User Interfaces

Lecture 16

Model - View - Controller

Hamza Bennani
hamza@hamzabennani.com

September 6, 2018

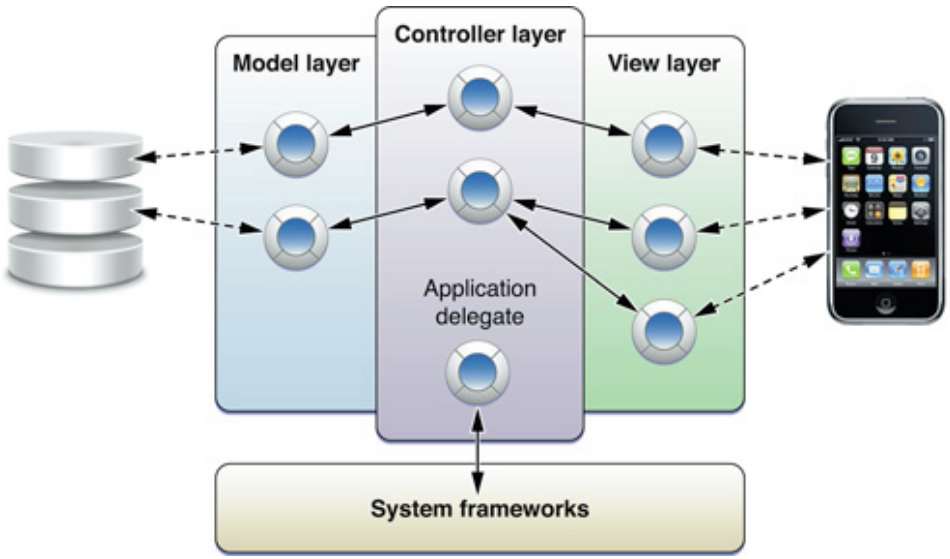
Last Lecture Summary

- Default Cocoa application.
- NSApplication
- NSApplicationDelegate
- XIB/NIB-file storing the visual elements
- RunLoop-event loop class, useful for running timers
- Outlet, target and action mechanism.
- @IBOutlet-annotation
- @IBAction-annotation
- Any questions ???

MVC

Definition

- Introduced in the late 70's in Smalltalk
- Prevalent in many frameworks:
  - Web development
  - Cocoa



