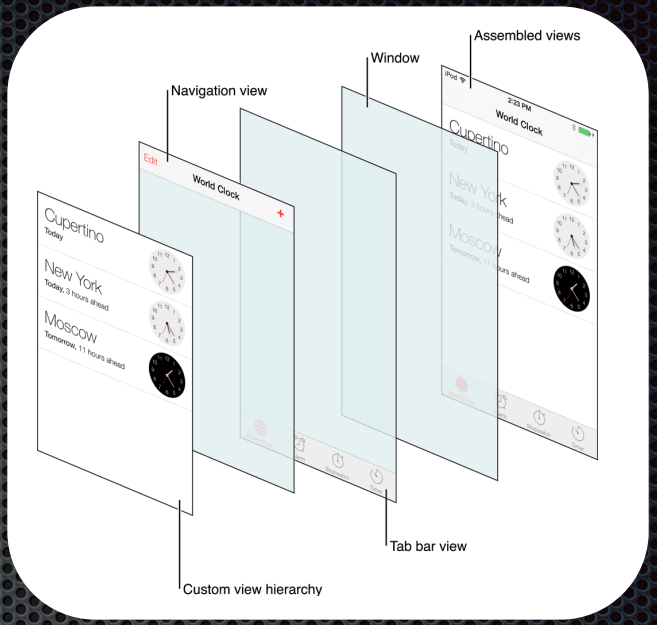
Mobile Application Programming

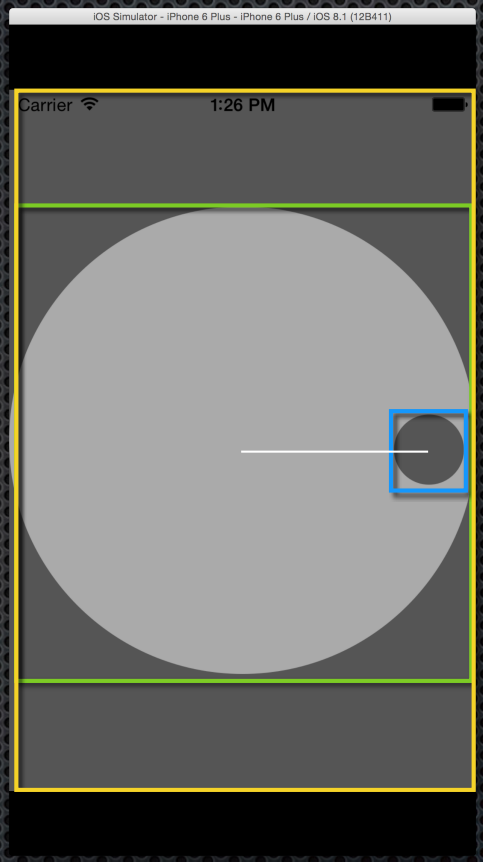
Controls

Views



- UIView instances and subclasses
- Form a tree rooted at the window
- Have a backing store of pixels that are drawn seldomly, then composited to form the full user interface
- Coordinate system for drawing uses bounds property
- Positioned within parent using center or frame property
- Subclasses typically override the draw method
- Receive touches overriding super-class methods touchesBegan, touchesMoved, and touchesEnded

