# RxSwift Basics – Day 2

Younghwan Kim

## RxSwift Basics

- Day 1 – Observable, Operator (Filter, Transform, Combine)
- **Day 2 – Subject (flatMap, flatMapFirst, flatMapLatest)**
- Day 3 – Two VCs communications with Subject, RxCocoa (Button)
- Day 4 – Sequential, Merged Observable Calls
- Day 5 – RxCocoa, UI Binding (Button, TextField, Label, TableView)

## Advanced RxSwift

- Day 1 – Protocol-Oriented Programming, Protocol Extension, Associatetype
- Day 2 – Network Call, Generic Enum
- Day 3 – Binding Track Activity (show / hide ‘Loading’ ), Scan Operator
- Day 4 – Adding a Reactive Extension to Custom UI Element,
- 2 Way Binding, Advanced TableView – RxDataSources
- Day 5 – Schedulers (observeOn, subscribeOn),

  Unit Test (RxTest, RxBlocking)

## Subject

**Observable**

**Observer**

**PublishSubject, BehaviorSubject, ReplaySubject**

**PublishRelay, BehaviorRelay (Variable)**

## Subject

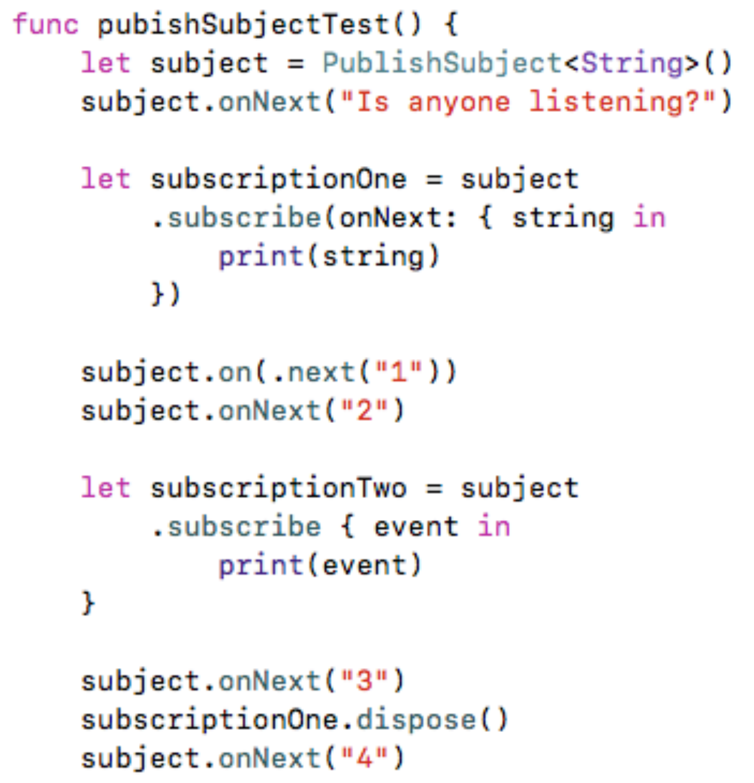
- **PublishSubject**: Starts empty and only emits new elements to subscribers.

- **BehaviorSubject**: Starts with an initial value and replays it or the latest element to new subscribers.

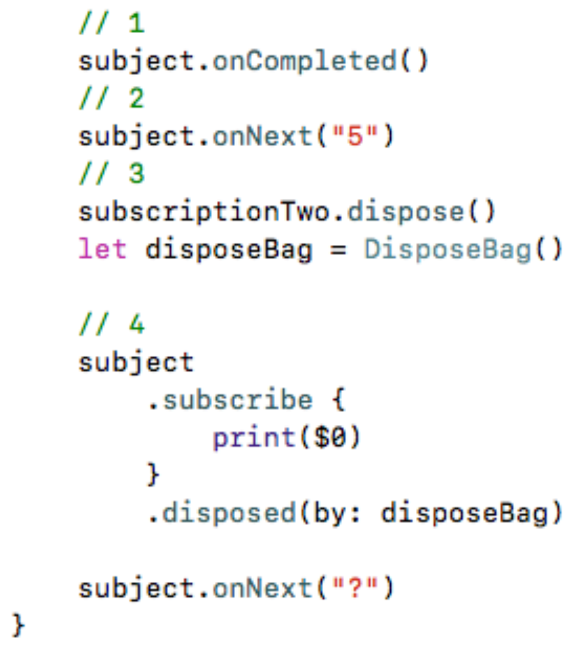
- **ReplaySubject**: Initialized with a buffer size and will maintain a buffer of elements up to that size and replay it to new subscribers.

