Today

- **Last Lecture!**
  It’s all final project from here on in

- **Integrating with UIKit**
  Not every feature from UIKit was transported into SwiftUI (not yet anyway)
  So occasionally we want to put some UIKit UI into our SwiftUI
  Luckily there is a very straightforward compatibility API for doing this
UIKit Integration

Views are not as “elegant” in UIKit

No MVVM either, MVC instead
In MVC, views are grouped together and controlled by a Controller
This Controller is the granularity at which you present views on screen
In other words, UIKit’s .sheet, .popover and NavigationLink destination equivalents
don’t present a view, they present a controller (which in turn controls views)

Integration

So there must be two points of integration for SwiftUI to UIKit ...
UIViewRepresentable and a UIViewControllerRepresentable
They each turn a view or controller into a SwiftUI View
They are extremely similar
The main work involved is interfacing with the given view or controller’s API
(Setting vars, dealing with “callback functions”, etc.)
UIKit Integration

Delegation

UIKit is based on object-oriented technology
It heavily uses a concept called “delegation”
We mentioned this briefly when talking about FileManager
Objects (controllers and views) often delegate some of their functionality to other objects
They do this by having a var called delegate
That delegate var is constrained via a protocol with all the delegatable functionality
We’re not here to learn UIKit, so that’s all we’ll really say about it
But both demos today will integrate things that have delegates so you can see it in action
UIKit Integration

**Representables**

UIViewRepresentable and UIViewControllerRepresentable are SwiftUI Views

They have 5 main components ...

1. a function which creates the UIKit thing in question (view or controller)
   
   ```
   func makeUIView(Controller)(context: Context) -> view/controller
   ```

2. a function which updates the UIKit thing when appropriate (bindings change, etc.)
   
   ```
   func updateUIView(Controller)(view/controller, context: Context)
   ```

3. a Coordinator object which handles any delegate activity that goes on
   
   ```
   func makeCoordinator() -> Coordinator // Coordinator is a don't care for Representables
   ```

4. a Context (contains the Coordinator, your SwiftUI's environment, animation transaction)
   
   // passed into the methods above

5. a “tear down” phase if you need to clean up when the view or controller disappears
   
   ```
   func dismantleUIView(Controller)(view/controller, coordinator: Coordinator)
   ```
Demos

Choose Destination Airport from a Map
No Map in SwiftUI, so let’s use the one from UIKit
This is a demo of integrating a `UIView` into SwiftUI

Set our EmojiArt Background from the Camera
No Camera API in SwiftUI either, but there’s one in UIKit we can use
This is a demo of integrating a `UIViewController` into SwiftUI
Focus on the integration here, not the (pretty bad) “feature” we’re adding to EmojiArt
(the image we set as our EmojiArt background only works on the device that created it!)