Data Flow Through SwiftUI

WWDC 2019

2023. 2. 25. Stat

Principles of Data Flow

Data Access as a Dependency

- 데이터가 변경될 때마다 뷰가 업데이트 되어야 하므로 뷰는 데이터에 의존적이다.
- 의존 관계에 따라 뷰가 업데이트 되는 과정을 직접 처리하는 대신 SwiftUI를 통해서 선언적으로 처리할 수 있도록 여러가지 도구 들을 제공한다.

Principles of Data Flow

Source of Truth

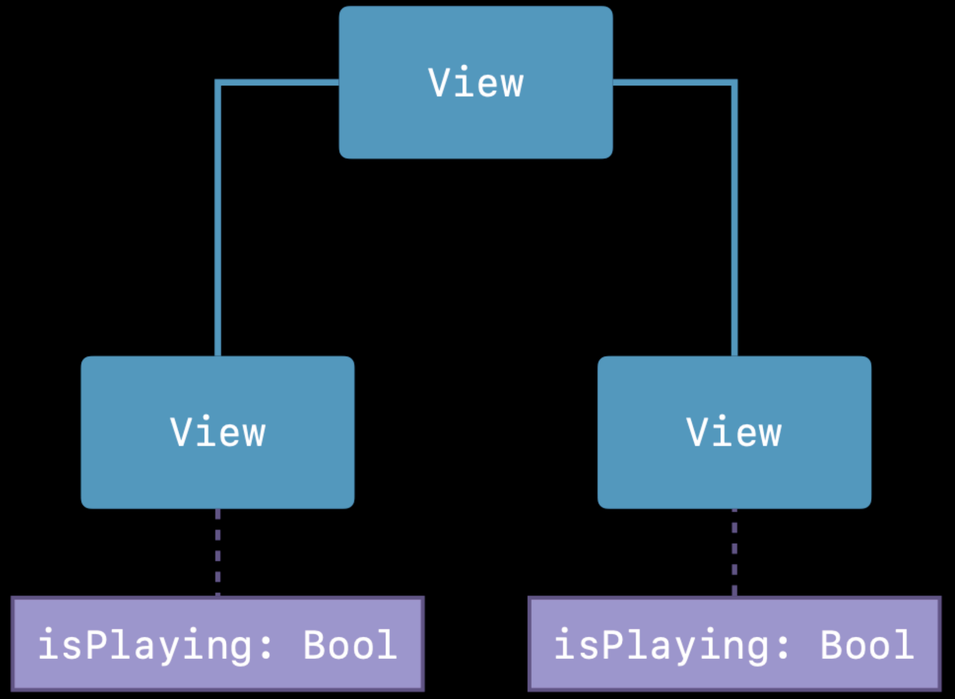
- 뷰 계층에서 각각의 뷰 컴포넌트들이 의존하는 데이터는 단일해야 한다.

