Thomas Sillmann, 02.12.2020

Status und SwiftUI

So geht es richtig!

Thomas Sillmann

Autor, Apple Developer, Trainer

- Seit 2015 Freiberufler
- App-Entwicklung für
  - App Store
  - Endkunden
- Autor von
  - „Das Swift-Handbuch“
  - „Einstieg in SwiftUI“

Agenda

Status und SwiftUI

Bedeutung des Status

Status-Arten

Umsetzung

Einbindung des eigenen Model

Best Practices

Bedeutung des Status

„Views are a function of state, not of a sequence of events.“

Apple, „Data Flow Through SwiftUI“ (WWDC 2019)

Aufgabe des Status

Information

- `Text`: Anzuzeigender Text
- `Image`: Anzuzeigendes Bild
- `Toggle`: An / Aus
- `ProgressView`: Aktueller Fortschritt
- `Link`: Aufzurufende URL

Aufgabe des Status

Umsetzung

- Jede View-Property entspricht einem Status
- Read-only / Read-write

```
struct TitleView: View {
    let text: String

    var body: some View {
        Text(text)
            .font(.largeTitle)
    }
}
```

```
struct TitleView: View {
    let text: String

    var body: some View {
        Text(text)
            .font(.largeTitle)
    }
}
```

Aufgabe des Status

Zweck

- Konfiguration einer View
- Automatische Aktualisierung einer View bei Änderung

Status-Arten

Status-Arten

- Source of Truth
- Derived Value

Source of Truth

- View besitzt die Daten
- Feste Verankerung der Daten mit der View
- Maximal 1 pro View (Idealfall)

Derived Value

- Weitergereichte Information
- Speicherung an anderer Stelle
- View *nutzt* die Daten, *besitzt* sie aber nicht
- Beliebig viele pro View möglich

```
struct TitleView: View {

    // Derived Value
    let text: String

    var body: some View {
        Text(text)
            .font(.largeTitle)
    }
}
```

Umsetzung

Property

Source of Truth / Derived Value

- Read-only
- Derived Value:
  - Wertzuweisung bei Initialisierung
- Source of Truth:
  - Konstante
  - Standardwert

State

Source of Truth

- Read-write
- Property Wrapper
- View speichert Daten
- Benötigt Standardwert
- Deklaration erfolgt typischerweise mit `private`

```
struct ReadingProgress: View {
    private var progress: Double = 0

    var body: some View {
        VStack {
            ProgressView(value: progress)
            Button("Finish reading") {
                progress = 1
            }
        }
    }
}
```

```
struct ReadingProgress: View {
    private var progress: Double = 0

    var body: some View {
        VStack {
            ProgressView(value: progress)
            Button("Finish reading") {
                progress = 1
            }
        }
    }
}
```

Cannot assign to property: 'self' is immutable

```
struct ReadingProgress: View {
    private var progress: Double = 0

    var body: some View {
        VStack {
            ProgressView(value: progress)
            Button("Finish reading") {
                progress = 1
            }
        }
    }
}
```

```
struct ReadingProgress: View {
    @State private var progress: Double = 0

    var body: some View {
        VStack {
            ProgressView(value: progress)
            Button("Finish reading") {
                progress = 1
            }
        }
    }
}
```

```
struct ReadingProgress: View {
    @State private var progress: Double = 0

    var body: some View {
        VStack {
            ProgressView(value: progress)
            Button("Finish reading") {
                progress = 1
            }
        }
    }
}
```

State ist für View-Daten

State

Beispiele

- Highlighted-State eines Buttons
- Automatische Animationen
- Sichtbarkeit eines Sheets oder Alerts

```
struct DeleteButton: View {
    @State private var showDeletionAlert = false

    var body: some View {
        Button("Delete") {
            showDeletionAlert = true
        }
        .alert(...)
    }
}
```

