Misc.

- **Credits & Acknowledgments:**
  - some materials based on Apple/Stanford
- **Apps developed during the course are yours:**
  - feel free to submit on the AppStore

Requirements

- **Required Hardware & Software:**
  - Recent Apple machine running MacOS (10.13 or newer
  - Lastest iOS SDK and Xcode (currently (iOS 11 and Xcode 10 or newer)
  - Assignments may be completed using iOS Simulator only
  - Loaner MacBooks (ageing) and iDevices may be available on limited numbers based on some priority criteria (see with Paul Martin)
- **Recommended Book:**
  - Apple documentation
  - http://developper.apple.com
  - iTunesU, Stanford Course
  - Some labs loosely based on iOS 8 for Programmers: An App-Driven Approach with Swift, Dutiel
- **Prerequisites: OOP, C, Graphics**
  - Class (description/template for an object)
  - Instance (manifestation of a class)
  - Message (sent to objects to make them act)

iOS Developer University Program

- **UCD is part of the iOS Developer University Program**
  - Do not sign up for this!
  - download latest iOS SDK and Xcode
- **Free on-device development for students**
  - ✓ Device
  - ✗ AppStore
  - Valid through the end of the term
  - Invites will be issued to your @ucd.ie email address
  - Read through a Student Agreement to get started
  - Follow the directions to join the Program
  - Submit your UDID to the UCD via e-mail (Paul Martin)

Why Are We Here?

... to learn how to build great iOS Apps ... distribute them on the AppStore!

Why Are We Here?

- **Module is not just about iOS**
  - It is also about Software Engineering, as well as object oriented architecture and design
  - Exposure to problems and solutions as already seen in some other modules
  - Based on Cocoa, Started with NEXT over 20 years ago. Mature, polished, highly consistent APIs
- **Provides a very rich starting point for exploring app design**
  - Easy to build even very complex applications
  - Result lives in your pocket!
  - Very easy to distribute your application through the AppStore
  - Great development community
- **Shows 'real-world' implementations of OO design patterns**
  - The heart of Cocoa Touch is 100% object-oriented
  - Application of MVC design pattern
  - Many computer science concepts applied in a commercial development platform: Databases, Graphics, Multimedia, Multithreading, Animation, Networking, and much, much more!
- **Designs learned on iOS translate directly to Mac OS X**

Apps You Will Build

	HelloUCD
	Tip Calculator
	Class Test
	Twitter Search
	Game / Animation: Connect4

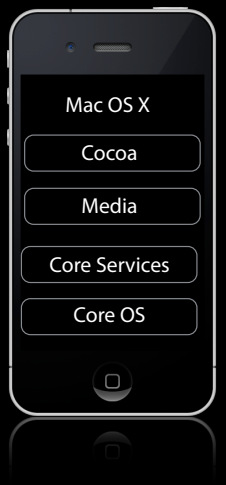
Platform Framework

Tools	 Xcode	 Instruments	 Interface Builder
Frameworks	 Foundation	 UIKit	 CoreData, MapKit, CoreMotion
Language (& Runtime)	`[textView setTextColor:[UIColor whiteColor]];` `textView.textColor = UIColor.white`		
Design Strategies		Model-View-Controller	

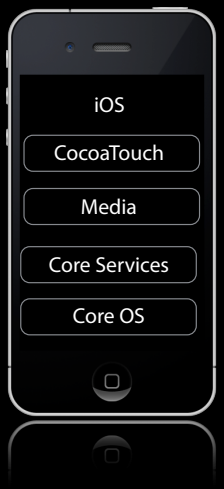
iOS Overview

Mac OS X

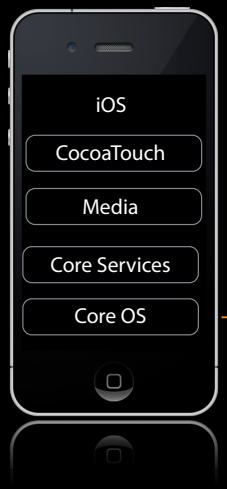
iOS



iOS



iOS



Core OS

OS X Kernel Power Management

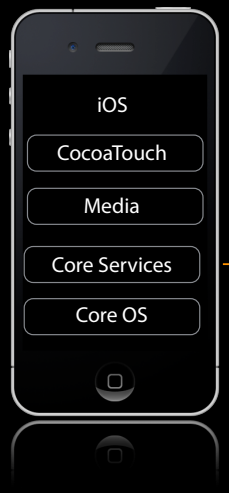
Mach 3.0 Keychain Access

BSD Certificates

Sockets File System

Security Bonjour

iOS



Core Services

Collections Core Location

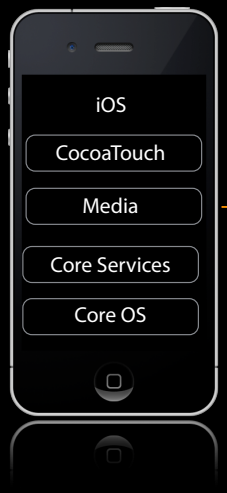
Address Book Net Services

Networking Threading

File Access Preferences

SQLite URL utilities

iOS



Media

Core Audio JPG, PNG, TIFF

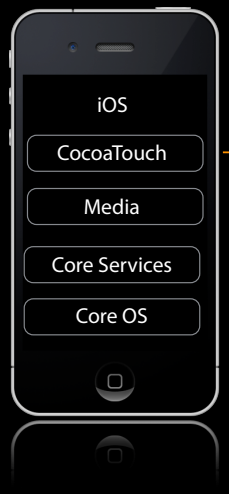
OpenAL PDF

Audio Mixing Quartz (2D)