Principles of Data Flow

Duplicated Source of Truth ❌

- 각각의 뷰 계층에서 복사된 데이터를 사용

  → 동기화를 직접 해주어야 하므로 버그가 발생할 수 있음

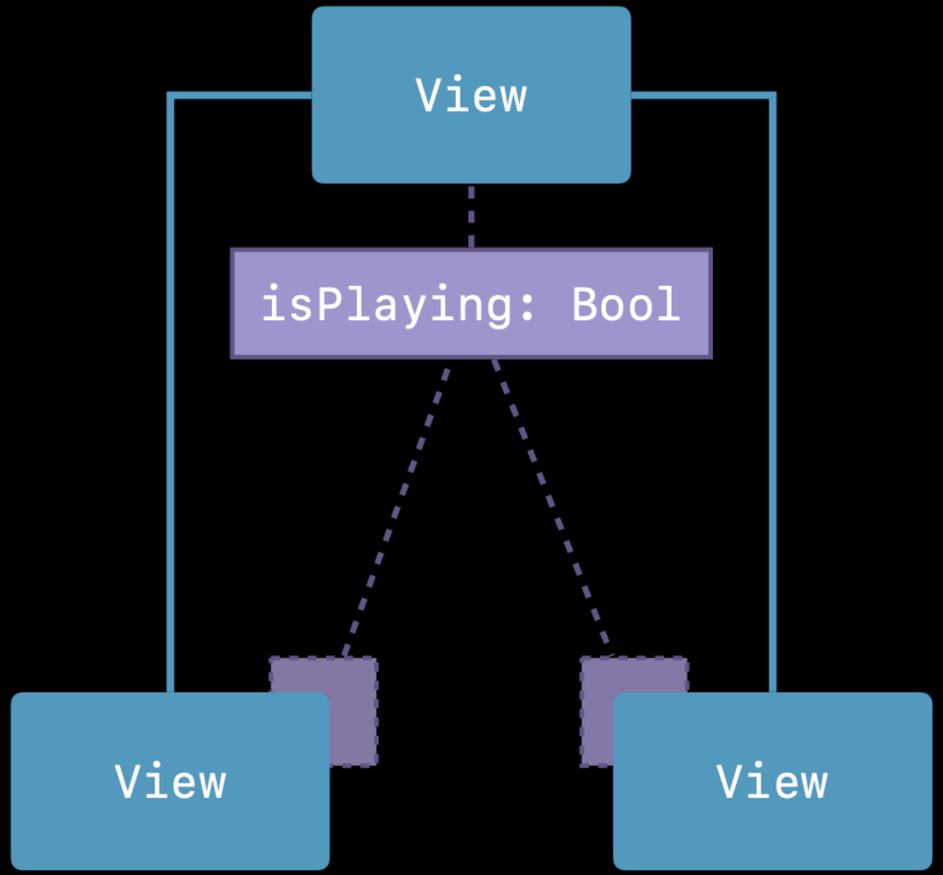


Principles of Data Flow

Single Source of Truth ✅

- 뷰 계층에서 단일한 데이터를 공유

  → 뷰와 데이터가 불일치 하는 버그 예방할 수 있음



9:41

시험지+정답을 올려주세요

관다 관다 관다 관다 관다 고등학교 2021년 중간고사 2학기 수학 두줄 이면....

STEP 1

시험지 촬영하기

짤리지 않고 순서대로 촬영해 주세요.

촬영하기

STEP 2

정답 촬영하기

시험지의 정확한 정답을 촬영해 주세요

촬영하기

2학년 수학 정답지

(2020 1학기 기말고사)

문항	정답	문항	정답
1	③	11	④
2	④	12	⑤
3	③	13	②
4	④	14	③
5	⑤	15	⑤
6	⑤	16	②
7	④	17	①, ④
8	③	18	④
9	②	19	①
10	④	20	④

제출하기

```swift
struct ContentView: View {

    var isCaptured: Bool = false

    var body: some View {
        HStack {
            VStack(alignment: .leading) {
                Text("STEP1")
                Spacer().frame(height: 4)
                Text("시험지 촬영하기")
                Spacer().frame(height: 12)
                Text("짤리지 않게 순서대로 촬영해주세요.")
                Spacer()
                Button(action: {
                    self.isCaptured.toggle()
                }, label: {
                    Text(isCaptured ? "수정하기" : "촬영하기")
                })
                .buttonStyle(.borderedProminent)
                .tint(isCaptured ? .black : .orange)
            }
            Image("sample")
        }
        .padding(16)
        .frame(width: 343, height: 232)
    }
}
```

```swift
struct ContentView: View {

    var isCaptured: Bool = false

    var body: some View {
        HStack {
            VStack(alignment: .leading) {
                Text("STEP1")
                Spacer().frame(height: 4)
                Text("시험지 촬영하기")
                Spacer().frame(height: 12)
                Text("짤리지 않게 순서대로 촬영해주세요.")
                Spacer()
                Button(action: {
                    self.isCaptured.toggle()
                }, label: {
                    Text(isCaptured ? "수정하기" : "촬영하기")
                })
                .buttonStyle(.borderedProminent)
                .tint(isCaptured ? .black : .orange)
            }
            Image("sample")
        }
        .padding(16)
        .frame(width: 343, height: 232)
    }
}
```