Example: Knob



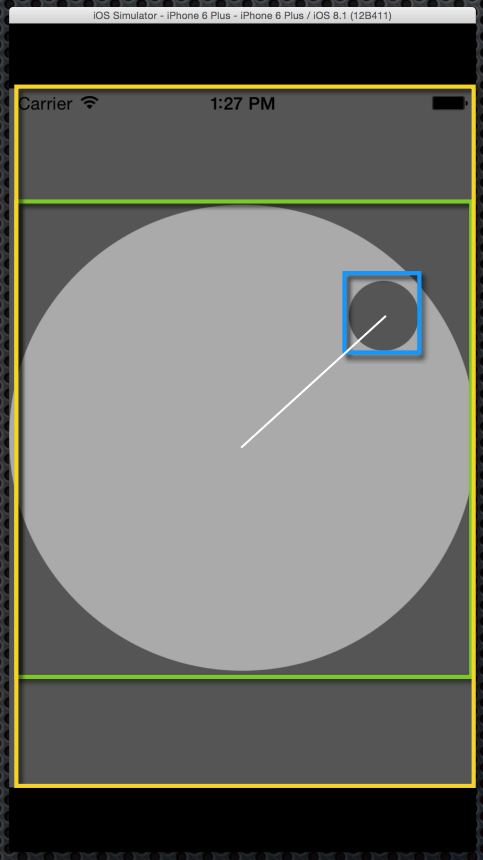
knobView.bounds

knobRect

nibRect

angle

Example: Knob



knobView.bounds

knobRect

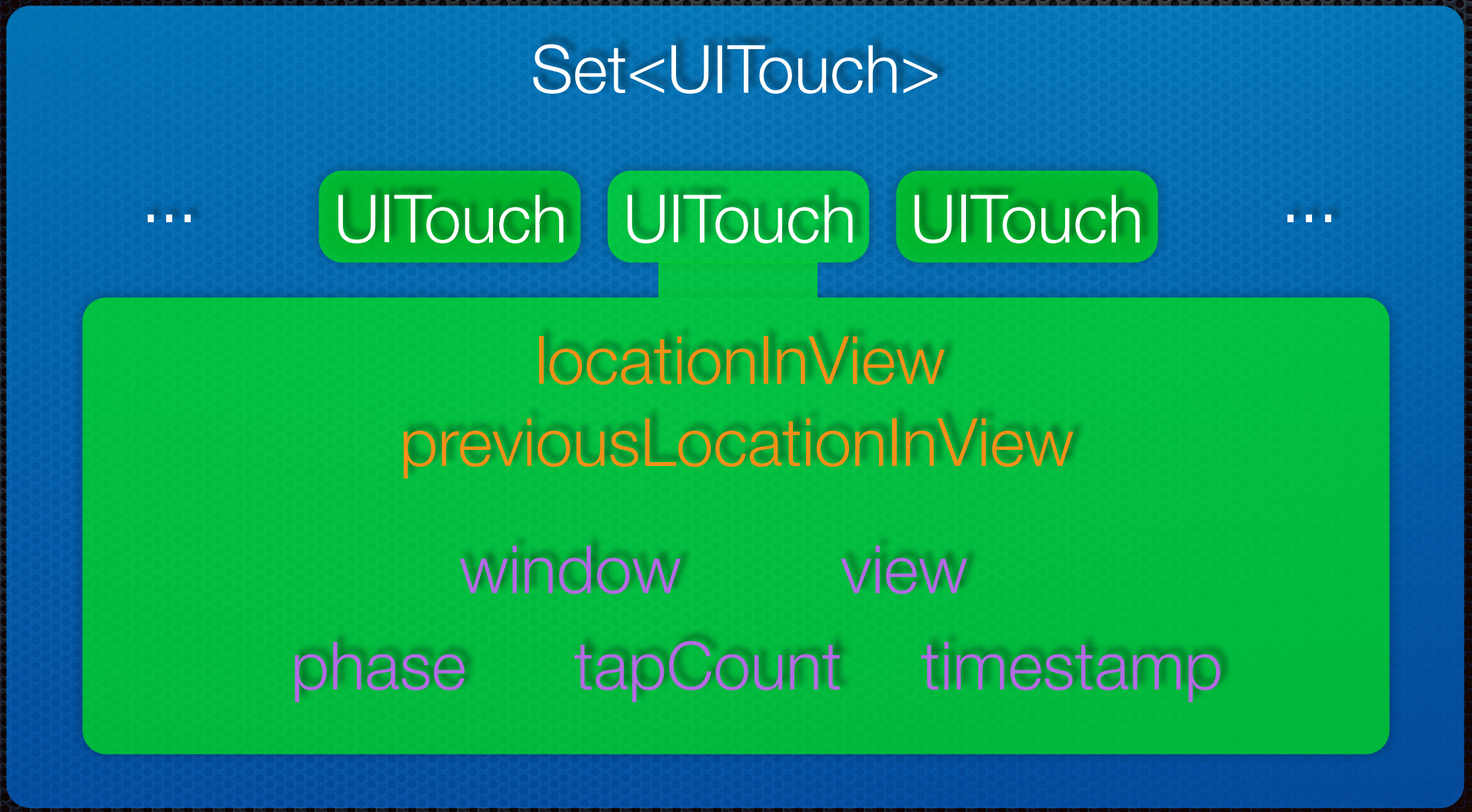
nibRect

angle

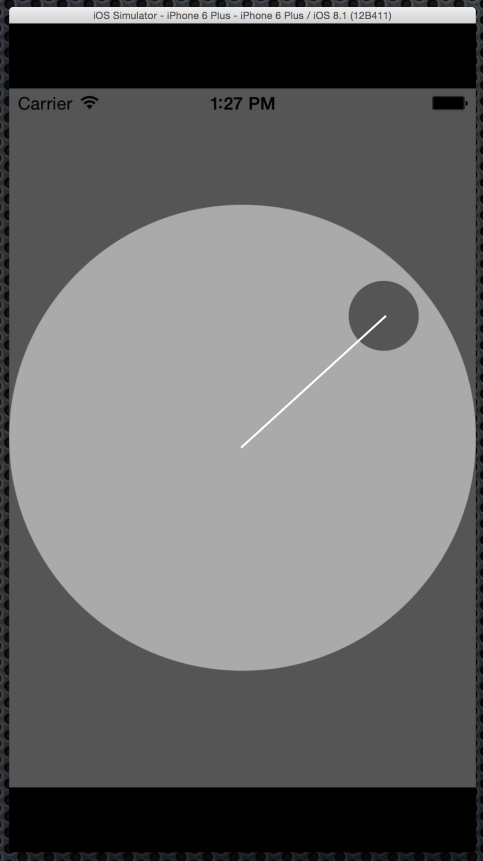
Touches

- Sub-class UIResponder or UIView
  - touchesBegan(touches: Set<UITouch> with event:)
  - touchesMoved(touches: Set<UITouch> with event:)
  - touchesEnded(touches: Set<UITouch> with event:)
  - touchesCancelled(touches: Set<UITouch> with event:)

Touches


Set<UITouch>
...
UITouch
UITouch
UITouch
...
locationInView
previousLocationInView
window
view
phase
tapCount
timestamp

Problem: Notifying of Change



Controlling Object

KnobView

Target-Action

Target-Action

Controlling Object

Reference ↓ stored property

Switch (UIControl)

Target-Action

addTarget(action, forControlEvents)