Audio Recording Core Animation

Video Playback OpenGL ES

iOS



Cocoa Touch

Multi-Touch Alerts

Core Motion Web View

Controls Map Kit

View Hierarchy Image Picker

Localization Camera

Cocoa Touch Architecture

UIKit	User interface elements Application runtime Event handling Hardware APIs
Foundation	Utility classes Collection classes Object wrappers for system services Subset of Foundation in Cocoa

Cocoa Touch Frameworks

CocoaTouch Layer	CoreOS Layer	CoreServices Layer		Media Layer	
AddressBookUI	Accelerate	Accounts	HealthKit	AVFoundation	ImageIO
EventKitUI	CoreBluetooth	AdSupport	HomeKit	AssetsLibrary	MediaAccessibility
GameKit	ExternalAccessory	AddressBook	JavaScriptCore	AudioToolbox	MediaPlayer
MapKit	LocalAuthentication	CFNetwork	MobileCoreServices	AudioUnit	Metal
MessageUI	Security	CloudKit	Multipeer-Connectivity	CoreAudio	OpenAL
NotificationCenter	System	CoreData	NewsstandKit	CoreGraphics	Photos
PhotosUI		CoreFoundation	PassKit	CoreImage	QuartzCore
Twitter		CoreLocation	QuickLook	CoreMIDI	SceneKit
UIKit		CoreMedia	Social StoreKit	CoreText	SpriteKit
iAd		CoreMotion	SystemConfiguration	CoreVideo	OpenGL ES
		CoreTelephony	UIAutomation	GLKit	GameController
		EventKit	WebKit		
		Foundation	CoreML		

New Programming Languages

- **Why? Exposure to other languages is always good**
- **Objective C (Legacy)**
  - language focused on simplicity and the elegance of OO design
  - Based on ANSI C
  - Brings many object oriented principles, but with a minimal amount of syntax
- **Swift**
  - New programming language created by Apple for building iOS and Mac apps
  - Powerful, intuitive and modern syntax
  - Easy to learn and use

Demo

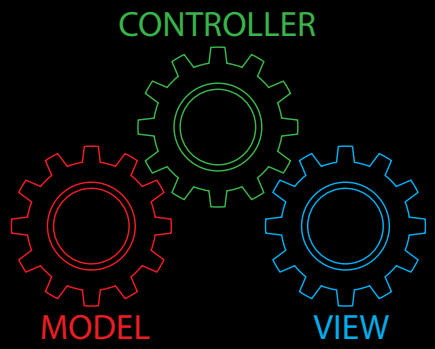
HelloUCD
GravityBubbles
RPN Calculator

Demos

- **Take away**
  - Creating a Project in Xcode 10
  - Building a responsive UI using Autolayout
  - The iOS Simulator
- **Obverse and take note (we'll come back over)**
  - language syntax
  - defining class in Swift, adding instance variables (properties) and behaviour (methods)
  - print and String interpolation
  - connecting Swift code and UI (outlets, actions)

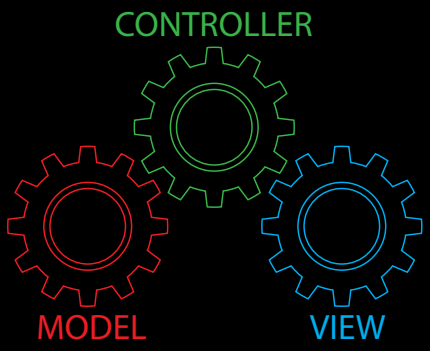
Model View Controller

MVC



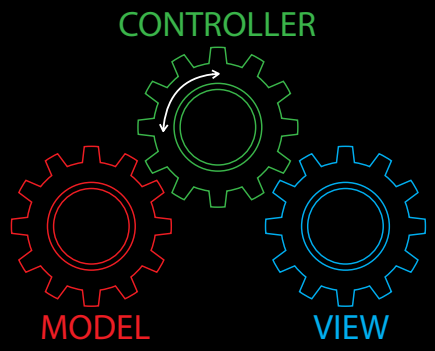
Divide objects in your program into **3 entities**

MVC



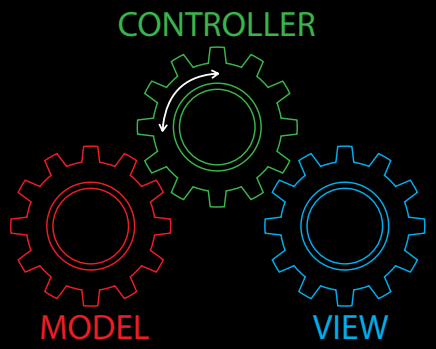
Model: **what** your application is (but **not how** it is displayed)

MVC



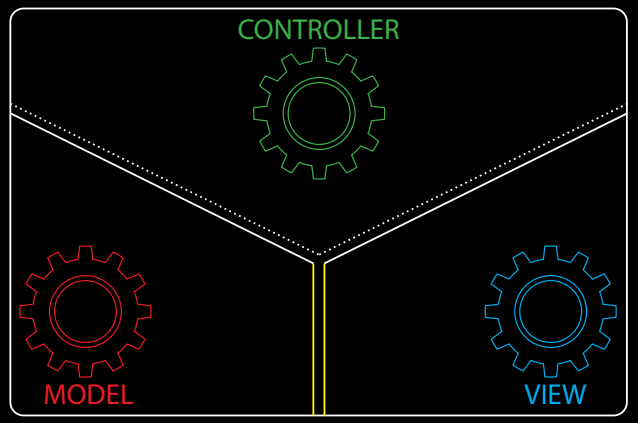
Controller: **How** your Model is presented to the user (UI logic)