Cannot use mutating member on immutable value: 'self' is immutable

Structure

State

A property wrapper type that can read and write a value managed by SwiftUI.

iOS 13.0+ | iPadOS 13.0+ | macOS 10.15+ | Mac Catalyst 13.0+ | tvOS 13.0+ | watchOS 6.0+

Declaration

```
@frozen @propertyWrapper struct State<Value>
```

Structure

State

A property wrapper type that can read and write a value managed by SwiftUI.

tvOS 13.0+ watchOS 6.0+

Wraps property access with additional behavior

Declaration

```
@frozen @propertyWrapper struct State<Value>
```

@State

- 값이 변경되면 이 값에 의존 하는 View들이 update
- Use state as the single source of truth for a given value stored in a view hierarchy
- State는 그 자체로 값을 나타내지는 않음; 내부의 값을 읽고 쓸 수 있게 해주는 아이
- var wrappedValue: Value -> State의 getter 메서드는 이 값을 리턴
- var projectedValue: Binding<Value> -> child view에서 데이터 바인딩할 때 사용, 달러사인 ($)을 State 변수 앞에 붙여서 이 변수에 접근 가능

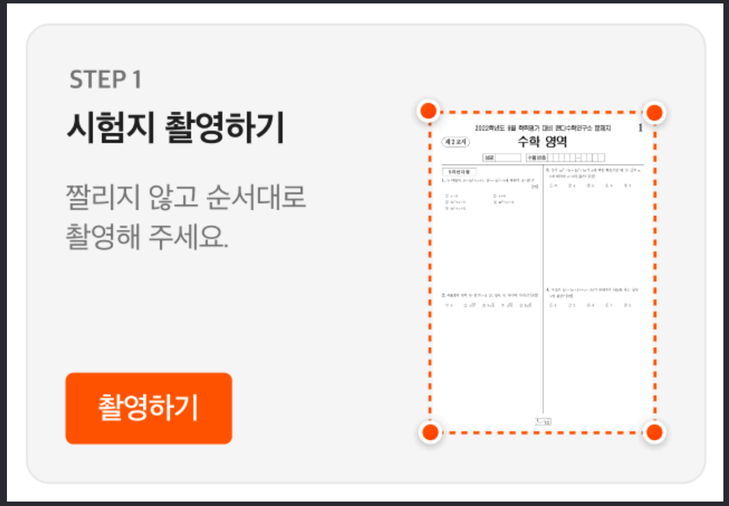
@State

주의할 점

- SwiftUI에서 제공하는 storage management가 고장날 수 있으므로 view heierarchy 에서 이니셜라이즈 해서는 안됩니다.

  -> 외부에서 주입 받는것 🙅‍♂️
  -> private으로 선언 🙆‍♂️

```swift
struct ContentView: View {

    @State private var isCaptured: Bool = false

    var body: some View {
        HStack {
            VStack(alignment: .leading) {
                Text("STEP1")
                Spacer().frame(height: 4)
                Text("시험지 촬영하기")
                Spacer().frame(height: 12)
                Text("짤리지 않게 순서대로 촬영해주세요.")
                Spacer()
                Button(action: {
                    isCaptured.toggle()
                }, label: {
                    Text(isCaptured ? "수정하기" : "촬영하기")
                })
                .buttonStyle(.borderedProminent)
                .tint(isCaptured ? .black : .orange)
            }
            Image("sample")
        }
        .padding(16)
        .frame(width: 343, height: 232)
    }
}
```