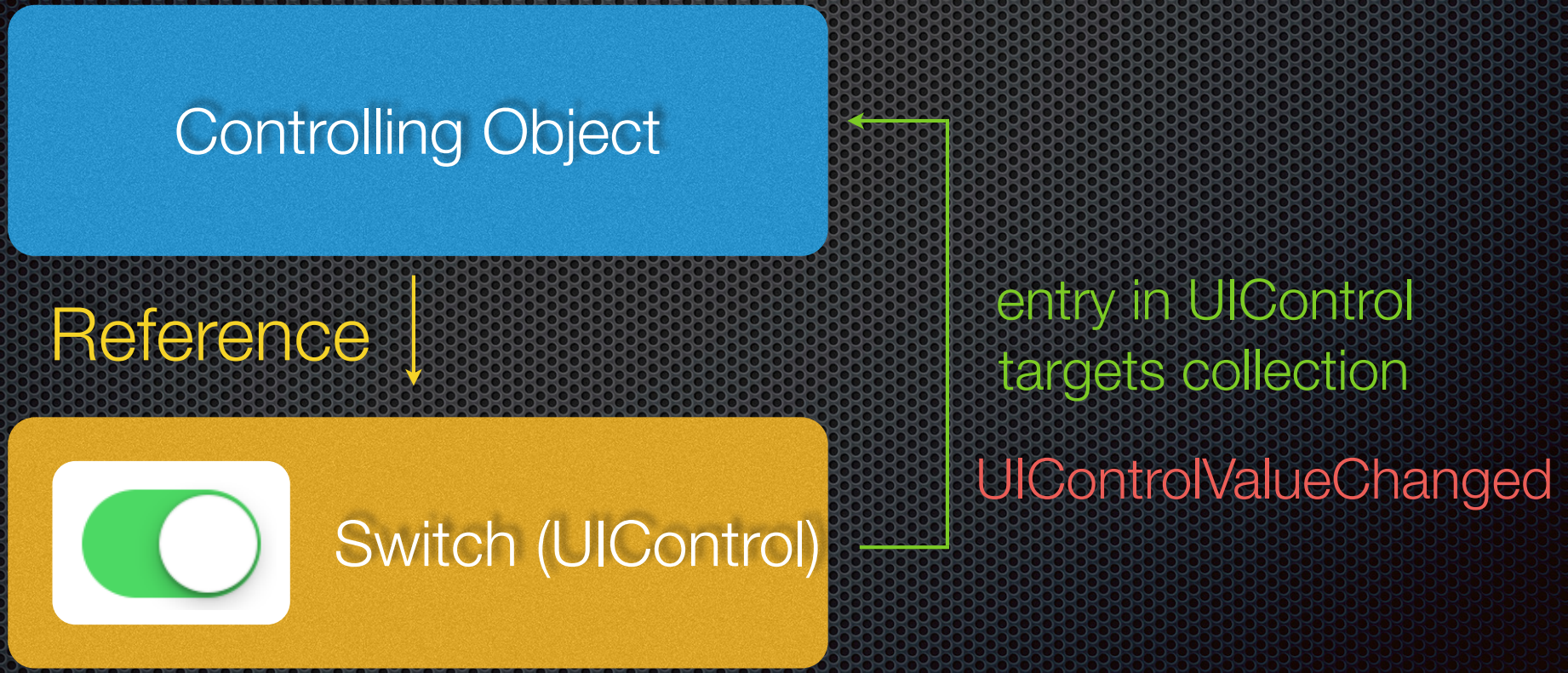
Controlling Object

Reference

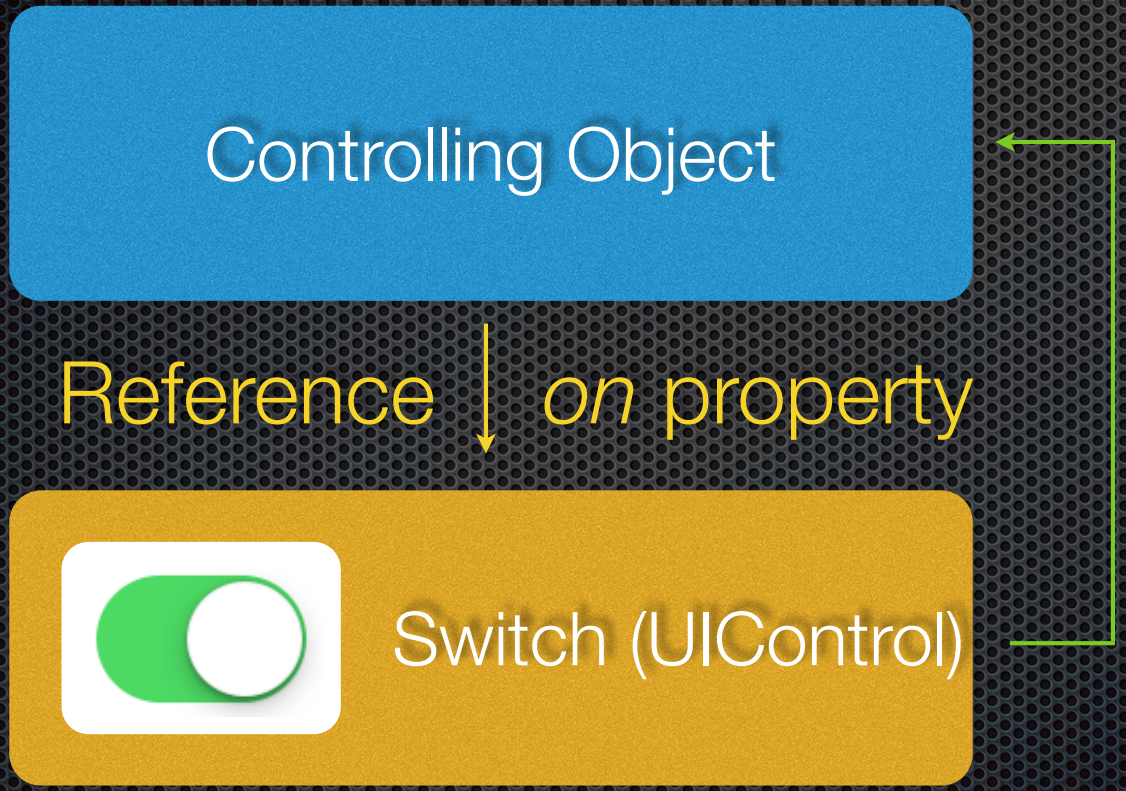
Switch (UIControl)