MVC



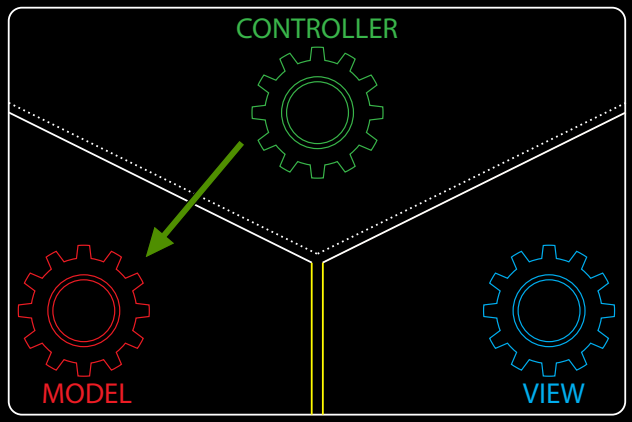
View: your controller's minion

MVC in Action



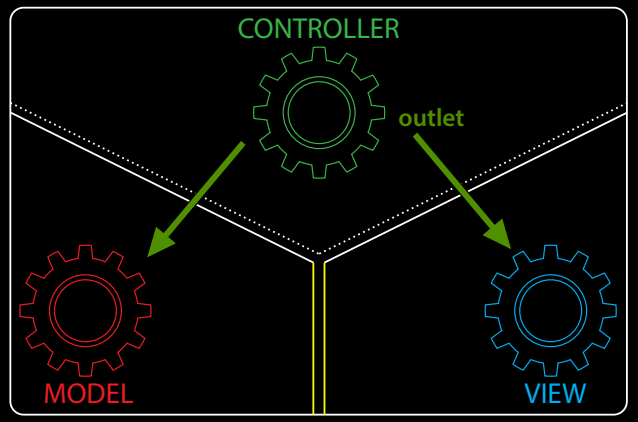
MVS is all about **managing communication** between camps

MVC in Action



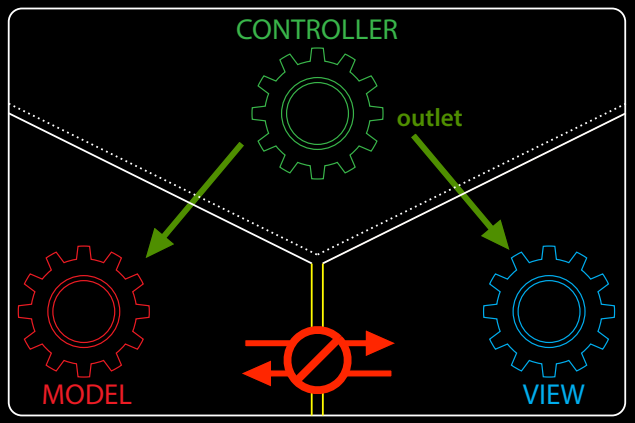
Controllers can always talk directly to their **Model**

MVC in Action



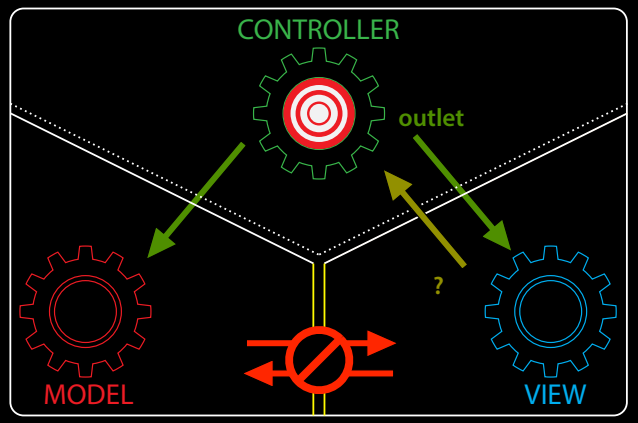
Controllers can also talk directly to their **View**

MVC in Action



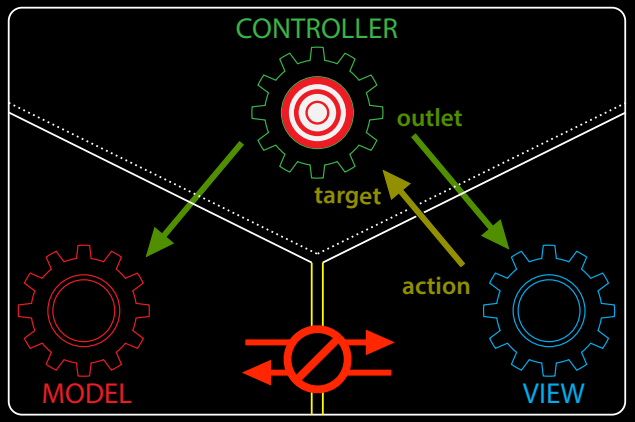
The Model and View should never speak to each other!

MVC in Action



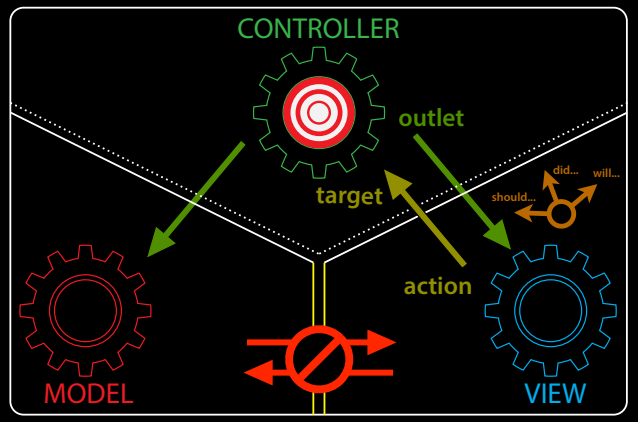
Can the View speak to its Controller?
–> sort of, communication is `blind' and structured

