This section explains operators with which you conditionally emit or transform Observables, or can do boolean evaluations of them:

Conditional Operators

Outline

amb

Available in:  `Flowable`,  `Observable`,  `Maybe`,  `Single`,  `Completable`

ReactiveX documentation: http://reactivex.io/documentation/operators/amb.html

given two or more source Observables, emits all of the items from the first of these Observables to emit an item

```
    Observable source1 = Observable.range(1, 5);
    Observable source2 = Observable.range(6, 5);
    Observable.amb(new ArrayList(Arrays.asList(source1, source2)))
        .subscribe(next -> System.out.printf("next: %s\n", next), // onNext
            throwable -> System.out.printf("error: %s\n", throwable), //onError
            () -> System.out.println("Completed") //onComplete
        );
```

defaultIfEmpty

Available in:  `Flowable`,  `Observable`,  `Maybe`,  `Single`,  `Completable`

ReactiveX documentation: http://reactivex.io/documentation/operators/defaultifempty.html

emit items from the source Observable, or emit a default item if the source Observable completes after emitting no items

```
 Observable.empty().defaultIfEmpty(1).blockingSubscribe(next ->
System.out.printf("next: %s\n", next), // onNext
                throwable -> System.out.printf("error: %s", throwable), //onError
                () -> System.out.println("Completed") //onComplete
        );
```

skipUntil

Available in:  `Flowable`,  `Observable`,  `Maybe`,  `Single`,  `Completable`

ReactiveX documentation: http://reactivex.io/documentation/operators/skipuntil.html

discard items emitted by a source Observable until a second Observable emits an item, then emit the remainder of the source Observable's items

```
Observable observable1 = Observable.range(1, 10).doOnNext(next ->
Thread.sleep(1000));

observable1.skipUntil(Observable.timer(3, TimeUnit.SECONDS))
    .subscribe(next -> System.out.printf("next: %s\n", next), // onNext
        throwable -> System.out.printf("error: %s", throwable), //onError
        () -> System.out.println("Completed") //onComplete
    );
```

skipWhile

Available in:  `Flowable`,  `Observable`,  `Maybe`,  `Single`,  `Completable`

ReactiveX documentation: http://reactivex.io/documentation/operators/skipwhile.html

discard items emitted by an Observable until a specified condition is false, then emit the remainder

```
Observable.range(1, 10).skipWhile(next -> next < 5)
    .subscribe(next -> System.out.printf("next: %s\n", next), // onNext
        throwable -> System.out.printf("error: %s", throwable), //onError
        () -> System.out.println("Completed") //onComplete
    );
```

takeUntil

Available in:  `Flowable`,  `Observable`,  `Maybe`,  `Single`,  `Completable`

ReactiveX documentation: http://reactivex.io/documentation/operators/takeuntil.html

emits the items from the source Observable until a second Observable emits an item or issues a notification

```
Observable.range(1, 10).takeUntil(value -> value >= 5)
    .subscribe(next -> System.out.printf("next: %s\n", next), // onNext
        throwable -> System.out.printf("error: %s", throwable), //onError
        () -> System.out.println("Completed") //onComplete
    );
```

takeWhile

Available in:  `Flowable`,  `Observable`,  `Maybe`,  `Single`,  `Completable`

ReactiveX documentation: http://reactivex.io/documentation/operators/takewhile.html

emit items emitted by an Observable as long as a specified condition is true, then skip the remainder

```
     Observable.range(1, 10).takeWhile(value -> value <= 5)
                .subscribe(next -> System.out.printf("next: %s\n", next), // onNext
                        throwable -> System.out.printf("error: %s", throwable),
//onError
                        () -> System.out.println("Completed") //onComplete
                );
```

Boolean Operators

Outline

- `all`
- `contains`
- `isEmpty`
- `sequenceEqual`

all

Available in:  `Flowable`,  `Observable`,  `Maybe`,  `Single`,  `Completable`

ReactiveX documentation: http://reactivex.io/documentation/operators/all.html

determine whether all items emitted by an Observable meet some criteria

```
Flowable.range(0,10).doOnNext(next -> System.out.println(next)).all(integer ->
integer<10).
    blockingSubscribe(success->System.out.println("Success: "+success));
```

contains

Available in:  `Flowable`,  `Observable`,  `Maybe`,  `Single`,  `Completable`

ReactiveX documentation: http://reactivex.io/documentation/operators/contains.html

determine whether an Observable emits a particular item or not

```
Flowable.range(1,10).doOnNext(next->System.out.println(next))
    .contains(4).blockingSubscribe(contains->System.out.println("contains:
"+contains));
```

isEmpty

Available in: ✅ `Flowable`, ✅ `Observable`, ✅ `Maybe`, ⭕ `Single`, ⭕ `Completable`

ReactiveX documentation: http://reactivex.io/documentation/operators/contains.html

determine whether the source Publisher is empty

```
Flowable.empty().isEmpty().subscribe(isEmpty -> System.out.printf("isEmpty: %s",
isEmpty));
```

sequenceEqual

Available in: ✅ `Flowable`, ✅ `Observable`, ✅ `Maybe`, ✅ `Single`, ✅ `Completable`

ReactiveX documentation: http://reactivex.io/documentation/operators/sequenceequal.html

test the equality of the sequences emitted by two Observables

```
Flowable<Integer> flowable1 = Flowable.range(1,3).doOnNext(next->
System.out.print("flowable1: "+next + " "));

Flowable<Integer> flowable2 = Flowable.range(1,3).doOnNext(next->
System.out.println("flowable2: "+next));

Flowable.sequenceEqual(Flowable.fromPublisher(flowable1),Flowable.fromPublisher(flowab

    .blockingSubscribe(sequenceEqual->System.out.println("sequenceEqual:
"+sequenceEqual));
```