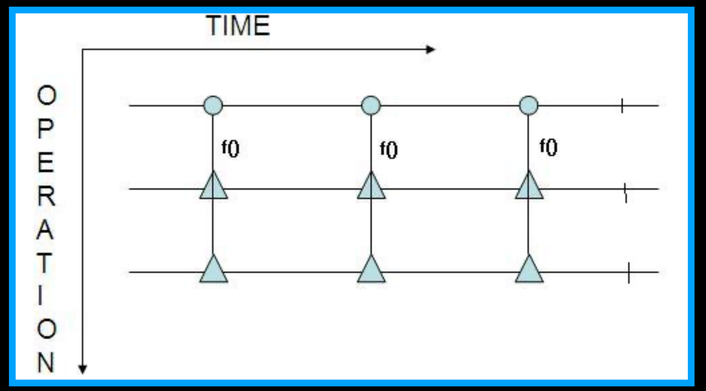
Combine

The Reactive Way

Onur H. Cantay 25.03.2022

What is Reactive Programming (FRP)

In a nutshell Reactive programming is a programming paradigm that makes use of functional programming methods (building blocks) such as map, reduce and filter to execute certain tasks on a specific **TIME.**



Combine

Apple Documentation

- Customize handling of asynchronous events by combining event-processing operators.

The Combine framework provides a declarative approach for how your app processes events.
Rather than potentially implementing multiple delegate callbacks or completion handler closures, you can create a single processing chain for a given event source.
Each part of the chain is a Combine operator that performs a distinct action on the elements received from the previous step.

