CSC 491 / 391
Mobile Application Development for iOS II

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

1

Outline

- Introducing Swift UI
- Developing apps with Swift UI

2

Introduction to Swift UI

3

What is Swift UI?

- A Swift base DSL (Domain Specific Language) for building user interfaces
- Introduced in 2019, available in iOS 13
- Not supported by any earlier version of iOS
- Purposes
  - A simpler approach to building UI
  - Cross-platform support
    - Natively supported on all Apple platforms
    - iOS, iPadOS, watchOS, tvOS, macOS

4

Swift UI – Benefits

- Declarative syntax
  - Developer declares views in a hierarchy that mirrors the desired layout of your interface.
  - Swift UI manages drawing and updating these views in response to events like user input or state changes.
- Less code, all in Swift
- Alternative to using UI Kit
  - No Storyboard
  - No Interface Builder
- Integrated support in Xcode

5

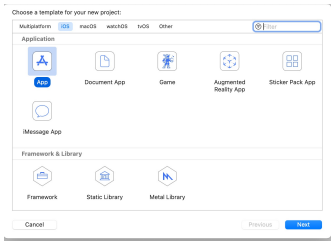
Swift UI - Limitations

- Requires iOS 13 or later
  - No support for earlier versions
- Not full feature set
  - Only a subset of widgets, compared with UIKit
  - Not all features are support in Swift UI
- Limited support
- Potential breaking changes

6

Hello, Swift UI! – A First Example

- New Project | App

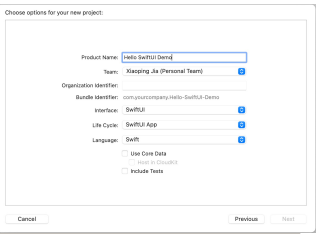


DEPAUL UNIVERSITY 7

7

Hello, Swift UI! – A First Example

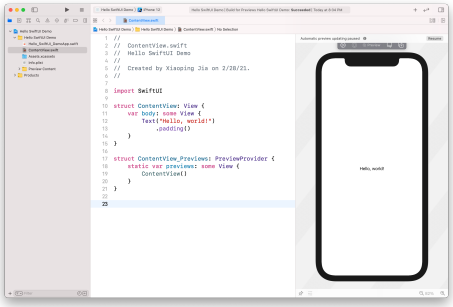
- Interface: SwiftUI
- Life Cycle: SwiftUI App
- Language: Swift



DEPAUL UNIVERSITY 8

8

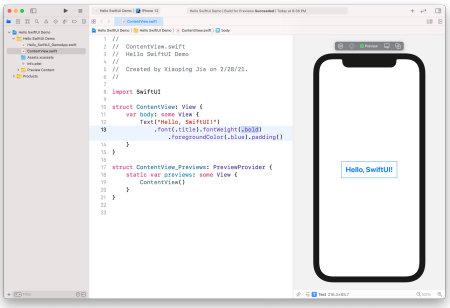
Hello, Swift UI! – A First Example



DEPAUL UNIVERSITY 9

9

Hello, Swift UI! – A First Example

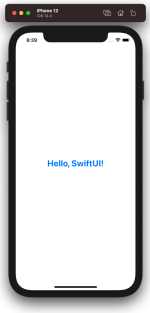


DEPAUL UNIVERSITY 10

10

Hello, Swift UI! – A First Example

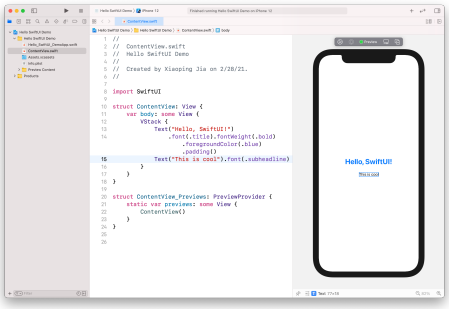
- Run in the simulator



DEPAUL UNIVERSITY 11

11

Hello, Swift UI! – A First Example

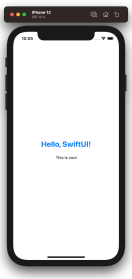


DEPAUL UNIVERSITY 12

12

Hello, Swift UI! – A First Example

- Run in the simulator



DePaul University 13

13

Hello, Swift UI! – A First Example

```
import SwiftUI
struct ContentView: View {
    var body: some View {
        VStack {
            Text("Hello, SwiftUI!")
                .font(.title).fontWeight(.bold)
                .foregroundColor(.blue).padding()
            Text("This is cool").font(.subheadline)
        }
    }
}
struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

14

14

Hello, Swift UI! – A First Example

```
import SwiftUI
struct ContentView: View {
    var body: some View {
        VStack {
            Text("Hello, SwiftUI!")
                .font(.title).fontWeight(.bold)
                .foregroundColor(.blue).padding()
            Text("This is cool").font(.subheadline)
        }
    }
}
struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

What are *ContentView* and *View*?
What is *some View*?
What is this block?
What are *VStack* and *Text*?
What is this syntax?
What is *Preview*?

