ackee edu

BI-IOS

Lukáš Hromadník, Jakub Olejník, Igor Rosocha

Lecture 4 - Navigation

Human Interface Guidelines

Offer invaluable information on how to design your app’s interface, navigate content, and manage interactions in your app

Reading and following these guidelines is essential

https://developer.apple.com/design/human-interface-guidelines/platforms/overview

Good resources to start with

https://developer.apple.com/ios/planning/

https://developer.apple.com/ipados/planning/

https://developer.apple.com/macos/planning/

https://developer.apple.com/tvos/planning/

https://developer.apple.com/watchos/planning/

Important shortcuts

Build: ⌘ + B

Run: ⌘ + R

Test: ⌘ + U

Stop: ⌘ + .

Clean (the build folder): ⌘ + ⇧ (+ ⌥) + K

Important shortcuts

Open Quickly: ⌘ + ⇧ + O

Show/Hide Navigator: ⌘ + 0

Show/Hide Utilities: ⌘ + ⌥ + 0

Show/Hide Debug Area: ⌘ + ⇧ + Y

Show/hide completions: ctrl + Space

Show/hide preview: ⌘ + ⌥ + Enter

List

A container that presents rows of data arranged in a single column

A scrollable list of data that user can interact with

Has some predefined styles and separators

Has native pull-to-refresh since iOS 15 🎉

Let’s code! 🤓

Property wrappers

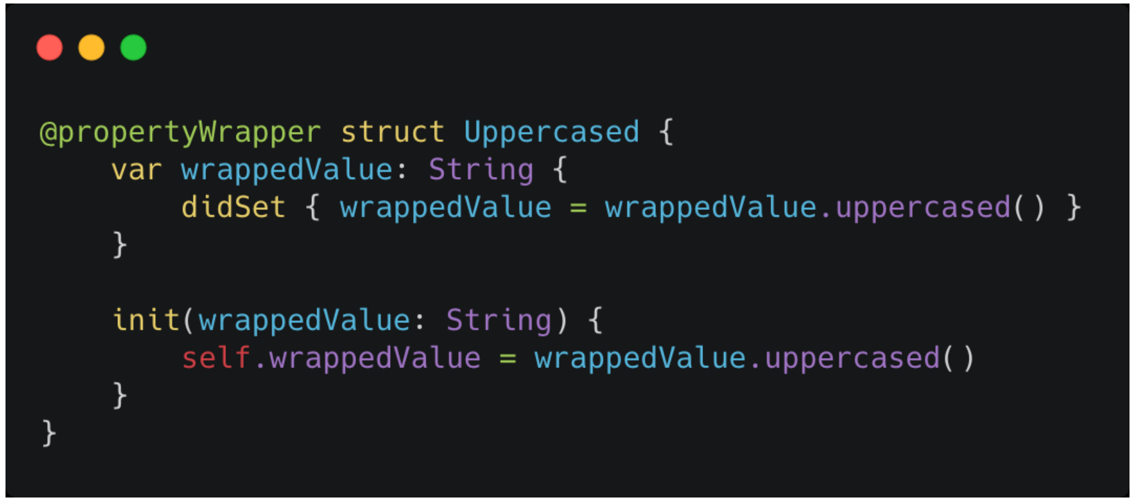
A type that wraps a given value in order to attach additional logic to it

Encapsulation of ”template” behavior applied to the vars they wrap

Can be implemented using struct (or a class)

Each property wrapper type should contain a stored property called `wrappedValue`

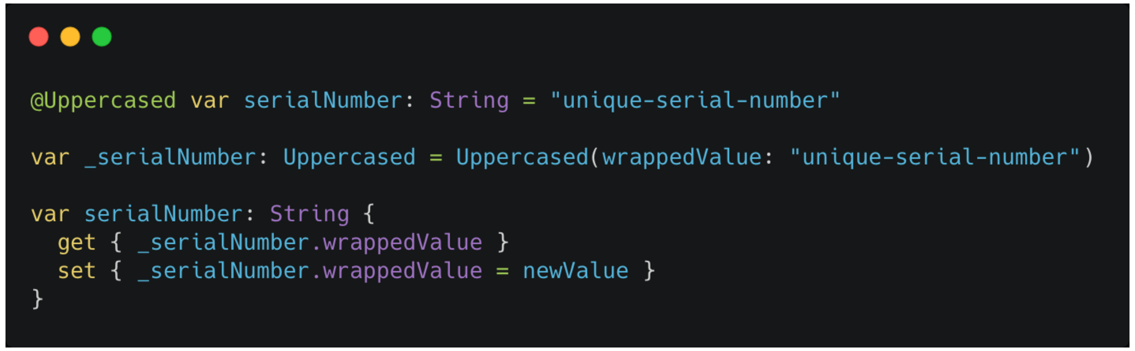
Property wrappers



```swift
@propertyWrapper struct Uppercased {
    var wrappedValue: String {
        didSet { wrappedValue = wrappedValue.uppercased() }
    }

    init(wrappedValue: String) {
        self.wrappedValue = wrappedValue.uppercased()
    }
}
```

Property wrappers

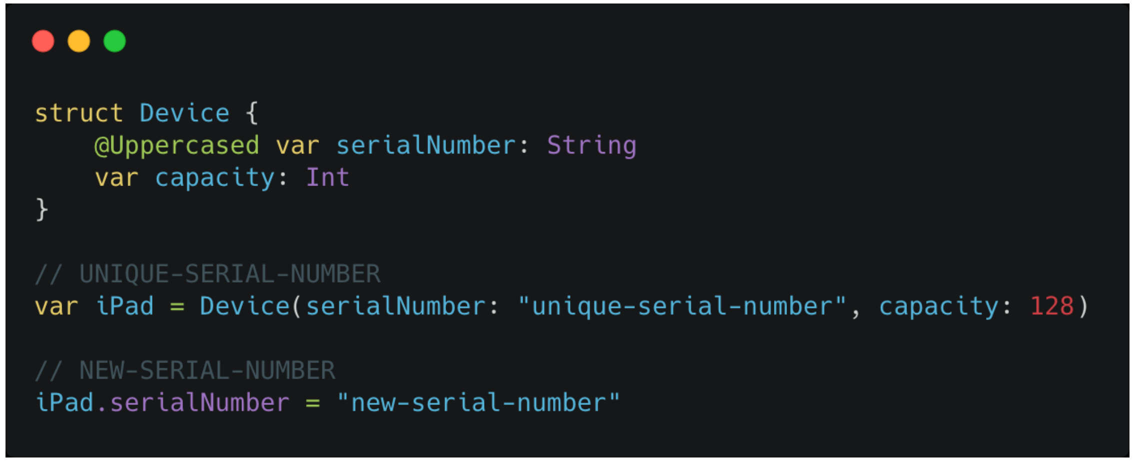


```swift
@Uppercased var serialNumber: String = "unique-serial-number"

var _serialNumber: Uppercased = Uppercased(wrappedValue: "unique-serial-number")

var serialNumber: String {
  get { _serialNumber.wrappedValue }
  set { _serialNumber.wrappedValue = newValue }
}
```

Property wrappers


```
struct Device {
    @Uppercased var serialNumber: String
    var capacity: Int
}

// UNIQUE-SERIAL-NUMBER
var iPad = Device(serialNumber: "unique-serial-number", capacity: 128)

// NEW-SERIAL-NUMBER
iPad.serialNumber = "new-serial-number"
```

@State

Allows us to modify values inside a struct, which would normally not be allowed because structs are value types

Effectively moves storage out from our struct and into shared storage managed by SwiftUI

Invalidates the View whenever wrappedValue changes

SwiftUI can destroy and recreate our struct whenever needed without losing the state

Alert

When you want the user to act in response to the state of the app

Nice, native look

Customizable since iOS 16

Sheet

Used to modally present new views over existing ones

Can be dragged down to dismiss

Dismiss can be handled in multiple ways

@Binding

A value that is bound to something else (another View)

Is able to get/set the value of the wrappedValue from some other source

Invalidates the View whenever bound-to value changes

@Environment

Specifically there to work with SwiftUI's own pre-defined keys

Great for reading out fixed properties that come from the system

Presentation mode, device dark/light mode, size class, etc.

fullScreenCover

Full screen modal presentation

Cannot be dragged down to dismiss

NavigationStack

Since iOS 16, replaced deprecated `NavigationView`

Can use `NavigationLink` to navigate through the app (push and pop screens)

Improved programmatic navigation using `NavigationPath`

TabView

Most user-intuitive approach to separate app logic/flow

Items have their own labels and images and lead to separate screens

Has maximum of 5 items

Questions?

Thank you very much,
see you next week!