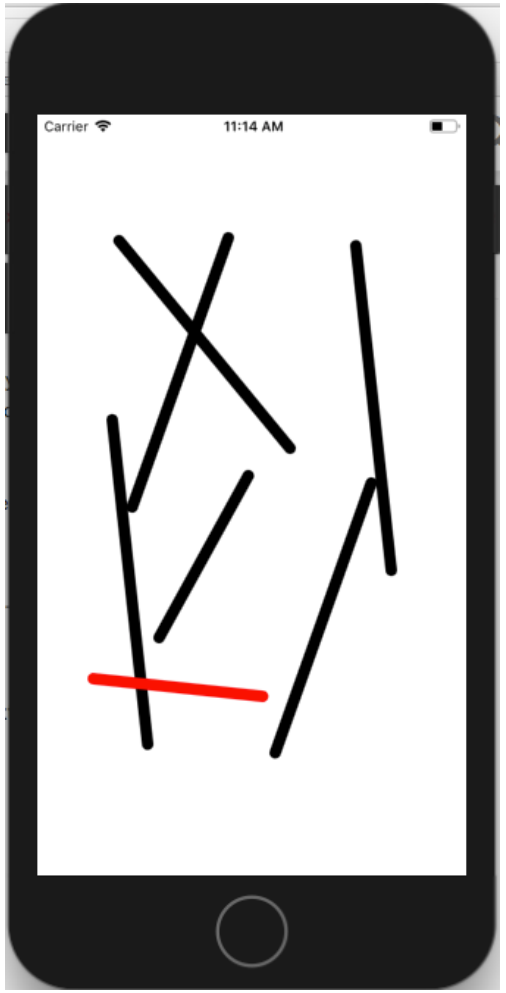
# iOS: Touch Events and `UIResponder`

BNRG CHAPTER 18

# TouchTracker

A drawing app

Lets users draw pictures by touching and dragging on the screen



# Touch Events

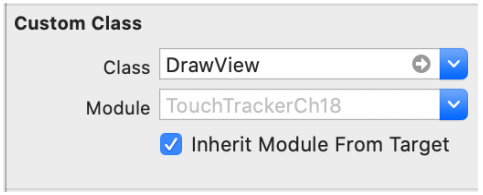
`UIView` is a subclass of `UIResponder`

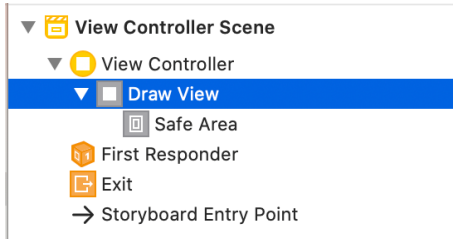
- and `UIView` is the parent class of all of all GUI widgets

`UIResponder` provides four methods to handle touch events

So to get direct access to touch events, we can do the following:

- create a custom class that subclasses `UIView`
- associate this class with the default view of a view controller
- override the touch-event methods in the class



# Touch Events

Here are the methods that handle raw touch events:

(1) one or more fingers touches the screen

```
func touchesBegan(_ touches: Set<UITouch>, with event UIEvent?)
```

# Touch Events

(2) one or more fingers move across the string; this message is sent repeatedly as a finger moves

```
func touchesMoved(_ touches: Set<UITouch>, with event UIEvent?)
```

# Touch Events

(3) one or more fingers are removed from the screen

```
func touchesEnded(_ touches: Set<UITouch>, with event UIEvent?)
```

# Touch Events

(4) a system event, such as an incoming phone call, interrupts a touch before it ends

```
func touchesCancelled(_ touches: Set<UITouch>, with event UIEvent?)
```

# Lifecycle of a Touch Event

When the user's finger touches the screen, an instance of `UITouch` is created

- the `UIView` that the finger touched is sent the `TouchesBegan(_:with:)` message
- the `UITouch` event is passed to the method in the set of touches

As the finger moves across the screen, the touch object is updated to contain the current location of the finger on the screen