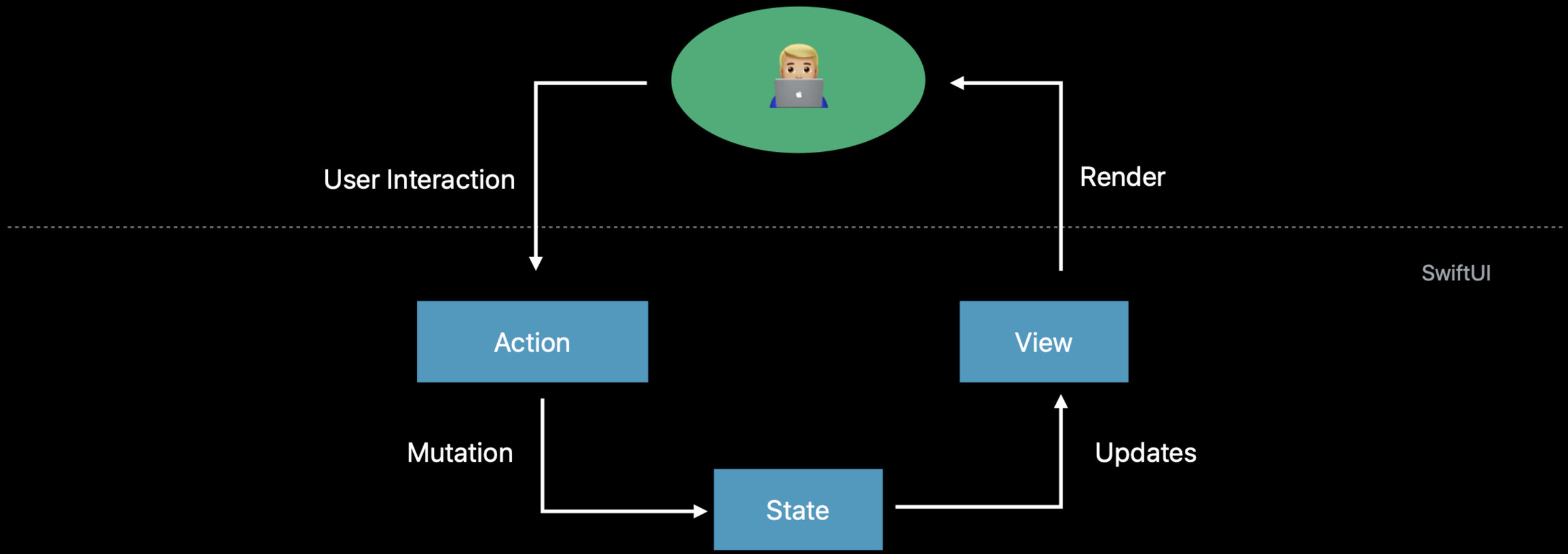


Anatomy of a View Update

뷰 업데이트가 어떻게 이루어 지는지

1. State가 선언되면 프레임워크 내부에서 persistent storage를 할당 (@State가 Struct 타입이더라도 wrappedValue를 share할 수 있는 이유)
2. View의 body에서 State가 사용된 경우 View의 렌더링이 해당 State에 의존하는 것으로 판단
3. 런타임에 State가 변경되면 해당 View의 모든 Body를 새로 연산
4. 렌더링 할 때는 diff를 체크하여 변경되는 부분만

```
struct ContentView: View {

    @State private var isCaptured: Bool = false

    var body: some View {
        HStack {
            VStack(alignment: .leading) {
                Text("STEP1")
                Spacer().frame(height: 4)
                Text("시험지 촬영하기")
                Spacer().frame(height: 12)
                Text("짤리지 않게 순서대로 촬영해주세요.")
                Spacer()
                Button(action: {
                    isCaptured.toggle()
                }, label: {
                    Text(isCaptured ? "수정하기" : "촬영하기")
                })
                .buttonStyle(.borderedProminent)
                .tint(isCaptured ? .black : .orange)
            }
            Image("sample")
        }
        .padding(16)
        .frame(width: 343, height: 232)
    }
}
```


User Interaction
Render
SwiftUI
Action
View
Mutation
State
Updates

Refactoring

Before

```
struct ContentView: View {

    @State private var isCaptured: Bool = false

    var body: some View {
        HStack {
            VStack(alignment: .leading) {
                Text("STEP1")
                Spacer().frame(height: 4)
                Text("시험지 촬영하기")
                Spacer().frame(height: 12)
                Text("짤리지 않게 순서대로 촬영해주세요.")
                Spacer()
                Button(action: {
                    isCaptured.toggle()
                }, label: {
                    Text(isCaptured ? "수정하기" : "촬영하기")
                })
                .buttonStyle(.borderedProminent)
                .tint(isCaptured ? .black : .orange)
            }
            Image("sample")
        }
        .padding(16)
        .frame(width: 343, height: 232)
    }
}
```

