iPhone Application Programming Lab 2: MVC and Delegation + A01 discussion

Nur Al-huda Hamdan
Media Computing Group
RWTH Aachen University

Winter Semester 2015/2016

http://hci.rwth-aachen.de/iphone

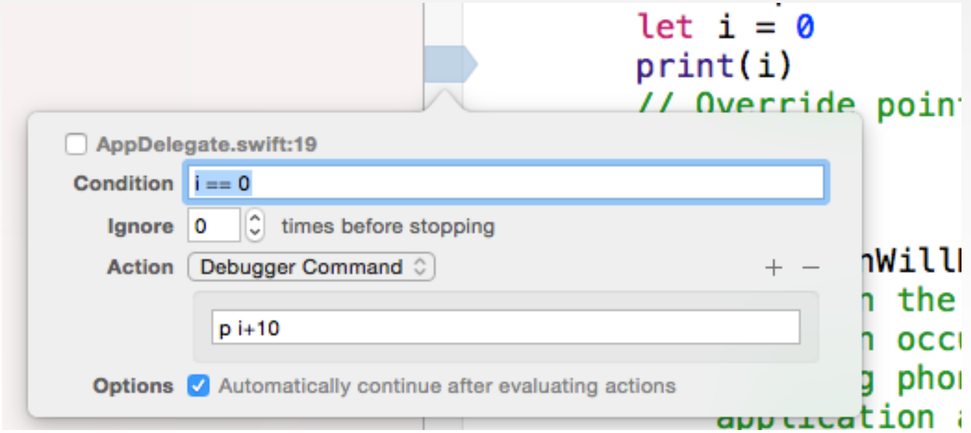
Media Computing Group | RWTH AACHEN UNIVERSITY

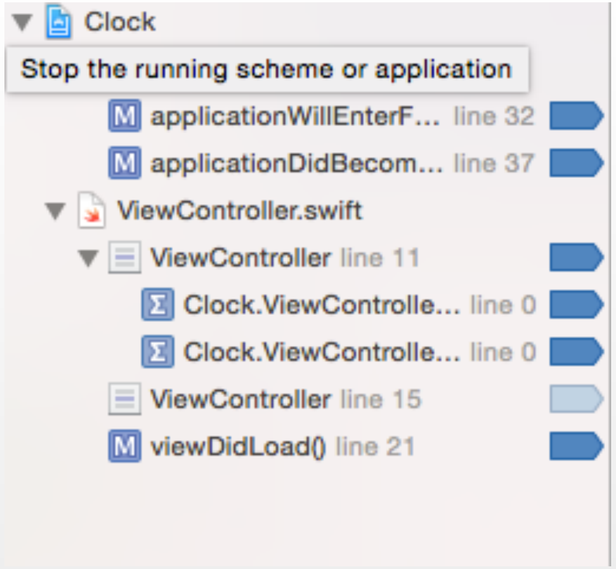
Learning Objectives

- Discuss A01 + demo
  - Concepts: debugging with breakpoints, app delegate, life cycle and transition states, notifications, deinitialization, object graph, optional chaining, type casting
- MVC model
  - Concepts: MVC, computed property, NSTimer
- Delegation and protocols + demo
- Introduce A02

Breakpoints for Debugging

- Better than print(), do not clutter code, and can be disable/enabled
- Customizable: can specify a condition for a break to happen; the actions at breakpoints, e.g., included log message and debug command, and specify "Automatically continue after evaluating"
- Breakpoint navigator gives you overview and control on breakpoints per class in the app
- More on Xcode debugging tools (reading assignment)





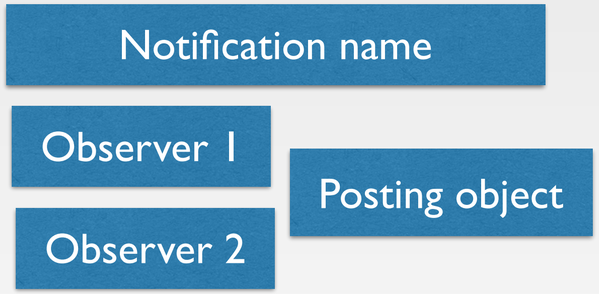
App Life Cycle and Transition States

- Two levels of states in the app life. When a state changes, an event is fired
  - App level in AppDelegate
    - UIApplication object reports to its delegate (AppDelegate) the app transitions and incoming push notifications
    - App delegate does app initializations, creates the root view controller on launch, posts notifications at state transition
    - App delegate has no view controller related responsibilities
  - **View controller level**
    - View controller encapsulates the coordination of updating the view
    - Implements UIViewController state transition functions
    - State transitions are evaluated from the view controller perspective

When the Clock.app is launched
viewDidLoad
viewWillAppear
applicationDidBecomeActive
viewDidAppear
If I send Clock.app to the background (⇧⌘H)
applicationWillResignActive
applicationDidEnterBackground
If I bring Clock.app to the foreground
applicationWillEnterForeground
applicationDidBecomeActive
If I try to kill Clock.app using the Multitasking Bar (⇧⌘H, twice quickly) while it is in the background
SIGKILL exception (Xcode bug, nothing you can do about it)
If I try to kill Clock.app using the Multitasking Bar (⇧⌘H, twice quickly) while it is in the foreground
applicationWillResignActive
If I open Clock.app (instead of killing it)
applicationDidBecomeActive
If I open another app instead
applicationDidEnterBackground
If I kill Clock.app
applicationDidEnterBackground
viewWillDisappear
viewDidDisappear
applicationWillTerminate

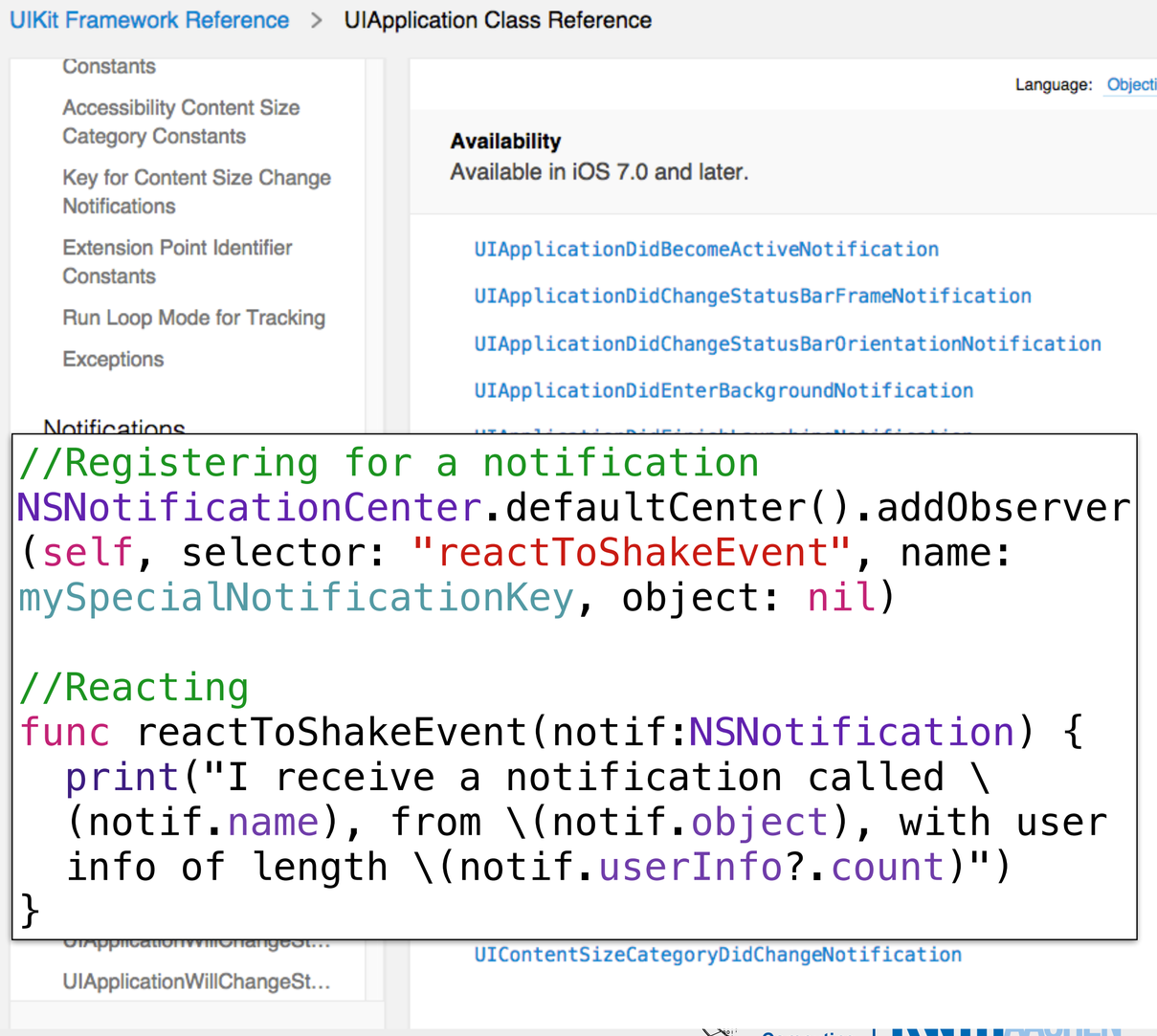
Responding to Events & Object-to-Object

- How to achieve object-to-object communication on special events, e.g., app delegate tells view controller when applicationWillEnterForeground event happens
- Four main design patterns: **notifications**, KVO, **delegation**, target-action



Notifications

- When an event occurs, an object posts notification in a broadcasting fashion (doesn't know who wants it)
- An object (observer) registers itself to receive a notification (by name) for some event
- The observer implements a function to respond to the event
- The observer should remove itself if it's no longer listening for notifications (deinit, called when the object will dealloc)
- Application notifications are NOT <u>push notifcations</u>




```
//Registering for a notification
NSNotificationCenter.defaultCenter().addObserver
(self, selector: "reactToShakeEvent", name:
mySpecialNotificationKey, object: nil)

//Reacting
func reactToShakeEvent(notif:NSNotification) {
  print("I receive a notification called \
  (notif.name), from \(notif.object), with user
  info of length \(notif.userInfo?.count)")
}
```

Object Graph, Optional Chaining, Type Casting

Using object graph is not recommended

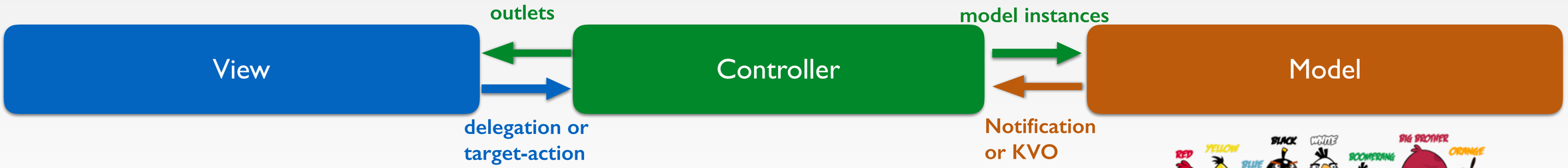
- This is optional chaining when one object or more are optional
  - self.window?.rootViewController?.view.subviews
  - If one of the options is nil, this fails graceful (no run time error)
  - If all optionals are set, the chain return an optional (even if the object in request, e.g., subviews, is not optional)
  - self.window?.rootViewController?.view.subviews! `//compiler error, subviews is not of type optional`
- This is type casting
  - Upcasting: object `as` superclass, or with a literal expression, e.g., 10 `as Int`
  - Downcasting
    - let subclassInstance = object `as`! subclass downcasts + force unwarp OR runtime error
    - if let subclassInstance = object `as`? subclass {…}, downcasts or nil

Question: Why isn't view declared as an optional?

```
  //Using the object graph to access deep objects hierarchies (optional chaining)
   //as means type casting the subview array to an array of UIView
for subview in (self.window?.rootViewController?.view.subviews)! as [UIView]
 {
       //type casting the subview to UILable
      if let labelView = subview as? UILabel
      {
        let formatter = NSDateFormatter()
        formatter.timeStyle = .MediumStyle
         labelView.text = formatter.stringFromDate(NSDate())
       }
 }
```

Type checking: object `is` subclass or variable `is` Double

MVC



Interpret user actions and communicate changes to models

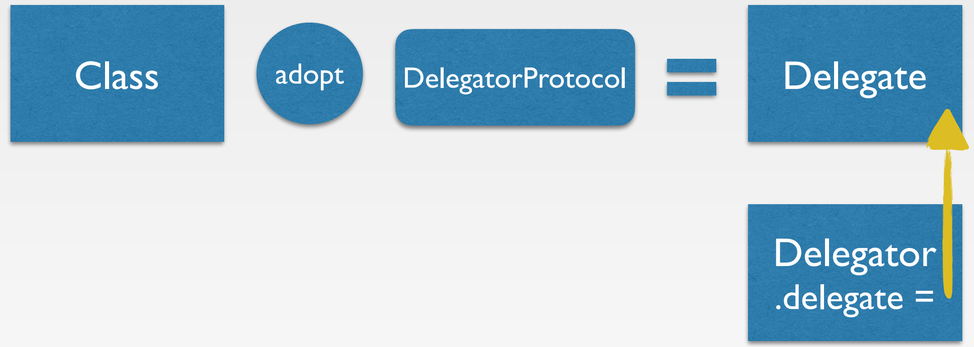
Decides how model data is displayed in the views

Each view controller is responsible of one screen of views

Representation of the models
Event handling (responds to users actions)

class RedBird:Bird
Attributes are properties let value =10
Behaviors/rules are methods func fly ()

Delegation



- Similar to notifications, delegation allows the *delegate* (observer) to respond to events on behalf of the *delegator* (posting object)
- Instead of registering for notifications, the delegate has to assign itself as the delegator's delegate and declare that will implement the required methods (*conform to protocol*)
- The main value of delegation is that it allows you to easily customize the behavior of several objects in one central object
- A delegate can be a data source for the delegator and respond to requests of data

NameOfProtocol (more like an interface, used instead of multiple inheritance)

NameofProtocolDeleagte
NameofProtocolDataSource

Protocols

- A protocol defines a blueprint of (instate/type) methods, (instance/type) properties that suit a particular task or piece of functionality
- The protocol can then be adopted by a class/structs/enum and provide actual implementation of those requirements (conform to that protocol)
  - Some elements of the protocols can be tagged as optional
- Any conforming type for the fullName protocol must be able to provide a full name for itself, any FullyNamed type must have a gettable instance property called fullName of type String.
- The RandomNumberGenerator protocol does not make assumptions how each random number will be generated, but requires the generator to provide a standard way to generate a new random number
- Swift reports an error at compile-time if a protocol requirement is not fulfilled

```
protocol FullyNamed {
    var fullName: String { get }
}

struct Person: FullyNamed {
    var fullName: String
}
let john = Person(fullName: "John Appleseed")
```

```
protocol RandomNumberGenerator {
    func random() -> Double
}

class LinearCongruentialGenerator:
RandomNumberGenerator {
    var lastRandom = 42.0
    let m = 139968.0
    let a = 3877.0
    let c = 29573.0
    func random() -> Double {
        lastRandom = ((lastRandom * a + c) % m)
        return lastRandom / m
    }
}
let generator = LinearCongruentialGenerator()
print("Here's a random number: \(generator.random())")
```

A02 Temperature Converter

- Part 1
  - Change A01 to work with an NSTimer as in the demo
  - Apply MVC to A01 by moving the NSTimer to a model class and key-value observing as a communication method
- Part 2
  - Implement a temperature converter app with a UIPickerView
  - Apply MVC and delegation
  - Create your custom operators for (celsius to fahrenheit) and (fahrenheit to celsius)