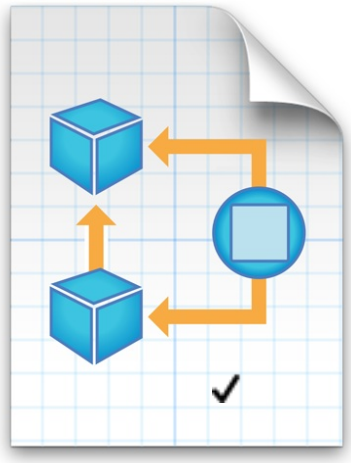
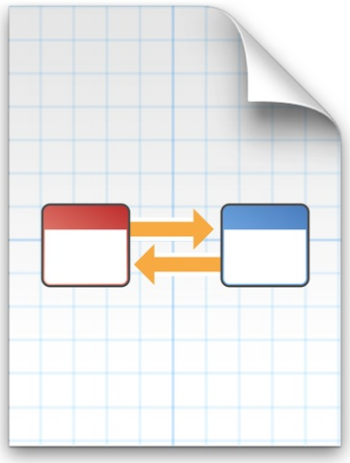
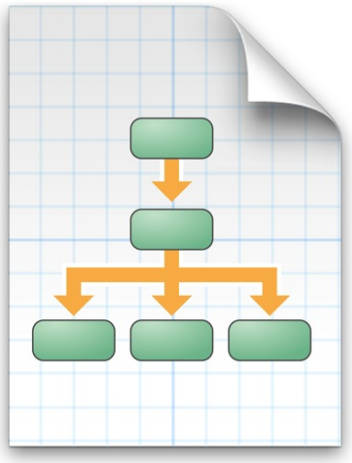
Design Patterns

There are a few standard design patterns that you will use in coding mobile applications. These are not necessarily specific to Swift or Apple, but are general templates you can utilize in your programming endeavors.

- MVC - Model View Controller pattern
- Singleton - Same instance of a class for all that request it.
- Delegate - Messaging system between two separate objects
- Notification - Messaging broadcast to all listeners

References

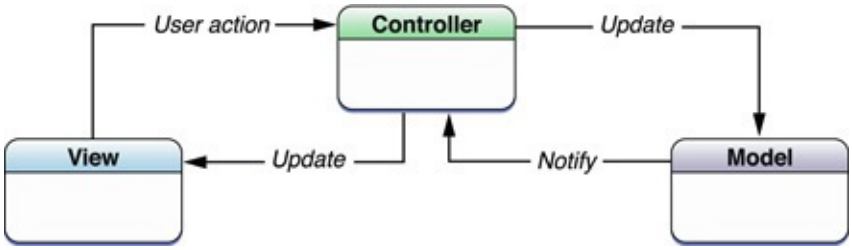
Start Developing iOS apps today - Using Design Patterns
https://developer.apple.com/library/ios/referencelibrary/GettingStarted/RoadMapiOS/DesignPatterns.html

iOS Core Competencies - https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/

