

# Combine Framework

Berdil Ilyada Karaçam

# Combine Nedir?

Combine, WWDC 2019'da tanıtılan, RxSwift, RxCocoa gibi kütüphanelere benzer bir yapıya sahip Apple tarafından hazırlanan yeni FRP Framework'üdür.

Apple'a göre Combine:

- *The framework provides a declarative Swift API for processing values over time.*
- *Customize handling of asynchronous events by combining event-processing operators.*

# Anahtar Kavramlar

- Publishers
- Subscribers
- Operators

# Publishers

*Publisher* sends sequences of values over time to one or more *Subscribers*.

```
@available(OSX 10.15, iOS 13.0, tvOS 13.0, watchOS 6.0, *)
public protocol Publisher {

    /// The kind of values published by this publisher.
    associatedtype Output

    /// The kind of errors this publisher might publish.
    ///
    /// Use `Never` if this `Publisher` does not publish errors.
    associatedtype Failure : Error

    /// This function is called to attach the specified `Subscriber` to this `Publisher` by `subscribe(_:)`
    ///
    /// - SeeAlso: `subscribe(_:)`
    /// - Parameters:
    ///   - subscriber: The subscriber to attach to this `Publisher`.
    ///     once attached it can begin to receive values.
    func receive<S>(subscriber: S) where S : Subscriber, Self.Failure == S.Failure, Self.Output == S.Input
}
```

Output, Publisher'ın göndereceği veri tipini belirler.

Failure, herhangi bir hata durumunda Publisher'ın döneceği hata tipini belirler.

receive(subscriber:) metodu ise Publisher'a Subscriber bağlamak için kullanılır. Ayrıca Publisher'ın Output'u ile Subscriber'ın Input tipinin ve Failure tiplerinin aynı olması gerektiğini tanımlar.

# Subscribers

*Subscriber* receives values from a *Publisher*.

```
protocol Subscriber {  
    associatedtype Input  
    associatedtype Failure: Error  
  
    func receive(subscription: Subscription)  
    func receive(_ input: Input) -> Subscribers.Demand  
    func receive(completion: Subscribers.Completion<Failure>)  
}
```

Input, Subscriber'ın alacağı veri tipini belirler.

Failure, herhangi bir hata durumunda Subscriber'ın edeceği hata tipini belirler.

Subscription ile publisher'a bağlantı başlatılır, input parametreli fonksiyon ile Publisher'dan Subscriber'a veri iletimi sağlanır ve iletim sonlandığında completion çağrırlar.

# Connecting Publisher to Subscriber

Combine'da publisherları subscriber'a bağlarken iki farklı method kullanılabilir.

- `.sink(receiveCompletion:receiveValue:)` : Publisher'dan gelen değeri kullanmamızı sağlayan ve bağlantı tamamlandığında kullanabileceğimiz bir closure sağlar.
- `.assign(to:on:)` : Herhangi bir property'e dönen değeri direkt olarak atamamızı sağlar.

# Combine – Swift API Entegrasyonu

Publisher ve Subscriber'lar protocol olarak tanımlandığından mevcut standart Swift API'da herhangi bir değişiklik yapılmadan extension'lar ile eklenebilir.

```
extension NotificationCenter {
    struct Publisher: Combine.Publisher {
        typealias Output = Notification
        typealias Failure = Never
        init(center: NotificationCenter, name: Notification.Name, object: Any? = nil)
    }
}
```