Binding
Derived Value

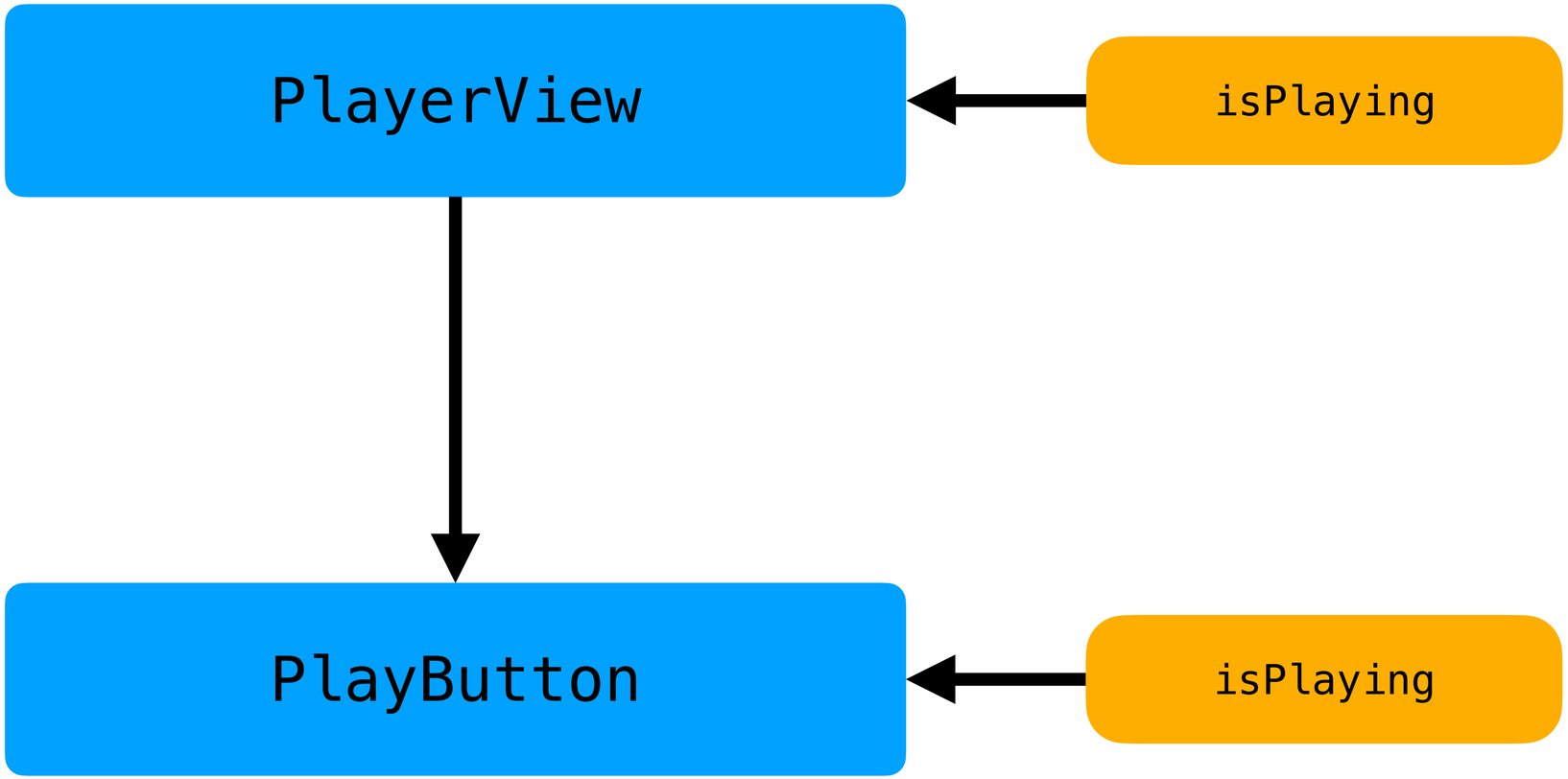
- Read-write
- Property Wrapper
- Erhält *Referenz* zu Source of Truth
- Speicherung zugrunde liegender Daten erfolgt an anderer Stelle

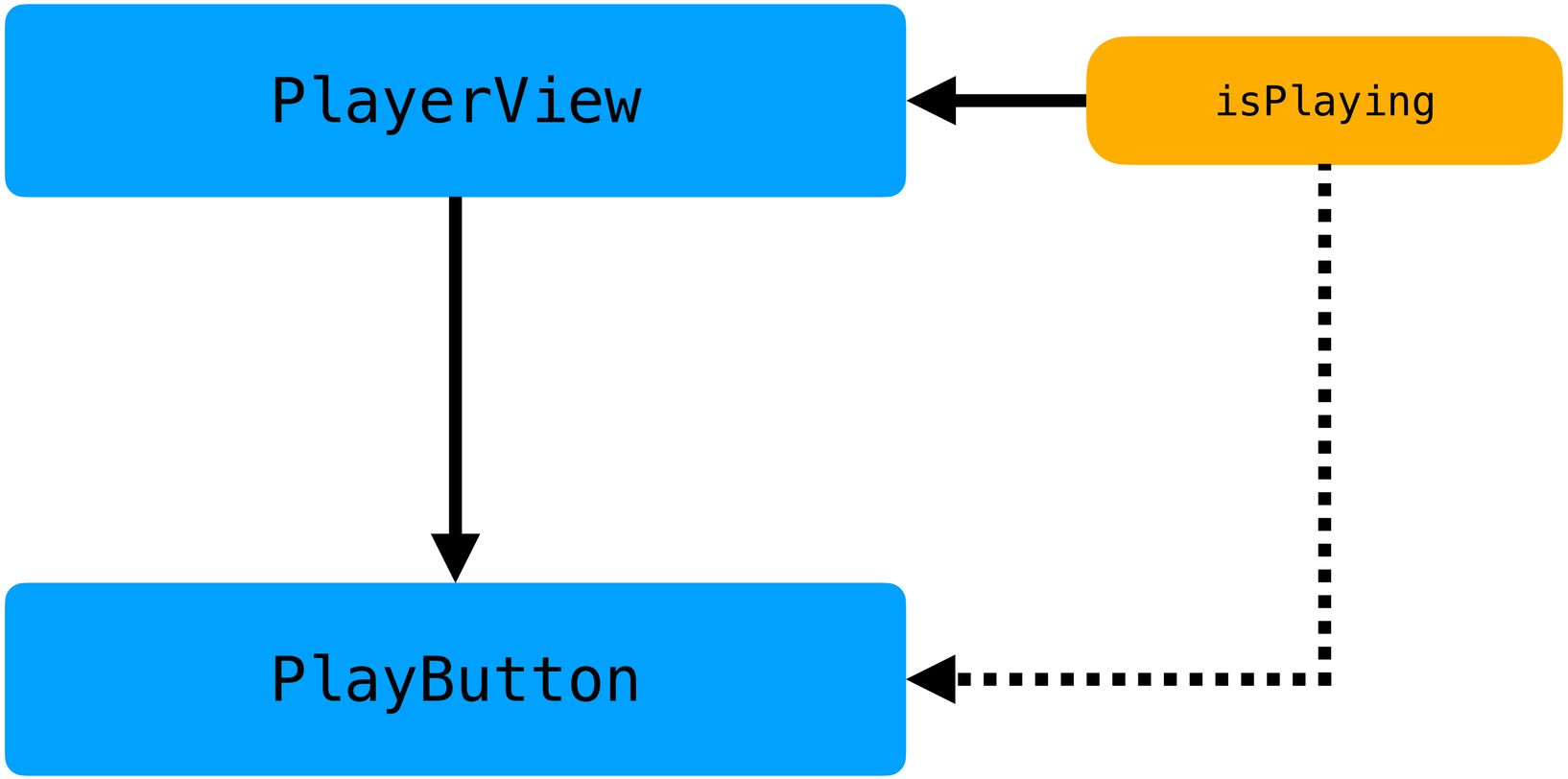
```
struct PlayerView: View {
    let track: String
    @State private var isPlaying = false

    var body: some View {
        VStack {
            Text(track)
            PlayButton()
        }
    }
}
```

```
struct PlayButton: View {
    @State private var isPlaying = false

    var body: some View {
        Button(action: {
            isPlaying.toggle()
        }, label: {
            Image(systemName: imageName())
        })
    }

    private func imageName() -> String {
        isPlaying ? "pause.circle" : "play.circle"
    }
}
```

```
struct PlayerView: View {
    let track: String
    @State private var isPlaying = false

    var body: some View { ... }
}

struct PlayButton: View {
    @State private var isPlaying = false

    var body: some View { ... }
}
```

```
struct PlayerView: View {
    let track: String
    @State private var isPlaying = false

    var body: some View { ... }
}

struct PlayButton: View {
    @State private var isPlaying = false

    var body: some View { ... }
}
```