entry in UIControl
targets collection

Target-Action



Target-Action



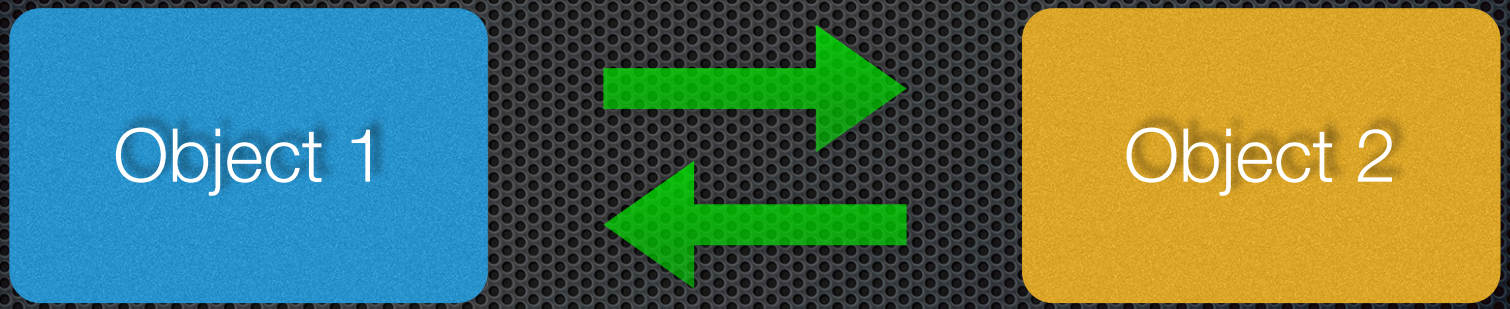
UIControl

- Uses the Target-Action Mechanism
- Allows simple distribution of events generated by controls, such as UIControlEvent.ValueChanged
- Interested parties call addTarget(action, forControlEvents) to receive updates
- Call sendActionsForControlEvents() to alert interested parties when your control has an event occur
- You can send custom events as well

UIControl

- State
  - Enabled, Highlighted, Selected
- Tracking
  - beginTracking, continueTracking, endTracking
- Content Alignment
  - Horizontal - Left, Center, Right, Fill
  - Vertical - Left, Center, Right, Fill

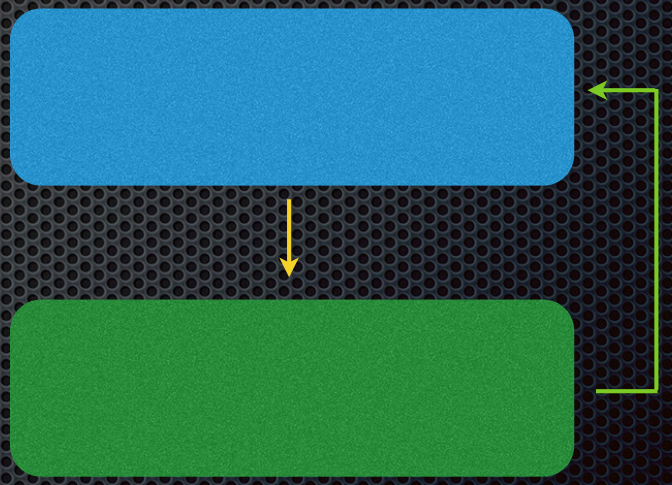
Problem: 2 Objects Talking



Problem: 2 Objects Talking

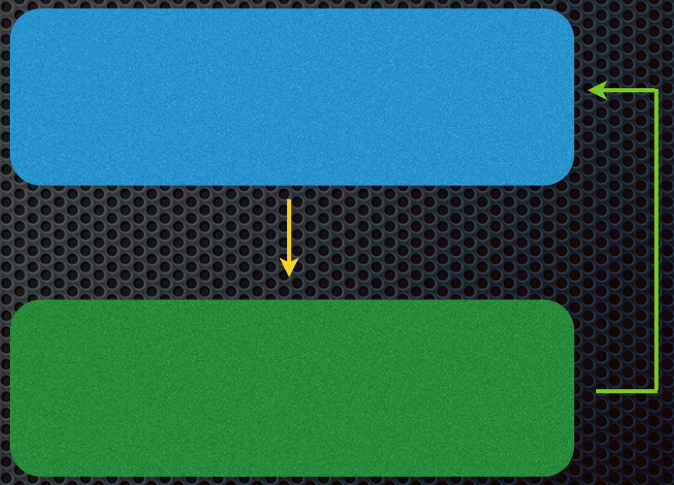
Delegates

- Assist in a task that a class doesn't know how to do itself, and thus is an alternative to subclassing complex classes
- Also useful for sending messages from a class to its containing class