# Sample 1 – Publisher & Subscriber

# Operators

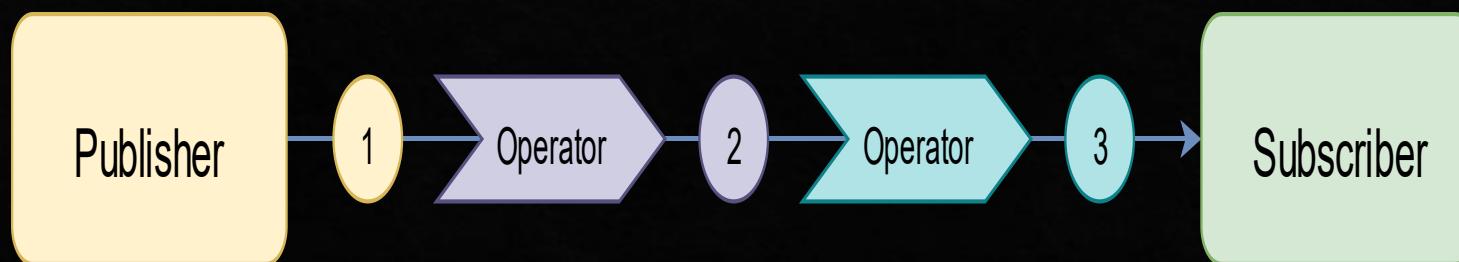
Operator'ler publisher'larda tanımlanan fonksiyonlardır.

Her Operator return type olarak yeni bir publisher döner.

Birbirlerine zincirleme şekilde işlem adımı olarak eklenebilirler.

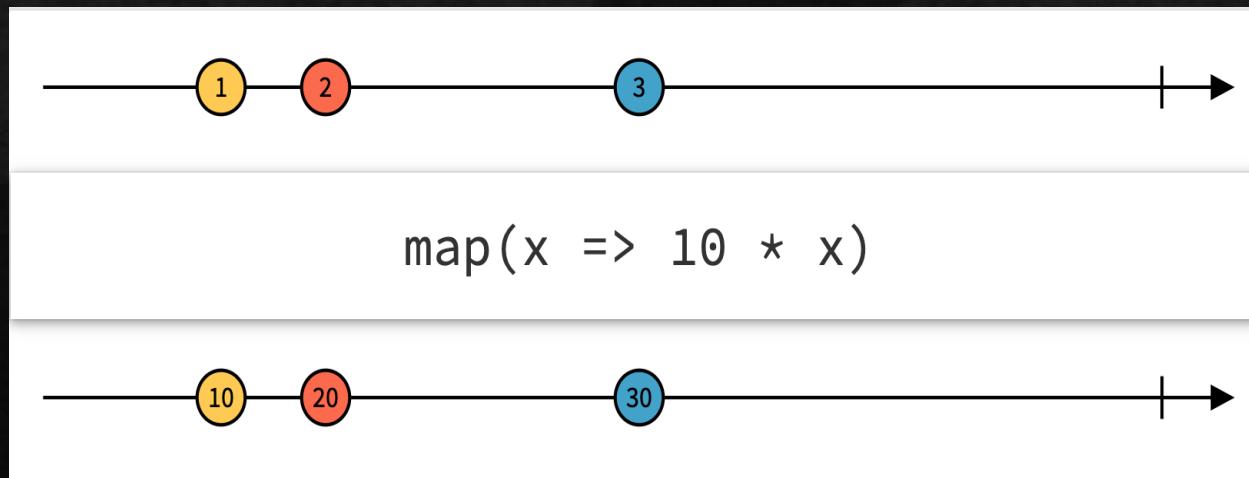
Operator'ler 3 grupta incelenebilir;

- Transforming,
- Filtering,
- Combinations Operators.



# Transforming Operators

Publisher'dan alınan değerlerin farklı değerlere dönüştürülmesinde kullanılır.