- **Variable**: Wraps a BehaviorSubject, preserves its current value as state, and replays only the latest/initial value to new subscribers.

## PublishSubject

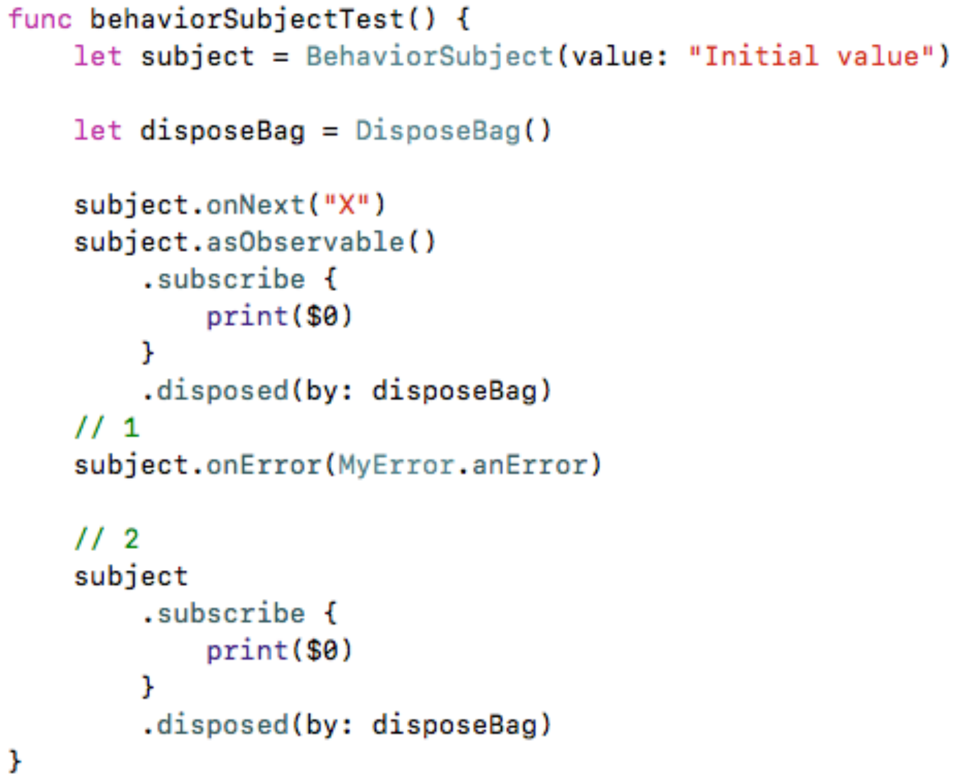


```swift
func pubishSubjectTest() {
    let subject = PublishSubject<String>()
    subject.onNext("Is anyone listening?")

    let subscriptionOne = subject
        .subscribe(onNext: { string in
            print(string)
        })

    subject.on(.next("1"))
    subject.onNext("2")

    let subscriptionTwo = subject
        .subscribe { event in
            print(event)
    }

    subject.onNext("3")
    subscriptionOne.dispose()
    subject.onNext("4")
```



```swift
    // 1
    subject.onCompleted()
    // 2
    subject.onNext("5")
    // 3
    subscriptionTwo.dispose()
    let disposeBag = DisposeBag()

    // 4
    subject
        .subscribe {
            print($0)
        }
        .disposed(by: disposeBag)

    subject.onNext("?")
}
```

## BehaviorSubject



```
func behaviorSubjectTest() {
    let subject = BehaviorSubject(value: "Initial value")

    let disposeBag = DisposeBag()

    subject.onNext("X")
    subject.asObservable()
        .subscribe {
            print($0)
        }
        .disposed(by: disposeBag)
    // 1
    subject.onError(MyError.anError)

    // 2
    subject
        .subscribe {
            print($0)
        }
        .disposed(by: disposeBag)
}
```