MVC in Action



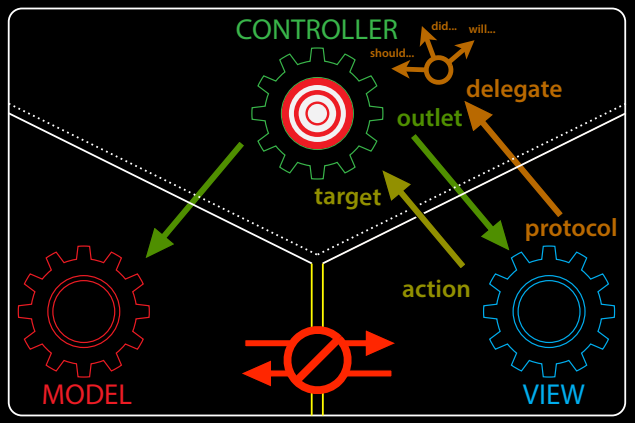
The Controller can drop a target on itself,
then hand out an action to the View.

MVC in Action



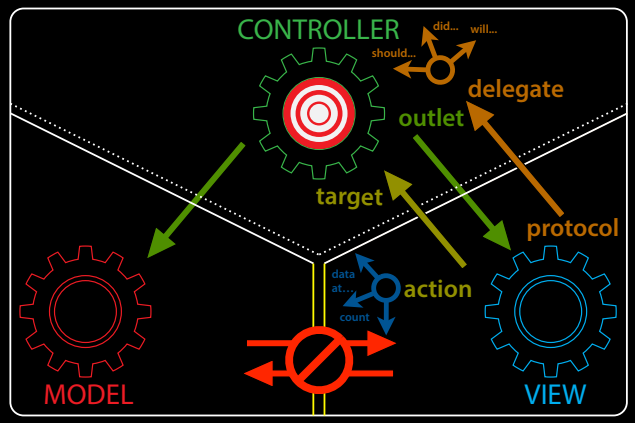
Sometimes the View needs to synchronize with the Controller

MVC in Action



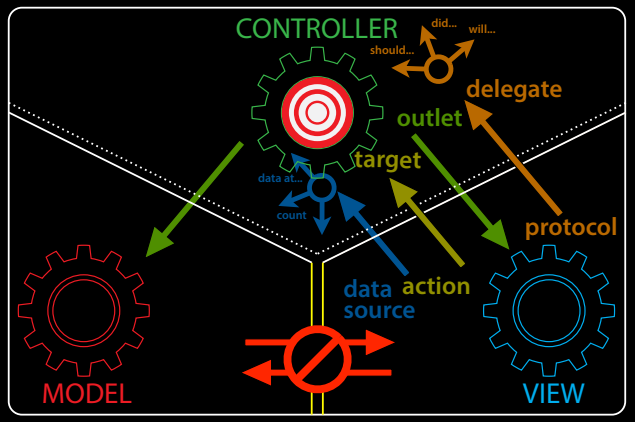
The **delegate** is set via a **protocol** i.e. it is blind to class

MVC in Action



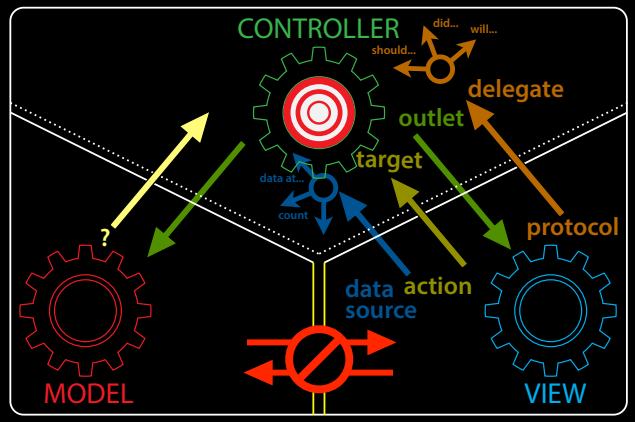
Views do not own the data they display.
So, if needed, they have a protocol to acquire it.

MVC in Action



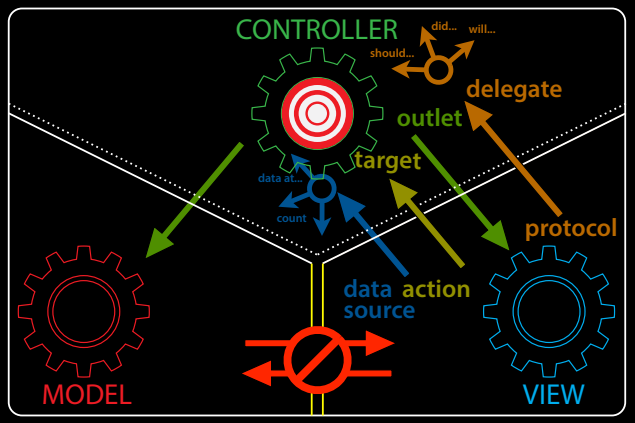
Controllers are almost always that data source (not Model).
Controllers interpret/format Model information for the View.

MVC in Action



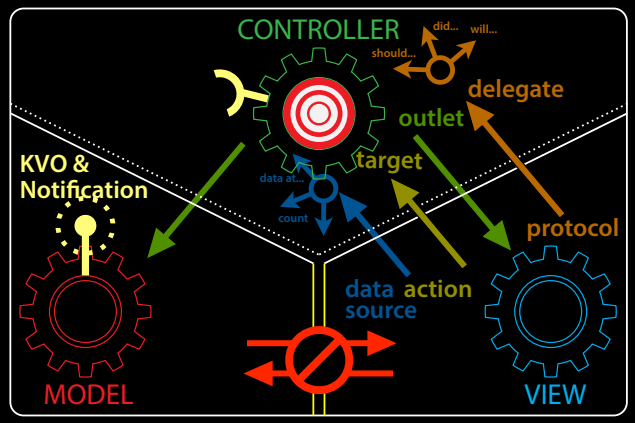
Can the Model talk directly to the Controller?