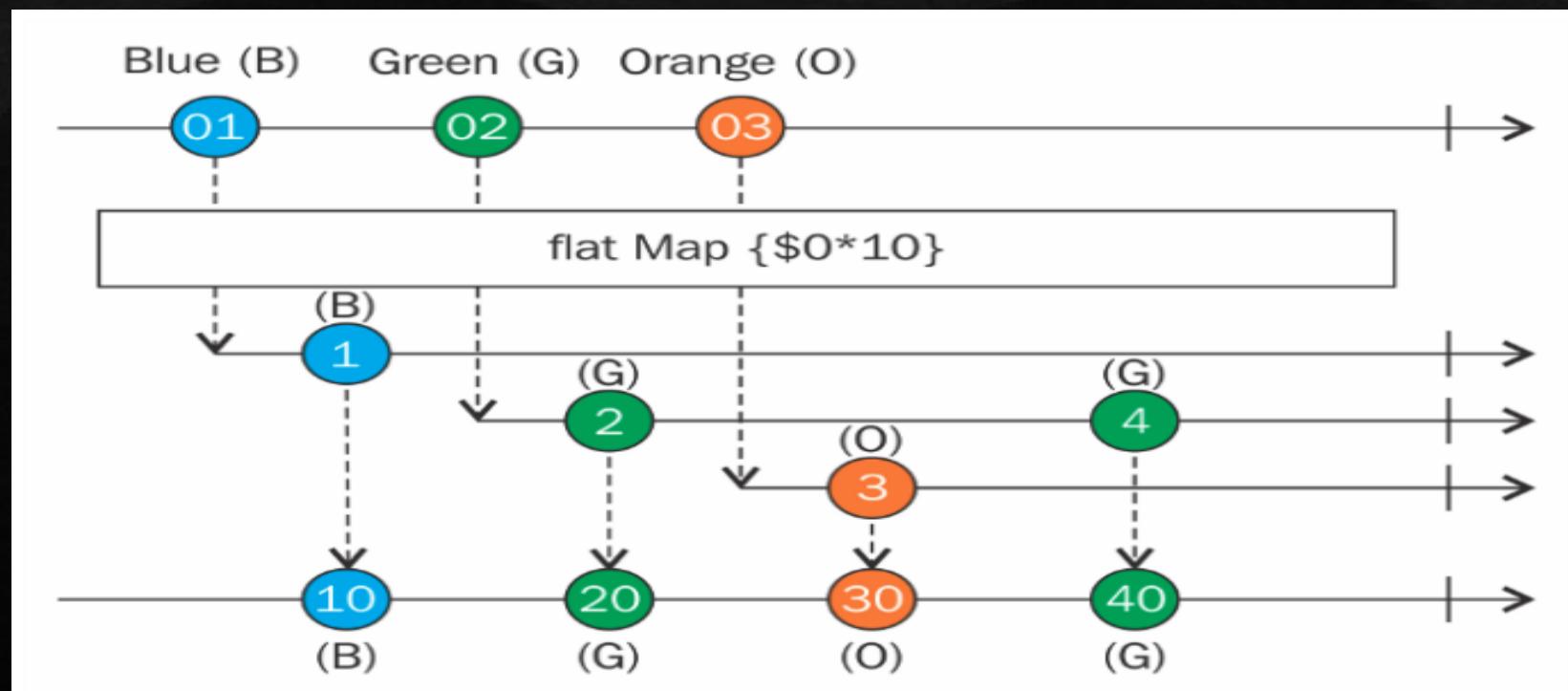


Transforming Operator örnekleri:

- Map: Swift'in standart kütüphanesiyle aynı yapıda, Publisher'lar üzerinde çalışan operator türü.
- Trymap: Herhangi hata durumunda Publisher'ın Failure yapısını otomatik olarak产生的 map türü.
- MapError: Error tiplerini kendi error tipimize dönüştürmemizi sağlayan map türü.