15

15

Views and Controls

- A view defines a piece of UI
  - Visual building blocks of your app's user interface
  - Presented on screen
  - Text, images, etc.
- Controls – views that enable user interactions
  - Buttons, sliders, etc.
- Primitive views – the basic building blocks
- Containers – building view hierarchies through composition
  - Define visual relationship of the components

DePaul University 16

16

View Protocol and Opaque Type

- View – a protocol
  - Requires a computed property – body
- Primitive views and controls
  - Structs that conform to the View protocol

```
struct ContentView: View {
    var body: some View { … }
}
```

What is *some View*?

- An opaque type: `some T`
  - A subtype of `T`
  - A function with multiple returns must return the same subtype of `T`

DePaul University 17

17

View Builders

- View Builders – define view containers
  - Structs that conform to the View protocol for building composite views from child-views
  - Provide a view building function in which each statement produce a child-view

```
var body: some View {
    VStack {
        Text("Hello, SwiftUI!")
        Text("This is cool")
    }
}
```

- Relying on a new feature in Swift 5.3
  - Result builder

DePaul University 18

18

Using View Builders to Define Custom Views

- View Container syntax

```
container {
    content
    content
    •••
}
```

- The `container` is a View Builder
- Each `content` is a child view
- Example

```
VStack {
    Image(...)
    Image(...)
    Text(...)
}
```

- `VStack`
  - A View Builder
- `Image` `Text`
  - Primitive Views

DePaul University 19

19

View Modifiers

- View modifiers customize the display and behavior of the views
- View modifiers are methods of a View
  - Returns a View
  - The altered view takes the place of the original view in the view hierarchy
  - Can be chained
- Examples:

```
Text("Hello!")
Text("Hello!").font(.title)
Text("Hello!").font(.title).fontWeight(.bold)
Text("Hello!").font(.title).fontWeight(.bold)
        .foregroundColor(.blue)
```

DePaul University 20

20

Swift UI Preview in Xcode

- Preview Provider – A protocol
  - Generates dynamic preview of custom views defined in Swift UI.

DePaul University 21

21

Developing Apps with Swift UI

22

22

Building Swift UI Apps

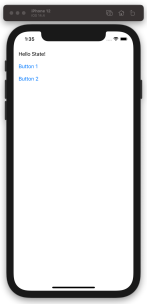
- Compose custom views out of
  - primitive views provided by Swift UI
  - other custom defined composite views
- Configure these views with view modifiers and connect them to your data model.
- Place your custom views in am app's view hierarchy.

DePaul University 23

23

Swift UI Example – Buttons

- A simple app
  - A Label
  - Two Buttons
- Action for Buttons
- A few variations with identical look and behavior



DePaul University 24

24

Buttons – Version 1

```
struct ContentView: View {
    @State var message = "Hello State!"
    var body: some View {
        VStack(alignment: .leading, spacing: 20.0) {
            Text(message)
                .frame(maxWidth: .infinity, alignment: .leading)
            Button(action: { message = "Button 1 pressed" }) {
                Text("Button 1")
            }
            Button(action: { message = "Button 2 pressed" }) {
                Text("Button 2")
            }
            Spacer()
        }
        .padding([.top, .leading, .trailing])
    }
}
```

DEPAUL UNIVERSITY 25

25

State Properties

- `@State` – a property wrapper
  - Indicate a property of a View is a *state*
- Property wrapper
  - Allows actions to be performed whenever a property is accessed or modified.
- A state property defines data dependencies within a custom view container
  - Child views can be dependent on state properties
  - Changes to state properties are propagated to dependent child views
  - Dependencies are managed by the system

DEPAUL UNIVERSITY 26

26

Buttons – Version 1 State Property

```
struct ContentView: View {
    @State var message = "Hello State!"
    var body: some View {
        VStack(alignment: .leading, spacing: 20.0) {
            Text(message)
                .frame(maxWidth: .infinity, alignment: .leading)
            Button(action: { message = "Button 1 pressed" }) {
                Text("Button 1")
            }
            Button(action: { message = "Button 2 pressed" }) {
                Text("Button 2")
            }
            Spacer()
        }
        .padding([.top, .leading, .trailing])
    }
}
```