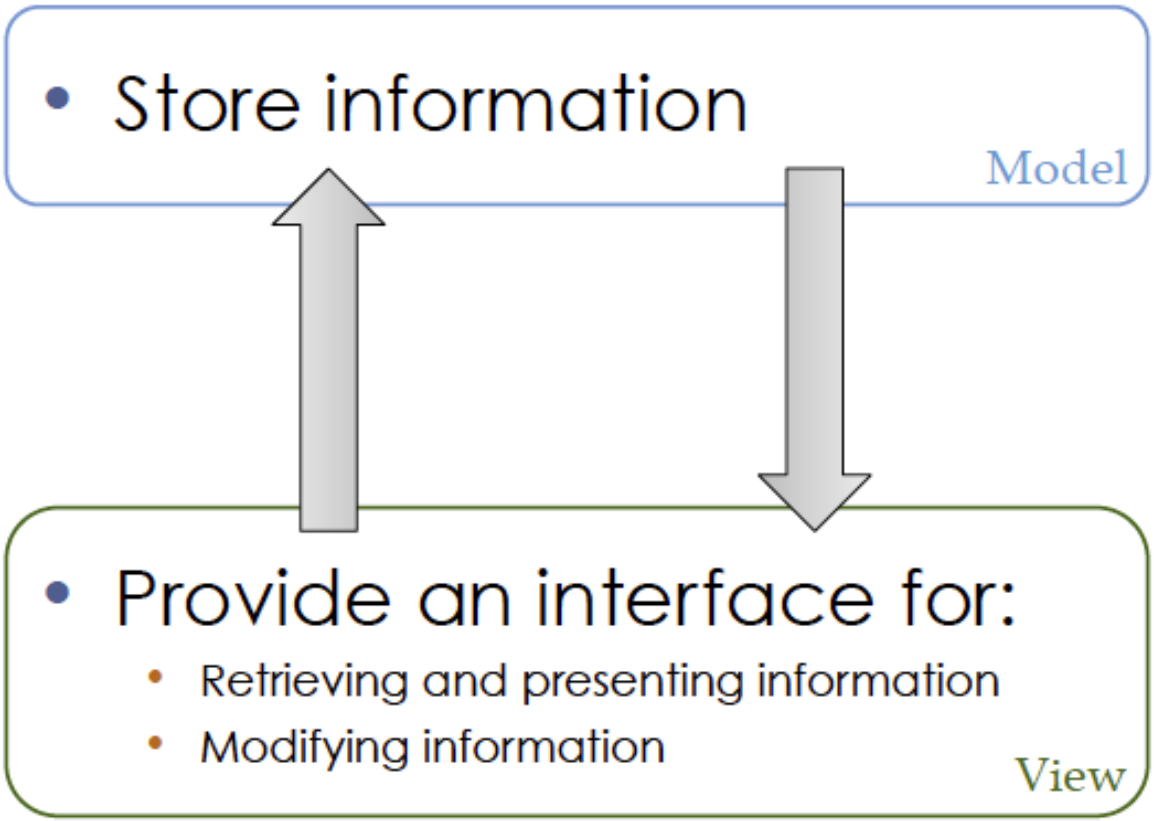
MVC in Applications

Generally, applications perform the following functions:

- Store information
- Provide an interface for:
  - Retrieving and presenting information
  - Modifying information

MVC in Applications

Generally, applications perform the following functions:

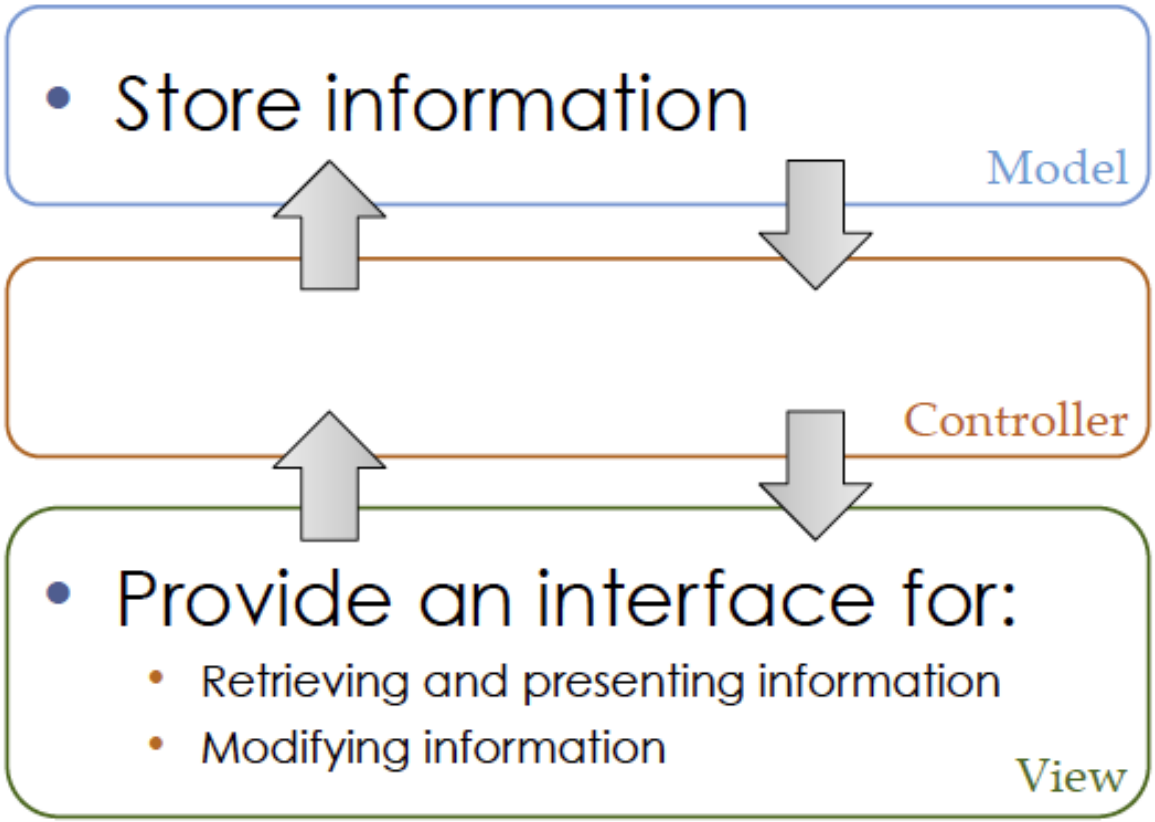


The nature and type of information changes fairly infrequently

The way we interact with information changes more often

MVC in Applications

Generally, applications perform the following functions:



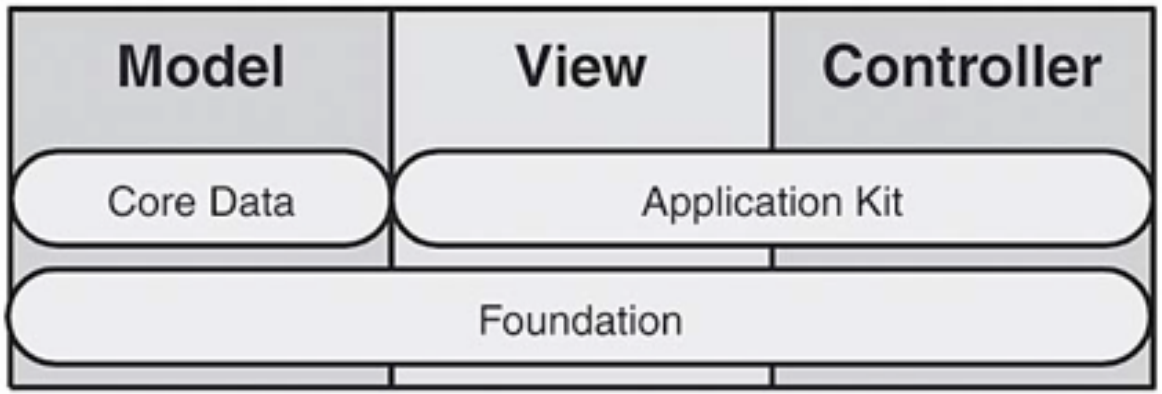
The nature and type of information changes rarely

The role of the controller is to decouple the model from the view

The way we interact with information changes more often

MVC in Cocoa

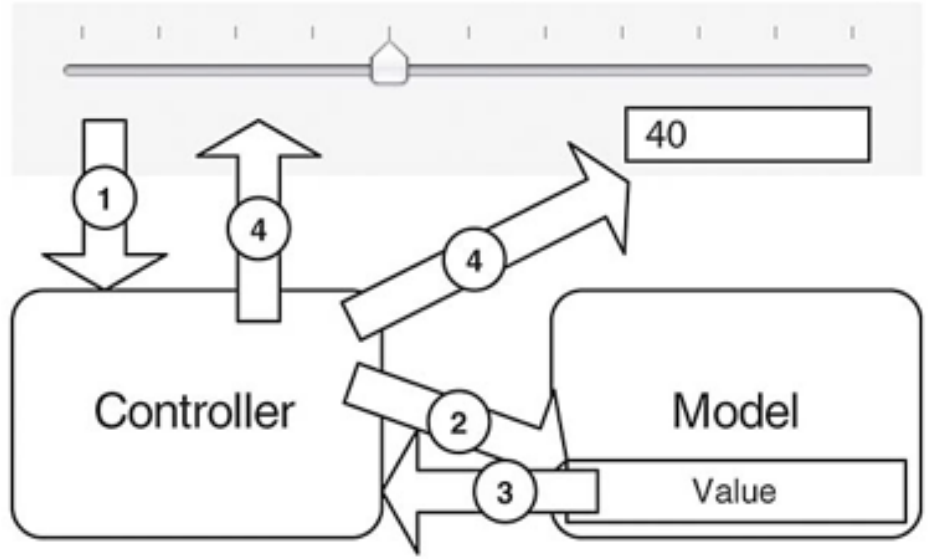
- **AppKit**: classes that provide the view and controller functionality
- **Core Data**: model that abstracts away details of archiving (In this paper we will be almost always creating our own models)
- **Foundation**: classes that glue and customise all the components together