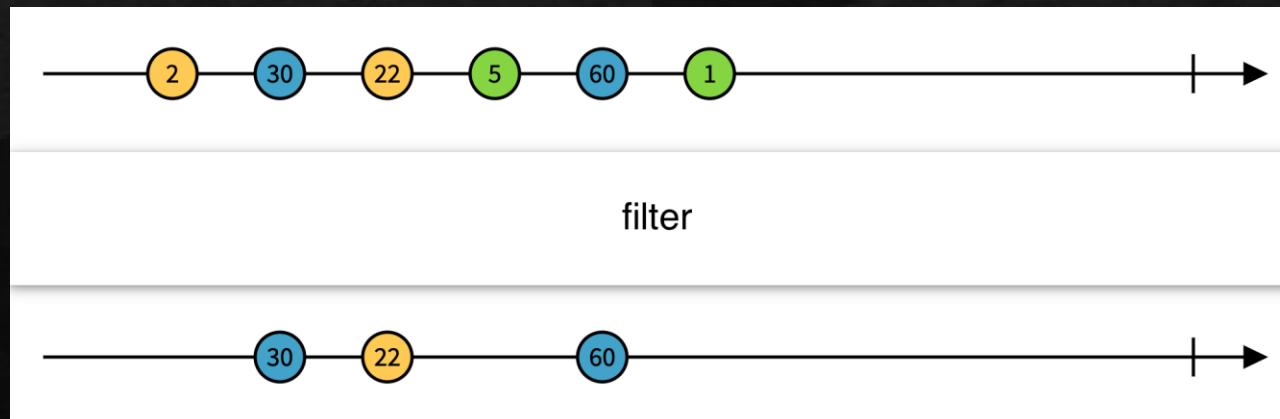
# Transforming Operators - FlatMap

Observable elemanlara sahip bir observable diziyi parçalayarak, her elemanı için bir observable dizi oluşturup, oluşturduğu observable dizileri tek bir observable dizi haline getirir. Daha çok error handling için kullanılır.



# Filtering Operators

Publisher'dan alınan bir dizi değerin içinden özel olarak belli filtreleme işlemleri gerçekleştirmek için kullanılan operator'lerdir.

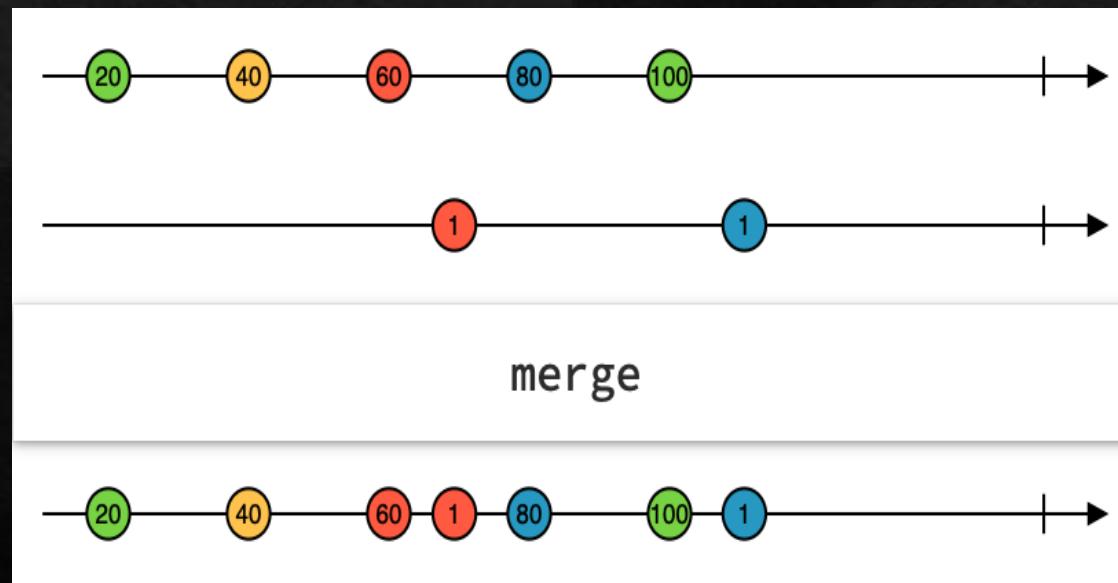


Filtering Operator örnekleri:

- RemoveDuplicates: Yalnızca daha önce eşsi olmayan elemanları döner.
- DropFirst: Parametrik olarak verilen integer değer kadar sayıyı çıkarır ve yeni subsequent döner.

