Thứ sáu 14/01/2022

Combining Operators

I. Định nghĩa

- Là các toán tử mà hoạt động với nhiều observable khác nhau để tạo ra 1 observable chung.

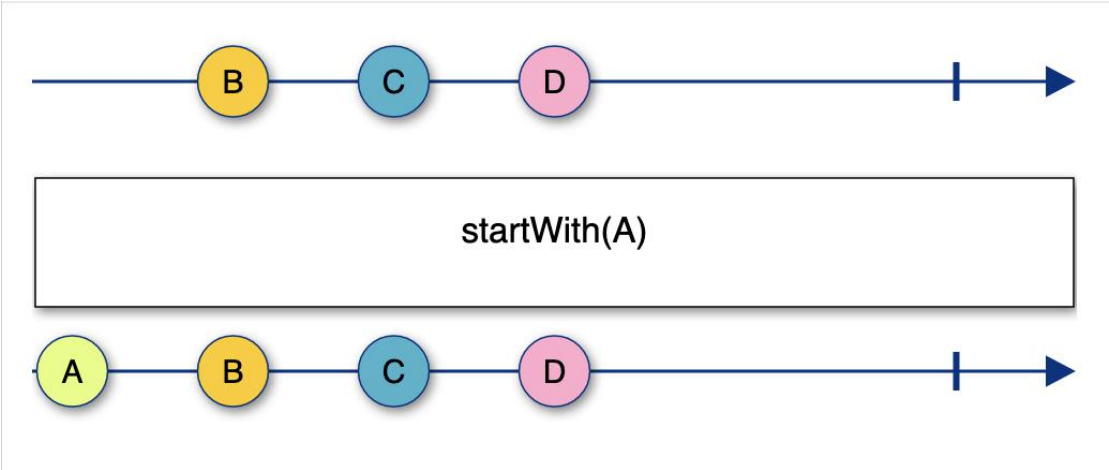
II. Prefixing & concatenating

1. startWith
    - Toán tử này sử dụng để thực hiện tạo 1 observable mới với 1 element hay sequence đằng trước sequence cũ.
    - Giá trị được thêm sẽ được emit trước khi mà emit các giá trị bên trong observable.
    - Giá trị được thêm phải cùng kiểu với các giá trị bên trong sequence.
    - Ví dụ:

```
func startWith(){
    let observable = Observable.of( "B", "C", "D", "E").startWith("A")
    //A will be emitted before the observable sequence

    observable.subscribe(onNext: { value in
        print(value)
    })
    .disposed(by: bag)
}
```

    - output:

- Sơ đồ marble:



2. Observable.concat
   - Khi dùng toán tử static concat của class Observable thì sẽ thực hiện gộp các element được emit bởi các observable lại thành 1 observable có chứa các element đó.
   - Concat hoạt động giống như startWith nhưng để sequence của observable kế tiếp đằng sau sequence của observable trước đó.
   - Do đó nên khi sử dụng ta không phải lo phần tử của 2 sequence bị xen kẽ với nhau.

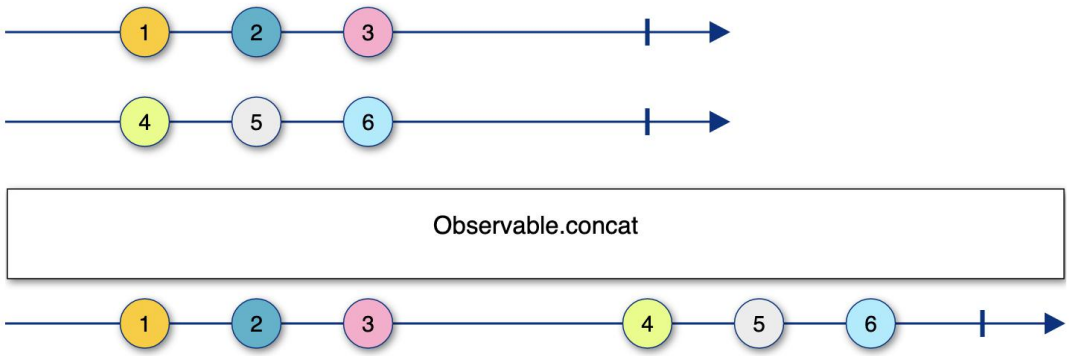
- Ví dụ:

```
func concatObservable(){
    let first = Observable.of(1, 2, 3)
    let second = Observable.of(4, 5, 6)

    let observable = Observable.concat([first, second])
    /*
     concat now will create a new observable using
     the first sequence then append the second sequence
     behind it.
     */
    observable.subscribe(onNext: { value in
        print(value)
    })
    .disposed(by: bag)
}
```