- and the same `UIView` that received the original message is sent the message `touchesMoved(_:with:)`
- even if the finger is no longer inside the frame of the original `UIView`, it's still this original `UIView` that receives the subsequent `touchesMoved(_:with:)` messages

# Lifecycle of a Touch Event

When the finger is removed from the screen, the touch object is updated one last time to contain the final location of the finger

- and the view in which the touch began is sent the message `touchesEnded(_:with:)`
- after that method finishes, the `UITouch` event is destroyed

# Notes

One `UITouch` object corresponds to one finger

- this object lives as long as the finger is is on the screen
- and it always contains the current position of the finger on the screen

Never keep a reference to a `UITouch` object in your code

- use the one that is passed to the touch methods

The `UIApplication` manages the `UITouch` objects

- it maintains a queue
- and it sends the touch messages to the various `UIView` objects that it controls

# Multiple Touches

If more than one finger does the same thing "at the same time", then messages for all of these touch events are delivered at once, in a set (a Swift `Set`)

- in this case, there is a `UITouch` object for each finger in the set that is passed to the touch method

But the window of opportunity for "at the same time" is fairly short

- so instead of one message for all of the touches, there could be more than one message for one or more of the touches

# Model Object: a Line

The model objects for this app are the lines that a user draws on the screen

```
import CoreGraphics

struct Line {
    var begin = CGPoint.zero
    var end = CGPoint.zero
}
```

`CGPoint`

- A structure that contains a point in a two-dimensional coordinate system.
- Part of the CoreGraphics framework

# Structs in Swift: Review

- Remember: Swift structs do not support inheritance
- Structs get a memberwise initializer if no other initializers are declared
  - for Line, it would be `init(begin: CGPoint, end: CGPoint)`
- If all properties are given default values and no other initializers are declared, then structs also get a free empty initializer that creates an instance and sets the properties to their default values
  - `init()`
- Structs are value types, as opposed to classes, which are reference types
  - recall the difference between reference types and value types: reference types are like pointers

# Custom Views

If we want a view that isn't one of the standard views available in the Interface Builder View Library

- the we have to build a custom view, in code

A custom view can be a combination of other views

- label + graphic element + status bar

It can also be a specialization of an existing view

- with added function, for example
- this is what we use for this app, by overriding the `UITouch` methods

# New Custom View: `DrawView`

Subclass of `UIView`

Will be the view of the application's rootViewController

Will turn touch events into lines on the screen:

- each `Line` has a begin point and an end point
- when a touch event happens, the app creates a `Line` and sets both of its points to the place where the touch event begins
- as the touch moves, the app updates the end point of the `Line`
- when the touch ends, we have a complete `Line`

# Drawing a View

The graphics subsystem will periodically (and efficiently) redraw all currently displayed views

- for text, or a button, or a slider, it's obvious how the view needs to be redrawn

For a custom view, we have to specify what should be redrawn

- and to do this, we override `draw(_:)`
- but we never call `draw(_:)` ourselves
- instead, we call `setNeedsDisplay()`
- this tells the UI system to call call our view's `draw(_:)` during the next refresh cycle

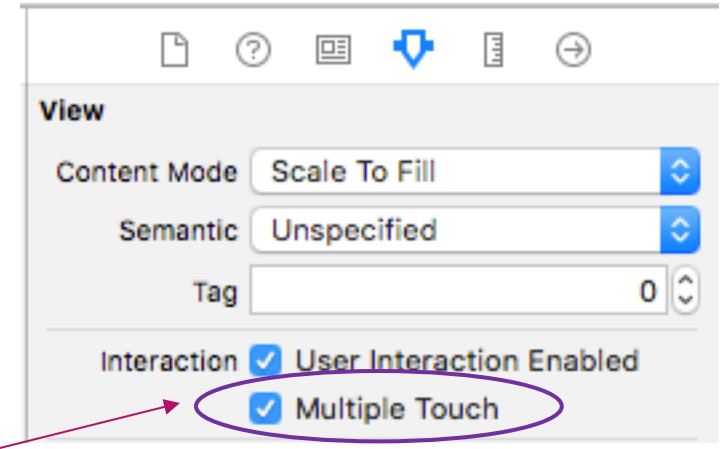
# Draw

Here's what the draw will do

```
override func draw(_ rect: CGRect) {
  // set current drawing color to black
  // for line in finishedLines {
  //   draw line
  // }
  // if there is a line currently being drawn {
  //   set color to red
  //   draw line
  // }
}
```

