User Interfaces

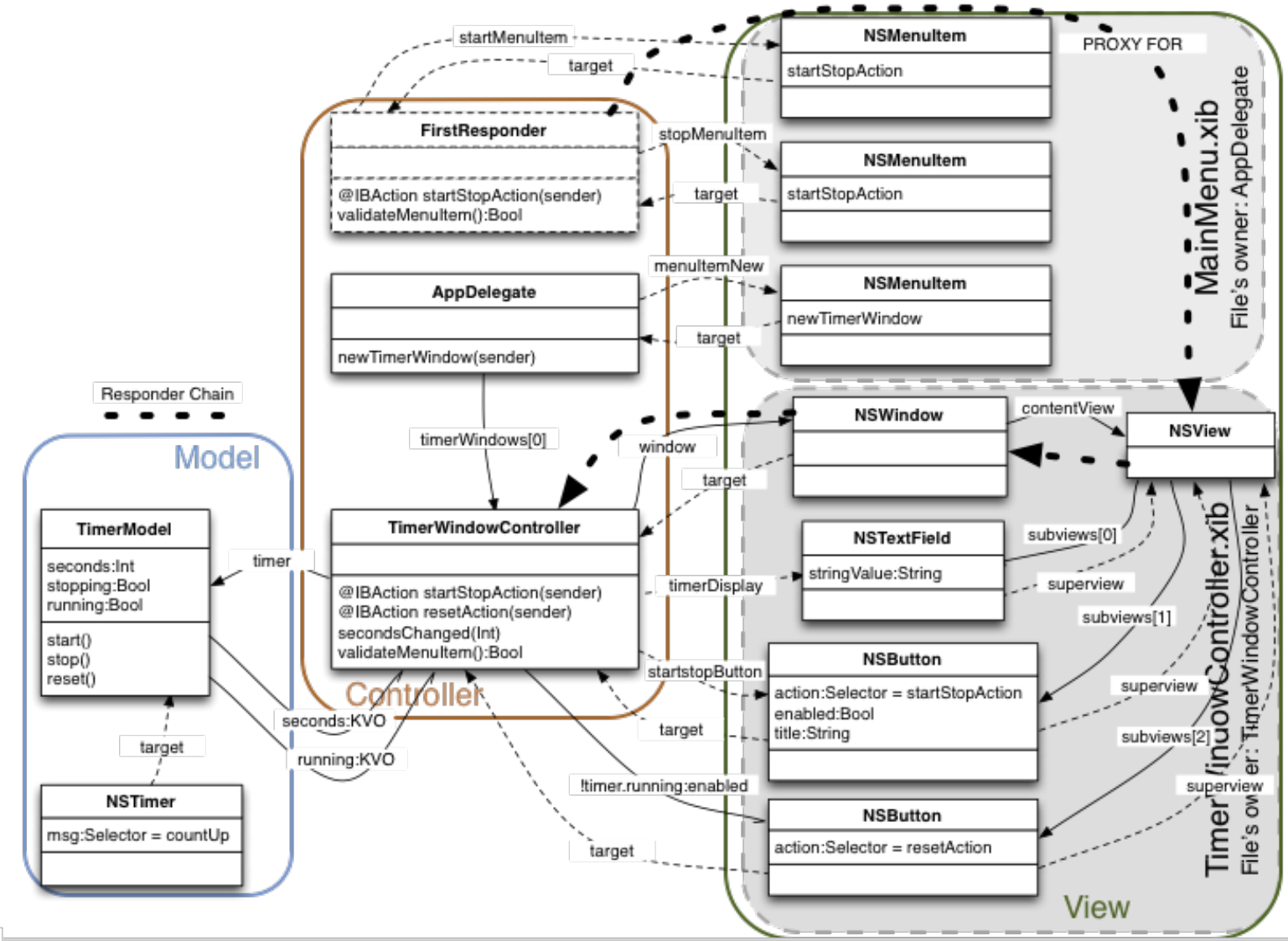
Lecture 21

Cocoa: Controllers & Undo/Redo

Hamza Bennani
hamza@hamzabennani.com

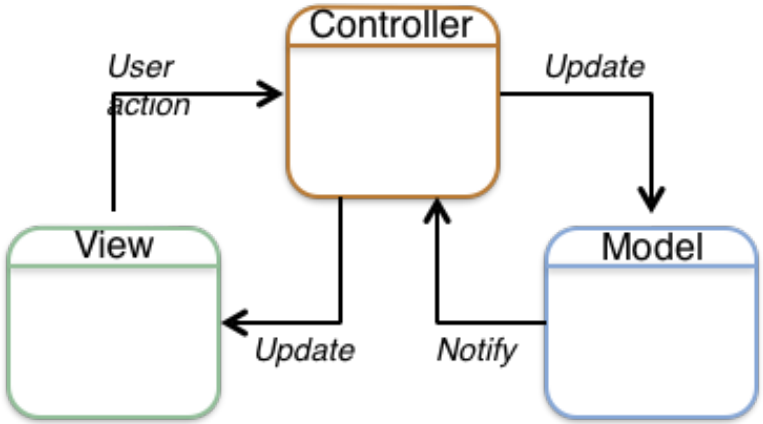
September 25, 2018

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

Model-View-Controller

Recall that the MVC pattern is a good way to organise your application code

