Operators

Study - Rx

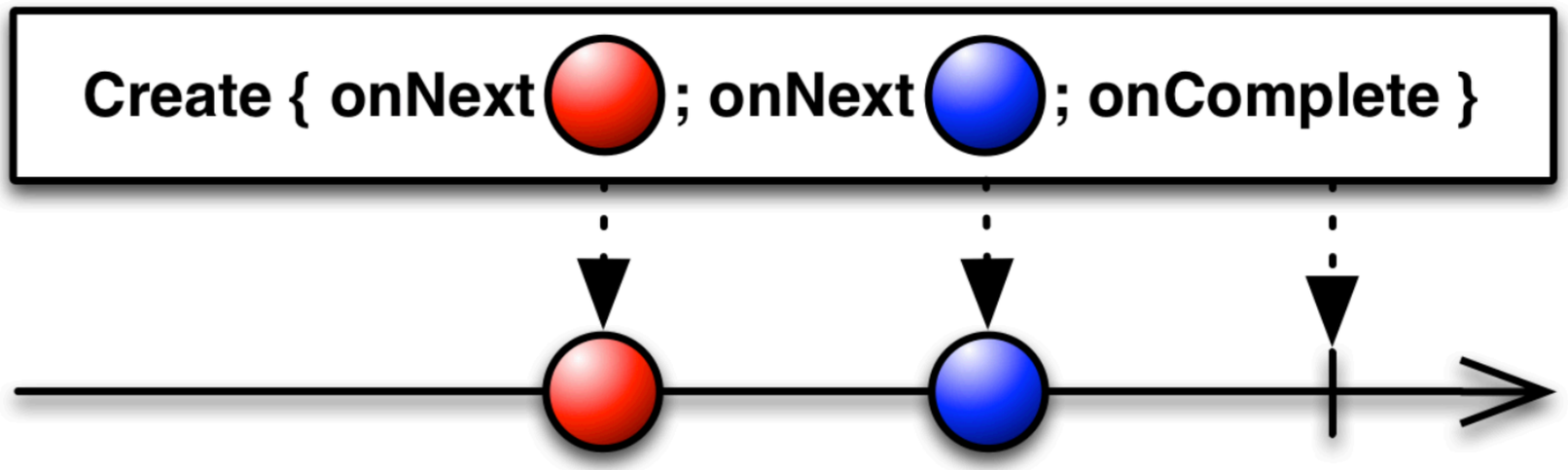
Operators

ReactiveX를 지원하는 언어 별 구현체들은 다양한 연산자들을 제공한다.

언어들은 유사한 형태로 연산자의 네이밍 컨벤션을 유지하고 있다.

직접 연산자를 구현할 수도 있다.

Create



직접적인 코드 구현을 통해 옵저버 메서드를 호출하여 Observable을 생성

Create

아이템을 발행하기 위해서는 onNext(), onError(), onCompleted()를 호출

주의 사항

- Observable이 구독 해지 되었을 때 등록된 콜백을 모두 해제 -> 메모리 누수 방지
- 구독자가 구독하는 동안에만 onNext, onComplete 이벤트 호출
- 에러가 발생했을 때는 오직 onError 이벤트로만 에러 전달

