Custom Controls in iOS

Hands-On Challenges

Custom Controls in iOS

Catie & Jessy Catterwaul

Challenge #6: Advanced Layers

By Catie & Jessy Catterwaul

Now that you've got a snazzy-looking ring layer, it's time to wrap it up inside a UIView along with two other instances of `RingLayer`, to make the three-ring control you want.

The version of `ThreeRingControl.playground` in the `Challenge Start` folder has a class waiting for you, that has been partially set up:

```
public final class ThreeRingView: UIView
```

Familiarize yourself a little bit with this class, including the `ringLayers` property, a dictionary whose keys are the cases of the `Ring` enumeration, which can be found in `Sources`.

```
fileprivate let ringLayers: [Ring: RingLayer] = [
  .inner: RingLayer(),
  .middle: RingLayer(),
  .outer: RingLayer()
]
```

Public API

`innerRingValue` and `outerRingValue` demonstrate `ringLayers` in action, using computed `get` and `set` accessors to encapsulate an API that wouldn't be suitable for public use. Follow their example for `middleRingValue`:

```
var middleRingValue: CGFloat {
  get {
    return ringLayers[.middle]!.value
  }
  set {
    ringLayers[.middle]!.value = newValue
  }
}
```

Similarly, use the code from `innerRingColor` and `outerRingColor` to learn what to do for `middleRingColor`. Wrapping up the conversion between `UIColor`, which you'll need to work with externally, and `CGColor`, will keep the public API tidy.

```
var middleRingColor: UIColor {
  get {
    return UIColor(cgColor: ringLayers[.middle]!.ringColor)
  }
  set {
    ringLayers[.middle]!.ringColor = newValue.cgColor
  }
}
```

Initialization

In `initPhase2`, each ring layer is given common default values. Use the three computed ring color properties to assign different default colors to each ring:

```
private func initPhase2() {
  backgroundColor = UIColor.black
  for ringLayer in ringLayers.values {
    layer.addSublayer(ringLayer)
    ringLayer.backgroundColor = UIColor.clear.cgColor
    ringLayer.ringBackgroundColor = ringBackgroundColor.cgColor
    ringLayer.ringWidth = ringWidth
    ringLayer.value = 0
  }

  innerRingColor = UIColor(cgColor: Color.pink)
  middleRingColor = UIColor(cgColor: Color.blue)
  outerRingColor = UIColor(cgColor: Color.green)
}
```

Positioning and Sizing

All that's left is to complete the `drawLayers` method. First, calculate two constants.

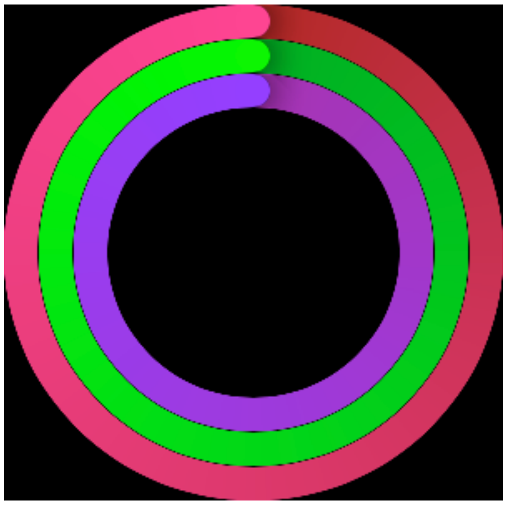
```
func drawLayers() {
  // the largest a ring can be,
  // and still fit withing the bounds of the current view
  let maxSize = min(bounds.width, bounds.height)

  // measured between adjacent rings
  let sizeDifference = (ringWidth + ringPadding) * 2
}
```

Then, loop through the ringLayers, assigning the bounds of each one according to its index, and setting the position to be the same for all rings.

```
func drawLayers() {
  // the largest a ring can be,
  // and still fit withing the bounds of the current view
  let maxSize = min(bounds.width, bounds.height)

  // measured between adjacent rings
```

```
    let sizeDifference = (ringWidth + ringPadding) * 2

    for (index, ringLayer) in ringLayers {
      let size =
        maxSize
        - sizeDifference * CGFloat(2 - index.rawValue)
      ringLayer.bounds = CGRect(
        x: 0,
        y: 0,
        width: size,
        height: size
      )

      ringLayer.position = layer.position
    }
  }
```

Congratulations! You've got three rings!



You're not quite done yet, though. Try changing `threeRingView`'s `ringWidth`, `ringPadding`, or `ringBackgroundColor` at the bottom of the Playround page, and you'll see that nothing happens. That won't do!

Property Observers

The three public properties defined at the top of the class need `didSet` observers. Each one needs to do a little different work, in order to update the view appropriately.

ringWidth

Changing `ringWidth` requires a call to `drawLayers`, and also an update to the `ringWidth` property of each ring layer.

```
  public var ringWidth: CGFloat = 20 {
    didSet {
      drawLayers()
      for ringLayer in ringLayers.values {
        ringLayer.ringWidth = ringWidth
      }
    }
  }
```

ringPadding

`ringPadding` changes also necessitate `drawLayers`, but no properties need to change on the ring layers themselves. You're only updating their arrangement, with this property.

```
  public var ringPadding: CGFloat = 1 {
    didSet {
      drawLayers()
    }
  }
```

ringBackgroundColor

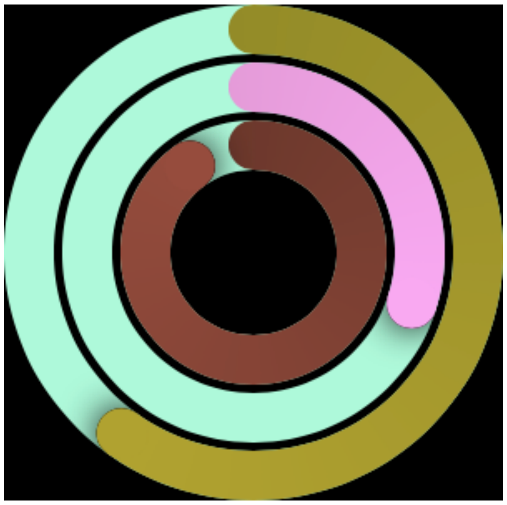
Changes to `ringBackgroundColor` are forwarded to the rings themselves, and `drawLayers` is not required.

```
  public var ringBackgroundColor = UIColor.darkGray {
    didSet {
      for ringLayer in ringLayers.values {
        ringLayer.ringBackgroundColor =
          ringBackgroundColor.cgColor
      }
    }
  }
```

You're Done!

Feel free to play around with the public API properties, that all work as expected now. You've earned it!

```
  threeRingView.innerRingValue
  threeRingView.middleRingValue
  threeRingView.outerRingValue

  threeRingView.ringWidth
  threeRingView.ringPadding

  threeRingView.ringBackgroundColor
  threeRingView.innerRingColor
  threeRingView.middleRingColor
  threeRingView.outerRingColor
```



Very modern-looking!