After

```
struct ContentView: View {

    @State private var isCaptured: Bool = false

    var body: some View {
        HStack {
            VStack(alignment: .leading) {
                Text("STEP1")
                Spacer().frame(height: 4)
                Text("시험지 촬영하기")
                Spacer().frame(height: 12)
                Text("짤리지 않게 순서대로 촬영해주세요.")
                Spacer()
                MyButton()
                .buttonStyle(.borderedProminent)
                .tint(isCaptured ? .black : .orange)
            }
            Image("sample")
        }
        .padding(16)
        .frame(width: 343, height: 232)
    }
}

struct MyButton: View {

    @State private var isCaptured: Bool = false

    var body: some View {
        Button(action: {
            isCaptured.toggle()
        }, label: {
            Text(isCaptured ? "수정하기" : "촬영하기")
        })
    }
}
```

Refactoring 🤦

Before

```
struct ContentView: View {

    @State private var isCaptured: Bool = false

    var body: some View {
        HStack {
            VStack(alignment: .leading) {
                Text("STEP1")
                Spacer().frame(height: 4)
                Text("시험지 촬영하기")
                Spacer().frame(heig
                Text("짤리지 않게 순서대
                Spacer()
                Button(action: {
                    isCaptured.toggl
                }, label: {
                    Text(isCaptured ? "수정하기" : "촬영하기")
                })
                .buttonStyle(.borderedProminent)
                .tint(isCaptured ? .black : .orange)
            }
            Image("sample")
        }
        .padding(16)
        .frame(width: 343, height: 232)
    }
}
```

1. not single source of truth

2. 초기값을 외부로부터 받아야 하는데...

After

```
struct ContentView: View {

    @State private var isCaptured: Bool = false

    var body: some View {
        HStack {
            VStack(alignment: .leading) {
                Text("STEP1")
                Spacer().frame(height: 4)
                Text("시험지 촬영하기")
                Spacer().frame(height: 12)
                Text("짤리지 않게 순서대로 촬영해주세요.")
                Spacer()
                MyButton()
                .buttonStyle(.borderedProminent)
                .tint(isCaptured ? .black : .orange)

            mage("sample")
        }
        .padding(16)
        .frame(width: 343, height: 232)
    }
}

struct MyButton: View {

    @State private var isCaptured: Bool = false

    var body: some View {
        Button(action: {
            isCaptured.toggle()
        }, label: {
            Text(isCaptured ? "수정하기" : "촬영하기")
        })
    }
}
```

Structure

Binding

A property wrapper type that can read and write a value owned by a source of truth.

iOS 13.0+ | iPadOS 13.0+ | macOS 10.15+ | Mac Catalyst 13.0+ | tvOS 13.0+ | watchOS 6.0+

Declaration

```
@frozen @propertyWrapper @dynamicMemberLookup struct Binding<Value>
```