MVC in Cocoa Example

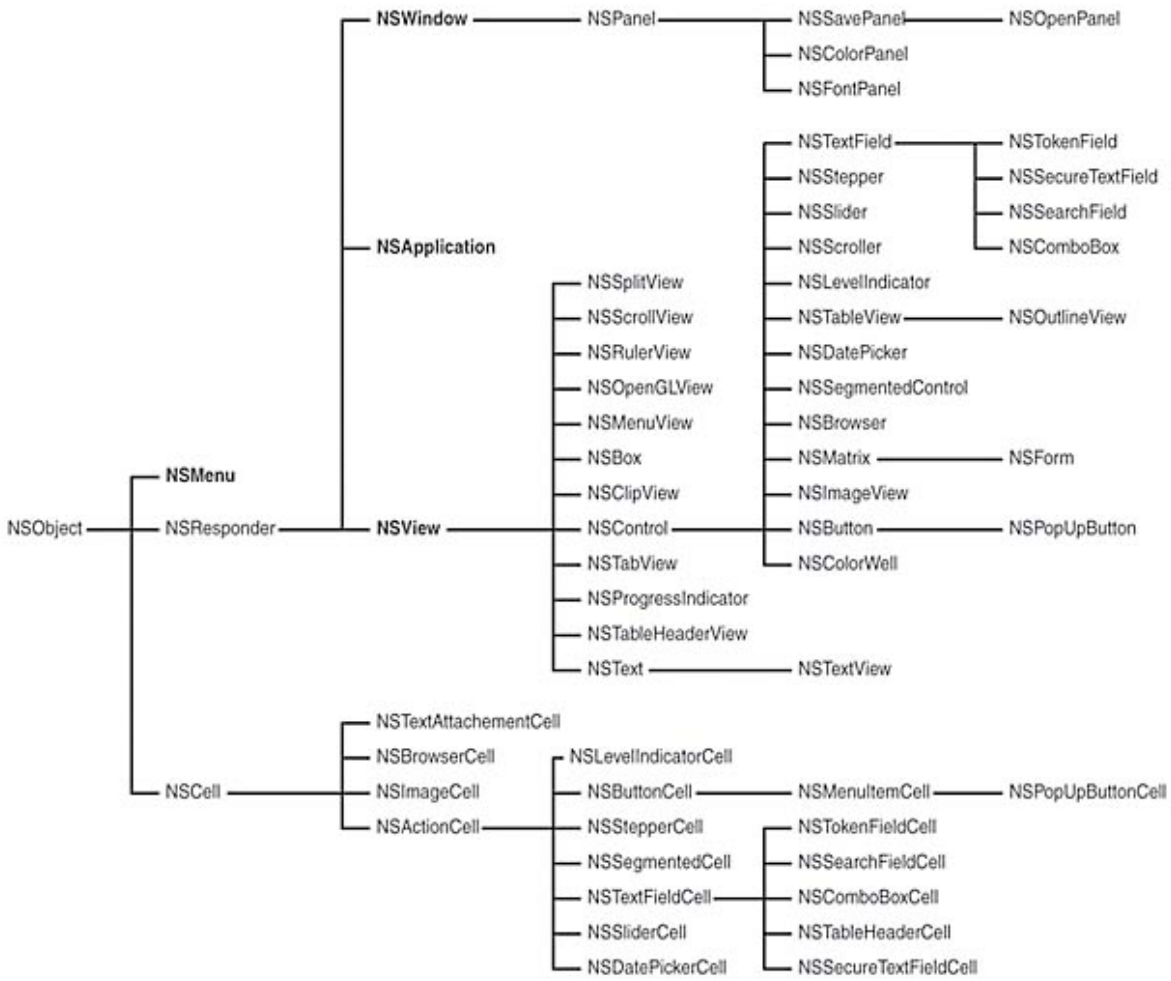
User interacts with a slider

1. Slider sends a msg to the controller with the new value
2. The controller notifies the model about the value change
3. The model updates the corresponding value
   - Constraints checked, Update, Notification
4. Controller notifies the view that the model has changed