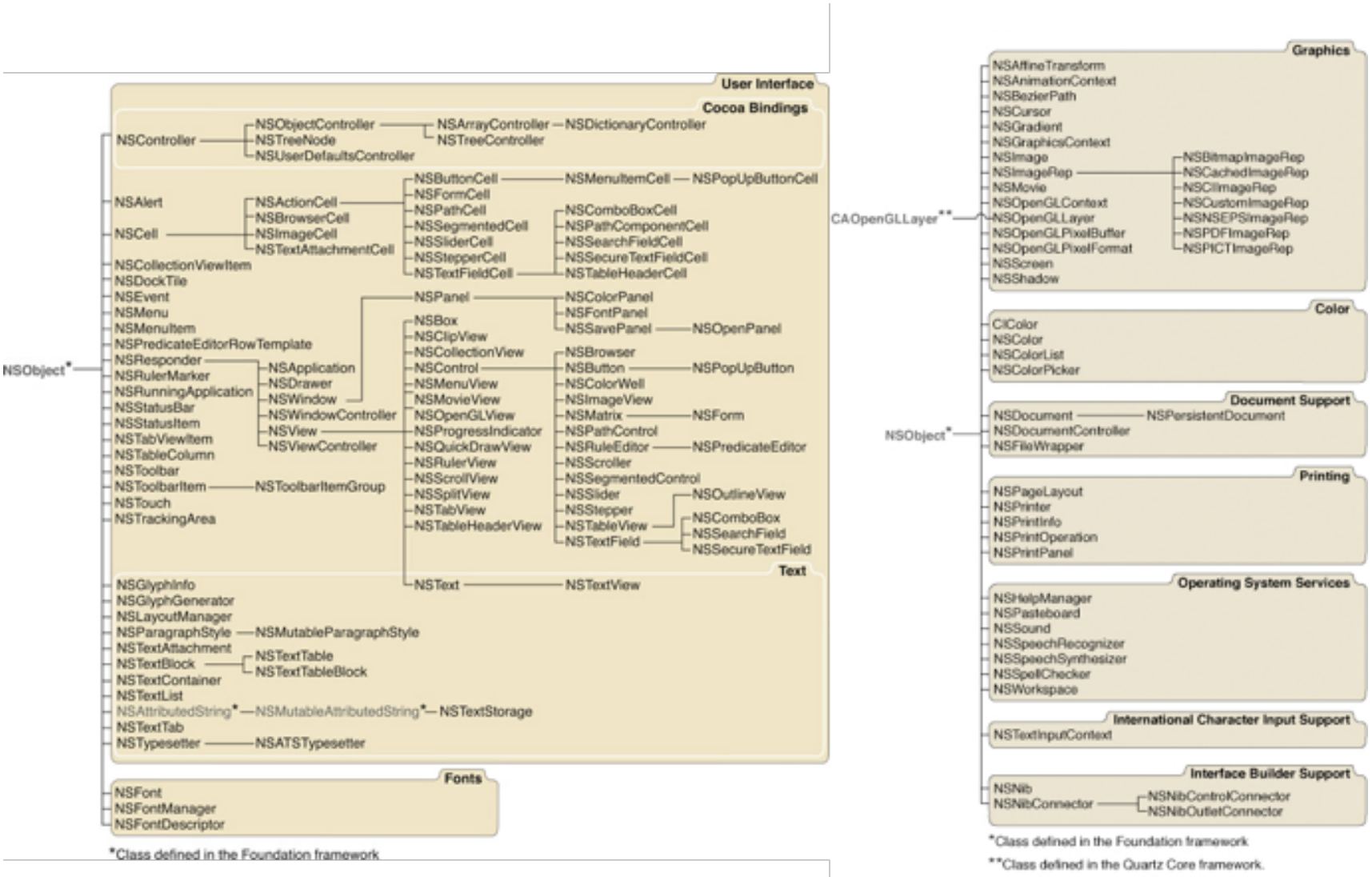


The AppKit Framework provides several controller classes that help with this task

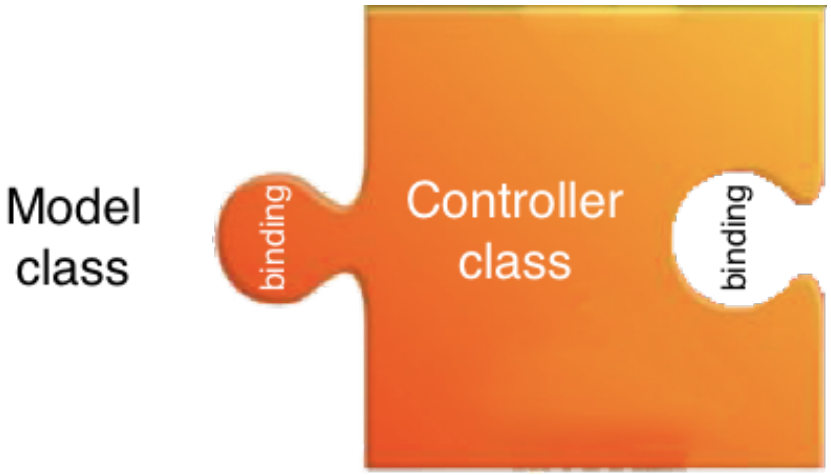
NSController

- Don't confuse with **NSControl**
- AppKit framework provides abstract **NSController** class that has several bindings-compatible classes
  - Are also other controller classes: NSViewController, NSWindowController, NSDocumentController
- Controllers offer the following advantages:
  - Less code (if using bindings)
  - Takes care of selection/editing for you (e.g., text fields in tables, aborted input, committing edits to model, etc.)
  - Controllers are compatible with other advanced features, such as undo/redo and Core Data