State property

Dependent on a state property

Update a state property

DEPAUL UNIVERSITY 27

27

Data Dependency Via State Properties

- When a state property of a view is updated
  - All dependent child-views are updated automatically
- Managed by Swift UI runtime
  - For every state property, watch for change
  - When a change happens, notify the container view that owns the state property
  - Notify all child views in the container that depend on the state property
- E.g., in Buttons
  1. Button action: update *message*, a state property
  2. Container: propagate changes to dependent child-views
  3. Text: a dependent child view, update the text label

DEPAUL UNIVERSITY 28

28

Buttons – Version 2

```
struct ContentView: View {
    let labels = [ "Button 1", "Button 2" ]
    @State var message = "Hello State!"
    var body: some View {
        VStack(alignment: .leading, spacing: 20.0) {
            Text(message)
                .frame(maxWidth: .infinity, alignment: .leading)
            ForEach (labels, id: \.self) { label in
                Button(action:  { message = "\(label) pressed" }) {
                    Text(label)
                }
            }
            Spacer()
        }
        .padding([.top, .leading, .trailing])
    }
}
```

DEPAUL UNIVERSITY 29

29

ForEach in View Builders

- A special function
  - Iterates over a collection
  - Each iteration provides a child-view to the view container
- The collection of data must either
  - Conform to the Identifiable protocol, or
  - Provide an id parameter using a *key path*
    - `\.self` refers to each element itself

DEPAUL UNIVERSITY 30

30

Buttons – Version 3

```
struct ContentView: View {
    let labels = [ "Button 1", "Button 2" ]
    @State private var buttonIndex = -1
    var body: some View {
        VStack(alignment: .leading, spacing: 20.0) {
            Text((buttonIndex < 0) ? "Hello State!"
                                   : "\(labels[buttonIndex]) pressed")
                .frame(maxWidth: .infinity, alignment: .leading)
            ForEach (0..<2) { i in
                Button(action: { buttonIndex = i }) {
                    Text(labels[i])
                }
            }
            Spacer()
        }
        .padding([.top, .leading, .trailing])
    }
}
```

31

Swift UI Example – Images

- An app with three buttons
- Each button selects a corresponding image to display
- The images are in the Asset Catalog with their names matching the titles of the buttons

DePaul University 32

32

Images in Swift UI

```
let names = [ "Lake Front", "Cloud Gate", "Picasso"]
struct ContentView: View {
    @State var imageName = names[0]
    var body: some View {
        VStack {
            HStack {
                Button(action: { self.imageName = names[0] }) {
                    Text(names[0]) }
                Spacer()
                Button(action: { self.imageName = names[1] }) {
                    Text(names[1]) }
                Spacer()
                Button(action: { self.imageName = names[2] }) {
                    Text(names[2]) }
            }.padding(.all)
            Spacer()
            Image(imageName)
            Spacer()
} } }
```

33

33

Images in Swift UI – Version 2

```
let names = [ "Lake Front", "Cloud Gate", "Picasso"]
struct ContentView: View {
    @State var imageName = names[0]
    var body: some View {
        VStack {
            HStack {
                ForEach (0 ..< names.count) { i in
                    if (i > 0) { Spacer() }
                    Button(action: {
                        self.imageName = names[i]
                    }) {
                        Text(names[i])
                    }
                }
            }.padding(.all)
            Spacer()
            Image(imageName)
            Spacer()
} } }
```

34

34

Swift UI Example – Controls

- An app showing various controls
  - Pickers
  - Switches, a.k.a. toggles
  - Sliders
  - Steppers

DePaul University 35

35

Controls – Pickers

```
let names = [ "Botticelli", "Raffaello", "Uccello" ]
struct ContentView: View {
    @State private var selectedNameIndex = 0
    …
    var body: some View {
        VStack {
            Picker("Names", selection: $selectedNameIndex) {
                ForEach(0 ..< names.count) { i in
                    Text(names[i]).tag(i)
                }
            }.pickerStyle(SegmentedPickerStyle())
            Text(names[selectedNameIndex]).frame(maxWidth: .infinity)
            …
            Image(names[selectedNameIndex])
                .resizable()
                .aspectRatio(contentMode: .fit)
        }.padding(.all).frame(alignment: .leading)
} }
```