iOS Design Patterns - http://www.raywenderlich.com/46988/ios-design-patterns

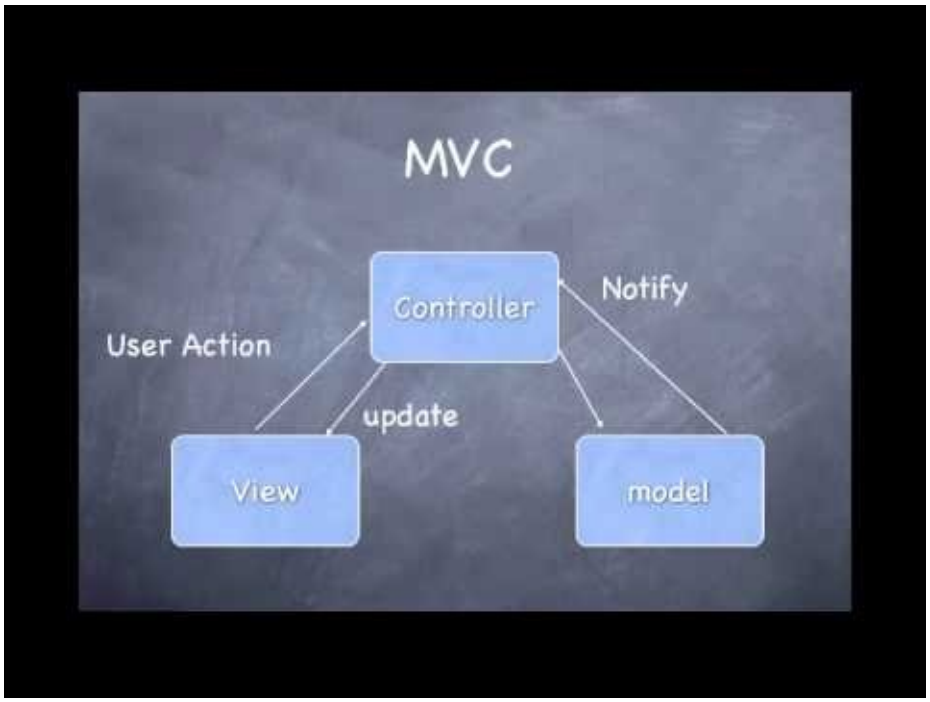
Model - View - Controller

iOS templates new projects in Xcode with the MVC pattern. That is to say, the give you a separate entity for each part of the MVC pattern. Keeping each part of MVC as separate as possible allows for reusability, maintenance and clean code.



1. Model - Data storage
2. View - UI as defined and created in a Storyboard
3. Controller - Controller classes which typically override a UIKit class.

[Video] Introduction to MVC



https://www.youtube.com/watch?v=Y09RvzZ1mY8

Model

A model in an iOS application can be a simple class or struct that persists in memory. It can also be a file that is written to disk, which would presist through subsequent launches of your application. CoreData is a system that Apple provides as a robust model layer in your iOS applications.

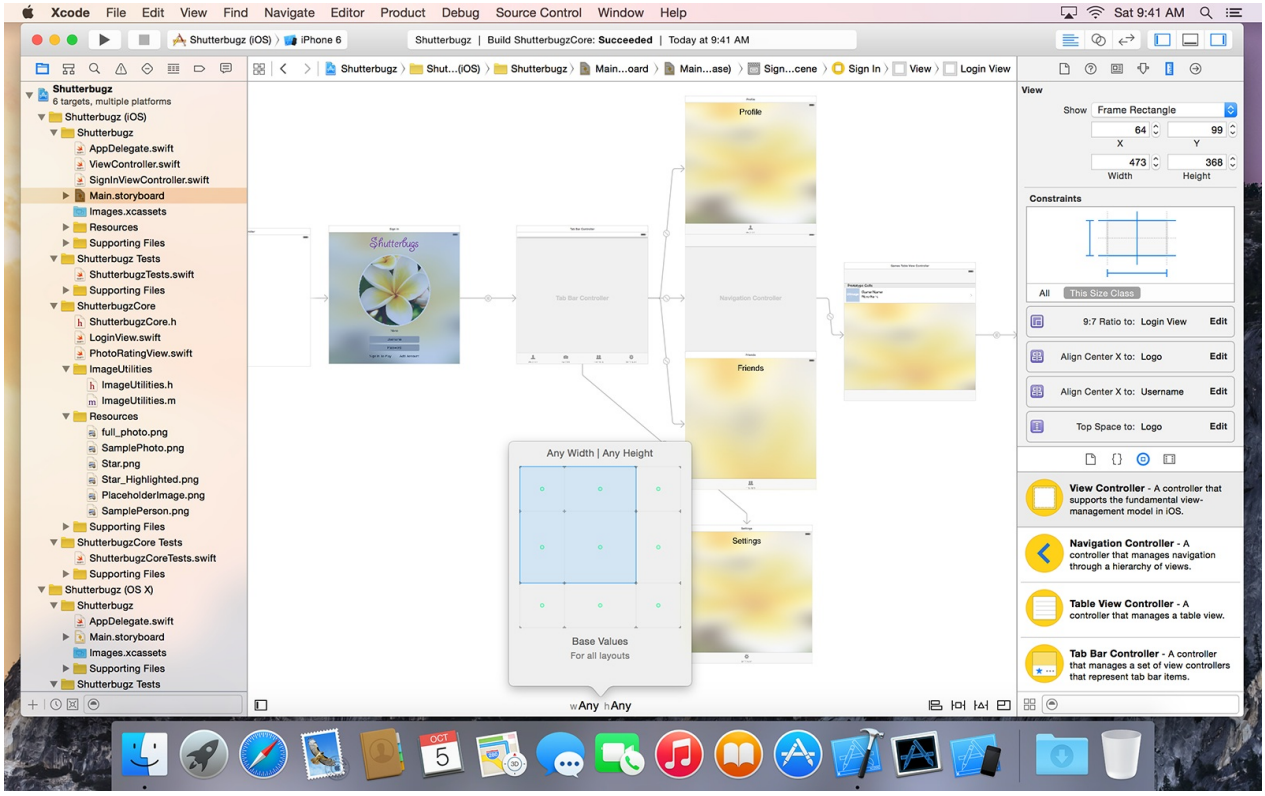
> Model objects encapsulate the data specific to an application and define the logic and computation that manipulate and process that data [1]