# Combination Operators

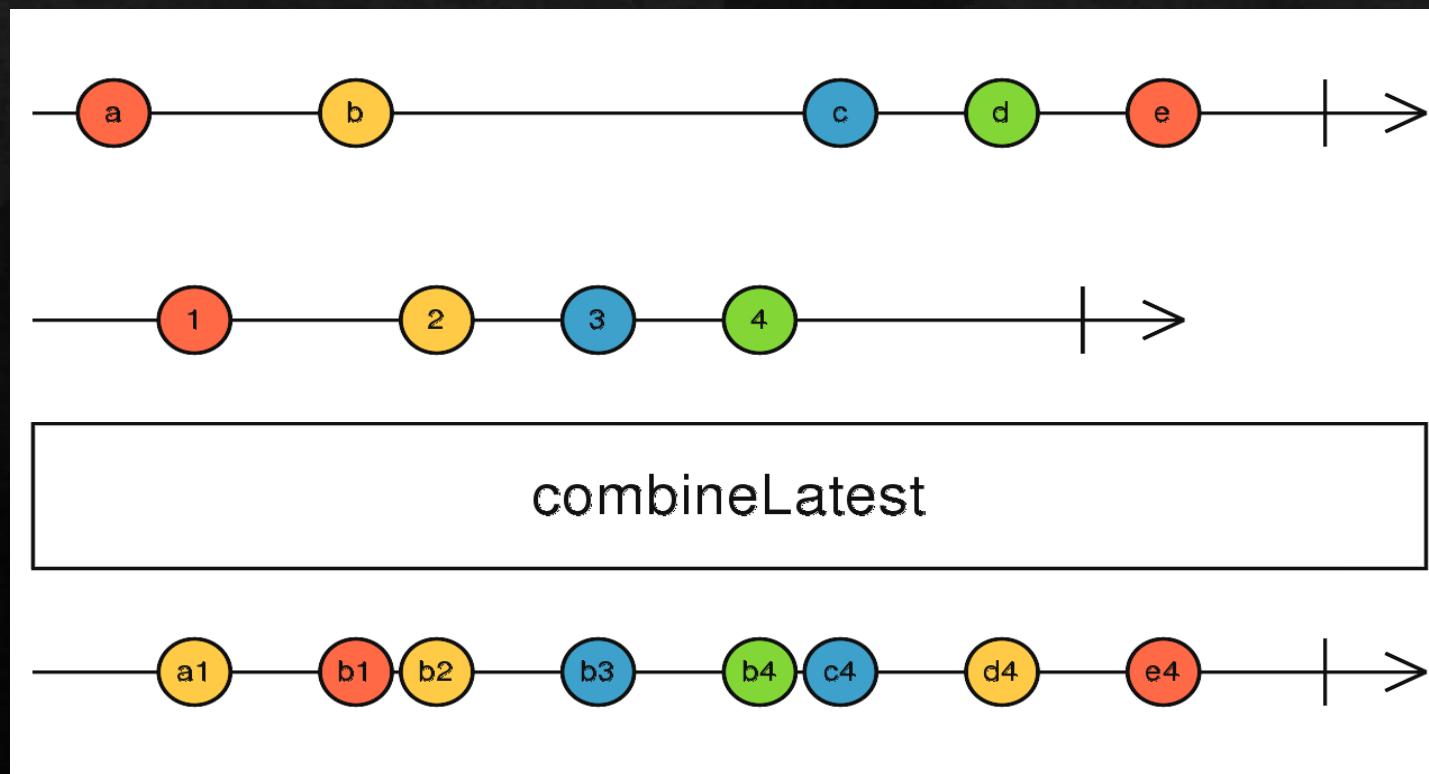
Asenkron dizilerdeki elemanları birleştirmemizi sağlayan operator'lerdir.



Merge: Birden fazla Publisher'ın elemanlarını birleştirerek tek bir Publisher'a dönüştürmemizi sağlar.