Delegates

- Assist in a task that a class doesn't know how to do itself, and thus is an alternative to subclassing complex classes
- Also useful for sending messages from a class to its containing class



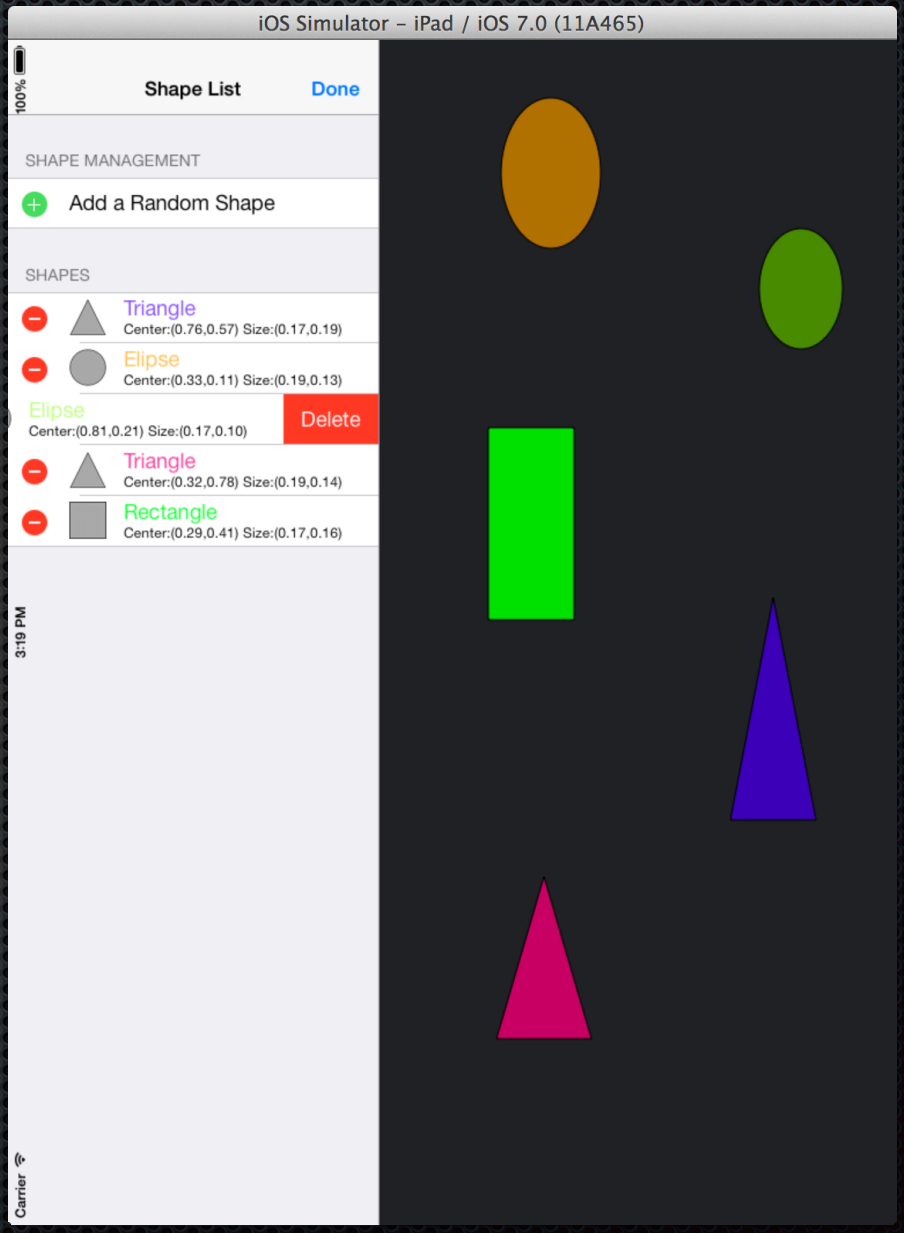
Delegates

UIApplication

application(didFinishLaunchingWithOptions:)