- Output:

```
1
2
3
4
5
6
```

- Sơ đồ marble:



3. concat

- Khác với toán tử static thì toán tử concat này

thay vì tạo 1 observable mới bằng cách nối 2 toán tử lại với nhau thì toán tử này sử dụng với 1 đối tượng Observable mà thực hiện nối sequence của observable khác vào.

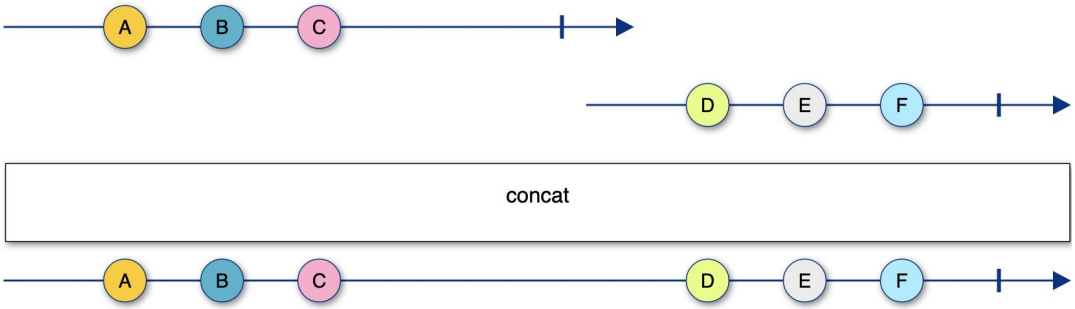
- Chú ý: concat sẽ chờ cho đến khi observable trước đó hoàn thành để nối observable kế tiếp vào.
- Ví dụ:

```
func concat(){
    let first = Observable.of("A" , "B", "C")
    let second = Observable.of("D" , "E", "F")

    let observable = first.concat(second)
    /*
     observable now will wait for when
     first finish to connect second sequence
     */
    observable.subscribe(onNext: { value in
        print(value)
    })
    .disposed(by: bag)
}
```

- Output:

```
A
B
C
D
E
F
```

- Sơ đồ marble:



4. concatMap

- Hoạt động giống như flatMap:
  + Biến đổi kiểu element của từng observable phát ra.
  + Tạo mới 1 observable chứa các phần tử mà được emit bởi các observable trong observable gốc
- Điểm khác biệt:
  + Đảm bảo mỗi observable emit xong mới subscribe cho observable kế tiếp cho đảm bảo trình tự phát của từng observable không bị xen kẽ lẫn nhau.
- Setup:

```
let cities = [
    "MienBac":Observable.of("HaNoi", "HaiPhong"),
    "MienTrung":Observable.of("Hue", "DaNang"),
    "MienNam":Observable.of("HoChiMinh", "CanTho")
]
```

- Sử dụng concatMap

```
let observable = Observable.of("MienBac", "MienTrung", "MienNam")
    .concatMap { (name) in      //(*)
        cities[name] ?? .empty()
    }

observable
    .subscribe(onNext: { print($0) })
    .disposed(by: bag)
```

+ Ở (*) ta thực hiện biến đổi các phần tử bên trong observable thành các Observable lấy từ bên trong dictionary.
+ Kế tiếp sẽ tạo 1 observable sequence mới sử dụng các phần tử mà các Observable bên trong observable emit ra
+ Các phần tử bên trong observable sequence mới sẽ có thứ tự theo đúng thứ tự mà từng observable emit.
+ Chú ý: .empty() dùng để trả về 1 observable kiểu empty nếu dictionary trả về nil.