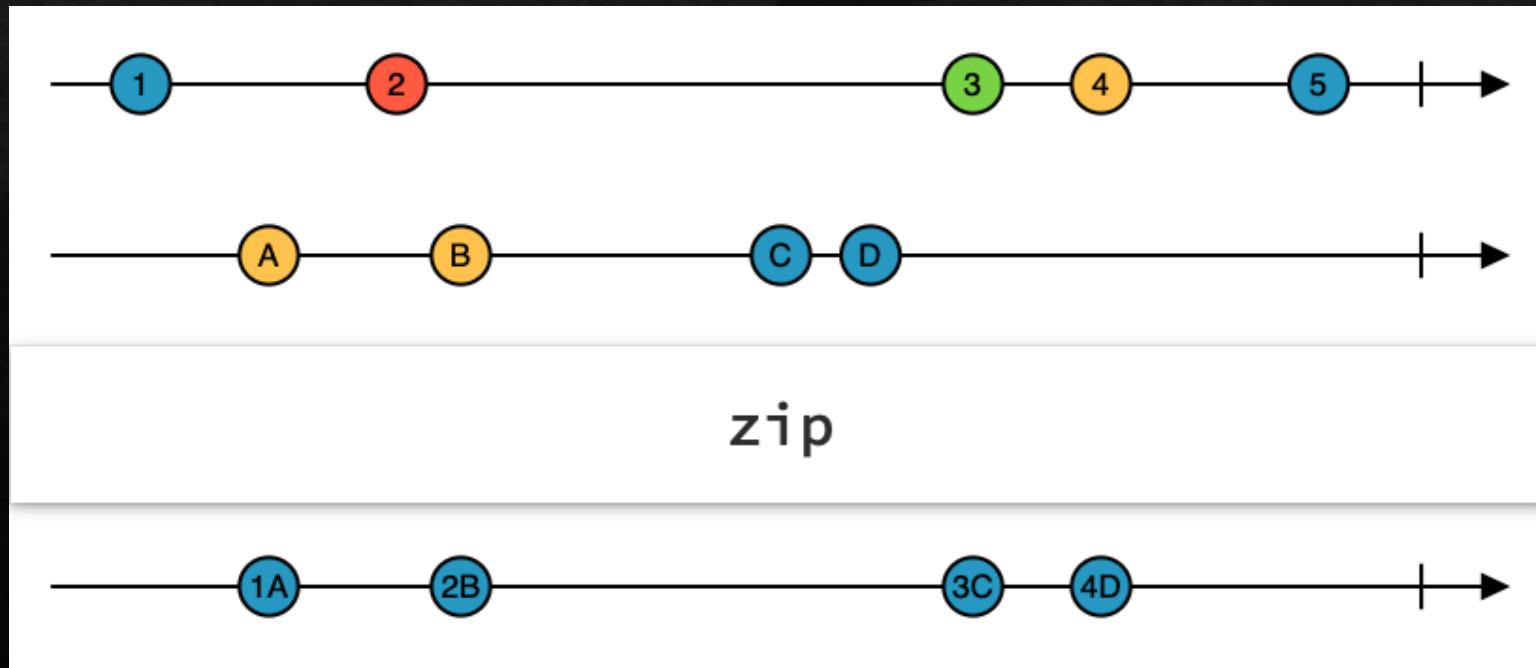
# Combination Operators – CombineLatest

Birden fazla Publisher'ın en son elemanlarını birleştirerek tek bir publisher oluşturur.



# Combination Operators – Zip

CombineLatest Operator'üne benzer bir yapıya sahiptir. Fakat Zip, ziplenecek her Publisher'ın yeni bir değer yayinallyamasını bekler.



catch

append

count

first

min

dropFirst

**map**

abortOnError

**merge**

log

last

allSatisfy

filter

breakpointOnError

handleEvents

mapError

output

breakpoint

**flatMap**

ignoreOutput

max

**drop**

prefix

setFailureType

removeDuplicates

**reduce**

combineLatest

replaceEmpty

prepend

**contains**

switchToLatest

**collect**

compactMap

replaceError

replaceNil

scan

retry

print

zip

# Subjects

Özel Publisher türleridir. Hem Publisher hem de Subscriber gibi davranabilir.

Send metodu sayesinde yeni değerleri manuel olarak göndermemizi sağlar.

Birden fazla Subscriber'a sahip olabilir.

2 tür Subject vardır:

- PassthroughSubject: Subscription'dan sonraki her elemanı iletir.
- CurrentValueSubject: PassthroughSubject'e benzer fakat tanımlanırken başlangıç değeri alır. Ayrıca CurrentValueSubject son değerleri saklayarak yeni subscriberlarına iletir.

# Sample 2 – Operators & Subjects

# Just & Future

Just ve Future Apple tarafından yazılan kullanıma hazır publisher tipleridir.

Just genel olarak error handling ve publisher chain oluşturulurken kullanılır. Tek bir result döner ve sonlanır. Error tipi olarak `<Never>` döner.