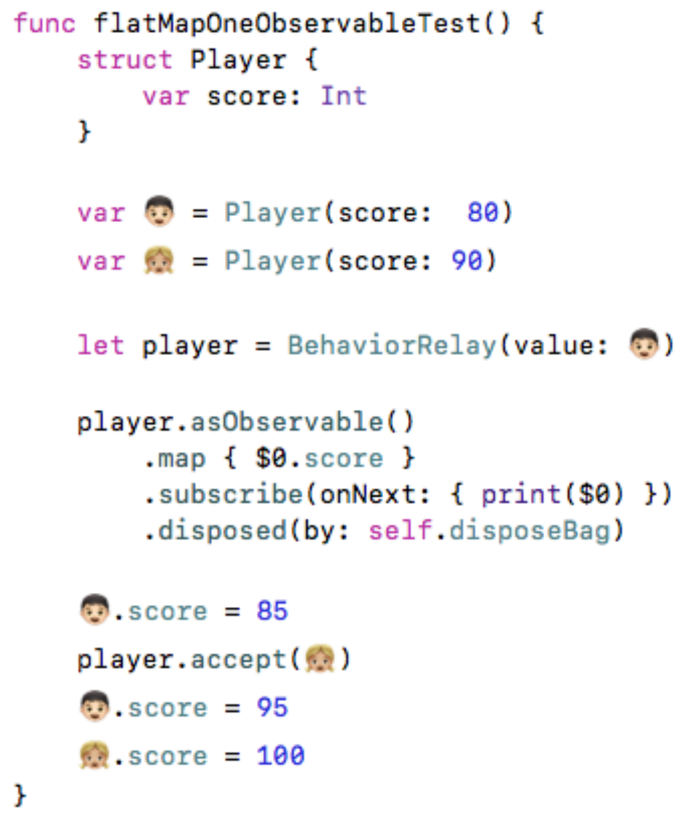
# ReplaySubject

```swift
func replaySubjectTest() {
    // 1
    let subject = ReplaySubject<String>.create(bufferSize: 2)

    let disposeBag = DisposeBag()

    // 2
    subject.onNext("1")
    subject.onNext("2")
    subject.onNext("3")

    // 3
    subject
        .subscribe {
            print($0)
        }
        .disposed(by: disposeBag)

    subject
        .subscribe {
            print($0)
        }
        .disposed(by: disposeBag)

    subject.onNext("4")
    subject.onError(MyError.anError)
    subject.dispose()

    subject
        .subscribe {
            print($0)
        }
        .disposed(by: disposeBag)
}
```

## Lab - 1

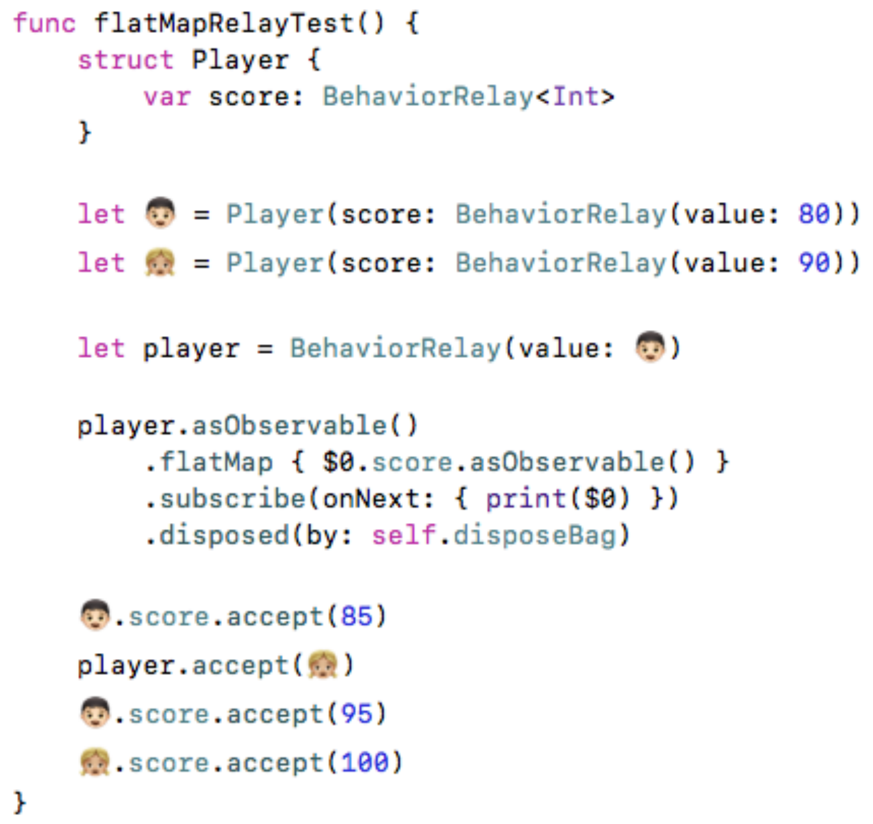
BehaviorSubject, PublishSubject, ReplaySubject