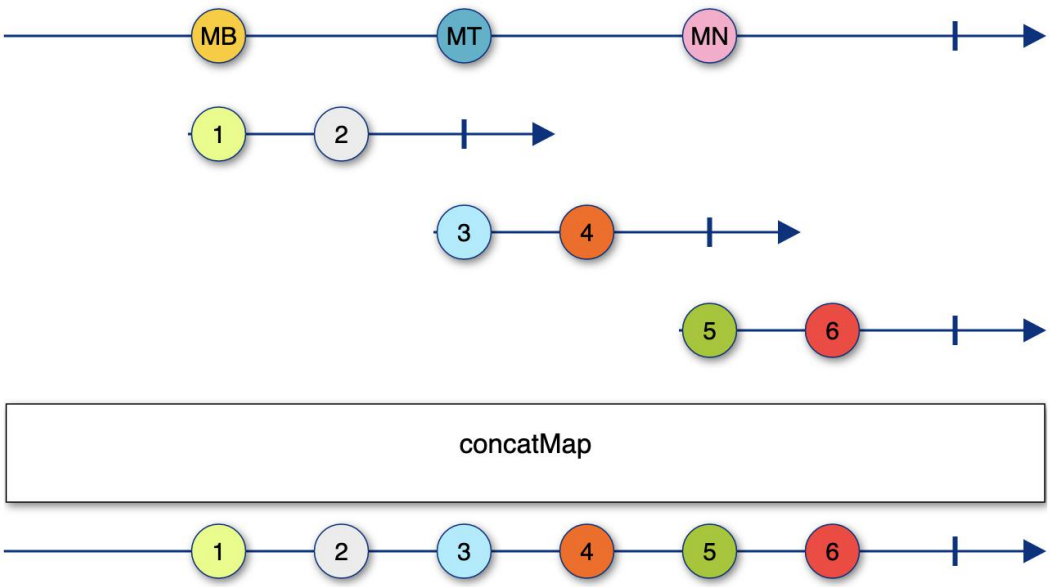
- Output:

```
HaNoi
HaiPhong
Hue
DaNang
HoChiMinh
CanTho
```

- `Thứ tự:
  + MienBac -> HaNoi -> HaiPhong

+ MienTrung -> Hue -> DaNang
+ MienNam -> HoChiMinh -> CanTho

- Sơ đồ marble:



- Chú thích:
  + MB, MT, MN: MienBac, MienTrung & MienNam.
  + 1, 2: HaNoi & HaiPhong.
  + 3, 4: Hue & DaNang.
  + 5, 6: HoChiMinh & CanTho.

III. Merging

- Dùng để kết hợp nhiều observable lại thành 1 observable bằng cách liên kết các element được emit bởi các observable đó.
- Toán tử này khác với toán tử concat ở việc

merge nhận element mà không quan tâm đến thứ tự emit của các observable .

- Do đó nên dẫn đến hiện tượng các element của 1 sequence sẽ đan xen với 1 hay nhiều element của sequence khác.
- Setup:

```
let chu = PublishSubject<String>()
let so = PublishSubject<String>()

let source = Observable.of(chu.asObserver(), so.asObserver())
let observable = source.merge()

observable.subscribe(onNext: { value in
    print(value)
})
.disposed(by: bag)
```

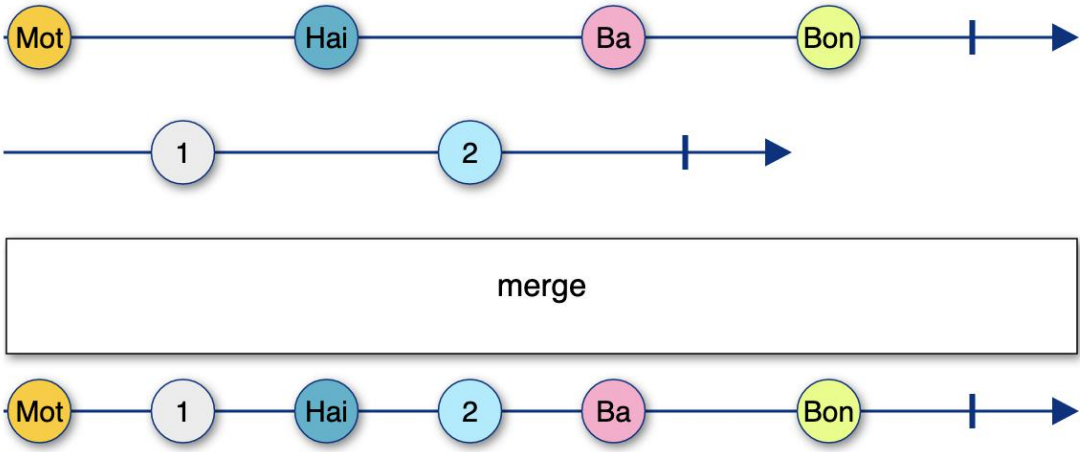
- Trình tự thực hiện:

```
chu.onNext("Mot")
so.onNext("1")
chu.onNext("Hai")
so.onNext("2")
chu.onNext("Ba")
//completed
so.onCompleted()

so.onNext("3")
chu.onNext("Bon")
//completed
chu.onCompleted()
```