PlayerView
isPlaying
PlayButton
isPlaying


PlayerView
isPlaying
PlayButton

```
struct PlayButton: View {
    @Binding var isPlaying: Bool

    var body: some View {
        Button(action: {
            isPlaying.toggle()
        }, label: {
            Image(systemName: imageName())
        })
    }

    private func imageName() -> String {
        isPlaying ? "pause.circle" : "play.circle"
    }
}
```

```swift
struct PlayButton: View {
    @Binding var isPlaying: Bool

    var body: some View {
        Button(action: {
            isPlaying.toggle()
        }, label: {
            Image(systemName: imageName())
        })
    }

    private func imageName() -> String {
        isPlaying ? "pause.circle" : "play.circle"
    }
}
```

```
struct PlayerView: View {
    let track: String
    @State private var isPlaying = false

    var body: some View {
        VStack {
            Text(track)
            PlayButton(isPlaying: <Parameter>)
        }
    }
}
```

```
struct PlayerView: View {
    let track: String
    @State private var isPlaying = false

    var body: some View {
        VStack {
            Text(track)
            PlayButton(isPlaying: <Parameter>)
        }
    }
}
```

State

Property Wrapper

- Wrapped Value entspricht eigentlichem Wert
- Projected Value verweist auf Binding
- `$isPlaying`
- Alternativ: Binding programmatisch erzeugen

```
struct PlayerView: View {
    let track: String
    @State private var isPlaying = false

    var body: some View {
        VStack {
            Text(track)
            PlayButton(isPlaying: $isPlaying)
        }
    }
}
```

```
struct PlayerView: View {
    let track: String
    @State private var isPlaying = false

    var body: some View {
        VStack {
            Text(track)
            PlayButton(isPlaying: $isPlaying)
        }
    }
}
```

Binding

Beispiele

- Schalter (an/aus)
- Picker (gewählter Wert)
- Textfeld (Quelle des Textes)

Status-Wahl

Welcher ist der richtige?

- Property
- State
- Binding

Status-Wahl

Fragen zur View-Erstellung

- Welche Daten benötigt eine View?
- Ändert die View die Daten?
- Woher kommen die Daten?

Status-Wahl

Welche Daten benötigt eine View?

- Informationen und Parameter
- Properties

Status-Wahl

Ändert die View die Daten?

- Read-only oder Read-write?
- `var`- oder `let`-Property

Status-Wahl

Woher kommen die Daten?

