This section explains operators you can use to combine multiple Observables.

Outline

startWith

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

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

Emit a specified sequence of items before beginning to emit the items from the Observable.

startWith Example

```
Observable<String> names = Observable.just("Spock", "McCoy");
names.startWith("Kirk").subscribe(item -> System.out.println(item));

// prints Kirk, Spock, McCoy
```

merge

Combines multiple Observables into one.

merge

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

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

Combines multiple Observables into one. Any `onError` notifications passed from any of the source observables will immediately be passed through to through to the observers and will terminate the merged `Observable`.

merge Example

```
Observable.just(1, 2, 3)
    .mergeWith(Observable.just(4, 5, 6))
    .subscribe(item -> System.out.println(item));
```

```
// prints 1, 2, 3, 4, 5, 6
```

mergeDelayError

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

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

Combines multiple Observables into one. Any `onError` notifications passed from any of the source observables will be withheld until all merged Observables complete, and only then will be passed along to the observers.

mergeDelayError Example

```
Observable<String> observable1 = Observable.error(new IllegalArgumentException(""));
Observable<String> observable2 = Observable.just("Four", "Five", "Six");
Observable.mergeDelayError(observable1, observable2)
        .subscribe(item -> System.out.println(item));

// emits 4, 5, 6 and then the IllegalArgumentException (in this specific
// example, this throws an `OnErrorNotImplementedException`).
```

zip

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

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

Combines sets of items emitted by two or more Observables together via a specified function and emit items based on the results of this function.

zip Example

```
Observable<String> firstNames = Observable.just("James", "Jean-Luc", "Benjamin");
Observable<String> lastNames = Observable.just("Kirk", "Picard", "Sisko");
firstNames.zipWith(lastNames, (first, last) -> first + " " + last)
    .subscribe(item -> System.out.println(item));

// prints James Kirk, Jean-Luc Picard, Benjamin Sisko
```

combineLatest

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

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

When an item is emitted by either of two Observables, combine the latest item emitted by each Observable via a specified function and emit items based on the results of this function.

combineLatest Example

```
Observable<Long> newsRefreshes = Observable.interval(100, TimeUnit.MILLISECONDS);
Observable<Long> weatherRefreshes = Observable.interval(50, TimeUnit.MILLISECONDS);
Observable.combineLatest(newsRefreshes, weatherRefreshes,
    (newsRefreshTimes, weatherRefreshTimes) ->
        "Refreshed news " + newsRefreshTimes + " times and weather " +
weatherRefreshTimes)
    .subscribe(item -> System.out.println(item));

// prints:
// Refreshed news 0 times and weather 0
// Refreshed news 0 times and weather 1
// Refreshed news 0 times and weather 2
// Refreshed news 1 times and weather 2
// Refreshed news 1 times and weather 3
// Refreshed news 1 times and weather 4
// Refreshed news 2 times and weather 4
// Refreshed news 2 times and weather 5
// ...
```

switchOnNext

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

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

Convert an Observable that emits Observables into a single Observable that emits the items emitted by the most-recently emitted of those Observables.

switchOnNext Example

```
Observable<Observable<String>> timeIntervals =
    Observable.interval(1, TimeUnit.SECONDS)
      .map(ticks -> Observable.interval(100, TimeUnit.MILLISECONDS)
                              .map(innerInterval -> "outer: " + ticks + " - inner: "
+ innerInterval));
Observable.switchOnNext(timeIntervals)
    .subscribe(item -> System.out.println(item));

// prints:
// outer: 0 - inner: 0
// outer: 0 - inner: 1
// outer: 0 - inner: 2
// outer: 0 - inner: 3
// outer: 0 - inner: 4
```

```
// outer: 0 - inner: 5
// outer: 0 - inner: 6
// outer: 0 - inner: 7
// outer: 0 - inner: 8
// outer: 1 - inner: 0
// outer: 1 - inner: 1
// outer: 1 - inner: 2
// outer: 1 - inner: 3
// ...
```

joins

- `join()` and `groupJoin()` — combine the items emitted by two Observables whenever one item from one Observable falls within a window of duration specified by an item emitted by the other Observable

rxjava-joins

- (`rxjava-joins`) `and()` , `then()` , and `when()` — combine sets of items emitted by two or more Observables by means of `Pattern` and `Plan` intermediaries

> *(`rxjava-joins`) — indicates that this operator is currently part of the optional `rxjava-joins` package under `rxjava-contrib` and is not included with the standard RxJava set of operators*