CSC 491 / 391
Mobile Application Development for iOS II

Prof. Xiaoping Jia
School of Computing, CDM
DePaul University
xjia@cdm.depaul.edu

1

Outline

- More Swift UI
- Multiple screens
- List and navigation
- Bindings
- Bindable and environment objects

2

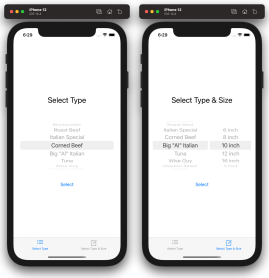
Swift UI

- A DSL for creating UI
- All textual, no storyboard
- Include many familiar features and components in UI Kit
- Built-in support in Xcode for previews and visual editing
- ***Different programming model*** from UI Kit
  - ***Data-driven*** in Swift UI
  - ***Event-driven*** in UI Kit

3

Tabs and Pickers – Order My Sub App

- A tabbed app with two tabs
- Single and double component pickers
- Alert popups



4

Order My Sub – State and Tabs

```
let subs = [ "Blockbuster", … ]
let sizes = [ "6 inch", … ]
struct ContentView: View {
    @State private var selectedTypeIndex = 0
    @State private var selectedSizeIndex = 0
    @State private var showTypeAlert = false
    @State private var showSizeAlert = false
    var body: some View {
        TabView {
            VStack {
                …
            }.tabItem { Label(…)
            }.alert(isPresented: $showTypeAlert) { Alert(…) }
            VStack {
                …
            }.tabItem { Label (…)
            }.alert(isPresented: $showSizeAlert) { Alert }
        } } }
```

5

Order My Sub – The First Tab

```
TabView {
    VStack {
        Text("Select Type").font(.title).padding()
        Picker("", selection: $selectedTypeIndex) {
            ForEach(0 ..< subs.count) { Text(subs[$0]) }
        }.padding()
        Button(action: { showTypeAlert = true }) { Text("Select") }
    }.tabItem {
        Label("Select Type", systemImage: "list.dash")
    }.alert(isPresented: $showTypeAlert) {
        Alert(title: Text("Order My Sub"),
        message: Text("You have selected \(subs[selectedTypeIndex])"),
          primaryButton: .default(Text("Confirm"), action: {}),
          secondaryButton: .destructive(Text("Cancel"), action: {}))
    }
    …
}
```


6

© Xiaoping Jia, 2015-2021

Order My Sub – The Second Tab

```
TabView {
    …
    VStack {
        Text("Select Type & Size").font(.title).padding()
        HStack {
            Picker("", selection: $selectedTypeIndex) {
                ForEach(0 ..< subs.count) { Text(subs[$0]) }
            } .frame(width: 200).clipped()
            Picker("", selection: $selectedSizeIndex) {
                ForEach(0 ..< sizes.count) { Text(sizes[$0]) }
            } .frame(width: 150).clipped()
        }.padding()
        Button(action: { showSizeAlert = true }) { Text("Select") }
    }.tabItem {
        Label("Select Type & Size", systemImage: "square.and.pencil")
    }.alert(isPresented: $showSizeAlert) {
        …
```


7

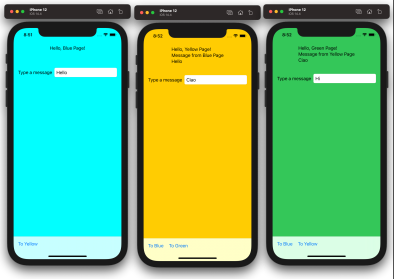
Order My Sub – The Second Tab

```
TabView {
    …
    VStack {
        Text("Select Type & Size").font(.title).padding()
        …
    }.tabItem {
        Label("Select Type & Size", systemImage: "square.and.pencil")
    }.alert(isPresented: $showSizeAlert) {
        Alert(title: Text("Order My Sub"),
            message: Text("You have selected
            \(sizes[selectedSizeIndex]) \(subs[selectedTypeIndex])"),
            primaryButton: .default(Text("Confirm"), action: {}),
            secondaryButton: .destructive(Text("Cancel"), action: {}))
    }
}
```