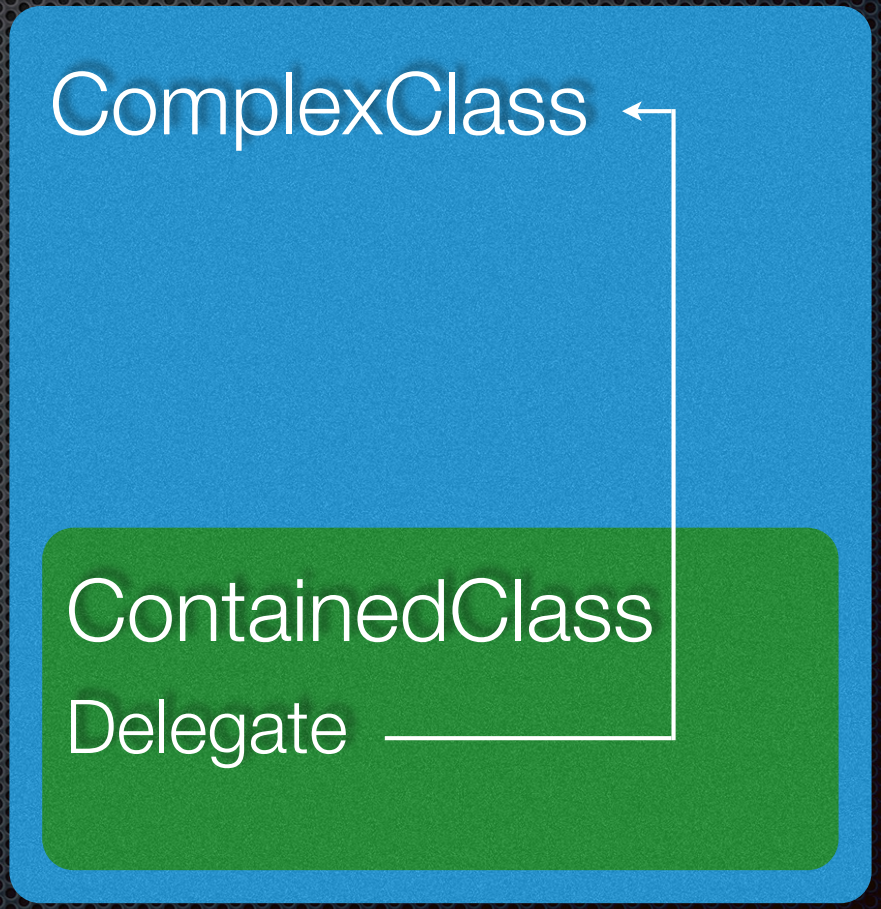
UIApplicationDelegate

buildUI



Delegates

- Assist in a task that a class doesn't know how to do itself, and thus is an alternative to subclassing complex classes
- Also useful for sending messages from a class to its containing class



Delegates

- A delegate is an object that performs actions on the behalf of another object
- A common use is a data model object alerting a controller of changes to its data, which then tells view objects about the change
- Another use of them is a view object having a controller object interact with the program data model on its behalf when the user triggers events
- 6 bits of code are required to properly set up both sides of a delegate connection between two objects

```swift
import UIKit

protocol KnobDelegate: class
{
    func knob(knob: Knob, rotatedToAngle angle: Float)
}

class Knob : UIView
{
    private var _knobRect: CGRect = CGRectZero
    private var _angle: Float = 3.0 * Float(M_PI) / 2.0

    var angle: Float
    {
        get { return _angle }
        set
        {
            _angle = newValue
            setNeedsDisplay()
        }
    }

    weak var delegate: KnobDelegate? = nil

    override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?)
    {
        let touch: UITouch = touches.anyObject() as UITouch
        let touchPoint: CGPoint = touch.locationInView(self)
        let touchAngle: Float = atan2f(
            Float(touchPoint.y - _knobRect.midY),
            Float(touchPoint.x - _knobRect.midX))

        angle = touchAngle
        delegate?.knob(self, rotatedToAngle: angle)
    }

    override func draw(_ rect: CGRect)
    {
    }
}
```