Example from Buck and Yacktman
Cocoa Design Patterns, 2009

AppKit
MVC


NSObject
NSMenu
NSResponder
NSWindow
NSPanel
NSSavePanel
NSOpenPanel
NSColorPanel
NSFontPanel
NSApplication
NSView
NSTextField
NSTokenField
NSStepper
NSSecureTextField
NSSlider
NSSearchField
NSScroller
NSComboBox
NSSplitView
NSLevelIndicator
NSScrollView
NSTableView
NSOutlineView
NSRulerView
NSDatePicker
NSOpenGLView
NSSegmentedControl
NSMenuView
NSBrowser
NSBox
NSMatrix
NSForm
NSClipView
NSImageView
NSControl
NSButton
NSPopUpButton
NSTabView
NSColorWell
NSProgressIndicator
NSTableHeaderView
NSText
NSTextView
NSCell
NSTextAttachementCell
NSBrowserCell
NSImageCell
NSActionCell
NSLevelIndicatorCell
NSButtonCell
NSMenuItemCell
NSPopUpButtonCell
NSStepperCell
NSTokenFieldCell
NSSegmentedCell
NSSearchFieldCell
NSTextFieldCell
NSComboBoxCell
NSSliderCell
NSTableHeaderCell
NSDatePickerCell
NSSecureTextFieldCell

AppKit
MVC

- **NSResponder**: handling of mouse and keyboard events
  Abstract class
- **NSView**: rectangle responsible for rendering visual information
  - Subclass of NSResponder
  - Abstract class: you either customise by extending it, or use one of predefined subclasses, such as: NSTabView, NSSplitView, NSScrollView, etc.
- **NSControl**: sending actions when user interacts with the control
  - Don’t confuse with a Controller
  - Subclass of NSView
  - Abstract class: you either customise by extending it, or use one of predefined subclasses, such as: NSButton, NSSlider, NSScroller, etc.

AppKit
MVC

- Bindings controllers:
  - NSController, NSObjectController, NSArrayController, NSUserDefaultsController, and NSTreeController
  - These are special classes that use Cocoa's bindings and key value encoding technology, allowing connection and binding data between view and model elements through Interface Builder - no code necessary
  - These objects implement common logic for mediating between the model and the view
- Application behaviour:
  - NSDocument, NSDocumentController, NSViewController, and NSWindowController
  - These control the fundamental aspects of a GUI application, such as multiple windows, views and documents
- Your custom controller: Typically, your application delegate