MVC in Action



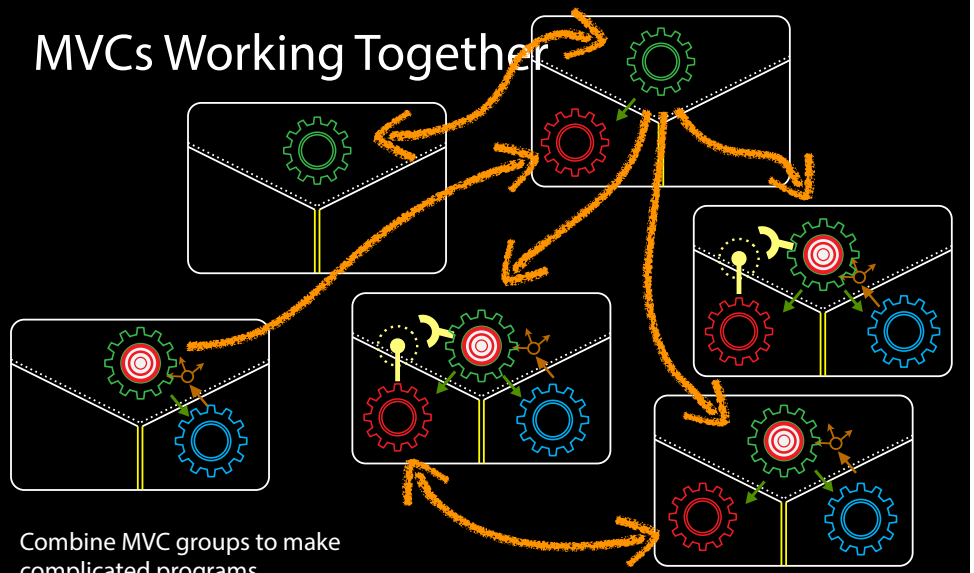
No. The Model is (should be) UI independent.
So what if the Model has information to update or something?

MVC in Action



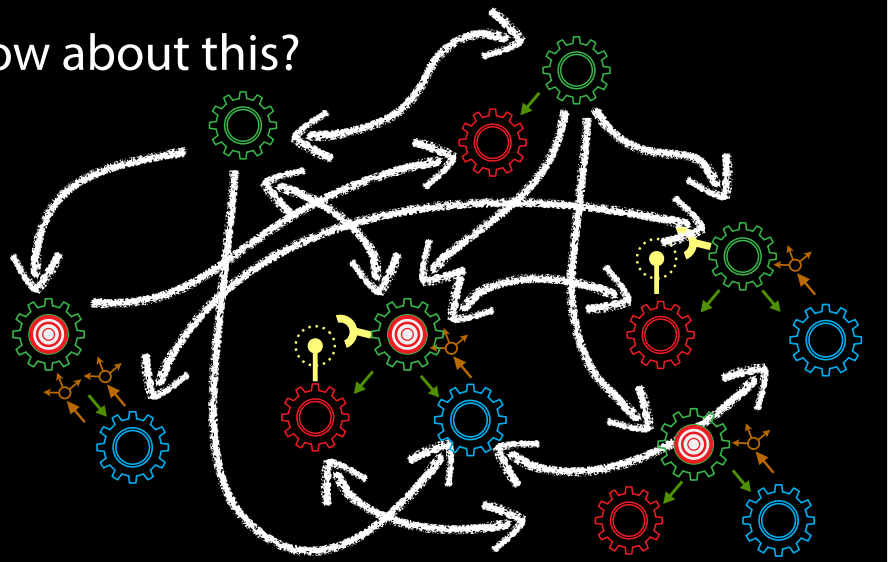
It uses a radio-like broadcast mechanism.
Controllers (or other Model) tune in to interesting stuff.
A View might tune in, but probably not to a Model's station.

MVCs Working Together



Combine MVC groups to make complicated programs ...

How about this?



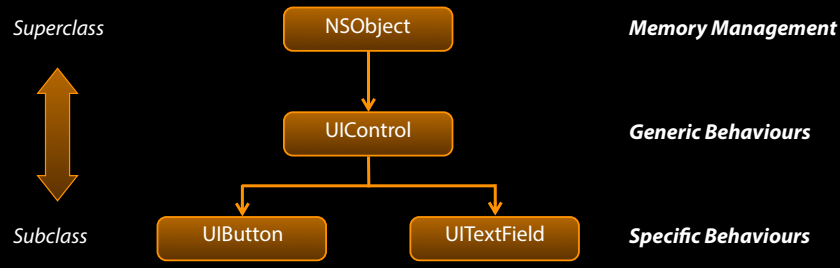
Architecture to Avoid

Objects & OOP

OOP Jargon

- **Class:** defines the grouping of data and code, the type of an object
- **Instance:** a specific allocation of a class
- **Method:** a function that an object knows how to perform
- **Instance Variable (or iVar):** a specific piece of data belonging to an object
- **Encapsulation:** keep implementation private and separate from interface
- **Polymorphism:** different objects, same interface
- **Inheritance:** hierarchical organization, share code, customize or extend behaviors