Create

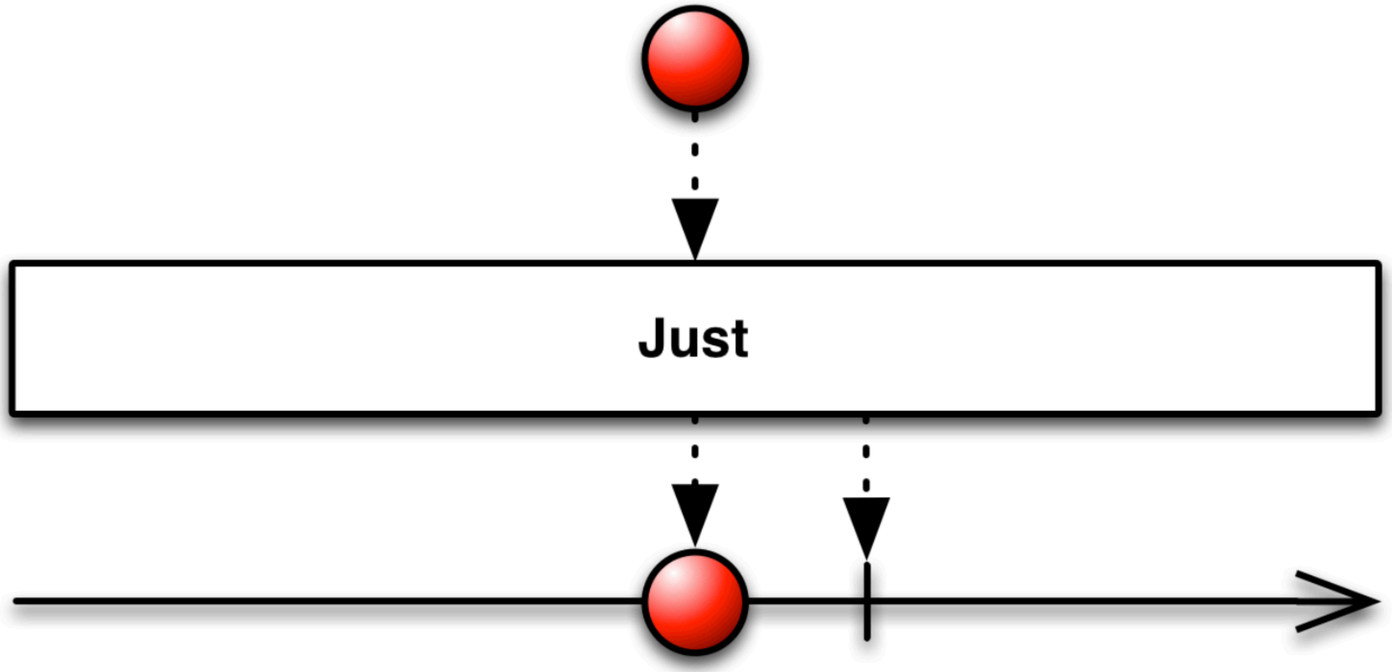
RxJava

```
Observable<Integer> source = observable.create(
  (observableEmitter<Integer> emitter) -> {
    emitter.onNext(100);
    emitter.onNext(200);
    emitter.onComplete();
  });

source.subscribe(System.out::println);
```

RxSwift

```
Observable<String>.create { observer in
    observer.onNext("A")
    observer.onCompleted()

    return Disposables.create()
}.subscribe(
    onNext: { print($0) },
    onError: { print($0) },
    onCompleted: { print("Completed") },
    onDisposed: { print("Disposed") }
).disposed(by: disposeBag)
```

Just



객체 하나 또는 객채집합을 Observable로 변환한다. 변환된 Observable은 원본 객체들을 발행

Just

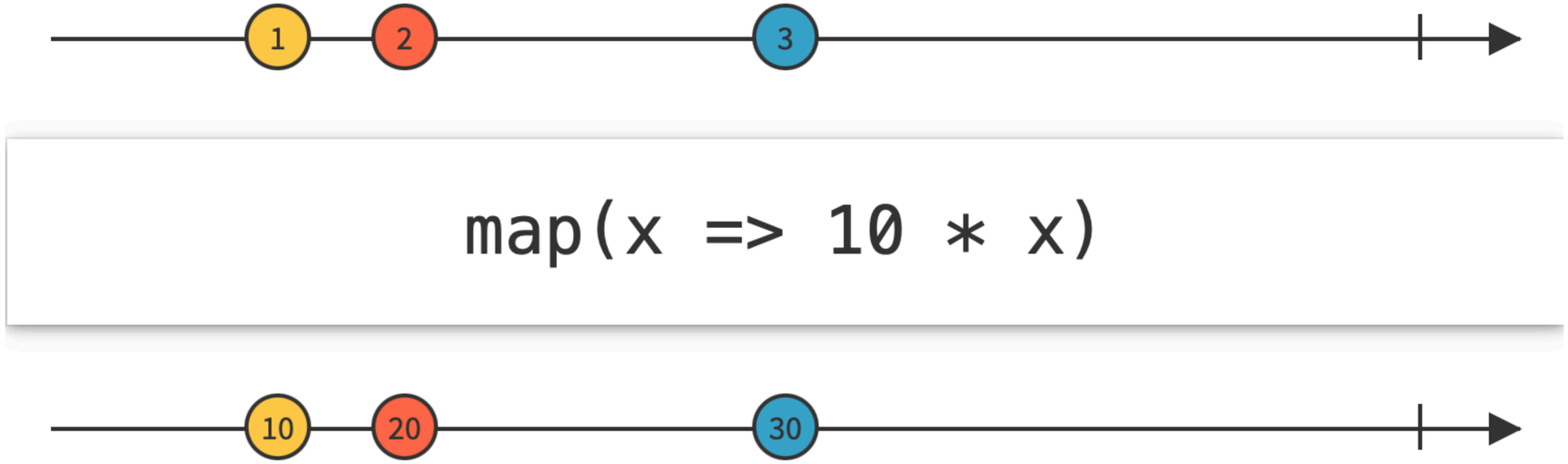
RxJava

```
public class Ex {
  public void emit(){
    Observable.just(1,2,3,4,5)
      .subscribe(System.out::println);
  }
}
```