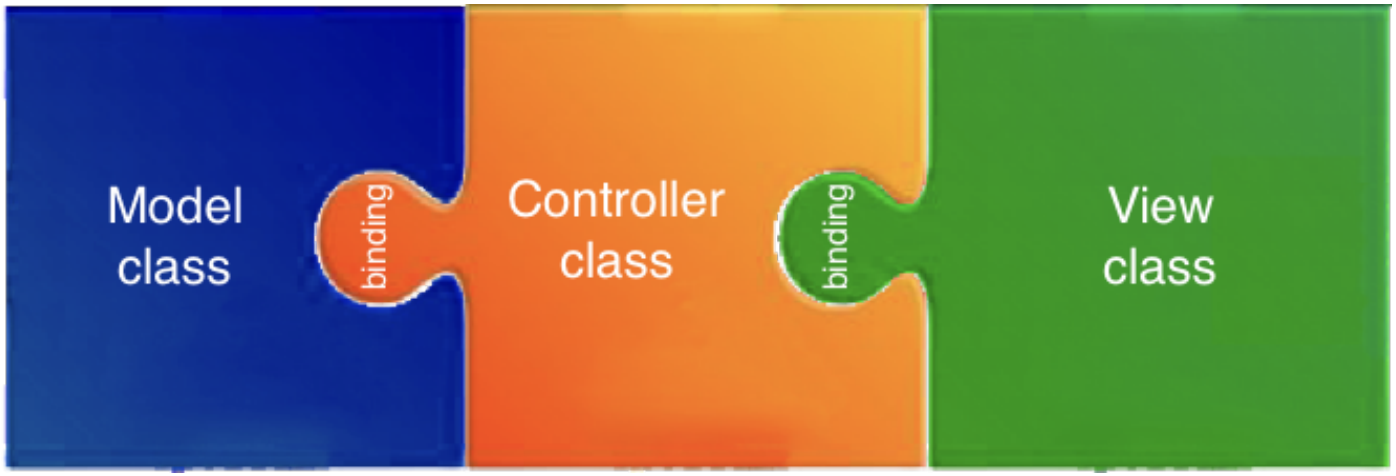
NSController



Controllers



Controllers

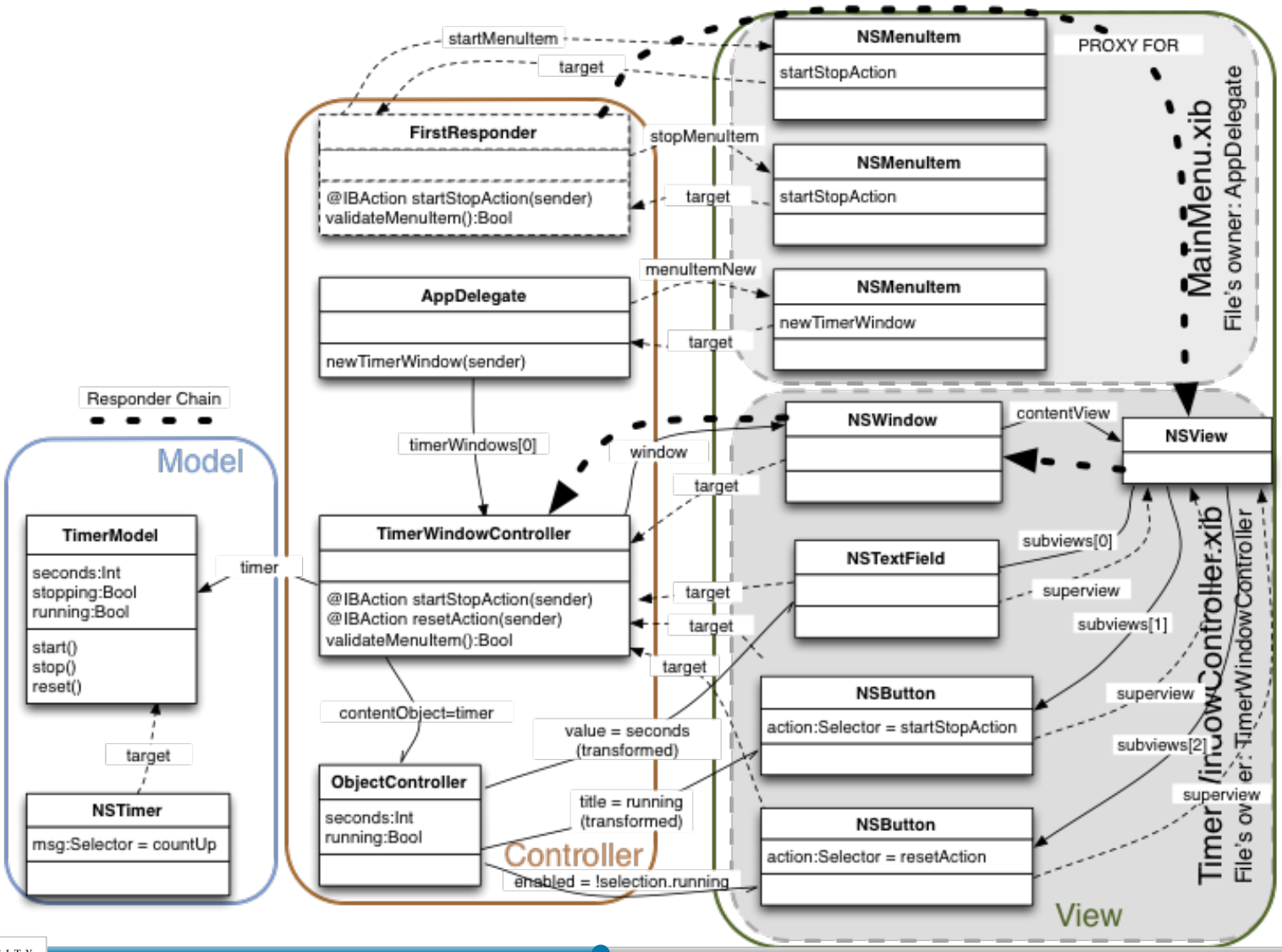


Using Controllers

- Step 1. Set Controller's Content:
  - Set object class (in Interface Builder's Attribute Inspector or by calling setObjectClass:)
  - Bind content (to contentObject)
- Step 2. Connect View to Controller:
  - Controller key points to content (e.g., **arrangedObjects** or **selection**)
  - Model key is key path into object
  - Complete key path is "<controller key>.<model key>"

Bindings can be done in Interface Builder or programmatically using bind:toObject:withKeyPath:options:

Timer Object Controller



Undo/redo

- Undo/Redo is handled by UndoManager in the Foundation framework
- An instance of UndoManager corresponds to two stacks for undo and redo actions
- To use UndoManager:
  1. Decide what should be undo-able
  2. Implement an "inverse" message for every message that causes an undo-able action
  3. Register your messages with the UndoManager

Undoable actions

- Not all actions are undoable
- Document/Image changes should be undoable:
  - Changes to the state of a document, e.g., inserts, removals, etc.
- Application changes should **NOT** be undoable:
  - Selections, changing modes (e.g., tool types, colours)
  - Changes to interface (e.g., font size, window size, preferences)

Undo/Redo Messages

Each undoable action should correspond to a message:

Make it **hotter** by 5° (makeItHotter)

For each undoable action there has to be a reverse message:

Make it **colder** by 5° (makeItColder)

How UndoManager Works

- Suppose the user does the following in the Hotter/Colder application:
  1. makeItColder
  2. makeItColder
  3. makeItHotter

How UndoManager Works

- Next the user invokes ‘undo’
  - Message on the top of the undo stack gets executed
  - The opposite message goes onto the redo stack

How Undo Manager Works

