit `TemperatureConverter2 Xcode project`

    ☐ Use `A01Part2-answers.txt` to 2 and 6. Submit this file.

Submission

Create a zip archive including the following items

- ☐ `Clock project`
- ☐ `A01Part1-answers.txt`
- ☐ `TemperatureConverter1 project`
- ☐ `TemperatureConverter2 project`
- ☐ `A01Part2-answers.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] A02 submission

Grading

We will grade this assignments using the following questions.

- A01 working as expected in all simulations?
- Answers to `A01Part1-answers.txt` are convincing?
- `TemperatureConverter2` and `TemperatureConverter2` are working with warning or errors?
- All projects apply the MVC model correctly?
- All projects provide modular implementation (use function for concrete tasks instead of a code jam)?
- Answers to `A01Part2-answers.txt` are convincing?

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.

- How can we update the interface to display the actual weather temp?
- How to extend this single view temp converter to a tab based unit converter?
- How to use NSUserDefaults to restore the user's state?