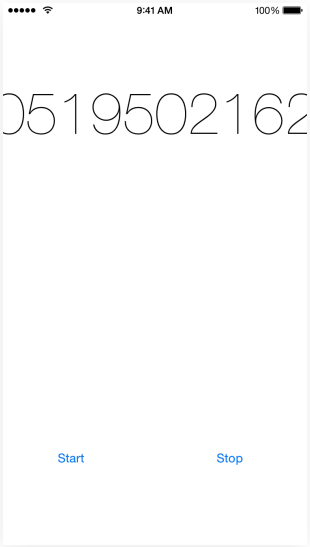
Stopwatch
Lesson 5

Description

Use an `NSTimer` to frequently update the elapsed time label.



Learning Outcomes

- Discover how an `NSTimer` is used to execute methods outside of the interface run loop.
- Relate the concept of a selector to a method name represented as a `String` value.
- Understand the importance of `NSTimer` invalidation.
- Practice writing conditional statements and accessing object properties.

Vocabulary

run loop	`NSTimer`	selector

Materials

- **Stopwatch Lesson 5** Xcode project
- **Run Loops and NSTimer** presentation

Opening

How can we continuously update the display of the elapsed time while the stopwatch is running?

Agenda

- Review the use case of continuously updating the label, and the technical flaw of using a loop in the `startButtonTapped:` controller action to continuously update the label.
- Present the concept of run loops and the `NSTimer` class.
- Using the Xcode Documentation and API Reference (⇧⌘0), explore the `NSTimer` class reference.
- Fix the previous implementation of `startButtonTapped:` to use an `NSTimer` that will call `updateElapsedTimeLabel:` once a second.

```
@IBAction func startButtonTapped(sender: UIButton) {
   println("Starting stopwatch")
   NSTimer.scheduledTimerWithTimeInterval(1.0, target: self,
     selector: "updateElapsedTimeLabel:", userInfo: nil, repeats: true)
   stopwatch.start()
}
```

- Discuss the arguments passed to the `scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:` method.
- Explain what a selector is, and how Swift allows the passing of a `String` that corresponds with the name of the method that will be invoked.
- Implement the `updateElapsedTimeLabel:` method.

```
func updateElapsedTimeLabel(timer: NSTimer) {
   println("updating...")
   elapsedTimeLabel.text ="\(stopwatch.elapsedTime)"
}
```

- Run the app (⌘R), tap the Start button, and observe the elapsed time label changing.
- Tap the Stop button, observe the label change to **0.0**, and observe how the console (⇧⌘C) indicates that the `NSTimer` is still repeatedly calling `updateElapsedTimeLabel:`.
- Discuss how the `updateElapsedTimeLabel:` method should not update the label if the `Stopwatch` is not running.
- Update the implementation of `updateElapsedTimeLabel:` to check the state of the `Stopwatch` model.

```
func updateElapsedTimeLabel(timer: NSTimer) {
   println("updating...")
   if stopwatch.isRunning {
      elapsedTimeLabel.text = "\(stopwatch.elapsedTime)"
   }
}
```

- Run the app (⌘R), tap the Start button, tap the Stop button, and observe that the label does not update while the `Stopwatch` is stopped. Observe how the console (⇧⌘C) indicates that the `NSTimer` still repeatedly calls `updateElapsedTimeLabel:`.
- Explain the best practice of invalidating an `NSTimer` when it is no longer needed, how an `NSTimer` passes itself to the designated selector, and the significance of the timer parameter of `updateElapsedTimeLabel:`.
- Using the Xcode Documentation and API Reference (⇧⌘0), explore the `NSTimer invalidate` method.
- Invalidate the `NSTimer` in `updateElapsedTimeLabel:`.

```
func updateElapsedTimeLabel(timer: NSTimer) {
   println("updating...")
   if stopwatch.isRunning {
      elapsedTimeLabel.text = "\(stopwatch.elapsedTime)"
   } else {
      timer.invalidate()
   }
}
```

- Run the app (⌘R), tap the Start button, observe the console (⇧⌘C) and the elapsed time label changing. Tap the Stop button, and notice the console output has ceased.

Closing

What would happen if we changed the numeric time interval passed to `scheduledTimerWithTimeInterval:`? What would happen if we used an invalid selector name for the `selector:` argument?

Modifications And Extensions

- Investigate the purpose of the `userInfo:` parameter of `scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:`.

Resources

NSTimer Class Reference https://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSTimer_Class/Reference/NSTimer.html

Cocoa Application Competencies for iOS: Main Event Loop https://developer.apple.com/library/ios/documentation/General/Conceptual/Devpedia-CocoaApp/MainEventLoop.html

Threading Programming Guide: Run Loops https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/Multithreading/RunLoopManagement/RunLoopManagement.html

Using Swift with Cocoa and Objective-C: Objective-C Selectors https://developer.apple.com/library/ios/documentation/Swift/Conceptual/BuildingCocoaApps/InteractingWithObjective-CAPIs.html#//apple_ref/doc/uid/TP40014216-CH4-XID_40

The Swift Programming Language: Conditional Statements https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/ControlFlow.html#//apple_ref/doc/uid/TP40014097-CH9-ID127

Start Developing iOS Apps Today: Finding Information https://developer.apple.com/library/ios/referencelibrary/GettingStarted/RoadMapiOS/FindingInformation.html