Important Terms

Important Types

- **Publisher**
- **Subscriber**
- **Cancellable**

A. **Future**

B. **Just**

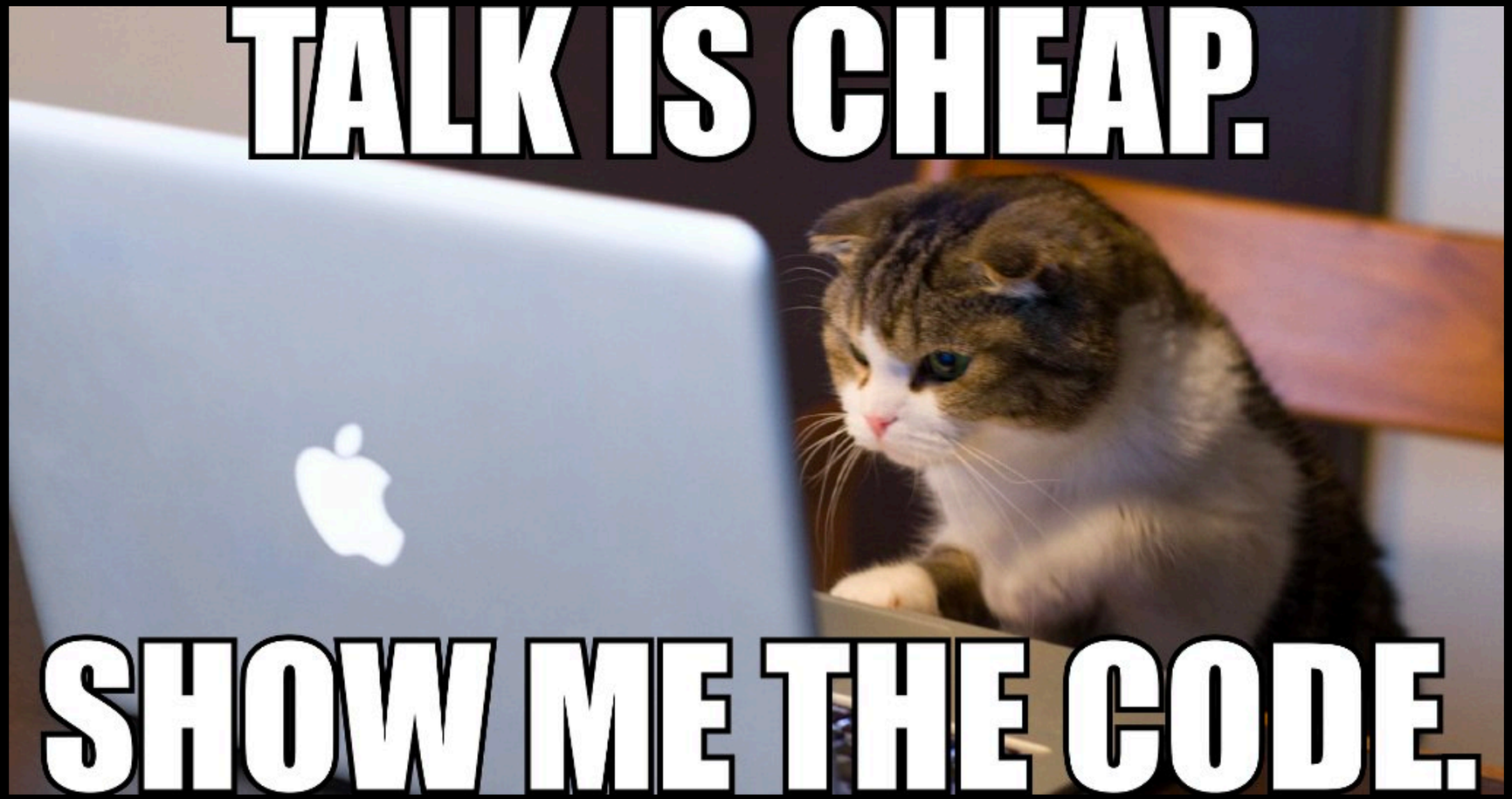
C. **Empty**

Important Functions

- **eraseToAnyPublisher() `Type Erasure`**
- **sink()**
- **receive(on:)**
- **mapError()**

Important Use Cases

- **Filtering Search History**
- **Validating Input Fields**
- **Making a network call**
- **Listening to a notification observation**
- **SwiftUI itself :)**


TALK IS CHEAP.
SHOW ME THE CODE.

Code Samples

Sample

```swift
import Combine

let sequencePublisher = [1, 2, 3, 4].publisher

let cancellable = sequencePublisher.sink { result in // -> AnyCancellable
    switch result {
    case .finished:
        print("Sequence Finished")
    case .failure:
        print("Imposible is Nothing")
    }
} receiveValue: { value in
    print("Current Sequence Value - ", value)
}
```

Output

```
Current Sequence Value -  1
Current Sequence Value -  2
Current Sequence Value -  3
Current Sequence Value -  4
Sequence Finished
```

Code Samples

Sample

```swift
import Combine

let sequencePublisher = [1, 2, 3, 4].publisher
let stringSequencePublisher = ["One", "Two", "Three", "Four"].publisher
var cancellables: Set<AnyCancellable> = []

stringSequencePublisher
    .zip(sequencePublisher)
    .flatMap { (stringPublisher, intPublisher) in
        Just("Int Value: \(intPublisher), String Value: \(stringPublisher)")
    }.sink { result in
        switch result {
        case .finished:
            print("Sequence Finished")
        case .failure:
            print("Imposible is Nothing")
        }
    } receiveValue: {
        print($0)
    }.store(in: &cancellables)
```

Output

```
Int Value: 1, String Value: One
Int Value: 2, String Value: Two
Int Value: 3, String Value: Three
Int Value: 4, String Value: Four
Sequence Finished
```