8

Multi-Screen – No Segues App

- A multi-screen app
- No segue in Swift UI
- Using state to show different screen



9

No Segues in Swift UI – Using State

```
struct ContentView: View {
    enum Page: String {
        case Blue = "Blue"
        case Yellow = "Yellow"
        case Green = "Green"
    }
    @State private var currentPage = Page.Blue
    var body: some View {
        if currentPage == .Blue {
            …  Draw Blue page
        } else if currentPage == .Yellow {
            …  Draw Yellow page
        } else {
            …  Draw Green page
        }
    }
}
```


10

Colors are Views

```
struct ContentView: View {
    @State private var currentPage = Page.Blue
    var pageColor : Color {
        switch currentPage{
        case .Blue:   return Color(UIColor.cyan)
        case .Yellow: return Color.yellow
        case .Green:  return Color.green } }
    var body: some View {
        ZStack {
            pageColor.ignoresSafeArea()
            if currentPage == .Blue { …
            } else if currentPage == .Yellow { …
            } else { …
            }
        }
    }
}
```


11

Draw Page Content

```
struct ContentView: View {
    …
    @State private var currentPage = Page.Blue
    @State private var message = ""
    @State private var messageToNextPage : String?
    var pageColor : Color { … }
    var body: some View {
        ZStack {
            pageColor.ignoresSafeArea()
            if currentPage == .Blue {
                VStack {
                    …
                }.toolbar(…)
            } else if currentPage == .Yellow { …
            } else { …
            }
        }
    }
```

12

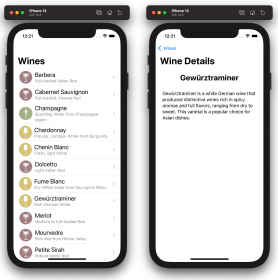
Draw Page Content – Blue Page

```
if currentPage == .Blue {
    VStack {
        Text("Hello, \(currentPage.rawValue) Page!\n
             \(messageToNextPage ?? "")").padding()
        HStack {
            Text("Type a message")
            TextField("Type a message here", text: $message,
                onCommit: { UIApplication.shared.endEditing() })
            .textFieldStyle(RoundedBorderTextFieldStyle())
        }.padding()
        Spacer()
    }.toolbar(content: {
        ToolbarItem(placement: .bottomBar) {
            Button(action: { messageToNextPage = "Message from
                \(currentPage.rawValue) Page\n\(message)"
                currentPage = .Yellow }) { Text("To Yellow") } } })
} else if currentPage == .Yellow { …
```


13

List and Navigation – Wine List App

- A list view and a detail view
- Define separate custom views
  - Each row in the list view
  - The detail view



14

Wine List – The Wine Class

```
class Wine : Identifiable {
    enum `Type`: String {
        case red = "red"
        case white = "white"
        case rosé = "rose"
        case sparkling = "sparkling"
    }
    var name: String
    var type: Type
    var shortDescription: String
    var longDescription: String
    var id: String { name }
    init(name: String, type: Type, shortDescription: String,
        longDescription: String) {
        self.name = name
        self.type = type
        self.shortDescription = shortDescription
        self.longDescription = longDescription
    }
}
```


15

Wine List – The Detail View

```
struct WineDetail: View {
    var wine: Wine

    var body: some View {
        VStack(alignment: .center) {
            Text(wine.name).font(.title).fontWeight(.bold)
            Spacer().frame(height: 30)
            Text(wine.longDescription)
            Spacer()
        }.padding()
         .navigationBarTitle("Wine Details")
    }
}
```


16

Wine List – A Single Row in the List

```
struct WineRow: View {
    var wine: Wine

    var body: some View {
        HStack {
            Image(wine.type.rawValue)
            VStack(alignment: .leading) {
                Text(wine.name).font(.title2)
                Text(wine.shortDescription).font(.subheadline)
                    .foregroundColor(.secondary)
            }
            Spacer()
        }
    }
}
```


17

Wine List – The List View

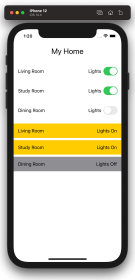
```
let wines = [ Wine(name: "Barbera", type: .red,
                   shortDescription: "Full-bodied Italian Red",
                   longDescription: "Barbera is a red wine …."), … ]

struct ContentView: View {
    var body: some View {
        NavigationView {
            List(wines) { wine in
                NavigationLink(destination: WineDetail(wine: wine)) {
                    WineRow(wine: wine)
                }
            }.navigationTitle("Wines")
        }
    }
}
```


18

© Xiaoping Jia, 2015-2021

My Home – Binding

- An app simulating controlling the lights in various rooms of your house
- Several variations



19

State vs. Binding

- Limitation of @State properties
  - Limited to a single custom view container
  - Should be declared as private
  - Not accessible outside the custom view in which they are declared
- If it is necessary to accessible a state outsize the custom view in which it is declared, use @Binding
- Each @State is a source of truth
- Each @Binding is derived from a @State or another @Binding
  - Data dependency without ownership
  - The truth is owned by another view

20

My Home – Using Binding

- @State – the single source of truth

```
struct Room {
    var name : String
    var lightsOn = false
}

struct ContentView: View {
    @State private var livingRoom = Room(name: "Living Room")
    @State private var studyRoom = Room(name: "Study Room")
    @State private var diningRoom = Room(name: "Dining Room")
    var body: some View {
        Text("My Home").font(.title).padding()
        …
    }
}
```


21

My Home – Using Binding

- @Binding must be derived from a @State or another @Binding

```
struct RoomView: View {
    @Binding var room : Room
    var body: some View {
        HStack {
            Text(room.name)
            Spacer()
            Text(room.lightsOn ? "Lights On" : "Lights Off")
        }.padding()
         .background(room.lightsOn ? Color.yellow : Color.gray)
    }
}
```


22

My Home – Using Binding

- Another @Binding

```
struct RoomControlView: View {
    @Binding var room : Room
    var body: some View {
        HStack {
            Text(room.name)
            Spacer()
            Text("Lights")
            Toggle("Lights", isOn: $room.lightsOn)
                .labelsHidden()
        }.padding()
    }
}
```


23

My Home – Using Binding

```
struct ContentView: View {
    @State private var livingRoom = Room(name: "Living Room")
    @State private var studyRoom = Room(name: "Study Room")
    @State private var diningRoom = Room(name: "Dining Room")
    var body: some View {
        Text("My Home").font(.title).padding()
        Group {
            RoomControlView(room: $livingRoom)
            RoomControlView(room: $studyRoom)
            RoomControlView(room: $diningRoom)
        }
        Divider()
        Group {
            RoomView(room: $livingRoom)
            RoomView(room: $studyRoom)
            RoomView(room: $diningRoom)
        }
        Spacer()
    }
}
```

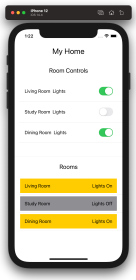
Connect @Binding to @State

Connect @Binding to @State

24

My Home – Using Observed Object

- A variation of My Home
- The truth lies outside the view, in the data model
- The data objects are not managed by Swift UI



25

My Home – Using Observed Object

- Data classes and objects not managed by Swift UI

```
class Room : Identifiable, ObservableObject {
    init(_ name: String) {
        self.name = name
    }
    let id: UUID = .init()
    var name : String
    @Published var lightsOn = false
}

var rooms = [
    Room("Living Room"),
    Room("Study Room"),
    Room("Dining Room")
]
```

Only this will be part of the UI

26

My Home – Using Observed Object

- Observe an object external to the view

```
struct RoomControlView: View {

    @ObservedObject var room : Room

    var body: some View {
        HStack {
            Text(room.name)
            Spacer()
            Toggle("Lights", isOn: $room.lightsOn)
        }.padding()
    }

}
```


27

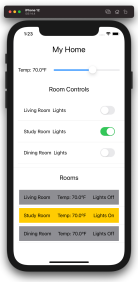
My Home – Observed Object

```
struct ContentView: View {
    var body: some View {
        Text("My Home").font(.title).padding()
        Group {
            Text("Room Controls").font(.title2).padding()
            List(rooms) { room in
                RoomControlView(room: room)
            }
        }
        Divider()
        Group {
            Text("Rooms").font(.title2).padding()
            List(rooms) { room in
                RoomView(room: room)
            }
        }
        Spacer()
    }
}
```

28

My Home – Using Environment Object

- Another variation of My Home
- Using environment objects
  - App-wide global objects
  - Observed by multiple views



29

Environment Objects

- The *Environment*
  - A collection of objects created by Swift UI automatically
  - Accessible to all views managed by Swift UI
- Swift UI uses the Environment to pass system-wide settings
  - Use system defined keys to access the system settings
- Environment objects
  - User defined objects stored in the Environment
  - Use class name as the key

30

© Xiaoping Jia, 2015-2021

My Home – Using Environment Object

```
class Room : Identifiable, ObservableObject {
    init(_ name: String) {
        self.name = name
    }
    let id: UUID = .init()
    var name : String
    @Published var lightsOn = false
}
class House : ObservableObject {
    init(_ name: String, rooms: [Room]) {
        self.name = name
        self.rooms = rooms
    }
    var name : String
    @Published var rooms : [Room]
    @Published var temp : Double = 70
}
```

31

31

My Home – Using Environment Object

- Declare app-wide objects for environment objects
- One environment object per class
- Register environment object at the top content view

```
let myHome = House("My Home",
                   rooms: [ Room("Living Room"),
                            Room("Study Room"),
                            Room("Dining Room") ])

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView().environmentObject(myHome)
    }
}
```

DePaul University 32

32

My Home – Using Environment Object

- Register environment objects at the top content view in the App delegate

```
@main
struct My_Home___Env___SUApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView().environmentObject(myHome)
        }
    }
}
```

DePaul University 33

33

My Home – Using Environment Object

```
struct RoomView: View {
    @EnvironmentObject var house : House
    @ObservedObject var room : Room

    var body: some View {
        HStack {
            Text(room.name)
            Spacer()
            Text(String(format: "Temp: %.1f°F", house.temp))
            Spacer()
            Text(room.lightsOn ? "Lights On" : "Lights Off")
        }.padding()
         .background(room.lightsOn ? Color.yellow : Color.gray)
    }

}
```

DePaul University 34

34

My Home – Using Environment Object

```
struct RoomControlView: View {
    @EnvironmentObject var house : House
    @ObservedObject var room : Room

    var body: some View {
        HStack {
            Text(room.name)
            Spacer()
            Toggle("Lights", isOn: $room.lightsOn)
        }.padding()
    }

}
```

DePaul University 35

35

My Home – Using Environment Object

```
struct ContentView: View {
    @EnvironmentObject var house : House
    var body: some View {
        VStack {
            Text(house.name).font(.title).padding()
            HStack {
                Text(String(format: "Temp: %.1f°F", house.temp))
                Slider(value: $house.temp, in: 40...90)
                    .padding(.leading)
            }.padding()
            Spacer()
            Group {
                Text("Room Controls").font(.title2).padding()
                List(house.rooms) { room in
                    RoomControlView(room: room)
                }
            }
            Divider()
```

36

36

My Home – Using Environment Object

```
struct ContentView: View {
    @EnvironmentObject var house : House
    var body: some View {
        VStack {
            Text(house.name).font(.title).padding()
            …
            Divider()
            Group {
                Text("Rooms").font(.title2).padding()
                List(house.rooms) { room in
                    RoomView(room: room)
                }
            }
        }
        Spacer()
    }
}
```

DEPAUL UNIVERSITY 37

37

Swift UI vs UI Kit

Swift UI

- Data driven
- Declarative
- More succinct
- Supports a large subset of UI features
- Supports multiple Apple platforms
- Still evolving

UI Kit

- Event driven
- Imperative
- More verbose
- Full featured
- iOS only
- Stable and extensible

DEPAUL UNIVERSITY 38

38