36

Controls – Pickers

```
struct ContentView: View {
    @State private var hiddenIndex = 1
    var hidden : Bool { hiddenIndex == 0 }
    @State var disableIndex = 1    …
    var body: some View {
        VStack {
            …
            Picker("hidden", selection: $hiddenIndex) {
                Text("Hide controls").tag(0)
                Text("Show controls").tag(1)
            }.pickerStyle(SegmentedPickerStyle())
            Picker("disable", selection: $disableIndex) {
                Text("Disbale controls").tag(0)
                Text("Enable controls").tag(1)
            }.pickerStyle(SegmentedPickerStyle())
            …
        }.padding(.all).frame(alignment: .leading)
} }
```

37

Controls – Toggles

```
struct ContentView: View {
    @State private var switchOn = false
    …
    var body: some View {
        let switch1 =  Toggle("Switch", isOn: $switchOn)
            .labelsHidden().disabled(disableIndex == 0)
        let switch2 =  Toggle("Switch", isOn: $switchOn)
            .labelsHidden().disabled(disableIndex == 0)
        VStack {
            …
            HStack {
                Text("Switch \(switchOn ? "on" : "off")").frame(…)
                if (!hidden) { switch1 }
                Spacer()
                if (!hidden) { switch2 }
            }
            …
```

DEPAUL UNIVERSITY 38

38

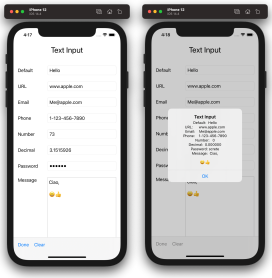
Controls – Slider and Stepper

```
struct ContentView: View {
    @State private var sliderValue = 0.0
    @State private var stepperValue = 0
    …
    var body: some View {
        let slider = Slider(value: $sliderValue, in: 0...100)
            .disabled(disableIndex == 0)
        VStack {
            …
            HStack {
                Text("Value \(Int(sliderValue))")
                    .frame(…)
                if (!hidden) { slider } else { Spacer() }
            }
            HStack {
                Stepper(value: $stepperValue, in: 0...100, label: {
                    Text("Value \(stepperValue)")
                })
```

39

Swift UI Example – Text Input

- An app with various text input fields
  - Default
  - URL
  - Email
  - Numbers
  - Password
- Display an alert popup



DEPAUL UNIVERSITY 40

40

Text Input – State Properties

```
struct ContentView: View {
    @State private var text = ""
    @State private var url = ""
    @State private var email = ""
    @State private var phone = ""
    @State private var number : Int = 0
    @State private var decimal : Double = 0
    @State private var password = ""
    @State private var message = ""
    @State private var isAlertPresent = false
    var body: some View {
        VStack {
            Text("Text Input").font(.title).padding()
            …
        }
    }
}
```

DEPAUL UNIVERSITY 41

41

Text Input – The Basic Text Fields

```
VStack {
    Text("Text Input").font(.title).padding()
    HStack {
        Text("Default").frame(width:100, alignment: .leading)
        TextField("Type text here", text: $text, onCommit:  {
                UIApplication.shared.endEditing() })
            .textFieldStyle(RoundedBorderTextFieldStyle())
    }.padding([.top, .leading, .trailing])
    HStack {
        Text("URL"). frame(width:100, alignment: .leading)
        TextField("Enter a URL", text: $url, onCommit:  {
                UIApplication.shared.endEditing()
        }).keyboardType(.URL)
          .textFieldStyle(RoundedBorderTextFieldStyle())
    }.padding([.top, .leading, .trailing])
    …
```

DEPAUL UNIVERSITY 42

42

Text Input – Extend UI Application

```
import SwiftUI
@main
struct Text_Input___SUApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView()
        }
    }
}

extension UIApplication {
    func endEditing() {
        sendAction(#selector(UIResponder.resignFirstResponder),
                   to: nil, from: nil, for: nil)
    }
}
```

DEPAUL UNIVERSITY 43

43

Text Input – Email and Phone

```
VStack { …
    HStack {
        Text("Email").frame(width:100, alignment: .leading)
        TextField("Enter an email address", text: $email, onCommit:  {
                UIApplication.shared.endEditing()
        }).keyboardType(.emailAddress)
          .textFieldStyle(RoundedBorderTextFieldStyle())
    }.padding([.top, .leading, .trailing])
    HStack {
        Text("Phone").frame(width:100, alignment: .leading)
        TextField("Enter a phone number", text: $phone)
            .keyboardType(.phonePad)
            .textFieldStyle(RoundedBorderTextFieldStyle())
    }.padding([.top, .leading, .trailing])
```