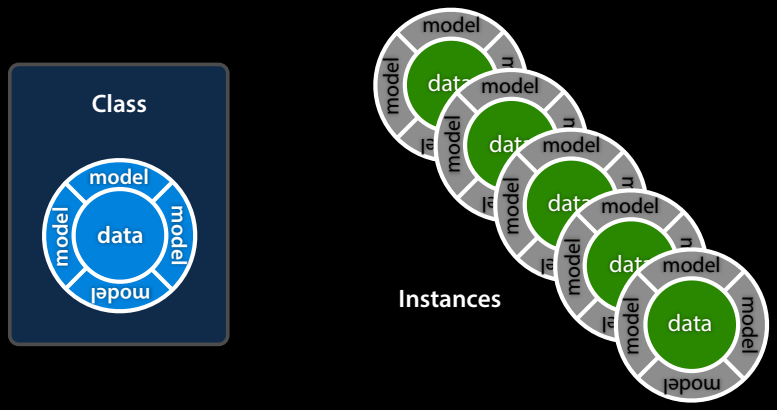
Inheritance



- Hierarchical relation between classes
- Subclass "inherit" behaviour and data from superclass
- Subclasses can use, augment or replace superclass methods
- Ref: Swift 4 Programming Language
  https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Inheritance.html

Classes and Instances

- In Objective–C and Swift, classes and instances are both objects
- Class is the blueprint to create instances



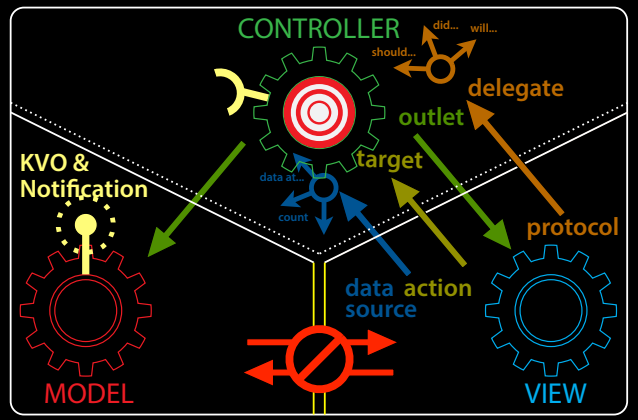
Classes and Objects

- Classes declare state and behavior
- State (data) is maintained using instance variables
- Behaviour is implemented using methods
- Instance variables typically hidden, accessible only using getter/setter methods

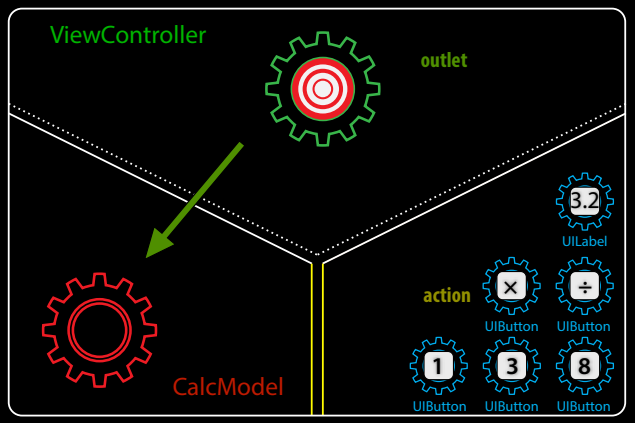
- Interesting read: Object-Oriented Programming with Objective–C
  https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/OOP_ObjC

MVC Example

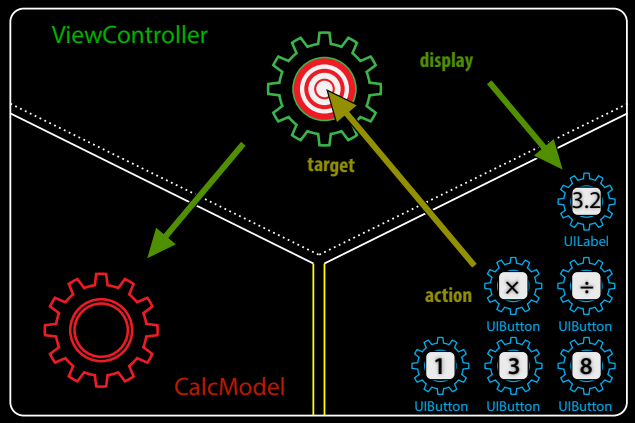
Calculator MVC



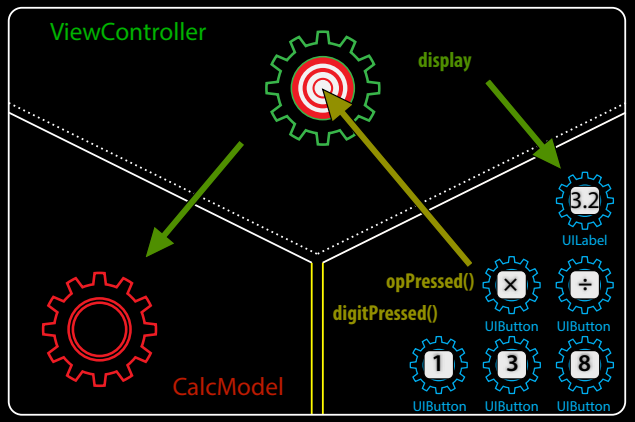
Calculator MVC



Calculator MVC



Calculator MVC



Model: CalcModel.swift

Swift file for this class. Unlike Objective-C It documents both public and private API

```
import Foundation

class CalcModel: NSObject {
    var stack = [Double]()

    func pushOperand(value: Double) {
        self.stack.append(value)
        print("stack = \(self.operandStack)")
    }

    func eval(operation: String) -> Double {
        ...
        return self.operandStack.last!
    }

    private func performOperation(operation: (Double, Double) -> Double) {
        ...
    }

    private func performOperation(operation: (Double) -> Double) {
        ...
    }

    func multiply(_ op1: Double,_ op2: Double) -> Double {
        return op1 * op2
    }
}
```