@Binding

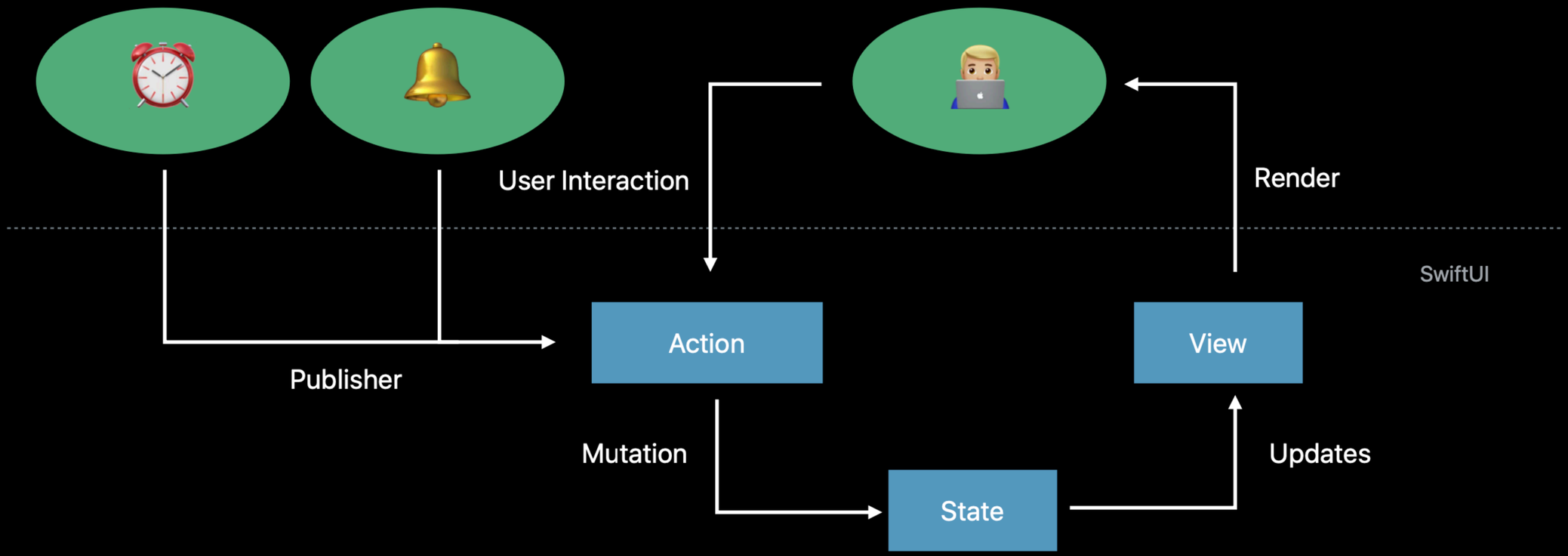
- 데이터를 직접 저장하지 않고 다른 곳(parent view)의 source of truth에 연결
- var wrappedValue: Value -> 연결된 source of truth의 value를 리턴
- var projectedValue: Binding<Value> -> 또 다시 하위의 View에 동일한 source of truth를 연결하고자 할 때 사용

Single source of truth with Binding

```swift
struct ContentView: View {

    @State private var isCaptured: Bool = false

    var body: some View {
        HStack {
            VStack(alignment: .leading) {
                Text("STEP1")
                Spacer().frame(height: 4)
                Text("시험지 촬영하기")
                Spacer().frame(height: 12)
                Text("짤리지 않게 순서대로 촬영해주세요.")
                Spacer()
                MyButton(isCaptured: $isCaptured)
                .buttonStyle(.borderedProminent)
                .tint(isCaptured ? .black : .orange)
            }
            Image("sample")
        }
        .padding(16)
        .frame(width: 343, height: 232)
    }
}
```

```swift
struct MyButton: View {

    @Binding var isCaptured: Bool

    var body: some View {
        Button(action: {
            isCaptured.toggle()
        }, label: {
            Text(isCaptured ? "수정하기" : "촬영하기")
        })
    }
}
```

Working With External Data



Combine Publisher

- 외부 이벤트에 대한 Single abstraction
- use onReceive(_:) for main thread

```swift
struct PlayerView : View {
    let episode: Episode
    @State private var isPlaying: Bool = true
    @State private var currentTime: TimeInterval = 0.0

    var body: some View {
        VStack {
            // …
            Text("\(playhead, formatter: currentTimeFormatter)")
        }
        .onReceive(PodcastPlayer.currentTimePublisher) { newCurrentTime in
            self.currentTime = newCurrentTime
        }
    }
}
```

Publisher 타입의 외부 데이터를 바인딩 하기 위해서onReceive 메서드를 호출하고 value change에 따라 업데이트 할 동작을 클로져에 정의하면 끝.