Future ise bir closure ile init edilir. Genelde promise diye adlandırılan bir parametre gönderir. Return tipi olarak belirtilen veri tipi (String, bool vb.) veya Error dönebilir. Custom bir fonksiyon çağrılmamızı sağlar ve parametresi sayesinde success ve failure olarak return tipi dönebiliriz.

# @Published Property Wrapper

@Published Property Wrapper'ı sayesinde herhangi bir property'e publisher ekleyebiliriz.

Örnek Kullanımı :

```
@Published var name: String = ""
```

@Published wrapperi sayesinde name stringi ne zaman değişirse event fırlatır ve herhangi bir subscriber tarafından dinlenebilir.

# Memory Management

Publisher'ların döndüğü value'ları kullanmak için sink veya assign kullanılır. Sink ve assign dönüş değeri olarak AnyCancellable tipini döner.

AnyCancellable sayesinde cancel() çağrılabılır ve bu sayede subscription'ı sonlandırmamızı sağlar.

Cancel() çağrılmadan subscription sonlanamayacağı için Publisher'in mevcut class'ı deinit edilse dahi subscription tarafından retain cycle oluşabilir.

# Debugging

Combine frameworkü ile yapılan işlemler çoğu zaman sıralı olmayabiliyor ve debugging işlemleri zorlayıcı oluyor. Bu tür sorunların çözümü için Combine bizlere farklı debugging yöntemleri sunuyor.

Loglama methodları:

- *print()* : Publisher 'ın lifecycle'ını ve her bir akışta gönderilen değerleri loglar.
- *handleEvents()* : Her bir lifecycle event'i için bir closure döner ve kendi loglarını yazmamızı sağlar.