RxSwift

```
let disposeBag = DisposeBag()

Observable.just(1)
    .subscribe { event in print(event) }
    .disposed(by: disposedBag)

next(1)
completed

Observable.just([1, 2, 3])
    .subscribe { event in print(event) }
    .disposed(by: disposedBag)
next([1, 2, 3])
completed
```

Map



Observable이 배출한 항목에 함수를 적용

Map

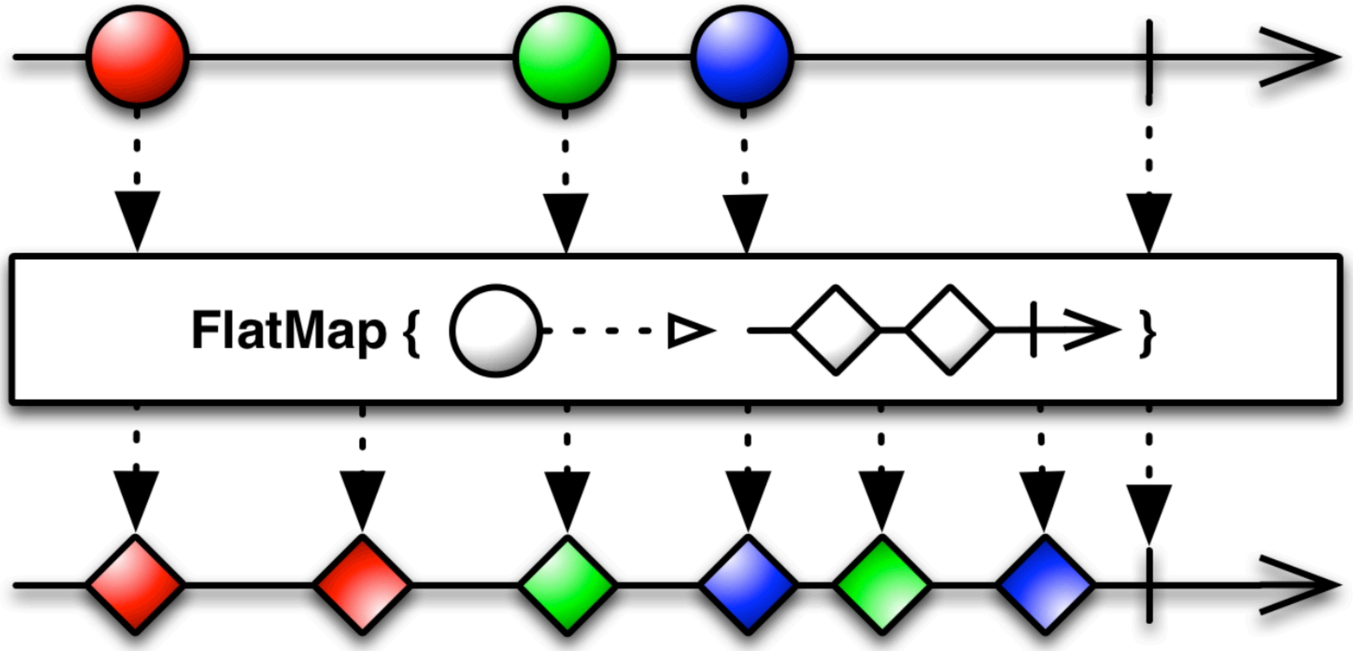
RxJava

```
Function<String, String> getDiamood = ball -> ball + "<>";

String[] balls = {"1", "2", "3"};
Observable<String> source = Observable.fromArray(balls)
  .map(getDiamood);
source.subseribe(Log::i)
```

RxSwift

```
slider.rx.value.asObservable()
            .map { String($0) }
            .bind(to: label.rx.text)
            .disposed(by: disposeBag)
```

FlatMap



Observable이 발행하는 항목들을 여러개의 Observable로

변환 후 줄 세워 하나의 Observable로 전달한다

FlatMap

```
Function<String, Observable<String>> getDoubleDiamonds =
  ball -> Observable.just(ball + "<>", ball + "<>");

String[] balls = {"1","3","5"};

Observable<String> source = Observable.fromArray(balls)
  .flatMap(getDoubleDiamonds);
source.subscribe(Log::i)
```

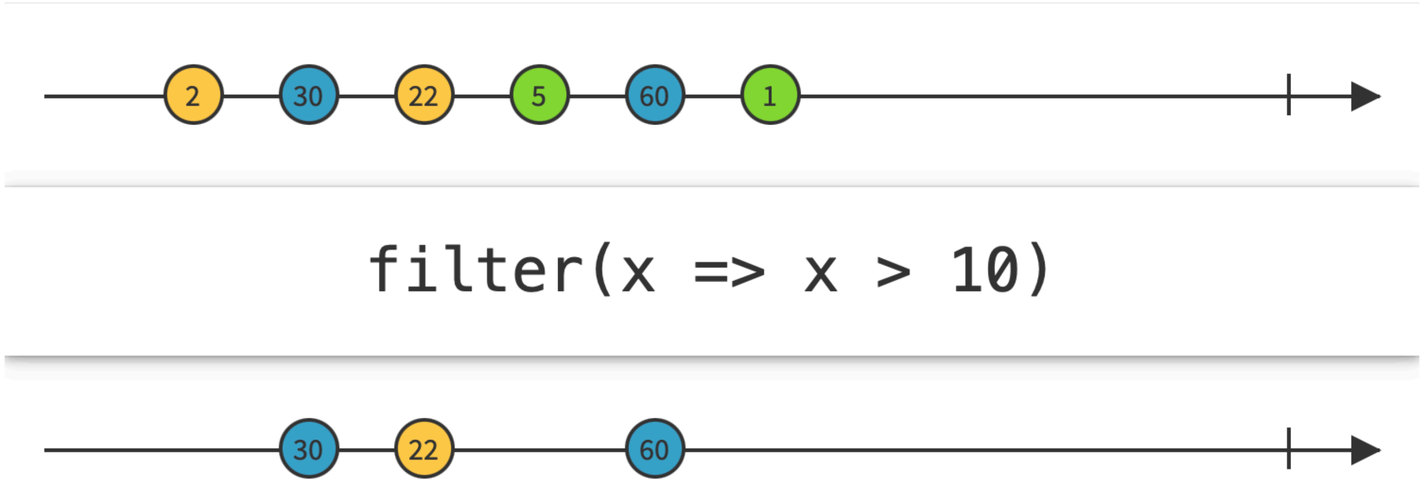
RxJava

FlatMap

```
let answers = Variable<[String]>.init([])

    button.rx.tap.asObservable().flatMap { _ -> Observable<[String]> in
            return fetchAllAnswers()
            }.subscribe(onNext: { newAnswers in
                answers.value = newAnswers
            }).disposed(by: disposeBag)


    func fetchAllAnswers() -> Observable<[String]> {
        let api = Observable<[String]>.create { observer in
            let answers = API.allAnswers()
            observer.onNext(answers)
            observer.onCompleted()
            return Disposables.create()
        }
        return api
    }
```

RxSwift

Filter



테스트 조건을 만족하는 항목들만 배출

Filter

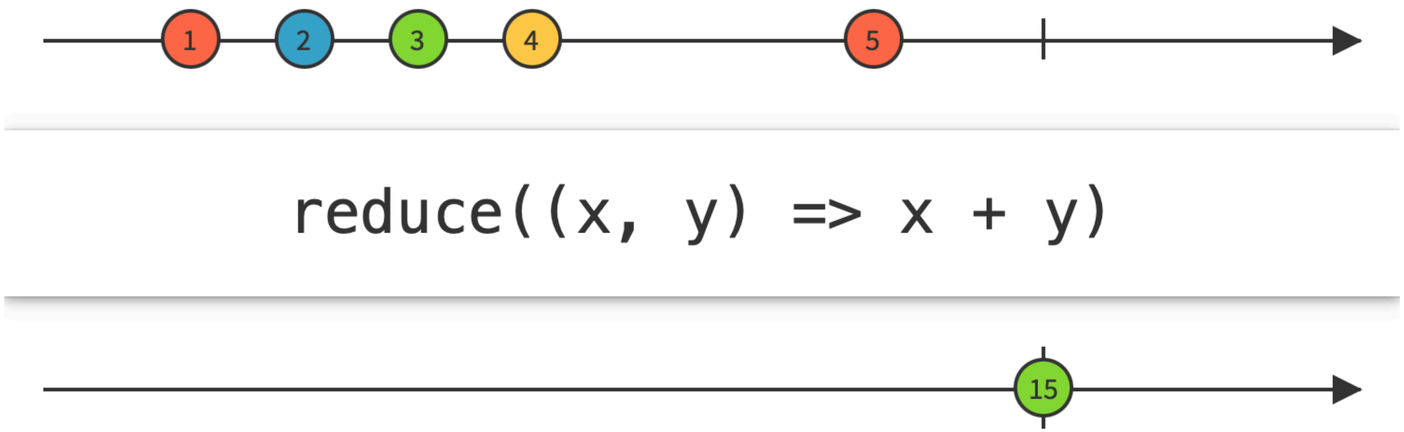
RxJava

```
Integer[] data = {"100", "34", "1", "35"};
Observable<Integer> source = Observable.fromArray(data)
  .filter(number -> number % 2 == 0);
source.subscribe(System.out::println);
```

RxSwift

```
let strikes = PublishSubject<String>()
let disposeBag = DisposeBag()
strikes
    .ignoreElements()
    .subscribe { _ in
    print("[Subsscription is called]")
}.disposed(by: disposeBag)

strikes.onNext("A")
strikes.onNext("B")
strikes.onNext("C")
```

Reduce



Observable이 배출한 항목에 함수를 순서대로 적용하고 함수를 연산한 후 최종 결과 발행

Reduce

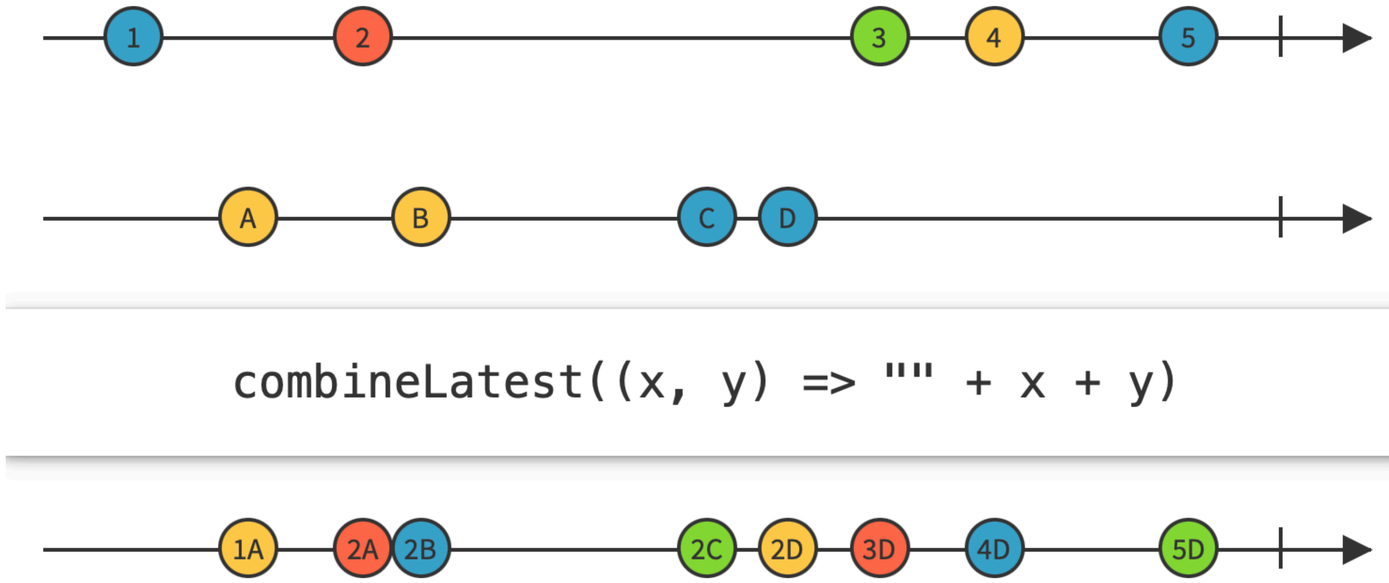
RxJava

```
String[] balls = {"1", "3", "5"};
Maybe<String> source = Observable.fromArray(balls)
  .reduce((ball1, ball2) -> ball2 + "(" + ball1 + ")");
source.subscribe(System.out::println);
```

RxSwift

```
Observable.of(1,2,3,4,5).reduce(0,accumulator: +)
    .subscribe(onNext: {
        print($0)
    }).disposed(by: disposeBag)
```

CombineLatest



두 개의 Observable 중 하나가 항목을 배출할 때 배출된 마지막 항목과

배출한 항목을 결합한 후 함수를 적용하여 실행 후 실행된 결과를 배출

CombineLatest

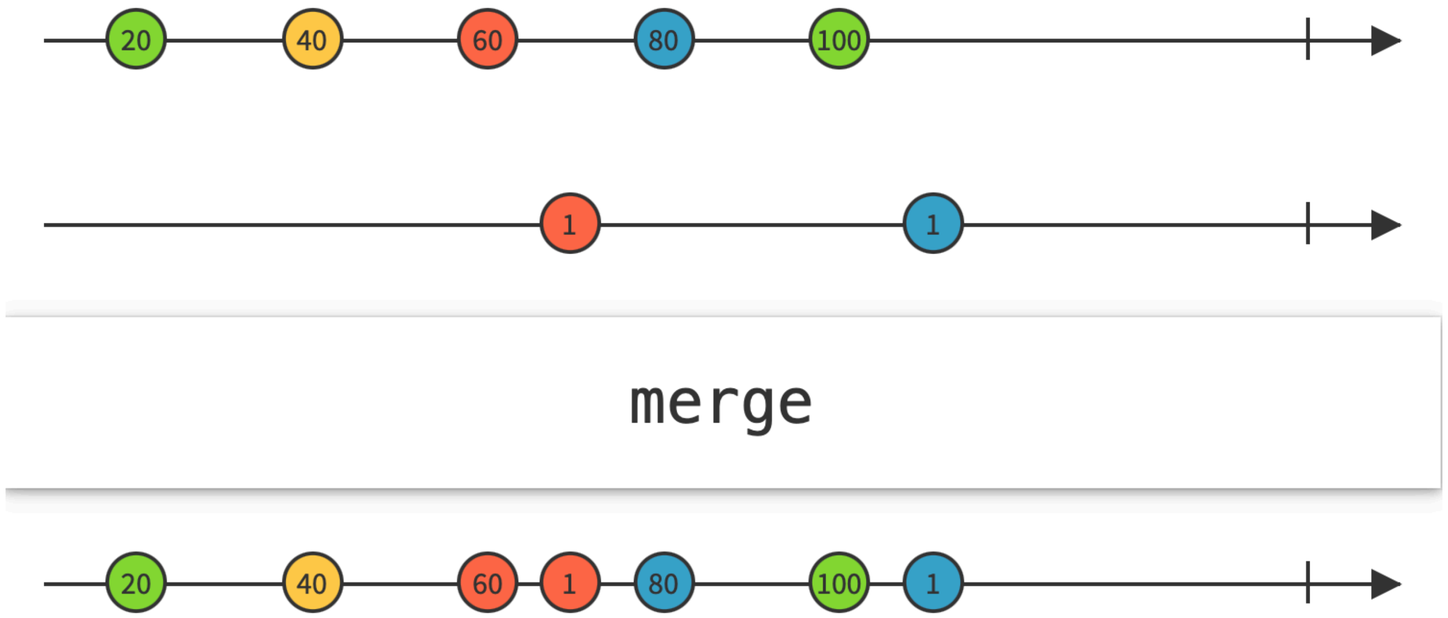
RxJava

```
@SchedulerSupport(SchedulerSupport.NONE)
public static <T1, T2, R> Observable<R> combineLatest(
  ObservableSource<? extends T1> source1,
  ObservableSource<? extends T2> source2,
  BiFunction<? super T1, ? super T2, ? extends R> combiner)
```