-> Publisher는 SwiftUI 타입이 아니라, 뷰에다가 바로 바인딩은 못함.
-> 뷰 외부의 이벤트를 뷰에다가 바로 바인딩 할 수 있는 방법 ?

External Data

BindableObject Protocol (deprecated)

- External
- Reference type
- 이미 사용중인 reference type의 모델에서 conform하여 SwiftUI의 View에서 사용가능

Protocol

ObservableObject

A type of object with a publisher that emits before the object has changed.

iOS 13.0+ | iPadOS 13.0+ | macOS 10.15+ | Mac Catalyst 13.0+ | tvOS 13.0+ | watchOS 6.0+

Declaration

```
protocol ObservableObject : AnyObject
```

External Data

ObservableObject

- var objectWillChange: Self.ObjectWilChangePublisher

  -> @Published 프로퍼티의 값이 변경되기 전에 Object를 emit하는 Publisher를 리턴하도록 기본 구현 제공.

Structure

Published

A type that publishes a property marked with an attribute.

iOS 13.0+ | iPadOS 13.0+ | macOS 10.15+ | Mac Catalyst 13.0+ | tvOS 13.0+ | watchOS 6.0+

Declaration

```
@propertyWrapper struct Published<Value>
```

External Data

@Published

- 변수명 앞에 달러사인($)을 붙이면 해당 타입의 publisher에 접근 가능
- wrappedValue의 값이 변경되기전에 willSet 블럭에서 publishing이 발생

  -> 해당 퍼블리셔를 구독하는 곳의 sink 클로져에 newValue가 전달됨
- class에서만 @Published를 프로퍼티로 사용가능

```
class Weather {
    @Published var temperature: Double
    init(temperature: Double) {
        self.temperature = temperature
    }
}

let weather = Weather(temperature: 20)
cancellable = weather.$temperature
    .sink() {
        print ("Temperature now: \($0)")
}
weather.temperature = 25

// Prints:
// Temperature now: 20.0
// Temperature now: 25.0
```

Structure

ObservedObject

A property wrapper type that subscribes to an observable object and invalidates a view whenever the observable object changes.

iOS 13.0+ | iPadOS 13.0+ | macOS 10.15+ | Mac Catalyst 13.0+ | tvOS 13.0+ | watchOS 6.0+

Declaration

```
@propertyWrapper @frozen struct ObservedObject<ObjectType> where ObjectType :
```

ObservedObject - Example

```
class ContentViewModel: ObservableObject {
    @Published var isCaptured: Bool = false
}

struct ContentView: View {

    @ObservedObject var viewModel = ContentViewModel()

    var body: some View {
        HStack {
            VStack(alignment: .leading) {
                Text("STEP1")
                Spacer().frame(height: 4)
                Text("시험지 촬영하기")
                Spacer().frame(height: 12)
                Text("짤리지 않게 순서대로 촬영해주세요.")
                Spacer()
                MyButton(isCaptured: $viewModel.isCaptured)
                .buttonStyle(.borderedProminent)
                .tint(viewModel.isCaptured ? .black : .orange)
            }
            Image("sample")
        }
        .padding(16)
        .frame(width: 343, height: 232)
    }
}
```

Structure

EnvironmentObject

A property wrapper type for an observable object supplied by a parent or ancestor view.

iOS 13.0+ | iPadOS 13.0+ | macOS 10.15+ | Mac Catalyst 13.0+ | tvOS 13.0+ | watchOS 6.0+

Declaration

```
@frozen @propertyWrapper struct EnvironmentObject<ObjectType> where ObjectType
```