# flatMap, flatMapFirst, flatMapLatest - 1



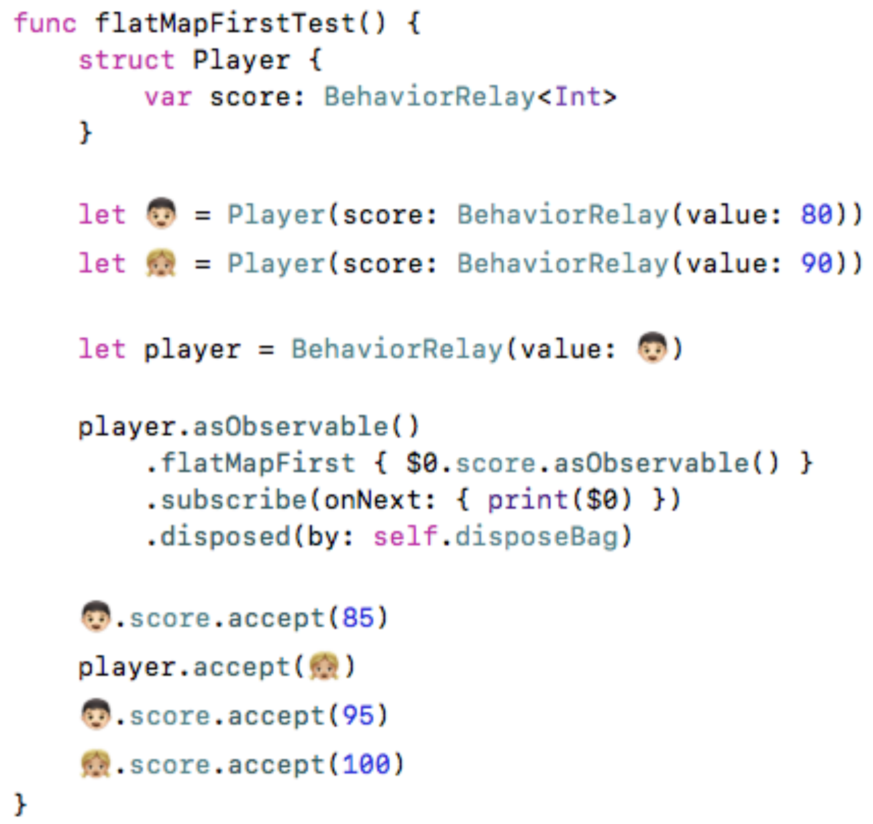
```
func flatMapOneObservableTest() {
    struct Player {
        var score: Int
    }

    var 👦🏻 = Player(score:  80)
    var 👧🏼 = Player(score: 90)

    let player = BehaviorRelay(value: 👦🏻)

    player.asObservable()
        .map { $0.score }
        .subscribe(onNext: { print($0) })
        .disposed(by: self.disposeBag)

    👦🏻.score = 85
    player.accept(👧🏼)
    👦🏻.score = 95
    👧🏼.score = 100
}
```