Model: CalcModel.swift

The name of this class.

```
import Foundation

class CalcModel: NSObject {
    var stack = [Double]()

    func pushOperand(value: Double) {
        self.stack.append(value)
        print("stack = \(self.operandStack)")
    }

    func eval(operation: String) -> Double {
        ...
        return self.operandStack.last!
    }

    private func performOperation(operation: (Double, Double) -> Double) {
        ...
    }

    private func performOperation(operation: (Double) -> Double) {
        ...
    }

    func multiply(_ op1: Double,_ op2: Double) -> Double {
        return op1 * op2
    }
}
```

Model: CalcModel.swift

Import header for our superclass

```
import Foundation

class CalcModel: NSObject {
    var stack = [Double]()

    func pushOperand(value: Double) {
        self.stack.append(value)
        print("stack = \(self.operandStack)")
    }

    func eval(operation: String) -> Double {
        ...
        return self.operandStack.last!
    }

    private func performOperation(operation: (Double, Double) -> Double) {
        ...
    }

    private func performOperation(operation: (Double) -> Double) {
        ...
    }

    func multiply(_ op1: Double, _ op2: Double) -> Double {
        return op1 * op2
    }
}
```

Model: CalcModel.swift

Properties go here
Array of Double in this case

```
import Foundation

class CalcModel: NSObject {
    var stack = [Double]()

    func pushOperand(value: Double) {
        self.stack.append(value)
        print("stack = \(self.operandStack)")
    }

    func eval(operation: String) -> Double {
        ...
        return self.operandStack.last!
    }

    private func performOperation(operation: (Double, Double) -> Double) {
        ...
    }

    private func performOperation(operation: (Double) -> Double) {
        ...
    }

    func multiply(_ op1: Double, _ op2: Double) -> Double {
        return op1 * op2
    }
}
```

Model: CalcModel.swift

Methods go here

```
import Foundation

class CalcModel: NSObject {
    var stack = [Double]()

    func pushOperand(value: Double) {
        self.stack.append(value)
        print("stack = \(self.operandStack)")
    }

    func eval(operation: String) -> Double {
        ...
        return self.operandStack.last!
    }

    func performBinaryOperation(operation: (Double, Double) -> Double) {
        ...
    }

    func performUnaryOperation(operation: (Double) -> Double) {
        ...
    }

    func multiply(_ op1: Double, _ op2: Double) -> Double {
        return op1 * op2
    }
}
```

Model: CalcModel.swift

```
import Foundation

class CalcModel: NSObject {
    var stack = [Double]()

    func pushOperand(value: Double) {
        self.stack.append(value)
        print("stack = \(self.operandStack)")
    }

    func eval(operation: String) -> Double {
        ...
        return self.operandStack.last!
    }

    func performBinaryOperation(operation: (Double, Double) -> Double) {
        ...
    }

    func performUnaryOperation(operation: (Double) -> Double) {
        ...
    }

    func multiply(_ op1: Double, _ op2: Double) -> Double {
        return op1 * op2
    }
}
```

Specifying Double as the return type

Model: CalcModel.swift

```
import Foundation

class CalcModel: NSObject {
    var stack = [Double]()

    func pushOperand(value: Double) {
        self.stack.append(value)
        print("stack = \(self.operandStack)")
    }

    func eval(operation: String) -> Double {
        ...
        return self.operandStack.last!
    }

    func performBinaryOperation(operation: (Double, Double) -> Double) {
        ...
    }

    func performUnaryOperation(operation: (Double) -> Double) {
        ...
    }

    func multiply(_ op1: Double, _ op2: Double) -> Double {
        return op1 * op2
    }
}
```

The name of this method is `multiply(_:_:)'

Model: CalcModel.swift

```
import Foundation

class CalcModel: NSObject {
    var stack = [Double]()

    func pushOperand(value: Double) {
        self.stack.append(value)
        print("stack = \(self.operandStack)")
    }

    func eval(operation: String) -> Double {
        ...
        return self.operandStack.last!
    }

    func performBinaryOperation(operation: (Double, Double) -> Double) {
        ...
    }

    func performUnaryOperation(operation: (Double) -> Double) {
        ...
    }

    func multiply(_ op1: Double, _ op2: Double) -> Double {
        return op1 * op2
    }
}
```

Semicolons are optional, do omit!

Controller: ViewController.swift

```
import UIKit

class ViewController: UIViewController {
    @IBOutlet weak var calcDisplay: UILabel!
    var calcModel = CalcModel()
    var inputMode = false

    @IBAction func digitPressed(sender: UIButton) {
        let digit = sender.currentTitle!
        ...
    }

    @IBAction func operationPressed(sender: UIButton)
    {
        let operation = sender.currentTitle!
        ...
        self.displayValue = self.calcModel.eval(operation)
    }

    var displayValue: Double {
        get {
            return NSNumberFormatter().numberFromString(self.calcDisplay.text!)!.doubleValue
        }
        set { self.calcDisplay.text = "\(newValue)"; self.inputMode = false }
    }
}
```

Our Controller inherits from UIViewController. UIKit supports MVC primarily through this class.