- Source of Truth
- State: View ist Quelle
- Binding: Daten kommen von anderer Stelle

Einbindung des eigenen Model

Einbindung des eigenen Model

Grundlagen

- Unterscheidung von View- und Model-Daten
- View-Daten basieren auf Value Types
- Model-Daten basieren auf Reference Types

Einbindung des eigenen Model

Model-Daten

- Basis: `ObservableObject`-Protokoll
- Einbindung eigener Model-Klassen als (änderbaren) Status in SwiftUI

```
class Person {
    var firstName: String = ""
    var lastName: String = ""

    var fullName: String {
        firstName + " " + lastName
    }
}
```

```
class Person: ObservableObject {
    var firstName: String = ""
    var lastName: String = ""

    var fullName: String {
        firstName + " " + lastName
    }
}
```

```
struct PersonView: View {
    var person: Person

    var body: some View {
        Form {
            Section {
                TextField("First name", text: <Parameter>)
                TextField("Last name", text: <Parameter>)
            }
            Section {
                Text("Name: \(person.fullName)")
            }
        }
    }
}
```

```
struct PersonView: View {
    var person: Person

    var body: some View {
        Form {
            Section {
                TextField("First name", text: <Parameter>)
                TextField("Last name", text: <Parameter>)
            }
            Section {
                Text("Name: \(person.fullName)")
            }
        }
    }
}
```

ObservedObject

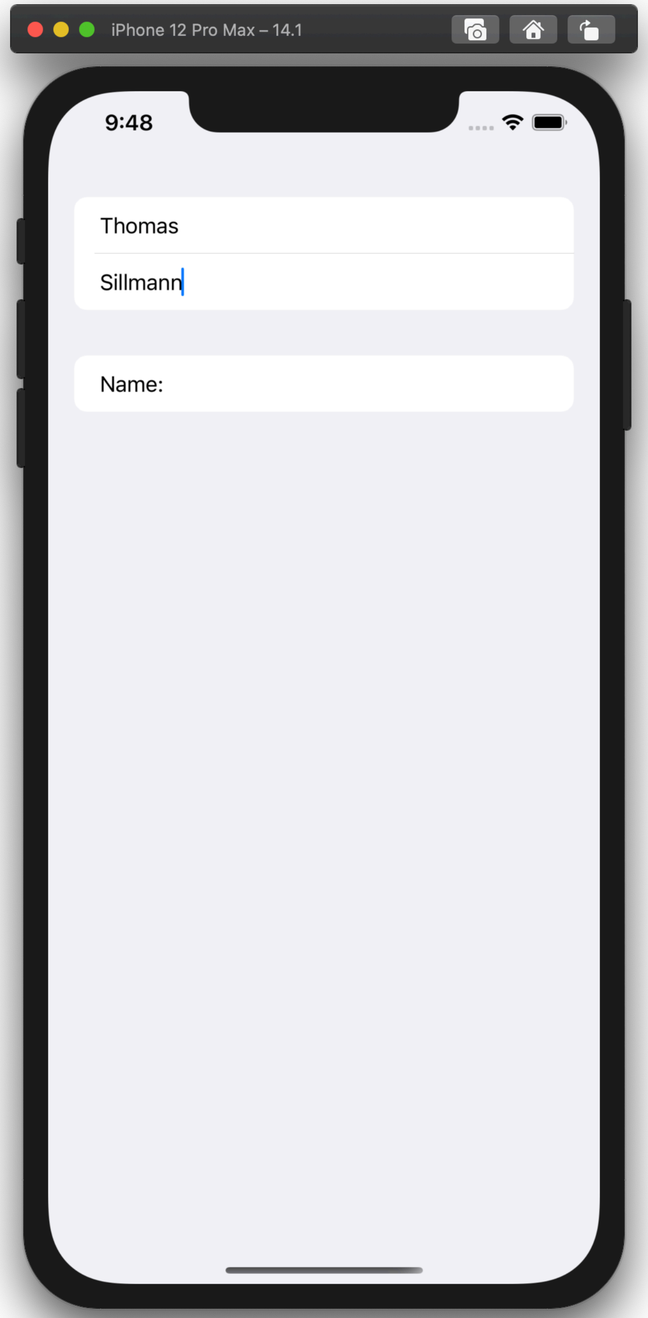
Source of Truth

- Read-write
- Status für eigene Model-Klassen
- Erzeugt Bindings für Eigenschaften der Model-Klasse
- `$person.firstName`
- Zuweisung erfolgt bei Initialisierung

```
struct PersonView: View {
    var person: Person

    var body: some View {
        Form {
            Section {
                TextField("First name", text: <Parameter>)
                TextField("Last name", text: <Parameter>)
            }
            Section {
                Text("Name: \(person.fullName)")
            }
        }
    }
}
```

```
struct PersonView: View {
    @ObservedObject var person: Person

    var body: some View {
        Form {
            Section {
                TextField("First name", text: $person.firstName)
                TextField("Last name", text: $person.lastName)
            }
            Section {
                Text("Name: \(person.fullName)")
            }
        }
    }
}
```

```
struct PersonView: View {
    @ObservedObject var person: Person

    var body: some View {
        Form {
            Section {
                TextField("First name", text: $person.firstName)
                TextField("Last name", text: $person.lastName)
            }
            Section {
                Text("Name: \(person.fullName)")
            }
        }
    }
}
```


iPhone 12 Pro Max – 14.1
9:48
Thomas
Sillmann
Name:

ObservedObject

Source of Truth

- Model-Änderungen führen nicht automatisch zu View-Aktualisierung
- Zwei Möglichkeiten:
  - Anstoßen mittels Publisher
  - Einsatz des Published-Property Wrappers

ObservableObject

Publisher

- Besitzt Publisher
- Zugriff über `objectWillChange`-Property
- Update via `send()`-Methode
- `person.objectWillChange.send()`

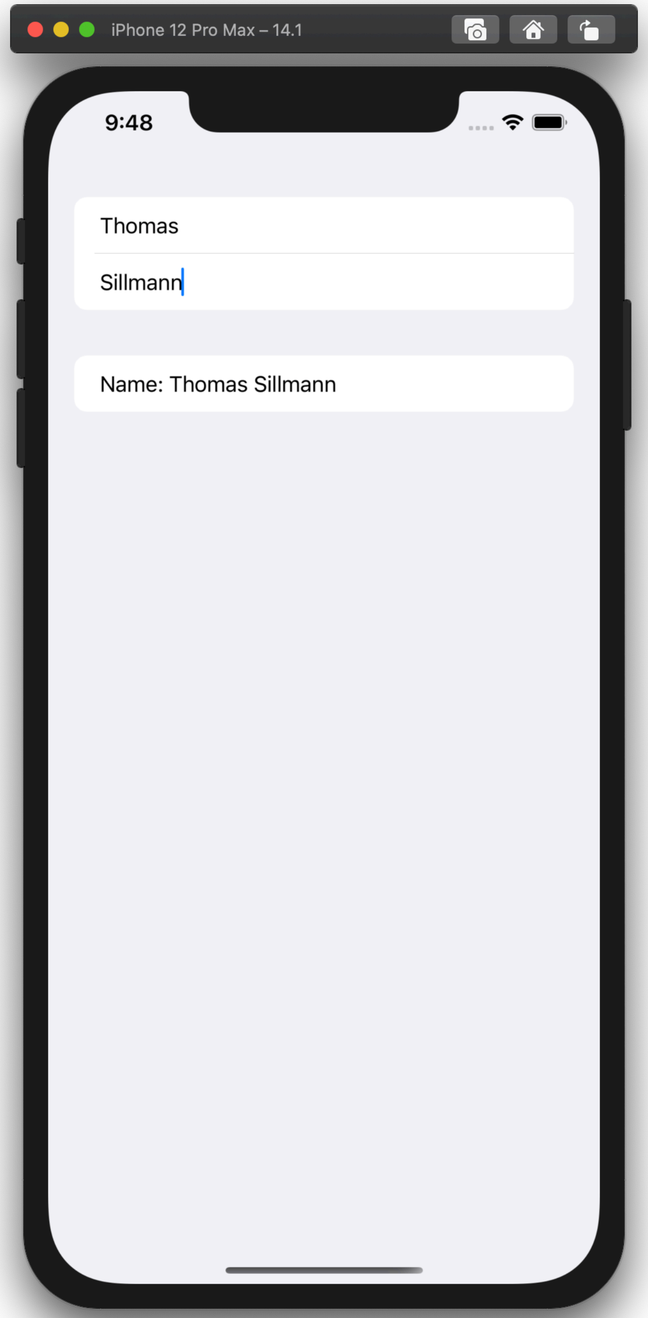
ObservableObject

Published-Property Wrapper

- Deklaration der änderbaren Eigenschaften als `Published`
- Automatische View-Aktualisierung bei Änderung der Eigenschaft

```
class Person: ObservableObject {
    var firstName: String = ""
    var lastName: String = ""

    var fullName: String {
        firstName + " " + lastName
    }
}
```

```
class Person: ObservableObject {
    @Published var firstName: String = ""
    @Published var lastName: String = ""

    var fullName: String {
        firstName + " " + lastName
    }
}
```


iPhone 12 Pro Max – 14.1
9:48
Thomas
Sillmann
Name: Thomas Sillmann

Zuweisung bei Initialisierung!

```
struct PersonView: View {
    @ObservedObject var person = Person()

    var body: some View {
        Form {
            Section {
                TextField("First name", text: $person.firstName)
                TextField("Last name", text: $person.lastName)
            }
            Section {
                Text("Name: \(person.fullName)")
            }
        }
    }
}
```

```
struct PersonView: View {
    @ObservedObject var person = Person()

    var body: some View {
        Form {
            Section {
                TextField("First name", text: $person.firstName)
                TextField("Last name", text: $person.lastName)
            }
            Section {
                Text("Name: \(person.fullName)")
            }
        }
    }
}
```

ObservedObject

Zuweisung

- Zuweisung bei Initialisierung, nicht mittels Standardwert
- `PersonView(person: Person())`
- Standardwert: Erzeugung einer neuen Instanz bei Aufruf von `body`
- Fehlerhafte Daten, Memory Leak
- Man selbst ist verantwortlich für Speicher-Management

StateObject

Source of Truth

- Read-write
- Mischung aus `State` und `ObservedObject`
- Status basiert auf Reference Type und `ObservableObject`-Protokoll
- Zuweisung erfolgt mittels Standardwert
- Kopplung der Daten mit View
- SwiftUI kümmert sich um Lebenszyklus der Daten

```
struct PersonView: View {
    @StateObject private var person = Person()

    var body: some View {
        Form {
            Section {
                TextField("First name", text: $person.firstName)
                TextField("Last name", text: $person.lastName)
            }
            Section {
                Text("Name: \(person.fullName)")
            }
        }
    }
}
```

```
struct PersonView: View {
    @StateObject private var person = Person()

    var body: some View {
        Form {
            Section {
                TextField("First name", text: $person.firstName)
                TextField("Last name", text: $person.lastName)
            }
            Section {
                Text("Name: \(person.fullName)")
            }
        }
    }
}
```

StateObject

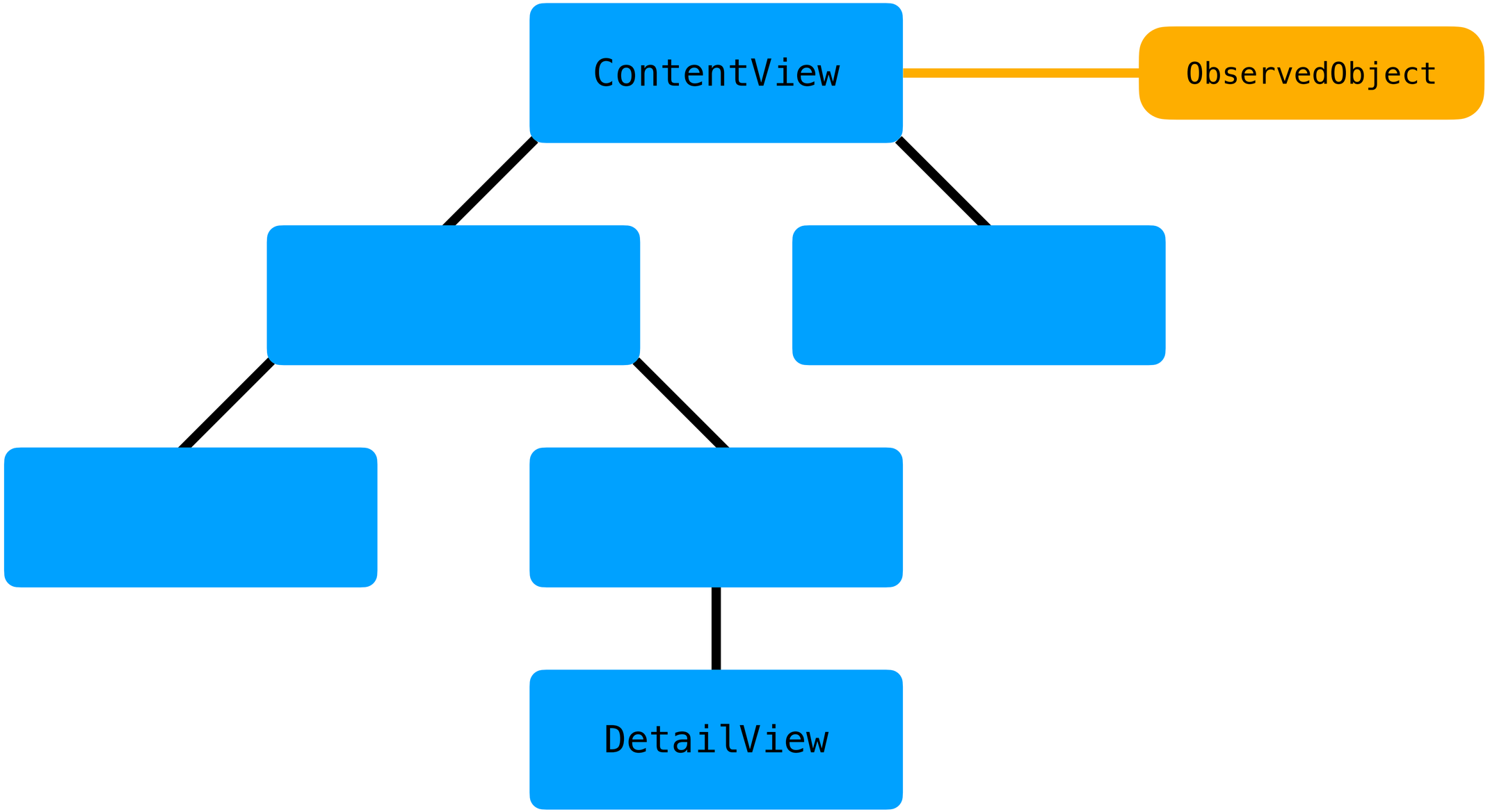
Einsatz

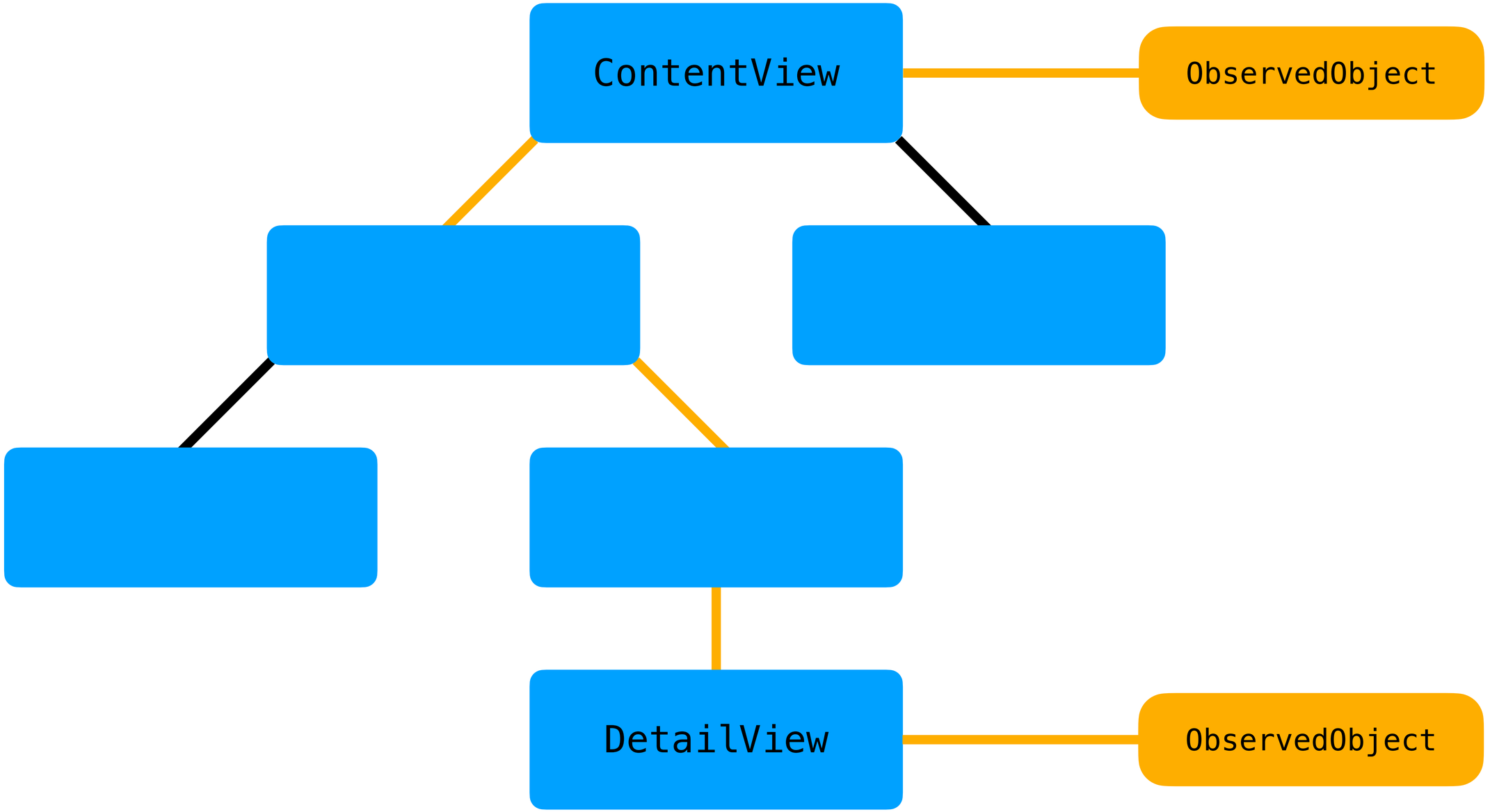
- Verzahnung des Models mit der View
- SwiftUI übernimmt Speicher-Management
- Ideal für:
  - Temporäre Model-Instanzen
  - Neu zu erstellende Model-Instanzen
  - Einbinden von Singletons

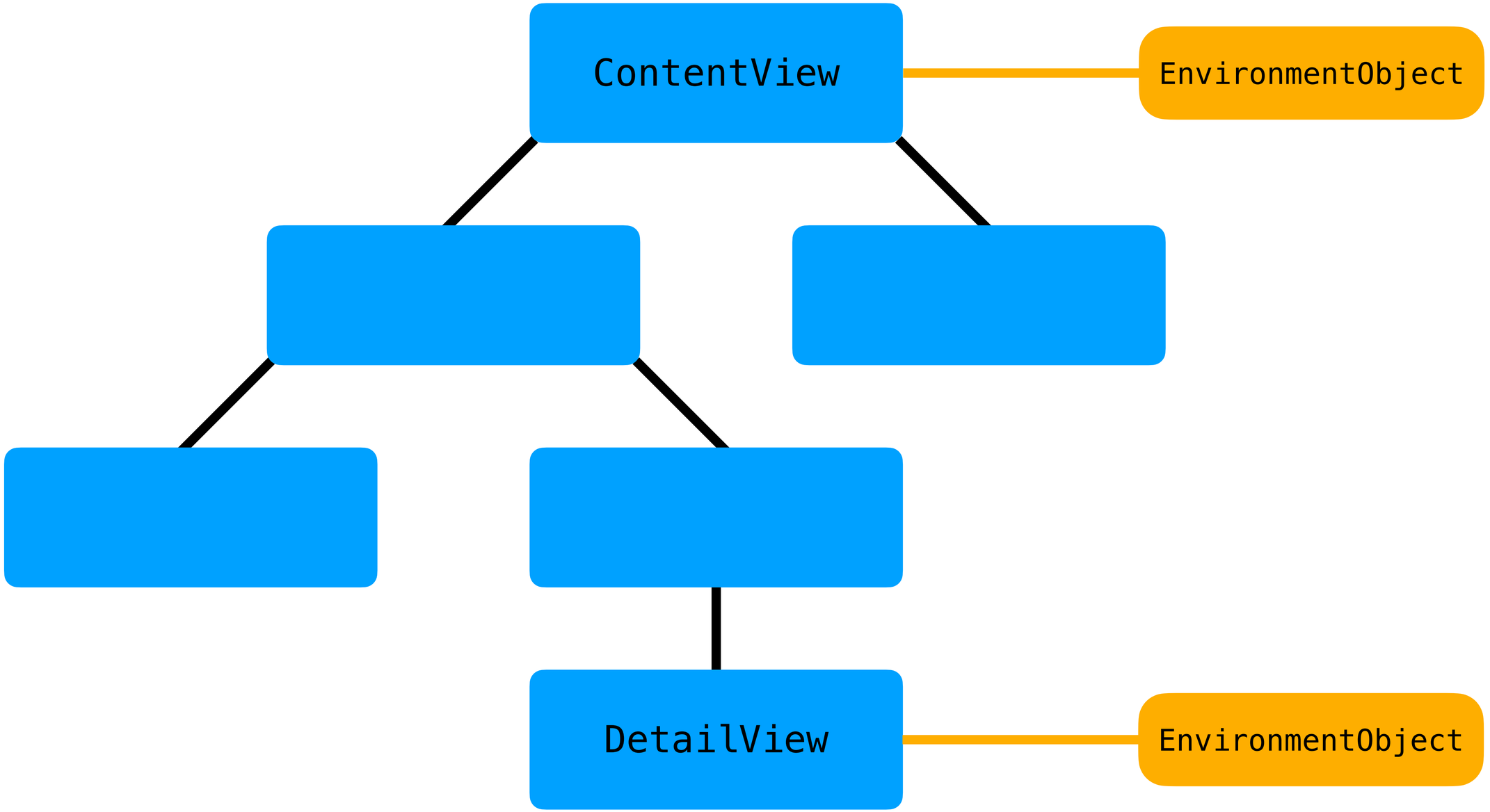
EnvironmentObject

Source of Truth

- Read-write
- Ähnlich `ObservedObject`
- Basiert auf Model-Daten und `ObservableObject`-Protokoll
- Zuweisung erfolgt mittels `environmentObject(_:)`-Modifier
- Status steht im gesamten View-Kontext zur Verfügung
- Kein Durchreichen von View zu View nötig


ContentView
ObservedObject
DetailView


ContentView
ObservedObject
DetailView
ObservedObject


ContentView
EnvironmentObject
DetailView
EnvironmentObject

```
struct TS_MyApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView()
                .environmentObject(Person())
        }
    }
}
```

```
struct PersonView: View {
    @EnvironmentObject var person: Person

    var body: some View {
        Form {
            Section {
                TextField("First name", text: $person.firstName)
                TextField("Last name", text: $person.lastName)
            }
            Section {
                Text("Name: \(person.fullName)")
            }
        }
    }
}
```

```
struct PersonView: View {
    @EnvironmentObject var person: Person

    var body: some View {
        Form {
            Section {
                TextField("First name", text: $person.firstName)
                TextField("Last name", text: $person.lastName)
            }
            Section {
                Text("Name: \(person.fullName)")
            }
        }
    }
}
```

EnvironmentObject

Wichtiges und beachtenswertes

- Bei Fehlen kommt es zur Zugriffs-Zeit zum Crash
- Compiler erkennt fehlendes Environment-Object nicht
- Mögliche Stolperfalle: Preview
- View-Kontext schließt Sheets aus

```
struct PersonView: View {
    @State private var showsSheet = false
    @EnvironmentObject var person: Person

    var body: some View {
        Form {...}
        .sheet(isPresented: $showsSheet, content: {
            AddPersonView()
            // Kein Zugriff auf person!
        })
    }
}
```

EnvironmentObject

Einsatz

- Mehrere Environment-Objects pro View-Kontext möglich
- Typ muss eindeutig sein
- Property-Name irrelevant

```
struct TS_MyApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView()
                .environmentObject(Person())
                .environmentObject(Address())
        }
    }
}
```

```
struct PersonView: View {
    @EnvironmentObject var person: Person
    @EnvironmentObject var address: Address

    var body: some View {...}
}
```

Best Practices

```
struct AddressView: View {
    @State private var name = ""
    @State private var street = ""
    @State private var postalCode = ""
    @State private var city = ""

    var body: some View {
        Form {...}
    }
}
```

```
struct AddressInformation {
    var name = ""
    var street = ""
    var postalCode = ""
    var city = ""
}

struct AddressView: View {
    @State private var address = AddressInformation()

    var body: some View {
        Form {...}
    }
}
```

Best Practices

Status und SwiftUI

- Views klein halten
- Nur die Views aktualisieren, die eine Änderung betrifft

Best Practices

Fragen zur View-Erstellung

- Welche Daten benötigt eine View?
- Ändert die View die Daten?
- Woher kommen die Daten?

Thomas Sillmann, 02.12.2020

Status und SwiftUI

So geht es richtig!