## flatMap, flatMapFirst, flatMapLatest - 2



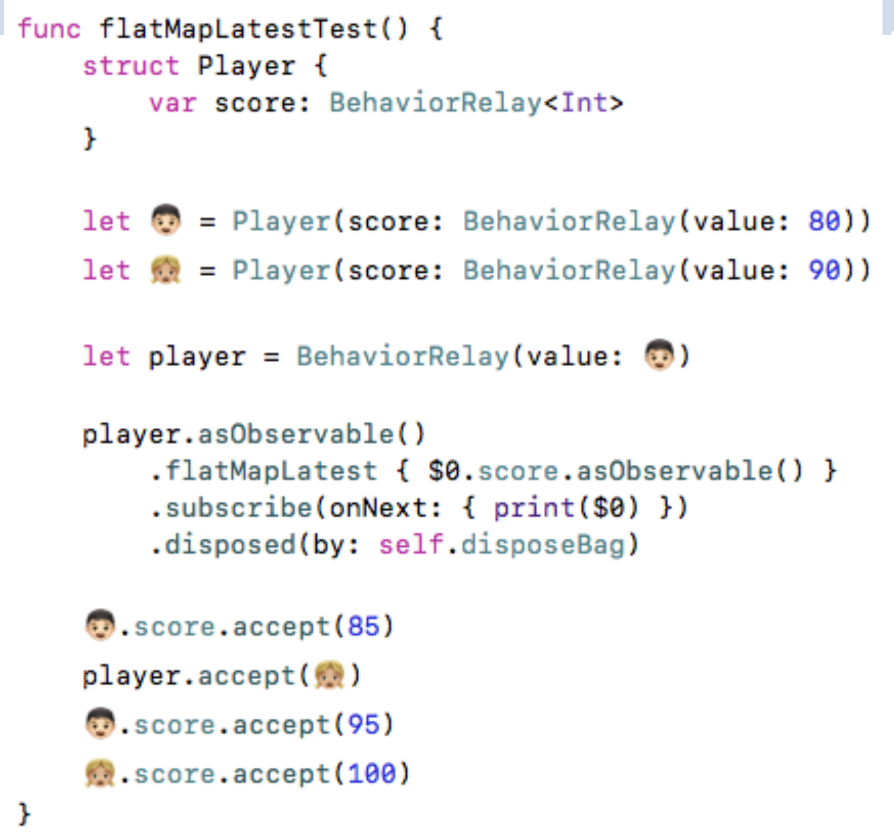
```swift
func flatMapRelayTest() {
    struct Player {
        var score: BehaviorRelay<Int>
    }

    let 👦🏻 = Player(score: BehaviorRelay(value: 80))
    let 👧🏼 = Player(score: BehaviorRelay(value: 90))

    let player = BehaviorRelay(value: 👦🏻)

    player.asObservable()
        .flatMap { $0.score.asObservable() }
        .subscribe(onNext: { print($0) })
        .disposed(by: self.disposeBag)

    👦🏻.score.accept(85)
    player.accept(👧🏼)
    👦🏻.score.accept(95)
    👧🏼.score.accept(100)
}
```

## flatMap, flatMapFirst, flatMapLatest - 3



```
func flatMapFirstTest() {
    struct Player {
        var score: BehaviorRelay<Int>
    }

    let 👦🏻 = Player(score: BehaviorRelay(value: 80))
    let 👧🏼 = Player(score: BehaviorRelay(value: 90))

    let player = BehaviorRelay(value: 👦🏻)

    player.asObservable()
        .flatMapFirst { $0.score.asObservable() }
        .subscribe(onNext: { print($0) })
        .disposed(by: self.disposeBag)

    👦🏻.score.accept(85)
    player.accept(👧🏼)
    👦🏻.score.accept(95)
    👧🏼.score.accept(100)
}
```

## flatMap, flatMapFirst, flatMapLatest - 4



```swift
func flatMapLatestTest() {
    struct Player {
        var score: BehaviorRelay<Int>
    }

    let 👦🏻 = Player(score: BehaviorRelay(value: 80))
    let 👧🏼 = Player(score: BehaviorRelay(value: 90))

    let player = BehaviorRelay(value: 👦🏻)

    player.asObservable()
        .flatMapLatest { $0.score.asObservable() }
        .subscribe(onNext: { print($0) })
        .disposed(by: self.disposeBag)

    👦🏻.score.accept(85)
    player.accept(👧🏼)
    👦🏻.score.accept(95)
    👧🏼.score.accept(100)
}
```

## Lab -2

flatMap, flatMapFirst, flatMapLatest

## BehaviorRelay

```
import RxSwift

/// BehaviorRelay is a wrapper for `BehaviorSubject`.
///
/// Unlike `BehaviorSubject` it can't terminate with error or completed.
public final class BehaviorRelay<Element>: ObservableType {
    public typealias E = Element

    private let _subject: BehaviorSubject<Element>

    // Accepts `event` and emits it to subscribers
    public func accept(_ event: Element) {
        _subject.onNext(event)
    }

    /// Current value of behavior subject
    public var value: Element {
        // this try! is ok because subject can't error out or be disposed
        return try! _subject.value()
    }
```

# BehaviorRelay

```
/// Initializes variable with initial value.
    public init(value: Element) {
        _subject = BehaviorSubject(value: value)
    }

    /// Subscribes observer
    public func subscribe<O: ObserverType>(_ observer: O) -> Disposable where O.E == E {
        return _subject.subscribe(observer)
    }

    /// - returns: Canonical interface for push style sequence
    public func asObservable() -> Observable<Element> {
        return _subject.asObservable()
    }
}
```