Reference: https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/ModelObject.html

View

Views are typically handled in a Storyboard, handled by Interface Builder within Xcode.

> A view object is an object in an application that users can see. A view object knows how to draw itself and can respond to user actions. [1]



Controller

Deciding what to do once your user interacts with your application is handled by the controller. In your projects, this is the subclass of UIViewControllers, UITableViewControllers, etc.

> A controller object acts as an intermediary between one or more of an application's view objects and one or more of its model objects. Controller objects are thus a conduit through which view objects learn about changes in model objects and vice versa. [1]

Reference: https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/ControllerObject.html

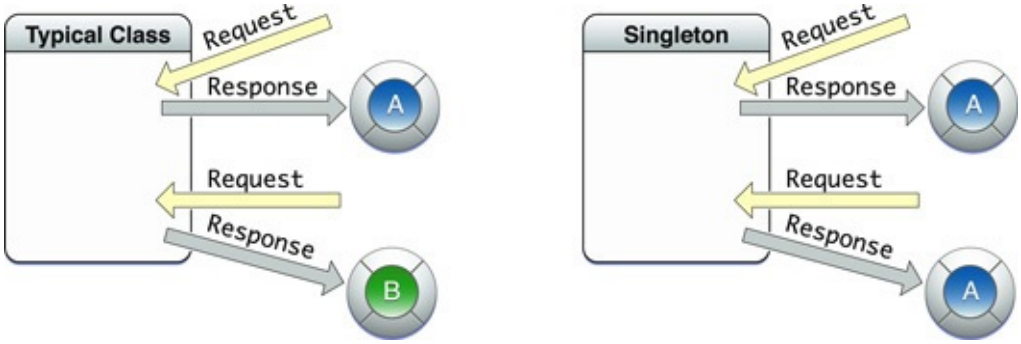
References

[1] - Cocoa Core - Model View Controller
https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

[2] Start Developing iOS apps today - Using Design Patterns
https://developer.apple.com/library/ios/referencelibrary/GettingStarted/RoadMapiOS/DesignPatterns.html

Singleton

A singleton is the same instance of a class for all that request it. Use a singleton to maintain a single point of control for your object.