# Multiple Touches

- The mouse will give you a single touch
- Note: in the simulator, hold down the option key and click the mouse
  - and this simulates two simultaneous touches
- But, at this point, the app doesn't handle multiple touches
- By default, a view will accept only one touch at a time
  - if one finger has already triggered `touchesBegan(_:with:)` but has not finished, the subsequent touches are ignored
  - i.e., no other touch methods are called for the second touch until the first touch finishes
- Must explicitly enable multiple touches in the view controller



# Multiple Touches

Even with multiple touches enabled in the view controller, there's still a problem

The code in DrawView (in the initial implementation) assumes that there will be only one line being drawn at any time (`currentLine`)

Need to modify DrawView to have a collection of active lines

- we'll use a dictionary
- the key will be something we derive from an active `UITouch`
- the value will be a `Line`
- so the dictionary will be something like this:

```
currentLines[UITouch_n] = currentLine_n
```

# The Actual Dictionary

Define the dictionary:

```
var currentLines = [NSValue:Line]()
```

Create a new line during touchesBegan(_:with:)

```
let newLine = Line(begin: location, end: location)
let key = NSValue(nonretainedObject: touch)
currentLines[key] = newLine
```

# nonretainedObject

Why not use the `UITouch` object itself as the key?

- "Never keep a strong reference to a `UITouch` object"
- this has to do with Swift's memory management
- and the bottom line is that we don't want to compromise Swift's efficient memory management

Safe Alternative: create an instance of `NSValue` that is derived from the `UITouch` object

- “wrap the memory address of the `UITouch` in an instance of `NSValue`”
- “useful if you want to add an object to a collection but don’t want the collection to create a strong reference to it”

# @IBInspectable

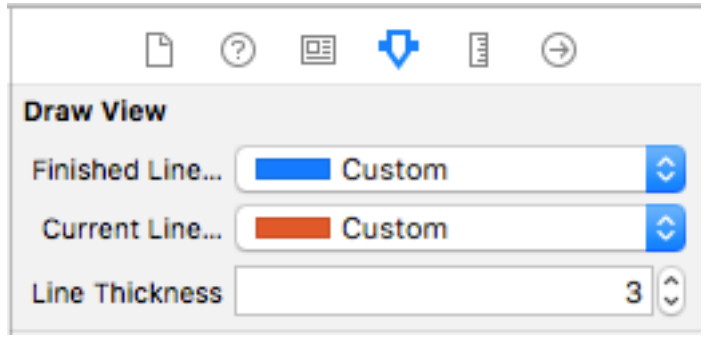
`@IBInspectable` exposes a property in a `UIView` to Interface Builder

- tells Interface Builder that it's a property that you want to customize through the attributes inspectors
- this way, we can change the property in the attributes inspect for the `UIView`

```
@IBInspectable
var finishedLineColor: UIColor = UIColor.black {
    didSet {
        setNeedsDisplay()
    }
}

@IBInspectable
var currentLineColor: UIColor = UIColor.red {
    didSet {
        setNeedsDisplay()
    }
}

@IBInspectable
var lineThickness: CGFloat = 10 {
    didSet {
        setNeedsDisplay()
    }
}
```



# Challenges

The Silver Challenge and Gold Challenge in Chapter 18 are excellent

- do them for practice!

# More Info

The book has a little more information about the responder chain

- how a UI event is passed to all of the objects in a the object hierarchy rooted at `UIApplication`

And also some more info about types of touch events