- Next the user invokes ‘undo’ again
  - Message on top of undo stack gets executed
  - The opposite message goes onto the redo stack
- If the user invokes ‘redo’, opposite messages go from the ‘redo’ to the ’undo’ stack

Invocation

- Undo/redo messages can be either "simple":
  - Callback to a selector with single argument with: “func registerUndo(withTarget: Any, selector: Selector, object: Any?)" (selector is called for undo).
    - Or with the potential for passing more state:

```
@IBAction func makeItHotter(_ sender: AnyObject) {
    thrmst.tmp+=1
    undomngr.setActionName("hotter")
    (undomngr.prepare(withInvocationTarget: self) as
        AnyObject).makeItCooler(self)
}
@IBAction func makeItCooler(_ sender: AnyObject) {
    thrmst.tmp-=1
    undomngr.setActionName("cooler")
    (undomngr.prepare(withInvocationTarget: self) as
        AnyObject).makeItHotter(self)
}
```

Labelling Undo/Redo actions

UndoManager also lets you describe the effect of undo to keep the user informed (this is displayed in the Edit menu's items).

```
@IBAction func makeItHotter(_ sender: AnyObject) {
    thrmst.tmp+=1
    undomngr.setActionName("hotter")
    (undomngr.prepare(withInvocationTarget: self) as
        AnyObject).makeItCooler(self)
}
@IBAction func makeItCooler(_ sender: AnyObject) {
    thrmst.tmp-=1
    undomngr.setActionName("cooler")
    (undomngr.prepare(withInvocationTarget: self) as
        AnyObject).makeItHotter(self)
}
```

Summary

Controllers

- are objects whose purpose is to bind model data and view controls (internal glue logic is already pre-defined: can be bound through Interface Builder)
- NSController - abstract class
  - **NSObjectController**
  - **NSArrayController**

Und/Redo actions

- are registered with undo manager by providing the invocations for the opposite actions
- **UndoManager** - a class implementing the undo and redo stacks - typically one per model

Timer App Undo/Redo