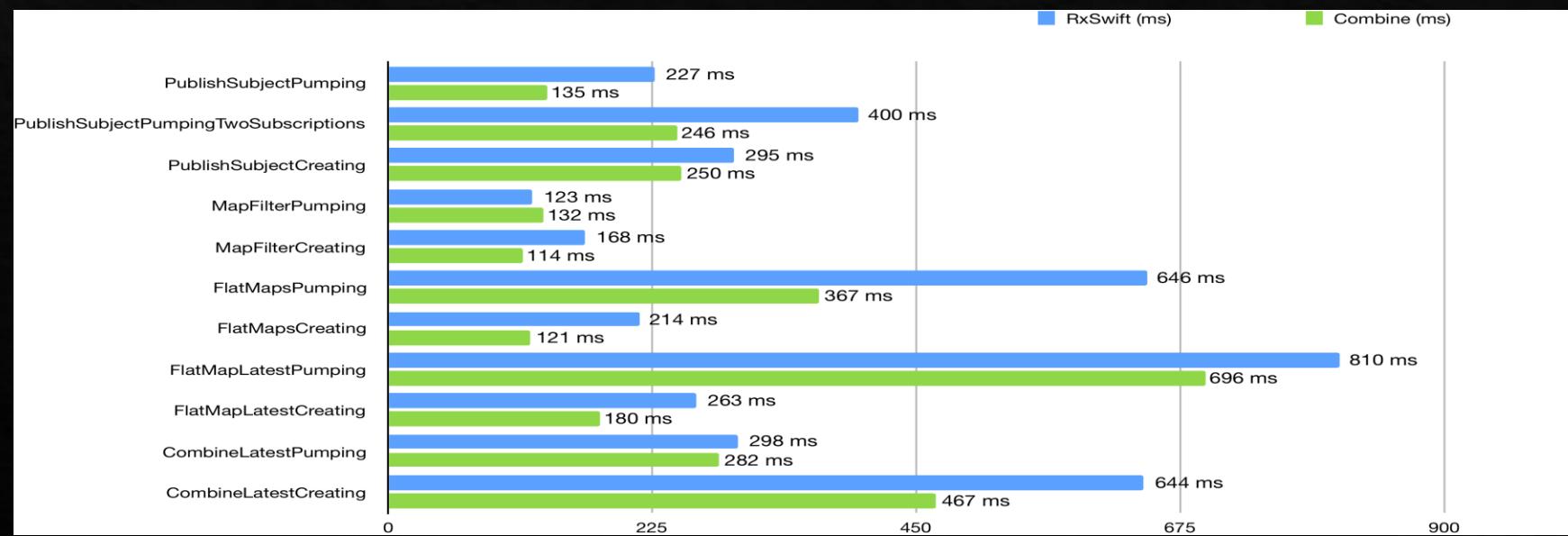
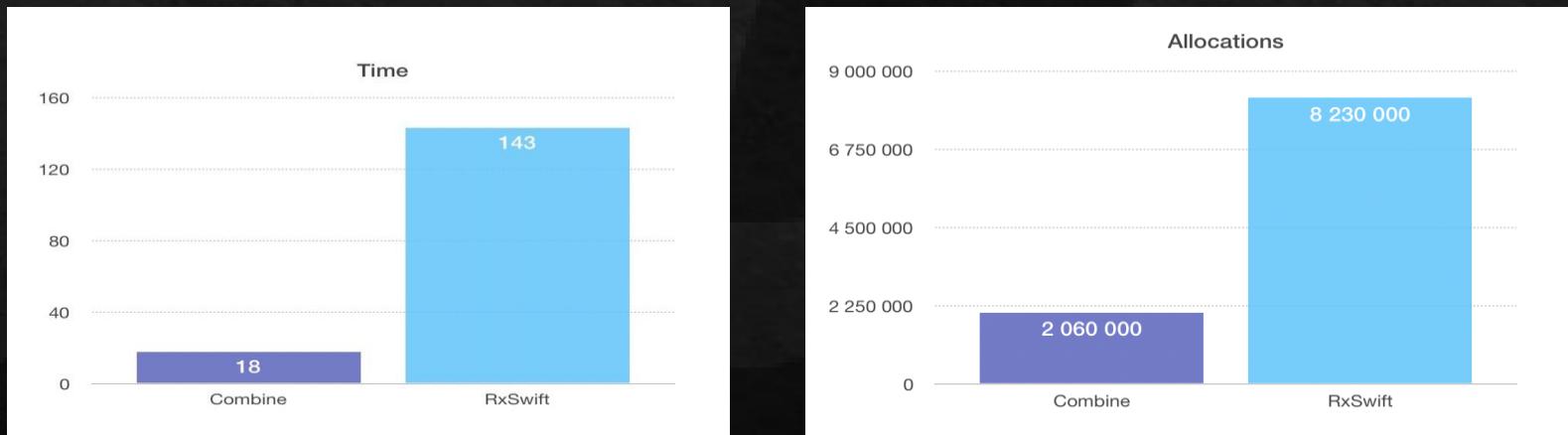
Generating Xcode BreakPoints:

- *breakpointOnError()* : Upstream bir publisher hata fırlattığında debugger'i durdurur.
- *breakpoint()* : Herhangi bir subscription anında, istenildiği zaman debugger'i durdurur.

# Combine vs RxSwift

	RxSwift	Combine
Deployment Target	iOS 8.0+	iOS 13.0+
Platforms supported	iOS, macOS, tvOS, watchOS, Linux	iOS, macOS, tvOS, watchOS, UIKit for Mac <sup>1</sup>
Spec	Reactive Extensions (ReactiveX)	Reactive Streams (+ adjustments)
Framework Consumption	Third-party	First-party (built-in)
Maintained by	Open-Source / Community	Apple
UI Bindings	RxCocoa	SwiftUI <sup>2</sup>

# Combine vs RxSwift



# Using Combine With MVVM

Combine frameworkü hem UIKit hem de SwiftUI için kullanılabilir. Fakat SwiftUI ile hem delegation metodlarına gerek kalmaması ve viewModel bağımlılığı oluşması hem de declarative syntax yapısıyla Combine ile daha uyumludur.

Ayrıca Combine ile birlikte observable yapılar öne çıktığinden, combine ile uygulama geliştirirken kullanılan mimariler içinde en uyumlu MVVM olarak öne çıkıyor.

# Sample 4 – DemoApp