> A typical class permits callers to create as many instances of the class as they want, whereas with a singleton class, there can be only one instance of the class per process [1]

An example

```
class ExampleSingleton {

    private var counter = 0

    class var sharedInstance : ExampleSingleton {
        struct Static {
            static let instance : ExampleSingleton = ExampleSingleton()
        }
        return Static.instance
    }

    func incrementCounter() -> Int {
        counter++
        return counter
    }
}

ExampleSingleton.sharedInstance.incrementCounter() // 1
ExampleSingleton.sharedInstance.incrementCounter() // 2
ExampleSingleton.sharedInstance.incrementCounter() // 3
ExampleSingleton.sharedInstance.incrementCounter() // 4
```

References

[1] Core Competency - Singleton - https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/Singleton.html

[2] Swift Singleton - GitHub - https://github.com/hpique/SwiftSingleton https://github.com/hpique/SwiftSingleton

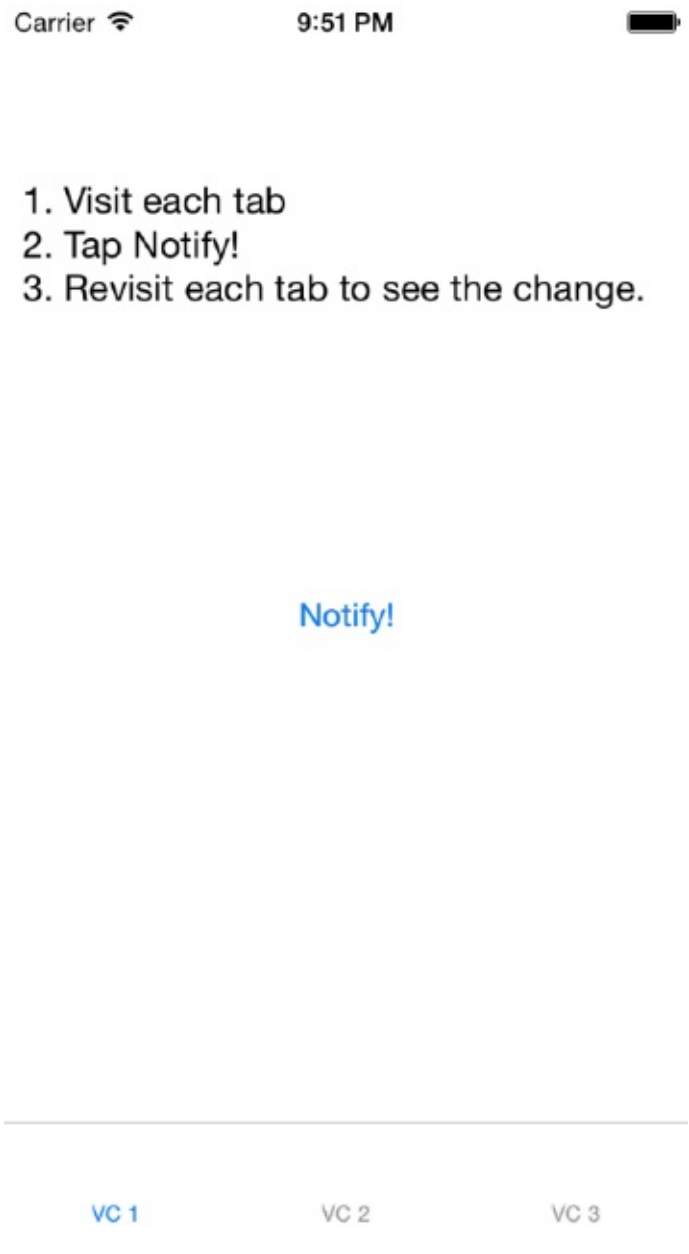
Notification

Notifications in iOS can be handled by the UINotificationCenter.

> An NSNotificationCenter object (or simply, notification center) provides a mechanism for broadcasting information within a program.

Notifications differ from delegates in that they are not a one-to-one connection point. Instead, they are a broadcast which objects can subscribe to.