Code Samples

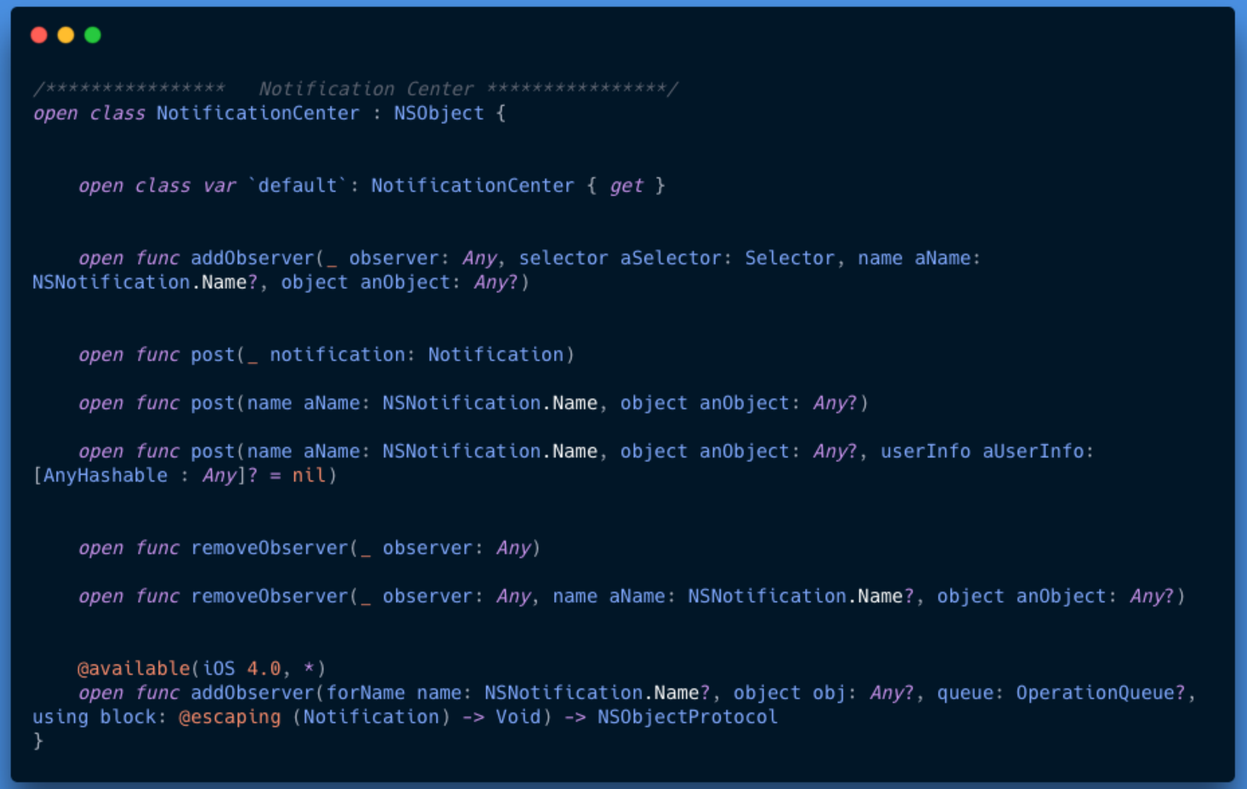
Sample

```swift
let mergedSequencePublisher = stringSequencePublisher
    .zip(sequencePublisher)
    .flatMap { (stringPublisher, intPublisher) in
        Just("Int Value: \(intPublisher), String Value: \(stringPublisher)")
    }.map { $0 as String? }
    .eraseToAnyPublisher()

import UIKit

private let sequenceLabel: UILabel = {
    let label: UILabel = .init()
    label.font = .systemFont(ofSize: 16, weight: .bold)
    label.textColor = .brown
    return label
}()

mergedSequencePublisher.assign(to: \.text, on: sequenceLabel)

mergedSequencePublisher.sink { result in
        switch result {
        case .finished:
            print("Sequence Finished")
        case .failure:
            print("Imposible is Nothing")
        }
    } receiveValue: {
        print($0)
    }.store(in: &cancellables)

print("------------")
print(sequenceLabel.text)
```

Output

```
Optional("Int Value: 1, String Value: One")
Optional("Int Value: 2, String Value: Two")
Optional("Int Value: 3, String Value: Three")
Optional("Int Value: 4, String Value: Four")
Sequence Finished
------------
Optional("Int Value: 4, String Value: Four")
```