Controller: ViewController.swift

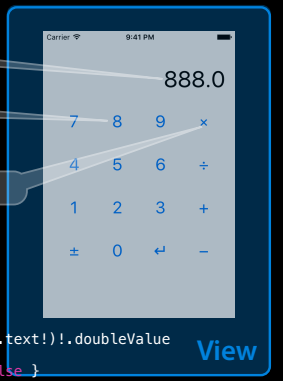

```
import UIKit

class ViewController: UIViewController {
    @IBOutlet weak var calcDisplay: UILabel!
    var calcModel = CalcModel()
    var inputMode = false

    @IBAction func digitPressed(sender: UIButton) {
        let digit = sender.currentTitle!
        ...
    }

    @IBAction func operationPressed(sender: UIButton)
    {
        let operation = sender.currentTitle!
        ...
        self.displayValue = self.calcModel.eval(operation)
    }

    var displayValue: Double {
        get {
            return NSNumberFormatter().numberFromString(self.calcDisplay.text!)!.doubleValue
        }
        set { self.calcDisplay.text = "\(newValue)"; self.inputMode = false }
    }
}
```

This is our model CalcModel

Controller: ViewController.swift

These hook up to our View

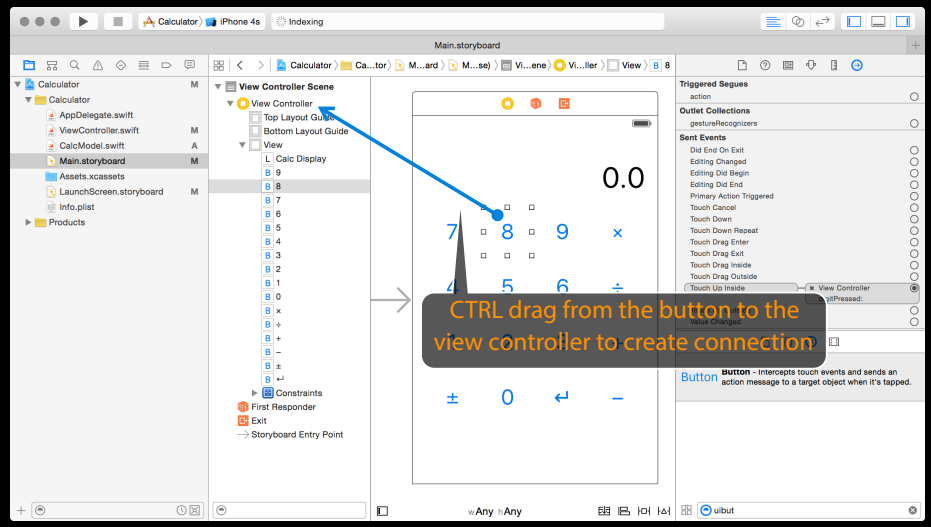
```
import UIKit

class ViewController: UIViewController {
    @IBOutlet weak var calcDisplay: UILabel!
    var calcModel = CalcModel()
    var inputMode = false

    @IBAction func digitPressed(sender: UIButton) {
        let digit = sender.currentTitle!
        ...
    }

    @IBAction func operationPressed(sender: UIButton)
    {
        let operation = sender.currentTitle!
        ...
        self.displayValue = self.calcModel.eval(operation)
    }

    var displayValue: Double {
        get {
            return NSNumberFormatter().numberFromString(self.calcDisplay.text!)!.doubleValue
        }
        set { self.calcDisplay.text = "\(newValue)"; self.inputMode = false }
    }
}
```

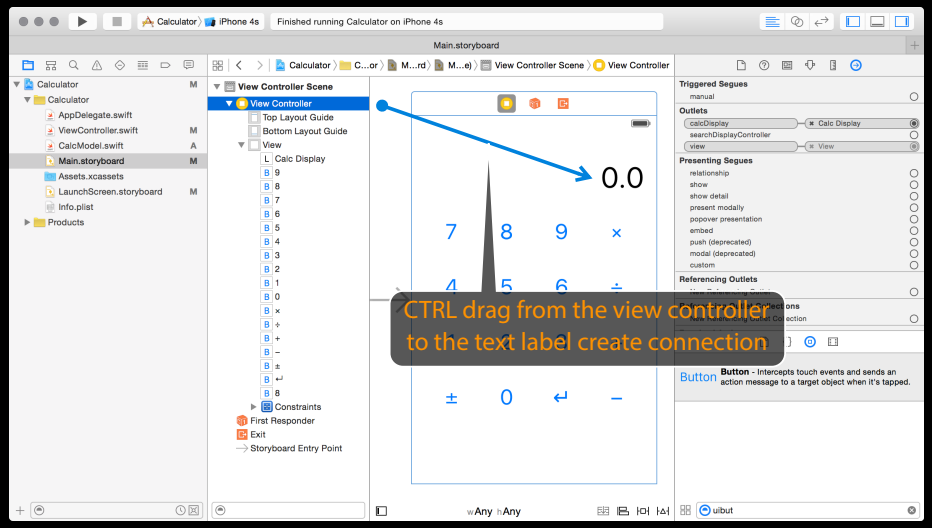


Controller: ViewController.swift


```
import UIKit

class ViewController: UIViewController {
    @IBOutlet weak var calcDisplay: UILabel!
    var calcModel = CalcModel()
    var inputMode = false

    @IBAction func digitPressed(sender: UIButton) {
        let digit = sender.currentTitle!
        ...
    }

    @IBAction func operationPressed(sender: UIButton)
    {
        let operation = sender.currentTitle!
        ...
        self.displayValue = self.calcModel.eval(operation)
    }

    var displayValue: Double {
        get {
            return NSNumberFormatter().numberFromString(self.calcDisplay.text!)!.doubleValue
        }
        set { self.calcDisplay.text = "\(newValue)"; self.inputMode = false }
    }
}
```

Model





Demo

RPN Calculator

Demos

- **Take away**
  - Creating a Project in Xcode 10
  - Building a responsive UI using Autolayout
  - The iOS Simulator
- **Obverse and take note (we'll come back over)**
  - language syntax
  - defining class in Swift, adding instance variables (properties) and behaviour (methods)
  - print and String interpolation
  - connecting Swift code and UI (outlets, actions)

Time for study