RxSwift

```
let left = PublishSubject<String>()
let right = PublishSubject<String>()

let observable = Observable
.combineLatest(left, right, resultSelector: { (lastLeft, lastRight) in
"\(lastLeft) \(lastRight)"
})

let disposable = observable.subscribe(onNext: { (string) in print(string) })
```

Merge



복수 개의 Observable들이 배출하는 항목들을 머지시켜 하나의 Observable로 만든다.

Merge

```
String[] data1 = {"1", "3"};
String[] data2 = {"2", "4", "6"};

Observable<String> source1 = Observable.interval(0L, 100L, TimeUnit.MILLISECONDS)
  .map(Long::intValue)
  .map(idx -> data1[idx])
  .take(data1.lenght);

Observable<String> source2 = Observable.interval(50L, TimeUnit.MILLISECONDS)
  .map(Long::intValue)
  .map(idx -> data2[idx])
  .take(data2.lenght);

Observable<String> source = Observable.merge(source1, source2);

source.subscribe(Log::i);
CommonUtils.sleep(1000);
```

RxJava

Merge

```
let left = PublishSubject<String>()
let right = PublishSubject<String>()

let source = Observable.of(left.asObservable(), right.asObservable())

let observable = source.merge()
let disposable = observable.subscribe(onNext: { value in
    print(value)
})

var leftValues = ["Berlin", "Munich", "Frankfurt"]
var rightValues = ["Madrid", "Barcelona", "Valencia"]
repeat {
    if arc4random_uniform(2) == 0 {
        if !leftValues.isEmpty {
            left.onNext("Left:  " + leftValues.removeFirst())
        }
    } else if !rightValues.isEmpty {
        right.onNext("Right: " + rightValues.removeFirst())
    }
} while !leftValues.isEmpty || !rightValues.isEmpty

disposable.dispose()
```

RxSwift

공식 문서

나는 새로운 Observable을 생성하고 싶은데 그 Observable이

…특정 항목을 생성해야 한다면:: Just

…구독 시점에 호출된 함수를 통해 생성된 항목을 리턴해야 한다면:: Start

…구독 시점에 호출된 `Action`, `Callable`, `Runnable` 또는 그와 유사한 함수 등을 통해 생성된 항목을 리턴해야 한다면:

: From

…지정된 시간 이후에 항목을 배출해야 한다면:: Timer

…특정 `Array`, `Iterable` 또는 유사한 형태의 소스로부터 항목들을 배출해야 한다면:: From

…퓨처(Future)에서 항목을 조회해서 배출해야 한다면:: Start

…퓨처에서 연속된 항목을 가져와야 한다면:: From

…반복적으로 연속된 항목을 배출해야 한다면:: Repeat

…사용자가 정의한 로직을 통해 생성되어야 한다면:: Create

…각각의 옵저버가 Observable을 구독한 후에 생성되어야 한다면:: Defer

…연속된 정수를 배출해야 한다면:: Range

…특정 시간 간격별로 항목을 배출해야 한다면:: Interval

…특정 시간 후에 항목을 배출해야 한다면:: Timer

…항목 배출 없이 실행을 완료해야 한다면:: Empty

…아무일도 하지 않아야 한다면:: Never