Notification Center



```swift
/*****************   Notification Center ****************/
open class NotificationCenter : NSObject {


    open class var `default`: NotificationCenter { get }


    open func addObserver(_ observer: Any, selector aSelector: Selector, name aName:
NSNotification.Name?, object anObject: Any?)


    open func post(_ notification: Notification)

    open func post(name aName: NSNotification.Name, object anObject: Any?)

    open func post(name aName: NSNotification.Name, object anObject: Any?, userInfo aUserInfo:
[AnyHashable : Any]? = nil)


    open func removeObserver(_ observer: Any)

    open func removeObserver(_ observer: Any, name aName: NSNotification.Name?, object anObject: Any?)


    @available(iOS 4.0, *)
    open func addObserver(forName name: NSNotification.Name?, object obj: Any?, queue: OperationQueue?,
using block: @escaping (Notification) -> Void) -> NSObjectProtocol
}
```

```swift
@available(macOS 10.15, iOS 13.0, tvOS 13.0, watchOS 6.0, *)
extension NotificationCenter {

    /// Returns a publisher that emits events when broadcasting notifications.
    ///
    /// - Parameters:
    ///   - name: The name of the notification to publish.
    ///   - object: The object posting the named notfication. If `nil`, the publisher emits elements for
any object producing a notification with the given name.
    /// - Returns: A publisher that emits events when broadcasting notifications.
    public func publisher(for name: Notification.Name, object: AnyObject? = nil) ->
NotificationCenter.Publisher
}
```

Notification Center

```swift
protocol NotificationCenterProxyProtocol {
    var notificationCenter: NotificationCenter { get }

    /// Generates a Publisher from a notification name and an object.
    /// - Returns: type erased publisher for the given name.
    func publisher(_ name: Notification.Name, object: AnyObject?) -> AnyPublisher<Notification, Never>
}
```

```swift
final class NotificationCenterProxy: NotificationCenterProxyProtocol {

    // MARK: - Public Variables

    private(set) var notificationCenter: NotificationCenter

    // MARK: - Object Lifecycle

    init(notificationCenter: NotificationCenter = .default) {
        self.notificationCenter = notificationCenter
    }

    // MARK: - Publisher

    func publisher(_ name: Notification.Name, object: AnyObject? = nil) -> AnyPublisher<Notification,
Never> {
        notificationCenter
            .publisher(for: name, object: object)
            .eraseToAnyPublisher()
    }
}
```

Writing Test Code
in Playground

Dependency Injection

SAMPLE PLAYGROUND

CombineGround

Testing

Schedulers

Stub

CustomError

RXSwift / Combine

Adaptation

Code Samples

```swift
// Thank you Jonas <3

@available(iOS 13.0, *)
public extension Promise {
    func toFuture() -> Future<T, Error> {
        return Future { promise in
            self.done { value in
                promise(.success(value))
            }.catch { error in
                promise(.failure(error))
            }
        }
    }
}

@available(iOS 13.0, *)
public extension Guarantee {
    func toFuture() -> Future<T, Never> {
        return Future { promise in
            self.done { value in
                promise(.success(value))
            }
        }
    }
}
```

Code Samples

```
// Firebase Database Listener
func startListening(
        euci: String,
        token: String?
    ) -> AnyPublisher<[ConversationsApiMessage], ChatMessageProviderError> {
        let publisher = PassthroughSubject<Data, ChatMessageProviderError>()
        var listenerRegistration: ListenerRegistration?
            // Some Data Preparation

        listenerRegistration = query.addSnapshotListener { querySnapshot, error in
           // Some Data Preparation
                if let data = try? environment
                    .jsonSerializationType
                    .data(
                        withJSONObject: messagesData,
                        options: []
                    ) {
                    publisher.send(data)
                }
            }
        })

        return publisher
            .decode(type: [ConversationsApiMessage].self, decoder: environment.jsonDecoder)
            .handleEvents(
                receiveCompletion: { _ in listenerRegistration?.remove() },
                receiveCancel: { listenerRegistration?.remove() }
            )
            .mapError(map)
            .eraseToAnyPublisher()
    }
```

Bonus

Async - Await VS Combine

Async - Awaits structured Concurrency provides us with new way to express this kind of logic, we can now write asynchronous code that split into smaller pieces and reads from top-to-bottom instead of as a series of chained transforms.

Its design focuses on providing a way to declaratively specify a chain of these operators transforming data as it moves from one end to the other. Sometimes this leads to call sites that are more complex than one might expect - especially when working with single values, errors or data that needs to be shared.

Thank You

Onur H. Cantay 25.03.2022