DEPAUL UNIVERSITY 44

44

Text Input – Numbers

```
VStack { …
    HStack {
        Text("Number").frame(width:100, alignment: .leading)
        TextField("Enter a number", value: $number,
                  formatter: NumberFormatter())
            .keyboardType(.numberPad)
            .textFieldStyle(RoundedBorderTextFieldStyle())
    }.padding([.top, .leading, .trailing])
    HStack {
        Text("Decimal").frame(width:100, alignment: .leading)
        TextField("Enter a decimal", value: $decimal,
                  formatter: NumberFormatter())
            .keyboardType(.decimalPad)
            .textFieldStyle(RoundedBorderTextFieldStyle())
    }.padding([.top, .leading, .trailing])
    …
```

DEPAUL UNIVERSITY 45

45

Text Input – Password and Multi-Line Editor

```
VStack {
    …
    HStack {
        Text("Password").frame(width:100, alignment: .leading)
        SecureField("Enter a pssword", text: $password)
            .textFieldStyle(RoundedBorderTextFieldStyle())
    }.padding([.top, .leading, .trailing])
    HStack(alignment: .top) {
         Text("Message").frame(width:100, alignment: .leading)
         TextEditor(text: $message).border(Color(UIColor.separator))
    }.frame(alignment: .top).padding([.top, .leading, .trailing])
    Spacer()
}
```

DEPAUL UNIVERSITY 46

46

Text Input – Toolbar

```
struct ContentView: View { …
   var body: some View {
       VStack {
           …
       }.onTapGesture {
           UIApplication.shared.endEditing()
       }.toolbar(content: {
            ToolbarItemGroup(placement: .bottomBar) {
                Button(action: { isAlertPresent = true } ) {
                    Text("Done") }
                Button(action: {
                    text = ""
                    …
                }) {
                   Text("Clear")
                }
            }
       })
```

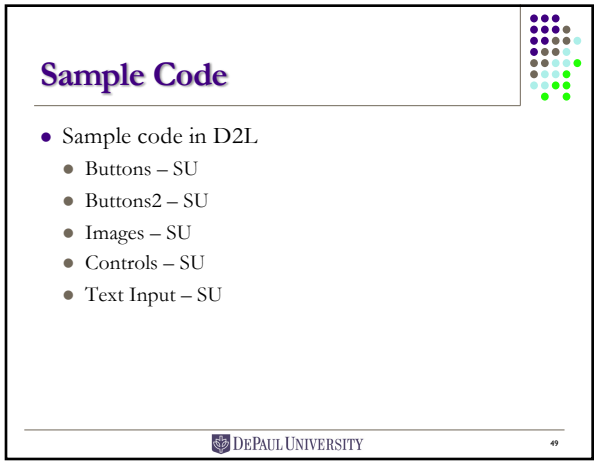
47

47

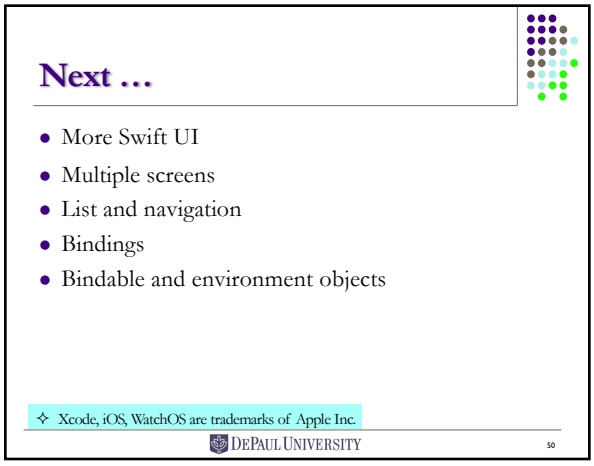
Text Input – Alert Popup

```
struct ContentView: View { …
    var body: some View {
        VStack {
        }.onTapGesture { …
        }.toolbar(…).alert(isPresented: $isAlertPresent) {
            Alert(title: Text("Text Input"),
                  message: Text("""
                    Default:  \(text)
                    URL:      \(url)
                    Email:    \(email)
                    Phone:    \(phone)
                    Number:   \(number)
                    Decimal:  \(decimal)
                    Password: \(password)
                    Message:  \(message)
                    """),
                  dismissButton: nil)
} } }
```

48


Sample Code
Sample code in D2L
Buttons – SU
Buttons2 – SU
Images – SU
Controls – SU
Text Input – SU
DePaul University
49

49


Next …
More Swift UI
Multiple screens
List and navigation
Bindings
Bindable and environment objects
Xcode, iOS, WatchOS are trademarks of Apple Inc.
DePaul University
50

50