An Example



Code and process an be found here: http://www.andrewcbancroft.com/2014/10/08/fundamentals-of-nsnotificationcenter-in-swift/

References

[1] UINotificationCenter - https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSNotificationCenter_Class/i

ndex.html

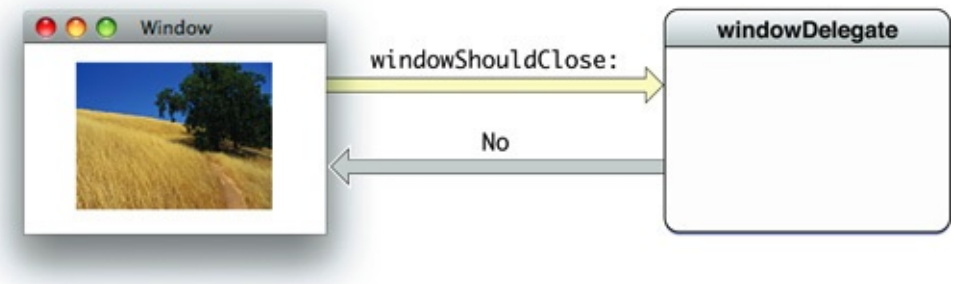
[2] Cocoa Core Competencies - Notification
https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/Notification.html

[3] NSNotification - NSHipster - http://nshipster.com/nsnotification-and-nsnotificationcenter/

Delegates (Protocols)

Delegates provide a messaging system between two separate objects. We use delegates to inform controller classes of some updated information or request that an action is to take place.

> Delegation is a design pattern that enables a class or structure to hand off (or delegate) some of its responsibilities to an instance of another type [1]



Example

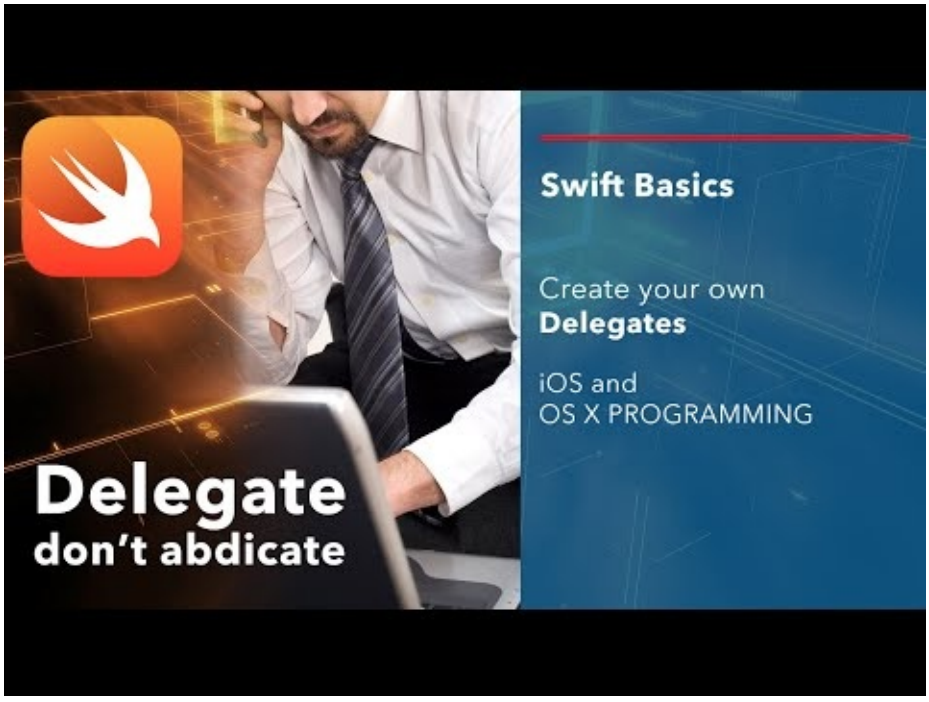
```
import Foundation

protocol CountingDelegate {
    func didBeginCounting()
    func didCount(currentValue:Int)
    func didEndCounting(finalCount: Int)
}

class CountTo {

    var delegate: CountingDelegate?

    func beginCounting(countUntil: Int) {
        var internalCounter = 0

        // OK, we're about to begin counting, let's inform our delegate
        delegate?.didBeginCounting()

        while ++internalCounter < countUntil {
            delegate?.didCount(internalCounter)
        }

        // Look's like we completed our while loop, let's inform our delegate
        delegate?.didEndCounting(internalCounter)
    }
}

class LearningToddler: CountingDelegate {
    var name: String
    var age: Int
    let maximumCountingAbility = 20
    let countingActivity = CountTo()

    init(toddlerName: String, toddlerAge: Int) {
        name = toddlerName
        age = toddlerAge
    }

    func beginCounting() {
        countingActivity.delegate = self
        countingActivity.beginCounting(maximumCountingAbility)
    }

    // Define our delegate methods which will be invoked
    func didBeginCounting() {
        NSLog("We have begun counting!")
    }
```

```
    func didCount(currentValue: Int) {
        NSLog("Our current count is \(currentValue) and we're counting until \(maximumCountingAbility)")
    }

    func didEndCounting(finalCount: Int) {
        NSLog("We successfully counted all the way to \(finalCount)!")
    }
}

// Running the example
let toddler = LearningToddler(toddlerName: "Raleigh", toddlerAge: 2)
toddler.beginCounting()
```