- Output:

```
Mot
1
Hai
2
Ba
Bon
```

- Sơ đồ marble:



IV. Combining elements

1. combineLatest

- Toán tử này có công dụng dùng để tạo 1 observable mà có sequence là 1 tuple chứa 2 giá trị mới nhất của 2 observable.
- Mỗi khi mà 1 trong 2 observable emit thì sẽ lấy giá trị mới nhất của mỗi observable
- Ví dụ:

```swift
let chu = PublishSubject<String>()
let so = PublishSubject<String>()

let observable = Observable.combineLatest(chu, so)

observable
    .subscribe(onNext: { value in
        print(value)
    })
    .disposed(by: bag)
```

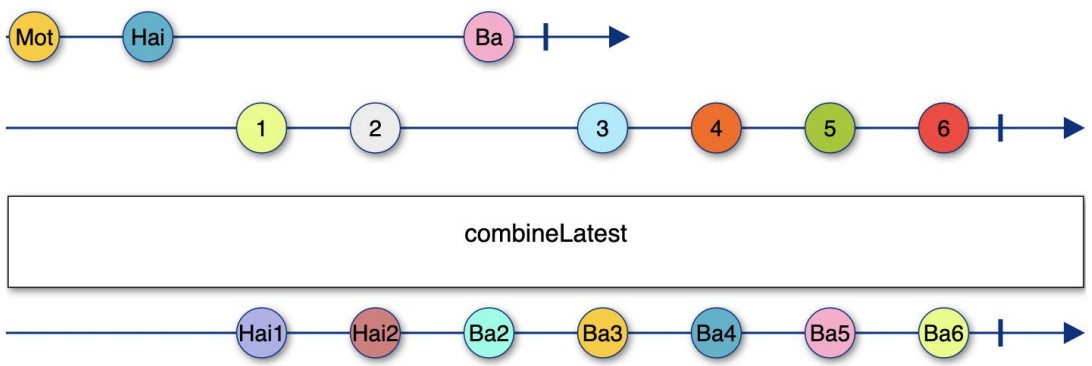
- Trình tự:

```swift
//start emitting
chu.onNext("Mot")
chu.onNext("Hai")
so.onNext("1")
so.onNext("2")

chu.onNext("Ba")
so.onNext("3")

//completed
chu.onCompleted()

chu.onNext("Bon")
so.onNext("4")
so.onNext("5")
so.onNext("6")

//completed
so.onCompleted()
```

- Output:

```
("Hai", "1")
("Hai", "2")
("Ba", "2")
("Ba", "3")
("Ba", "4")
("Ba", "5")
("Ba", "6")
```

- Chú ý:
  - + Để kết thúc observable tạo bởi combineLatest phải dùng .dispose().
  - + Nếu 1 trong 2 observable mà kết thúc mà observable kia vẫn còn emit thì combineLatest sẽ lấy giá trị cuối cùng của observable kia trước khi mà nó emit .completed.
  - + Do chu kết thúc ở Ba nên combineLatest sẽ lấy giá trị Ba.
  - + Giá trị Mot bị loại bỏ do bên so chưa bắt đầu emit.
- Sơ đồ marble:



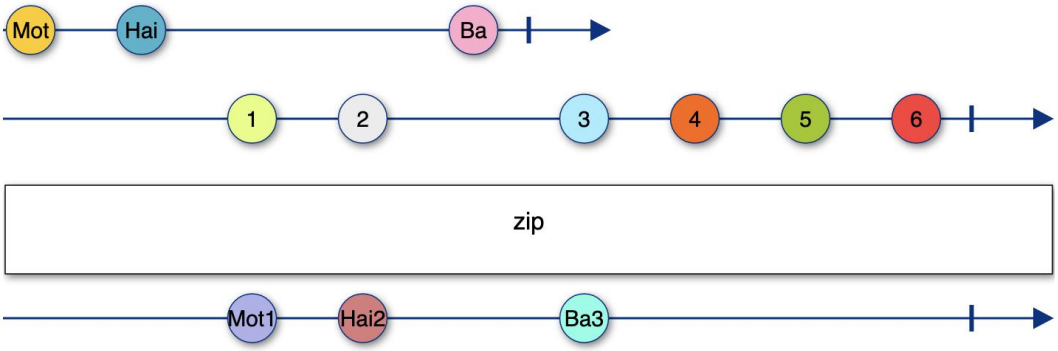
2. combineLatest(_:_:resultSelector:)
    - Giống như toán tử map mà sẽ thực hiện thay đổi kiểu trả về cho subscriber của observable tạo ra bởi combineLatest.
3. zip
    - Hoạt động giống như combineLatest nhưng chỉ lấy các element dựa theo thứ tự emit của từng observable
    - Setup:

```
let observable = Observable
    .zip(chu, so) { (chu, so) in
        "\(chu):\(so)"
    }
```

    - Output:

```
Mot:1
Hai:2
Ba:3
```

        + Không nhận 4, 5, 6, do không có phần tử nào bên sequence của chu có vị trí bằng với các element đó.
    - Sơ đồ marble:



V. Trigger

1. Định nghĩa: Là các observable mà hoạt động dựa theo các observable khác
2. withLatestFrom
   - Là trigger mà có tham số là dữ liệu mới nhất của observable khác.
   - Setup:

```
let button = PublishSubject<Void>()
let textField = PublishSubject<String>()

let observable = button.withLatestFrom(textField)

_ = observable.subscribe(onNext: { value in
    print(value)
})
```

   - Trình tự:

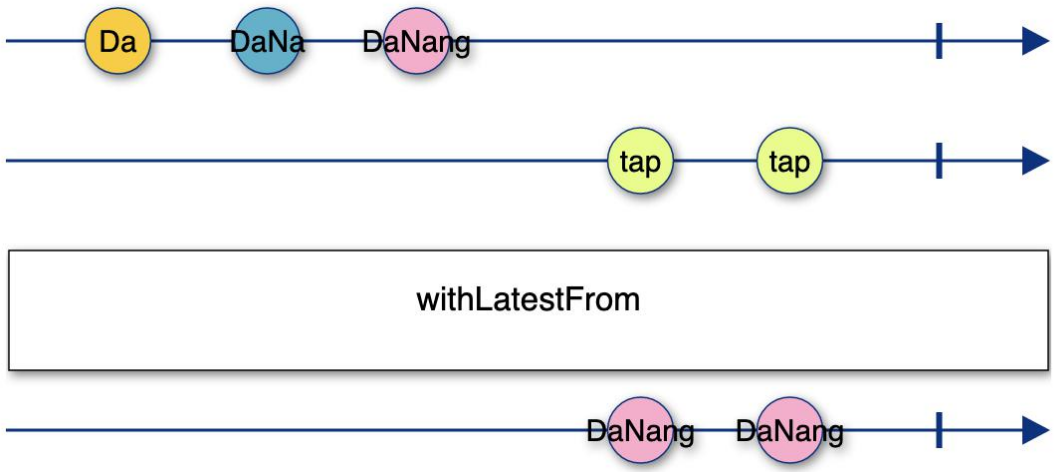
```
textField.onNext("Da")
textField.onNext("DaNa")
textField.onNext("DaNang")

button.onNext(())
button.onNext(())
```

- Output:

```
DaNang
DaNang
```

+ Do trigger của observable button nên mỗi lần mà button thực hiện emit thì observable sẽ bắt đầu emit giá trị mới nhất của textField.

- Marble diagram:



3. sample

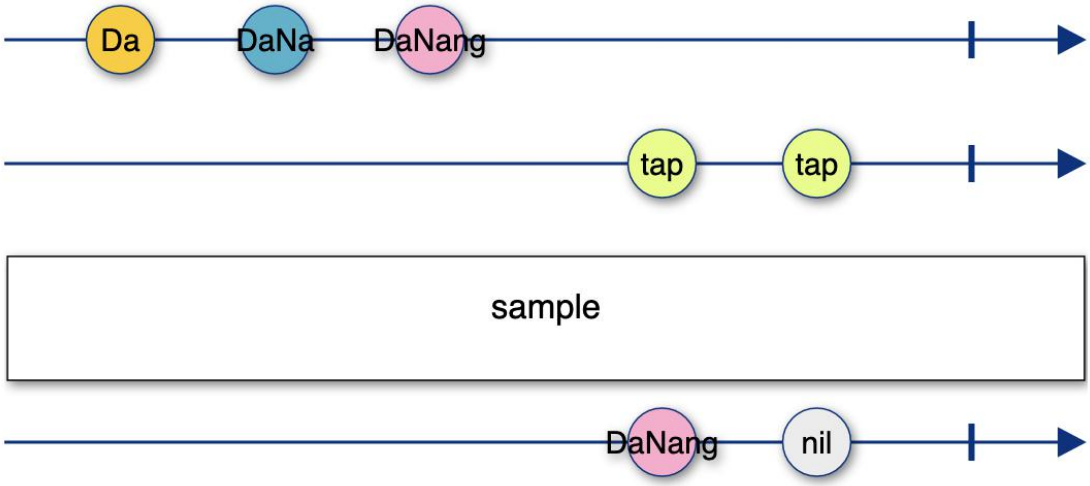
- Cũng là 1 trigger nhưng lấy tham số là 1 observable.
- Nếu như observable mà được trigger không có emit giá trị mới thì sẽ emit nil.
- Có thể cho giá trị mặc định để emit sử dụng .sample(_:defaultValue:)
- Setup:

```
let observable = textField.sample(button,defaultValue: nil)
```

- Output:

```
DaNang
```

- Sơ đồ marble:



VI. Switches

1. Định nghĩa:
   - Các switches dùng để tạo 1 observable từ nhiều observable với sự lựa chọn observable nào là nguồn phát dữ liệu cho các subscriber.
2. Amb
   - Sử dụng để chọn 1 observable để toàn quyền emit dựa theo observable nào emit đầu tiên.
   - Toán tử này chỉ lựa chọn 1 observable làm nguồn phát.

- Setup:

```
let chu = PublishSubject<String>()
let so = PublishSubject<String>()

let observable = chu.amb(so)

observable
    .subscribe(onNext: { value in
        print(value)
    })
    .disposed(by: bag)
```

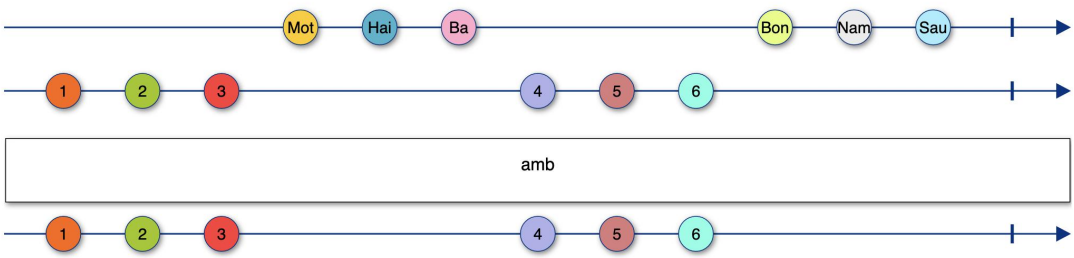
- Trình tự:

```
so.onNext("1")
so.onNext("2")
so.onNext("3")

chu.onNext("Mot")
chu.onNext("Hai")
chu.onNext("Ba")

so.onNext("4")
so.onNext("5")
so.onNext("6")

chu.onNext("Bon")
chu.onNext("Nam")
chu.onNext("Sau")
```

- Output:

```
1
2
3
4
5
6
```

- Sơ đồ marble:



3. switchLatest

- Là toán tử tạo ra 1 observable sequence mà emit các observable.
- Observable tạo ra đang emit observable nào thì subscriber sẽ nhận được giá trị mà observable đó đang emit ra.
- Để kết thúc switchLatest ta phải dùng .dispose() do toán tử này không kết thúc kể cả khi các tất cả các observable element đều đã onCompleted.
- Setup:

```
typealias StringStream = PublishSubject<String>

let chu = StringStream()
let so = StringStream()
let dau = StringStream()

let observable = PublishSubject<StringStream>() //original observable

observable
    .switchLatest()
    .subscribe(onNext: { print($0) },
               onCompleted: {print("completed")})
    .disposed(by: bag)

observable.onNext(so)
```

- Trình tự:

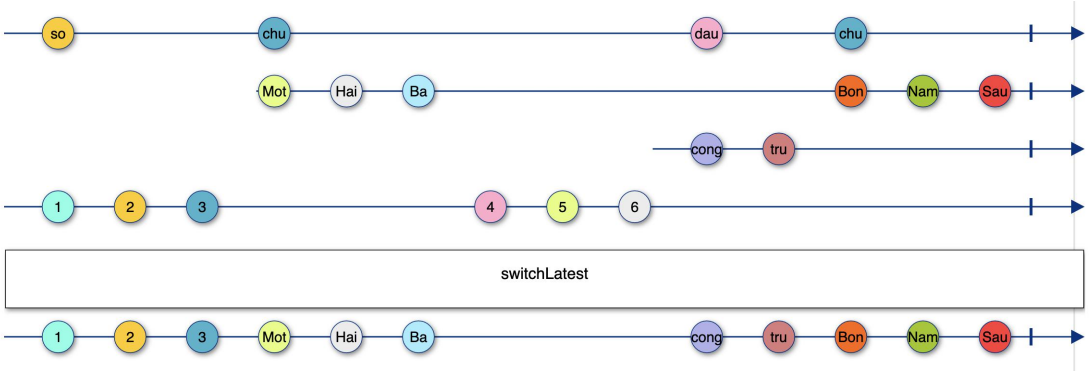
-

```
observable.onNext(so)

so.onNext("1")
so.onNext("2")
so.onNext("3")

observable.onNext(chu)

chu.onNext("Mot")
chu.onNext("Hai")
chu.onNext("Ba")

so.onNext("4")
so.onNext("5")
so.onNext("6")

observable.onNext(dau)

dau.onNext("+")
dau.onNext("-")

observable.onNext(chu)

chu.onNext("Bon")
chu.onNext("Nam")
chu.onNext("Sau")

observable.dispose() //call to terminate
```

- Output

```
1
2
3
Mot
Hai
Ba
+
-
Bon
Nam
Sau
```

- Sơ đồ marble:



VII. Combining elements within a sequence

1. Định nghĩa:
   - Là các toán tử thực hiện kết hợp các phần tử trong 1 sequence
   - Các toán tử này thường được chạy đệ quy do không biết trước khi nào mà observable sẽ ngừng emit.
2. reduce
   - Toán tử này có chức năng thực hiện đưa 1 biến tham số (seed) và 1 closures (accumulator) vào bên trong.

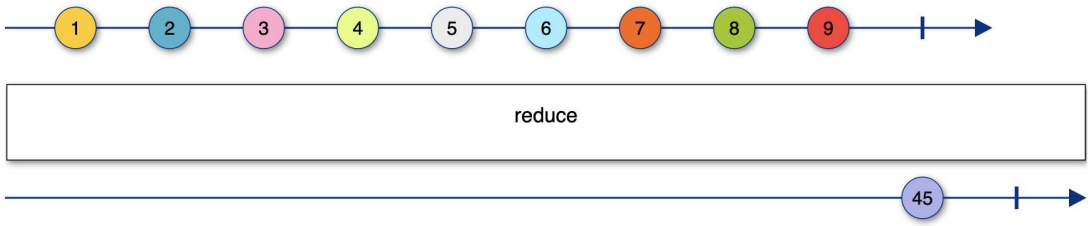
- Toán tử sẽ thực hiện công việc tính toán sử dụng accumulator và gán kết quả vào biến seed.
- Toán tử sẽ thực hiện vòng lặp này cho đến khi mà observable emit onComplete. Khi đó observable sẽ emit giá trị của seed cho subscriber.
- Setup:

```
func reduce() {
    let source = Observable.of( 1, 2, 3, 4, 5, 6, 7, 8, 9)
    let observable = source.reduce(0, accumulator: +)

    _ = observable.subscribe(onNext: { print($0) })
}
```

- Output:

```
45
```

- Marble diagram



3. scan(_:accumulator:)
   - Hoạt động giống như toán tử reduce nhưng thay vì emit seed khi mà observable emit onCompleted thì sẽ thực hiện emit seed kết

quả mỗi lần thực hiện tính toán xong.

- Cách này hoạt động giống như 1 dòng for..in cho RxSwift.
- Sửa lại:

```
let observable = source.scan(0) { summary, newValue in
  return summary + newValue
}
```

- Output:

```
1
3
6
10
15
21
28
36
45
```

- Sơ đồ marble: