**Combine Framework - Notes**

Combine is Apple's framework for handling asynchronous events using declarative Swift code. It uses **Publishers** and **Subscribers** to manage data streams and handle state changes reactively.

**Key Concepts**

**1️⃣ Publisher & Subscriber**

* **Publisher**: Emits values over time.
* **Subscriber**: Receives and processes emitted values.

swift

CopyEdit

import Combine

let publisher = Just("Hello, Combine!") // Emits a single value

let subscriber = publisher.sink { value in

print(value) // Prints: Hello, Combine!

}

**2️⃣ Operators**

Operators modify values before they reach the subscriber.

swift

CopyEdit

let numbers = [1, 2, 3, 4, 5].publisher

let subscriber = numbers

.map { $0 \* 10 } // Multiply each value by 10

.filter { $0 > 20 } // Only allow values greater than 20

.sink { print($0) } // Prints: 30, 40, 50

**3️⃣ Subjects**

Subjects act as both **publishers** and **subscribers**.

* PassthroughSubject: Emits new values dynamically.
* CurrentValueSubject: Starts with an initial value and updates over time.

swift

CopyEdit

let subject = PassthroughSubject<String, Never>()

let subscription = subject.sink { print($0) }

subject.send("First message") // Prints: First message

subject.send("Second message") // Prints: Second message

**4️⃣ Combine with SwiftUI**

Use @Published and ObservableObject to bind Combine to SwiftUI.

swift

CopyEdit

import SwiftUI

import Combine

class NoteViewModel: ObservableObject {

@Published var note = "Initial Note"

}

struct NoteView: View {

@StateObject var viewModel = NoteViewModel()

var body: some View {

VStack {

Text(viewModel.note)

Button("Update Note") {

viewModel.note = "Updated Note"

}

}

}

}

**5️⃣ Combine with URLSession (Network Requests)**

swift

CopyEdit

struct Post: Codable {

let title: String

}

let url = URL(string: "https://jsonplaceholder.typicode.com/posts/1")!

let publisher = URLSession.shared.dataTaskPublisher(for: url)

.map(\.data)

.decode(type: Post.self, decoder: JSONDecoder())

let subscription = publisher.sink(receiveCompletion: { print($0) },

receiveValue: { print($0.title) })

**Summary**

✅ **Publishers** emit values  
✅ **Subscribers** receive values  
✅ **Operators** transform values  
✅ **Subjects** send new values  
✅ **SwiftUI + Combine** keeps UI reactive  
✅ **Network calls** can use Combine