User Interfaces

Lecture 20

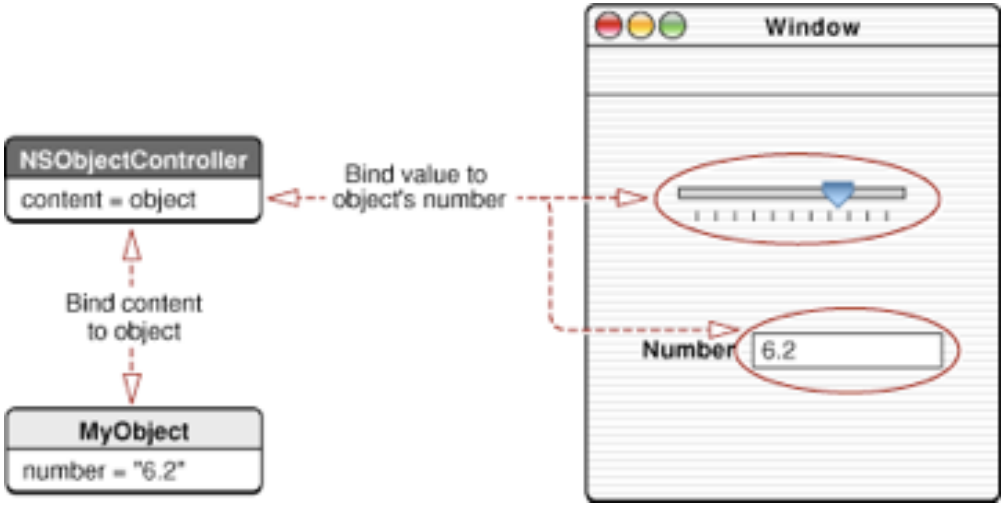
Bindings

Hamza Bennani
hamza@hamzabennani.com

September 4, 2018

Binding

- Often you want a GUI to directly control an instance variable
  - You want the controller to bind the view data to the model data
- This can be done using target/action and outlets, or . . .
  - You can use Cocoa bindings to generate all the "glue code" automatically



key-Value Coding

- Key-Value Coding (KVC) allows us to get and set instance variables using key strings
  - To get a variable use valueForKey:
  - To set a variable use setValue:forKey:
- The key string is the same name as the instance variable
  - (Think about the difference here: variable names are known at compile time, but the string values are only known at runtime.)

KVC

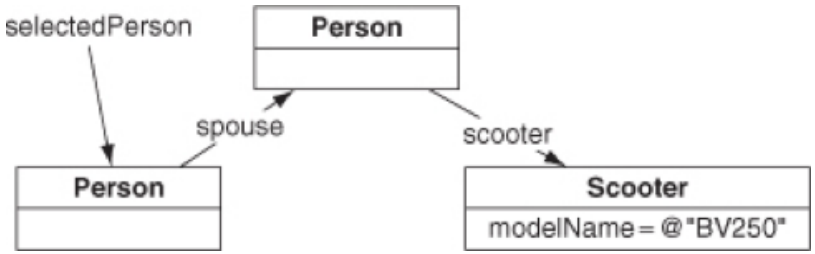
```
14 @objcMembers class ClassA : NSObject {
15     @objc var hm : Int = 0;
16 }
17
18 let anObject = ClassA();
19 // set hm using standard setters
20 anObject.hm = 27;
21 print("hm value using standard setter \(anObject.hm)" );
22
23 // set hm using KVC
24 anObject.setValue(27, forKey: "hm");
25 let val = anObject.value(forKey: "hm") as! NSObject;
26 print("using KVC \(val)" );
```

KVC

- NSObject implements Key-Value Coding
- It calls key-value compliant setters/getters:
  - Key-value coding automatically wraps non-objects as NSNumber or NSValue
  - More of a concern for Objective-C than for Swift, which will usually take care of things for you
- Key-Value Coding:
  - Enables application scripting
  - Underlies variable binding to GUI elements
  - Can help you simplify your code

Key Paths

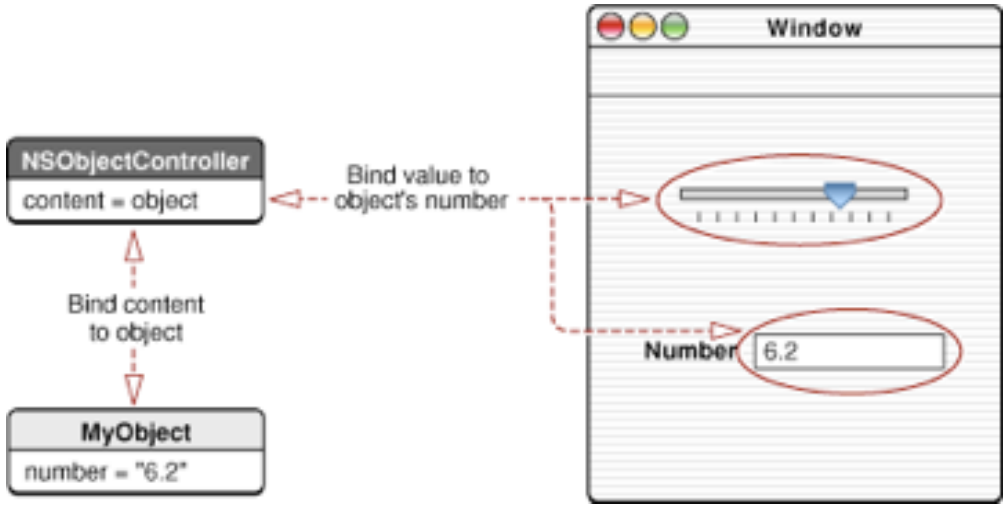
- Key paths can be used to access instance variables indirectly
- Key paths consist of keys separated by dots




```
95 let model = selectedPerson.value( forKeyPath:
       "spouse.scooter.modelName") as! NSObject;
```

Binding

- Key-Value Coding allows you to get and set instance variables in your model or view (This is half of the controller glue code)
- You also need to know when something has changed (i.e., not by you)



Key-Value Observing

- Key-Value Observing (KVO) allows notifications to be set up when a variable (specified by a key string) changes
- To register an observer to be notified when an instance variable of data changes, call the addObserver:forKeyPath:options:context: method on the data-holding object
- To get the notification, the observer object must implement the method:

```
override func observeValue(forKeyPath keyPath: String?, of
object: Any?, change: [NSKeyValueChangeKey : Any]?,
context: UnsafeMutableRawPointer?)
```

Key-Value Observing

```
34 class ClassB : NSObject {
35     // b can receive notification from keyvalue observing
36     override func observeValue(forKeyPath keyPath: String?,
37                                of object: Any?, change:
                                       [NSKeyValueChangeKey : Any]?,
38                                context: UnsafeMutableRawPointer?) {
39         /// check the right reason. fido updated
40         if keyPath!.compare("fido")==ComparisonResult.orderedSame {
41             // we care about the newkey value in the change
                   dictionnary
42             if let newValue = change?[.newKey] as? NSObject {
43                 print("The value is \(newValue)")
44             }
45         }
46     }
47 }
```

Key-Value Observing

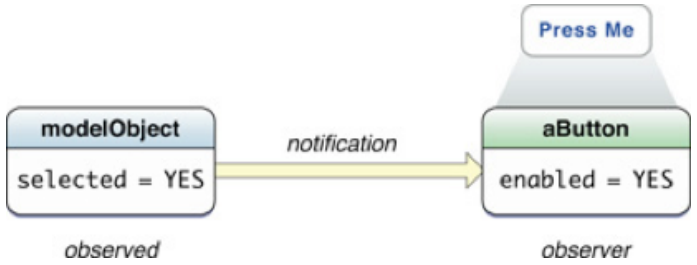
```
49 class ClassC : NSObject {
50     @objc dynamic var fido : Int = 10
51     @objc dynamic var fido2 : Int = 10
52 }
53 let dataObject = ClassC()
54 let observerObject = ClassB()
55 dataObject.addObserver(observerObject, forKeyPath: "fido",
56                        options:  NSKeyValueObservingOptions.new,
                               context: nil)
57 //set fido using standard setters
58 dataObject.fido = 9
59 dataObject.removeObserver(observerObject, forKeyPath: "fido")
60 dataObject.fido = 90
```

KVO

- KVO also implemented at the NSObject level
- However, changes to the variable must occur via an appropriate set accessor
- Objective-C, for single variable: setValue:
  - For NSArray: setValue:forKey:
  - For key path: setValue:forKeyPath:
- In Swift, use the dynamic keyword on properties
- Notifications can also be produced manually via willChangeValueForKey: and didChangeValueForKey:

KVO

- Key-value observing notifications are not Notifications (were NSNotifications)
- Not all classes allow key-value observing
- Only those that support the NSKeyValueCoding informal protocol



Advanced KVC Collections

- What if the instance variable is a collection?
  - You can use key-value coding on a proxy
- For an indexed collection
  - To get a proxy use mutableArrayValueForKey:
  - The proxy provides methods: count:, objectAtIndex:, insertObject:atIndex:, and removeObjectAtIndex:.
- For an unordered collection
  - To get a proxy use mutableSetValueForKey:
  - The proxy provides methods: count:, addObject:, and removeObject:.

Advanced KVC Collections

- For a class that inherits from NSObject with properties that are single attributes, or a to-one relationship:
  - KVC support is already provided by NSObject
  - Ensure the variable in question has set<Name> setter and <name> getter, where <name> is the variable name
- Otherwise, you must implement the methods from the NSKeyValueCoding informal protocol
  - To-one relationship: setValue:ForKey, valueForKey:, validateValue:forKey:error:
  - To-many relationship: dictionaryWithValuesForKey:, mutableArrayValueForKey:, mutableOrderedSetValueForKey:, mutableSetValueForKey:, etc.

Advanced KVC Collections

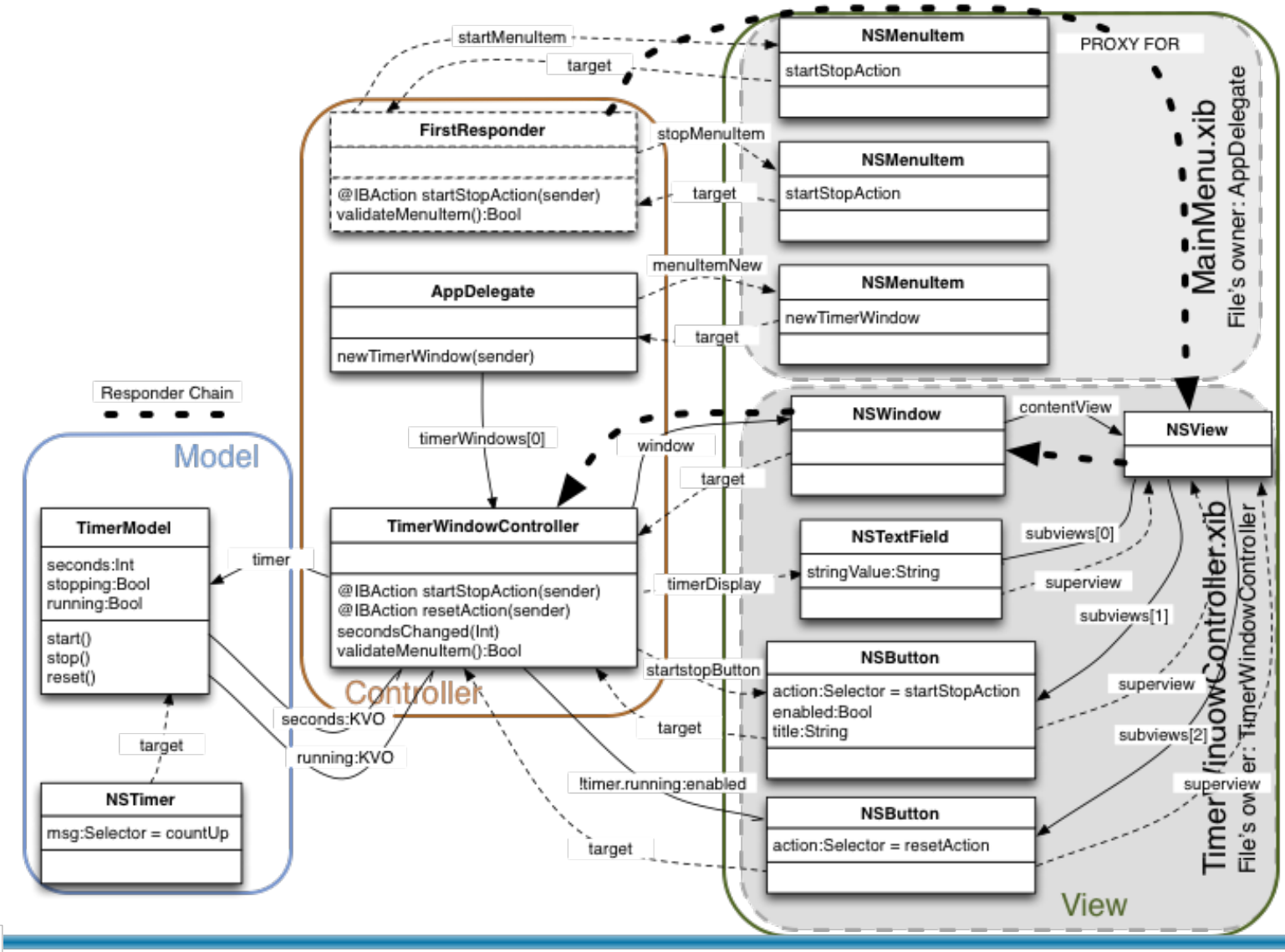
- You can use simple operators with key-value strings
- @avg, @count, @max, @min, @sum
- There are also operators for sets of objects
- @distinctUnionOfObjects, @unionOfObjects, @distinctUnionOfArrays, @unionOfArrays, @distinctUnionOfSets

Summary

In this lecture we learned how to bind data between MVC's model and MVC's view:

- KVC - key value coding, ability to get a variable value at runtime by referring to its name (encoded as a string)
- KVO - framework for notification of an object interested in change in value of another object's instance variable
- Binding - connecting of model data and view controls using KVC and KVO (Note: above features are often from NSObject)

KVO


startMenuItem
target
NSMenuItem
startStopAction
PROXY FOR
MainMenu.xib
File's owner: AppDelegate
FirstResponder
@IBAction startStopAction(sender)
validateMenuItem():Bool
stopMenuItem
target
NSMenuItem
startStopAction
menuItemNew
AppDelegate
newTimerWindow(sender)
target
NSMenuItem
newTimerWindow
Responder Chain
Model
timerWindows[0]
window
target
NSWindow
contentView
NSView
TimerModel
seconds:Int
stopping:Bool
running:Bool
start()
stop()
reset()
timer
TimerWindowController
@IBAction startStopAction(sender)
@IBAction resetAction(sender)
secondsChanged(Int)
validateMenuItem():Bool
NSTextField
stringValue:String
subviews[0]
superview
timerDisplay
subviews[1]
NSButton
action:Selector = startStopAction
enabled:Bool
title:String
startstopButton
superview
Controller
seconds:KVO
target
subviews[2]
target
running:KVO
!timer.running:enabled
superview
NSTimer
msg:Selector = countUp
NSButton
action:Selector = resetAction
target
TimerWindowController.xib
File's owner: TimerWindowController
View