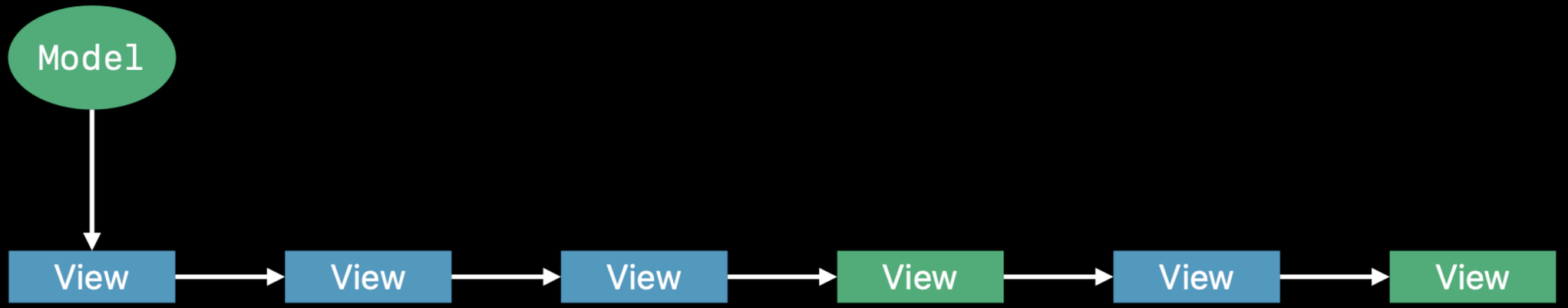
External Data

EnvironmentObject

- 부모 or 조상 뷰의 ObservableObject를 SSOT로 사용
- value가 변경될 때 마다 현재 뷰가 업데이트 됨
- 상위뷰에서는 반드시 environmentObject(_:) modifier를 호출되어야함

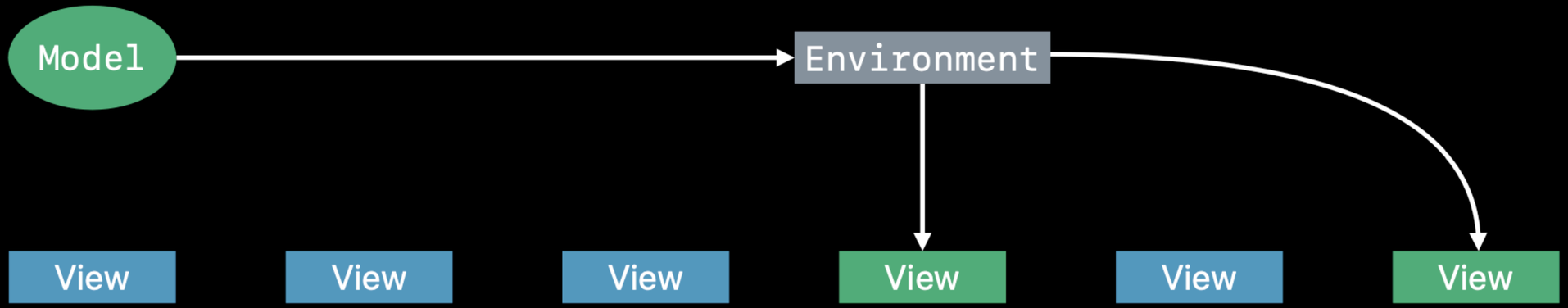
  -> 하위 뷰 계층으로 ObservableObject를 supply하는 modifier메서드

@ObservedObject



- 원하는 View까지 전달하기 위해서 모든 뷰 계층을 통해서 전부 pass해 줘야함

@EnvironmentObject



func environmentObject

- environmentObject를 호출한 뷰 이하로는 @EnvironmentObject 프로퍼티를 선언하는 것만으로 동일한 데이터를 사용 할 수 있음
- accent color, layout direction, theme 등에서 활용할 수 있음

Sourth of truth

올바른 데이터 도구 사용하기

`@State`

View-local

Value

Framework Managed

ObservedObject

External

Reference

Developer Managed

Building Reusable Components

- Swift property, Environment : 뷰에서 사용하는 데이터가 변경할 필요가 없는 경우 사용 (Read-only)
- @Binding : 뷰에서 사용하는 데이터가 변경될 필요가 있는 경우 사용 (Read-Write)
  - 데이터를 직접 소유하지 않더라도 읽고 쓸 수 있음 -> 재사용성 up

@Binding

다른 여러 데이터도 달러사인을 통해서 바인딩 변수로 접근 가능함