TP 4 - Clock

Adrien Humilière 07/05/2019

Part 1

- In Interface Builder, add a text label for displaying the current time onto the view.
- Apply styles, adjust position, add autolayout constraints, and set the initial text value to **00:00**.
- Some methods of the view controllers are automatically called by the application, following it's lifecycle events, such as `viewDidLoad`. Experiment with generating an explicit console message with `print()` during `viewDidLoad`.
- Run the app, and witness the print message on the console.
- Experiment changing the label text during `viewDidLoad`.

Part 2

- Most of iOS applications follow the MVC pattern. Model, views and controllers have separated roles :

  Model Manages data and only data.

  View Manages the display of informations to the user.

  Controller Picks data from the model, format it and send it to the view for display.
- In that case, we need a model to encapsulate the representation of a clock. Add a `Clock` class to the project.
- Using the Xcode Documentation and API Reference, explore the `Date` class.
- Define and implement a `currentTime` method that will return a `Date` object of the current time. This method should always return a new instance of `Date`.
- For case like this `currentTime` method, Swift provides a feature known as "computed properties" that represent properties whose values are computed each time they are accessed. Replace the currentTime method definition with a computed property.

```swift
var currentTime: Date {
    return ...
}
```

- Declare a `clock` property within the `ViewController` class, with an instance of `Clock` as default value.
- Update `viewDidLoad` to set the label text with the raw `Date` object returned by the `Clock` `currentTime` property.
- Run the app. You may witness that we need to customize the format of the `Date` as a string.
- Using the Xcode Documentation and API Reference, explore the `DateFormatter` class and the `DateFormatterStyle` constants. Use a `DateFormatter` to display a properly formatted time on the screen.

```
let formatter = DateFormatter()
formatter.timeStyle = .short
timeLabel.text = formatter.string(from: clock.currentTime)
```

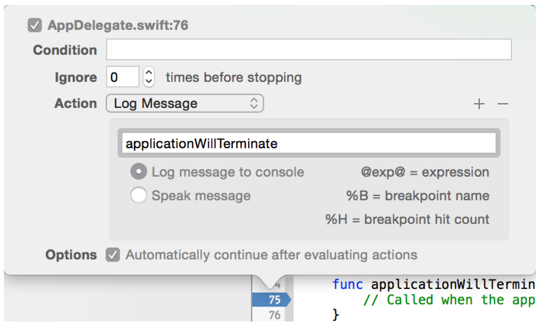
- Run the app, and witness the correctly formatted time on the screen. Experiment changing the simulator locale in various languages/regions (in iOS settings) and run the application again.

Part 3

- Using the Simulator, send the app to the background (⇧+⌘+h), wait until the OS X menu bar time indicator has changed, and bring the app to the foreground. Observe that the time is not current.
- Using the Multitasking Bar (⇧+⌘+h, twice quickly), force quit the app and start it again. Notice the time is now correct. The time is correct only when starting the application.
- Add a `print()` call in `viewDidLoad`.
- Run the app, and observe the Xcode console while repeating the starting, backgrounding, foregrounding and quitting of the app. When does the iOS app seems to execute this `viewDidLoad` method?
- Examine the class declaration for `ViewController` and not that it extends `UIViewController`.
- Using the Xcode Documentation and API Reference, explore the `UIViewController` class reference and notice its life cycle methods.
- Experiment with attempting to set the current time by overriding `viewWillAppear:`.
- Run the application. Observe the Xcode console while foregrounding and backgrounding the app. Notice how `viewWillAppear:` is also not the appropriate lifecycle method.

- Using the Project Navigator, examine `AppDelegate.swift`. The app delegate implements the UIApplicationDelegate protocol and will receive all events corresponding to the application lifecycle. Check this methods in the documentation.

- Instead of adding a print call to all `AppDelegate` methods, use Xcode to add breakpoints that automatically continue after writing a message to the console.



☑ AppDelegate.swift:76

Condition

Ignore 0 times before stopping

Action Log Message

applicationWillTerminate

● Log message to console — @exp@ = expression

○ Speak message — %B = breakpoint name

%H = breakpoint hit count

Options ☑ Automatically continue after evaluating actions

```
func applicationWillTermin
    // Called when the app
}
```

- Observe the Xcode console while starting, backgrounding, foregrounding, quitting and restarting the app.

- Choose the event that suite best for the feature of updating the currently displayed time. However, the controller should be responsible for communicating with the view, and writing view-related code in the `AppDelegate` violate the separation of concerns of the MVC pattern.

Part 4

- The `NotificationCenter` allow us to observe system and application events, from anywhere in the application (this is not the same as user notifications).

- Explore the `NotificationCenter` class documentation, its `default` class method and the `addObserver:selector:name:object:` method.

- Register the controller as an observer in `viewDidLoad`. This registration will make the system call a `updateTimeLabel` when the application will enter foreground.

```
NotificationCenter.default.addObserver(self,
    selector: Selector("updateTimeLabel"),
    name: NSNotification.Name.UIApplicationWillEnterForeground,
    object: nil)
```

- Implement the `updateTimeLabel` method.

- Refactor `viewWillAppear` to use `updateTimeLabel`.

- Run the app and use the Simulator to send the app to the background (⇧+⌘+h). Wait until the OS X menu bar time indicator has changed, and bring the app to the foreground. Observe that the time is current.
- Experiment with using an invalid selector name when registering an observer in viewDidLoad. Run the app, send the app to the background, bring the app to the foreground, and observe the app crashing. Restore the correct selector name.
- It is a best practice to unregister observers when an application quits or is "destroyed" from memory Unregister the observer in a deinitializer.

```
deinit {
    NSNotificationCenter.default.removeObserver(self)
}
```

- The app delegate has no controller-related responsibilities, and the view controller encapsulates the coordination of updating the view.

Part 5

- Imagine a real user of the Clock application. What is the main flaw of the app? time is only updated when bringing the app into the foreground, and the displayed time does not continuously change while the app is running.
- Add a new controller property for an optional `Timer`. The `timer` property is declared as an optional, because the `ViewController` initializer will not initialize the property.
- Explore the `Timer` class documentation and its `scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:` class method.
- Replace the observer registration in `viewDidLoad` with the creation of a `Timer` that will call `updateTimeLabel` every second.

```
Timer.scheduledTimer(timeInterval: 1.0,
    target: self,
    selector: Selector("updateTimeLabel"),
    userInfo: nil,
    repeats: true)
```

- Modify the `updateTimeLabel` method's format of the displayed time (`formatter.timeStyle`), such that it displays seconds. Choose the relevant time style.
- Replace the observer removal in the deinitializer with an invalidation of the timer.

```swift
deinit {
    if let timer = self.timer {
        timer.invalidate()
    }
}
```

- Run the app and observe that it continuously displays the current time.

Part 6

- The iOS Human Interface Guidelines, or "HIG", describes best practices for consistent, high quality user experience. Explore this documentation : https://developer.apple.com/ios/human-interface-guidelines. It should be followed for your project.

- It is best practice to not hiding the iOS status bar, but we should make the design decision to hide the status bar for this app in order to remove the redundancy of the status bar's time display.

- An individual view controller can override a `prefersStatusBarHidden` method, and the status bar can be disabled application-wide through configuration.

- Using the Project Navigator, select `Info.plist`, add a new `Boolean` item called `Status bar is initially hidden` and assign it the value `YES`. Add a second `Boolean` item called `View controller-based status bar appearance` and assign it the value NO.

- Run the app, and observe how the status bar is now hidden.