Copying the above code into Playgrounds should yield the following result:

```
2015-01-04 14:47:35.685 MyPlayground[21014:3462547] We have begun counting!
2015-01-04 14:47:35.686 MyPlayground[21014:3462547] Our current count is 1 and we're counting until 20
2015-01-04 14:47:35.687 MyPlayground[21014:3462547] Our current count is 2 and we're counting until 20
2015-01-04 14:47:35.689 MyPlayground[21014:3462547] Our current count is 3 and we're counting until 20
2015-01-04 14:47:35.690 MyPlayground[21014:3462547] Our current count is 4 and we're counting until 20
2015-01-04 14:47:35.692 MyPlayground[21014:3462547] Our current count is 5 and we're counting until 20
2015-01-04 14:47:35.694 MyPlayground[21014:3462547] Our current count is 6 and we're counting until 20
2015-01-04 14:47:35.695 MyPlayground[21014:3462547] Our current count is 7 and we're counting until 20
2015-01-04 14:47:35.697 MyPlayground[21014:3462547] Our current count is 8 and we're counting until 20
2015-01-04 14:47:35.699 MyPlayground[21014:3462547] Our current count is 9 and we're counting until 20
2015-01-04 14:47:35.701 MyPlayground[21014:3462547] Our current count is 10 and we're counting until 20
2015-01-04 14:47:35.703 MyPlayground[21014:3462547] Our current count is 11 and we're counting until 20
2015-01-04 14:47:35.704 MyPlayground[21014:3462547] Our current count is 12 and we're counting until 20
2015-01-04 14:47:35.706 MyPlayground[21014:3462547] Our current count is 13 and we're counting until 20
2015-01-04 14:47:35.708 MyPlayground[21014:3462547] Our current count is 14 and we're counting until 20
2015-01-04 14:47:35.709 MyPlayground[21014:3462547] Our current count is 15 and we're counting until 20
2015-01-04 14:47:35.711 MyPlayground[21014:3462547] Our current count is 16 and we're counting until 20
2015-01-04 14:47:35.712 MyPlayground[21014:3462547] Our current count is 17 and we're counting until 20
2015-01-04 14:47:35.714 MyPlayground[21014:3462547] Our current count is 18 and we're counting until 20
2015-01-04 14:47:35.716 MyPlayground[21014:3462547] Our current count is 19 and we're counting until 20
2015-01-04 14:47:35.717 MyPlayground[21014:3462547] We successfully counted all the way to 20!
```

[Video] Protocols and Delegates



https://www.youtube.com/watch?v=9LHDsSWc680

References

[1] Swift Programming Guide -

https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Protocols.html

[2] Cocoa Core - Delegation https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/Delegation.htm