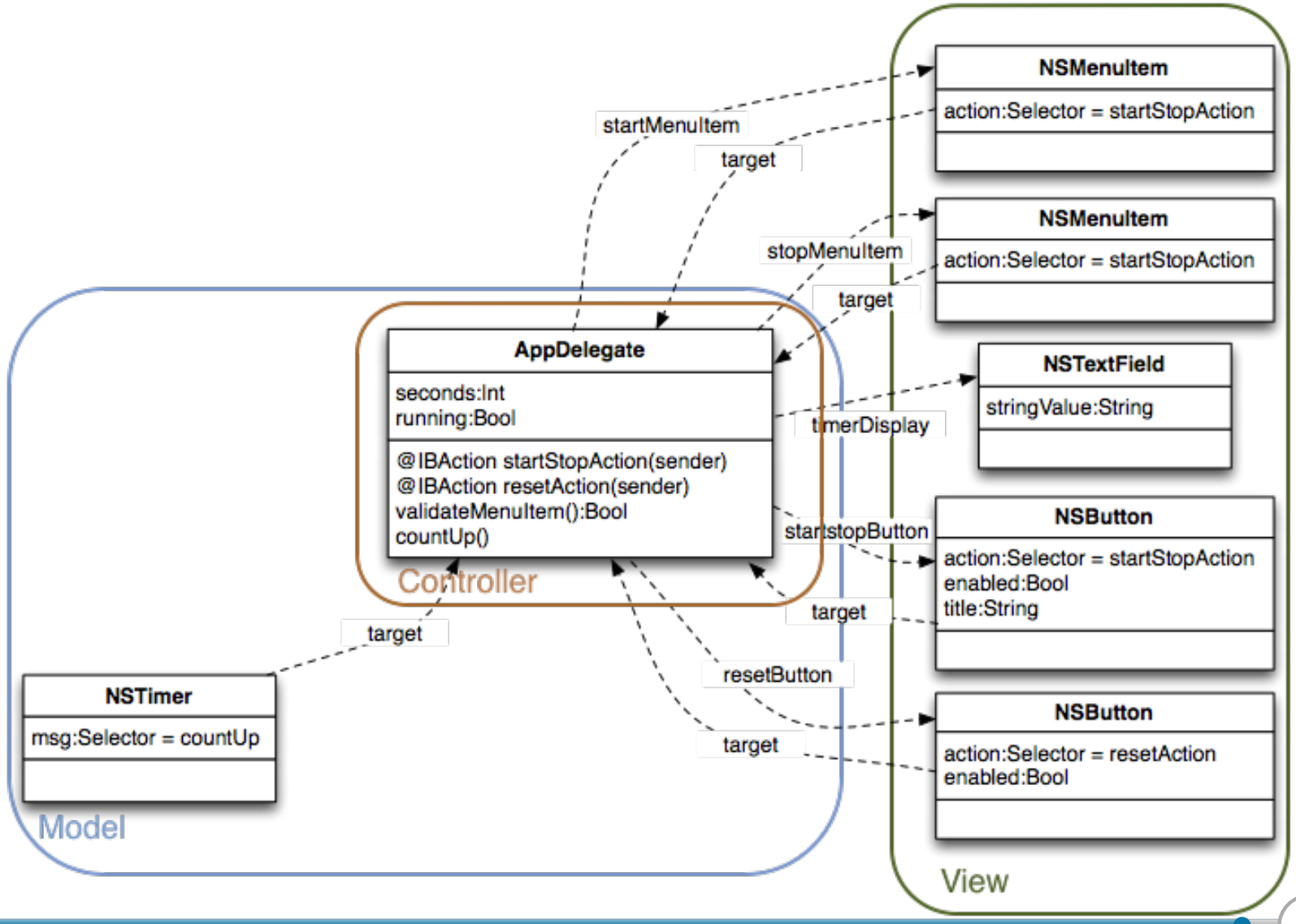
Core Data
MVC

- Collection of classes that abstract away the details of how data is stored and manipulated
  You work with entities (a little bit like a class), attributes (like instance variables in a class), and relationships
- Possible to bind data to controllers in the Interface Builder without needing code
- For persistent storage can store data as XML, binary format, or in an SQLite database

Summary

- **Model-View-Controller** pattern underlines the design of Cocoa framework.
- **View**: code that deals with the visual presentation side of the program Tends to change often
- **Model**: code that implements the underlying representation that stores the information/content Tends to remain unchanged
- **Controller**: the code logic that connects the View and Model components

TimerApp


NSMenuItem
action:Selector = startStopAction
startMenuItem
target
NSMenuItem
action:Selector = startStopAction
stopMenuItem
target
AppDelegate
seconds:Int
running:Bool
@IBAction startStopAction(sender)
@IBAction resetAction(sender)
validateMenuItem():Bool
countUp()
Controller
NSTextField
stringValue:String
timerDisplay
NSButton
action:Selector = startStopAction
enabled:Bool
title:String
startstopButton
target
target
NSTimer
msg:Selector = countUp
resetButton
target
NSButton
action:Selector = resetAction
enabled:Bool
Model
View

TimerApp MVC