CS275
Fall 2021

iOS: IBDesignable and Customizing Views

BONUS MATERIAL

Standard Views

Standard UI widgets include `UILabel`, `UIButton`, `UISlider` etc.

- these are OK
- but from Interface Builder they're a little boring
- more specifically: they just don't have very many aspects that can be customized

Customizing the Standard Views

One way to customize a `UIView` widget

- by creating programmatic views
- in other words, by creating UI elements in code
- this then gives us full control over the appearance and behavior of views
- you'll recall that we did this in Ch. 5 :-)

Programmatic Views

Example from Ch. 5

```
let segmentedControl = UISegmentedControl(items: ["Standard", "Hybrid", "Satellite"])
segmentedControl.backgroundColor = UIColor.systemBackground
segmentedControl.selectedSegmentIndex = 0

segmentedControl.translatesAutoresizingMaskIntoConstraints = false
view.addSubview(segmentedControl)
let topConstraint = segmentedControl.topAnchor.constraint(equalTo: view.safeAreaLayoutGuide.topAnchor, constant: 8)

let margins = view.layoutMarginsGuide
let leadingConstraint = segmentedControl.leadingAnchor.constraint(equalTo: margins.leadingAnchor)
let trailingConstraint = segmentedControl.trailingAnchor.constraint(equalTo: margins.trailingAnchor)

topConstraint.isActive = true
leadingConstraint.isActive = true
trailingConstraint.isActive = true

segmentedControl.addTarget(self, action: #selector(mapTypeChanged(_:)), for: .valueChanged)
```

Programmatic Views

Advantages

- gives us full control over the appearance and behavior of the UI
- enables some features and behavior that just aren't possible using an IB-designed interface

Disadvantages

- requires a lot of code: for the view, for the constraints
- makes it impossible to preview the interface: the only way to see how the interface will look and act is by running the app

`IBDesignable` Views

There is a great compromise

- between a 100% IB-designed view
- and a 100% programmatic view

It's called an `IBDesignable` view

`IBDesignable` Views

Basic idea

- create a custom class that subclasses an existing `UIView`, such as `UIButton`
- put hooks in the class for Interface Builder for different UI styles

Then, in Interface Builder

- create an UI element
- set its Custom Class to the new class
- and the UI styles from the class show up in Interface Builder

Example

Simple, but very useful, example: `RoundedCornerView`

In code, in a file that I created called `RoundedCornerView.swift`:

```
@IBDesignable
class RoundedCornerView: UIButton {

}
```

here, I am subclassing `UIButton` to make my own, specialized `UIButton`

Example

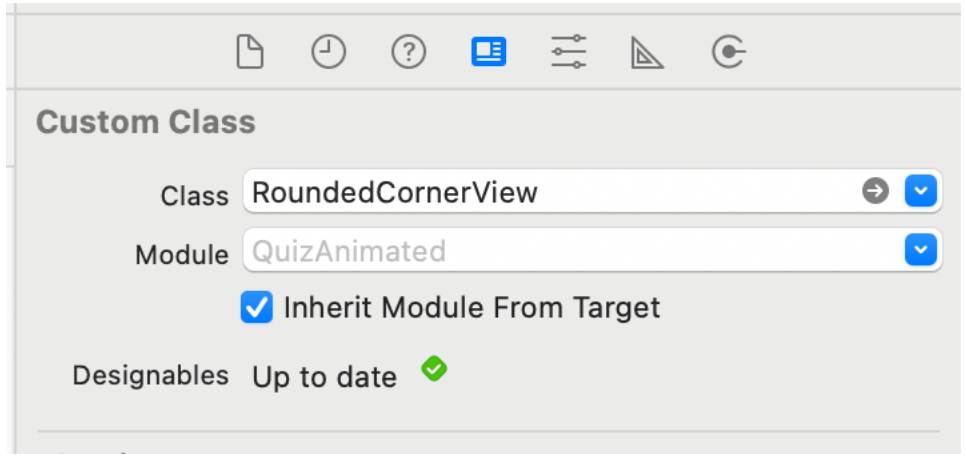
To tell IB about the customizable properties:

```
@IBDesignable
class RoundedCornerView: UIButton {

    @IBInspectable var cornerRadius: CGFloat = 0 {
        didSet {
            layer.cornerRadius = cornerRadius
            layer.masksToBounds = cornerRadius > 0
        }
    }

    @IBInspectable var borderWidth: CGFloat = 0 {
        didSet {
            layer.borderWidth = borderWidth
        }
    }

    @IBInspectable var borderColor: UIColor? {
        didSet {
            layer.borderColor = borderColor?.cgColor
        }
    }

    @IBInspectable override var backgroundColor: UIColor? {
        didSet {
            layer.backgroundColor = backgroundColor?.cgColor
        }
    }
}
```

Example

(1) Place a `UIButton` in Interface Builder

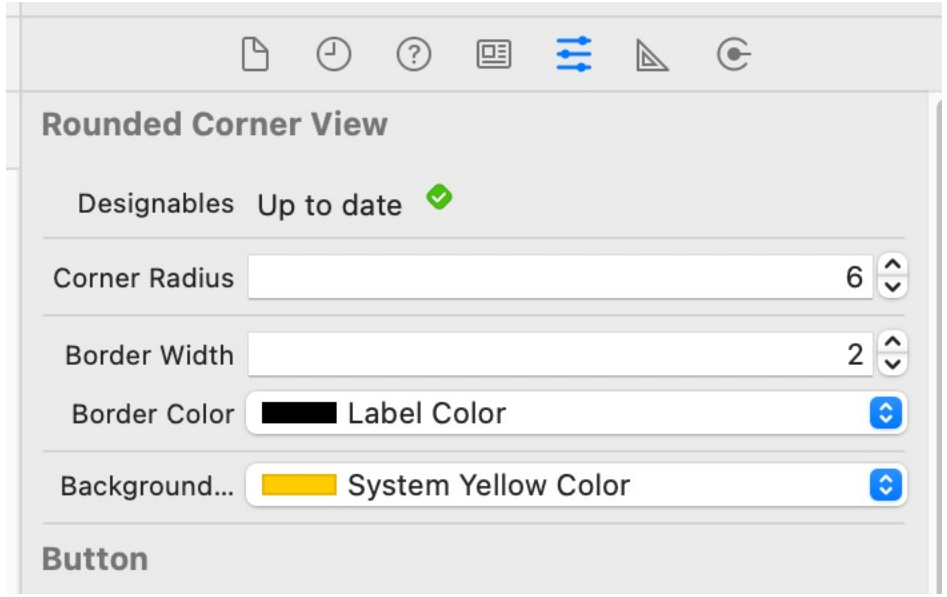
(2) And then set its custom class to the class you define



Example

And here's what IB shows:

- the properties that we have annotated with `@IBInspectable`



Have to build one time for IB to see the actual values and render the view correctly on the storyboard

Summary

Use the `IBDesignable` elements

- they can really help distinguish a boring, so-so interface from a handsome interface!