1. Delegate Protocol
2. Delegate Property
3. Delegate Invocation

```swift
import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, KnobDelegate
{
    var window: UIWindow?

    func application(application: UIApplication,
       didFinishLaunchingWithOptions l: [NSObject: AnyObject]?) -> Bool
    {
        window = UIWindow(frame: UIScreen.mainScreen().bounds)
        window?.makeKeyAndVisible()

        var knob: Knob = Knob(frame: window!.frame)
        knob.backgroundColor = UIColor.darkGrayColor()
        knob.delegate = self
        window?.addSubview(knob)

        return true
    }

    func knob(knob: Knob, rotatedToAngle angle: Float)
    {
        println("Knob rotated to angle: \(angle)")
    }
}
```

```swift
import UIKit

protocol KnobDelegate: class
{
    func knob(knob: Knob, rotatedToAngle angle: Float)
}

class Knob : UIView
{
    private var _knobRect: CGRect = CGRectZero
    private var _angle: Float = 3.0 * Float(M_PI) / 2.0

    var angle: Float
    {
        get { return _angle }
        set
        {
            _angle = newValue
            setNeedsDisplay()
        }
    }

    weak var delegate: KnobDelegate? = nil

    override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?)
    {
        let touch: UITouch = touches.anyObject() as UITouch
        let touchPoint: CGPoint = touch.locationInView(self)
        let touchAngle: Float = atan2f(
            Float(touchPoint.y - _knobRect.midY),
            Float(touchPoint.x - _knobRect.midX))

        angle = touchAngle
        delegate?.knob(self, rotatedToAngle: angle)
    }

    override func draw(_ rect: CGRect)
    {
    }
}
```

1. Delegate Protocol
2. Delegate Property
3. Delegate Invocation

```swift
import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, KnobDelegate
{
    var window: UIWindow?

    func application(application: UIApplication,
       didFinishLaunchingWithOptions l: [NSObject: AnyObject]?) -> Bool
    {
        window = UIWindow(frame: UIScreen.mainScreen().bounds)
        window?.makeKeyAndVisible()

        var knob: Knob = Knob(frame: window!.frame)
        knob.backgroundColor = UIColor.darkGrayColor()
        knob.delegate = self
        window?.addSubview(knob)

        return true
    }

    func knob(knob: Knob, rotatedToAngle angle: Float)
    {
        println("Knob rotated to angle: \(angle)")
    }
}
```

4. Delegate Protocol Conformity
5. Delegate Assignment
6. Delegate Protocol Method(s)

```swift
import UIKit

protocol KnobDelegate: class
{
    func knob(knob: Knob, rotatedToAngle angle: Float)
}

class Knob : UIView
{
    private var _knobRect: CGRect = CGRectZero
    private var _angle: Float = 3.0 * Float(M_PI) / 2.0

    var angle: Float
    {
        get { return _angle }
        set
        {
            _angle = newValue
            setNeedsDisplay()
        }
    }

    weak var delegate: KnobDelegate? = nil

    override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?)
    {
        let touch: UITouch = touches.anyObject() as UITouch
        let touchPoint: CGPoint = touch.locationInView(self)
        let touchAngle: Float = atan2f(
            Float(touchPoint.y - _knobRect.midY),
            Float(touchPoint.x - _knobRect.midX))

        angle = touchAngle
        delegate?.knob(self, rotatedToAngle: angle)
    }

    override func draw(_ rect: CGRect)
    {
    }
}
```

1. Delegate Protocol

2. Delegate Property

3. Delegate Invocation

The method invocation here...

```swift
import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, KnobDelegate
{
    var window: UIWindow?

    func application(application: UIApplication,
       didFinishLaunchingWithOptions l: [NSObject: AnyObject]?) -> Bool
    {
        window = UIWindow(frame: UIScreen.mainScreen().bounds)
        window?.makeKeyAndVisible()

        var knob: Knob = Knob(frame: window!.frame)
        knob.backgroundColor = UIColor.darkGrayColor()
        knob.delegate = self
        window?.addSubview(knob)

        return true
    }

    func knob(knob: Knob, rotatedToAngle angle: Float)
    {
        println("Knob rotated to angle: \(angle)")
    }
}
```

goes here.

4. Delegate Protocol Conformity

5. Delegate Assignment

6. Delegate Protocol Method(s)

Closure Properties

- Self-contained code snippets that can be invoked at a later time to accomplish a task
- Have a syntax that looks odd at first, but is consistent
- Are similar to listener objects with one method in Java or lambda expressions in other languages
- Are called “closures” because they store the current values of referenced variables at the time of their creation, thus “closing over” the referenced data
- Result in very disorganized code if used imprudently