Applying Key Methods in the Observable Class (Part 4)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Recognize key methods in the Observable class & how they are applied in the case studies

Class Observable<T>

java.lang.Object
    io.reactivex.rxjava3.core.Observable<T>

Type Parameters:
    `T` - the type of the items emitted by the `Observable`

All Implemented Interfaces:
    ObservableSource<T>

Direct Known Subclasses:
    ConnectableObservable, GroupedObservable, Subject

```
public abstract class Observable<T>
extends Object
implements ObservableSource<T>
```

The `Observable` class is the non-backpressured, optionally multi-valued base reactive class that offers factory methods, intermediate operators and the ability to consume synchronous and/or asynchronous reactive dataflows.

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html

Learning Objectives in this Part of the Lesson

- Case study ex3 shows how to apply various RxJava operations *asynchronously* to reduce & multiply BigFraction objects
  - e.g., fromIterable(), map(), create(), flatMap(), flatMapCompletable(), filter(), collectInto(), subscribeOn(), onErrorReturn(), & Schedulers.computation(), ambArray(), & doOnSuccess()

```
return Observable
   .create(ObservableEx::bFEmitter)

   .flatMap(unreducedFraction ->
      reduceAndMultiplyFraction
        (unreducedFraction,
         Schedulers.computation()))

   .collectInto(new ArrayList
     <BigFraction>(), List::add)

   .flatMapCompletable(list ->
         BigFractionUtils
         .sortAndPrintList(list,
                           sb));
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/Observable/ex3

Applying Key Methods in the Observable Class to ex3

Applying Key Methods in the Observable Class to ex3

- testFractionExceptions()
  - Use an asynchronous Observable stream & a pool of threads to showcase exception handling of BigFraction objects

```
return Observable
   .fromIterable(denominators)
   .flatMap(denominator -> {
      return Observable
         .fromCallable(() -> ...))
         .subscribeOn(...)
         .onErrorReturn(...)
         .map(multiplyBigFractions);
       })
   .filter(...)
   .collectInto(...)
   .flatMapCompletable
      (list -> BigFractionUtils.
            sortAndPrintList(list,
                             sb));
```

See Reactive/Observable/ex3/src/main/java/ObservableEx.java

Applying Key Methods in the Observable Class to ex2

- testFractionExceptions()
  - Use an asynchronous Observable stream & a pool of threads to showcase exception handling of BigFraction objects
  - Demonstrates Observable methods
    - e.g., fromIterable(), create(), fromCallable(), map(), flatMap(), flatMapCompletable(), filter(), collectInto(), subscribeOn(), onErrorReturn(), & Schedulers. computation()

```
return Observable
   .fromIterable(denominators)
   .flatMap(denominator -> {
      return Observable
         .fromCallable(() -> ...))
         .subscribeOn(...)
         .onErrorReturn(...)
         .map(multiplyBigFractions);
       })
   .filter(...)
   .collectInto(...)
   .flatMapCompletable
      (list -> BigFractionUtils.
            sortAndPrintList(list,
                             sb));
```

Applying Key Methods in the Observable Class to ex2

- testFractionExceptions()
  - Use an asynchronous Observable stream & a pool of threads to showcase exception handling of BigFraction objects
  - Demonstrates Observable methods
  - Also demonstrates Single methods
    - e.g., ambArray(), doOnSuccess(), & ignoreElement()

```
return Single
   .ambArray(quickSortS,
             heapSortS)
   .doOnSuccess(displayList)

   .ignoreElement();
```

Applying Key Methods in the Observable Class to ex3

- The fromIterable() method
  - Create an Observable that emits the items contained in the given Iterable

```
static <T> Observable<T>
fromIterable
   (Iterable<? extends T> it)
```

Applying Key Methods in the Observable Class to ex3

- The fromIterable() method
  - Create an Observable that emits the items contained in the given Iterable
    - The Iterable.iterator() method will be invoked at least once & at most twice for each subscriber

```
static <T> Observable<T>
fromIterable
   (Iterable<? extends T> it)
```

Interface Iterable<T>

Type Parameters:

T - the type of elements returned by the iterator

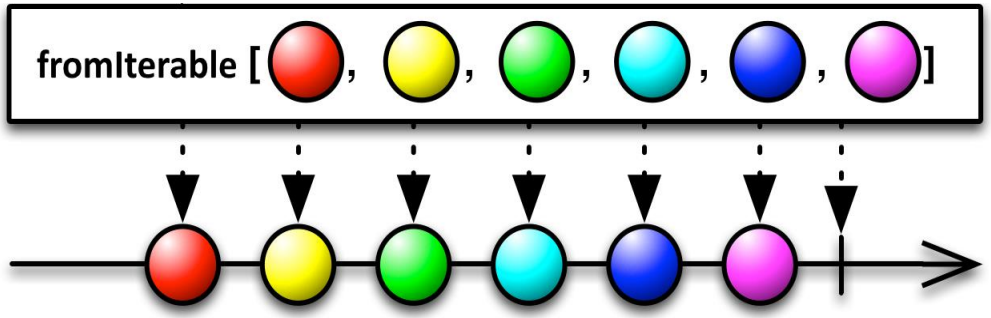
All Known Subinterfaces:

BeanContext, BeanContextServices, BlockingDeque<E>, BlockingQueue<E>, Collection<E>, Deque<E>, DirectoryStream<T>, List<E>, NavigableSet<E>, Path, Queue<E>, SecureDirectoryStream<T>, Set<E>, SortedSet<E>, TransferQueue<E>

See docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html

Applying Key Methods in the Observable Class to ex3

- The fromIterable() method
  - Create an Observable that emits the items contained in the given Iterable
  - This factory method adapts non-reactive input sources into the reactive model
    - e.g., Java collections



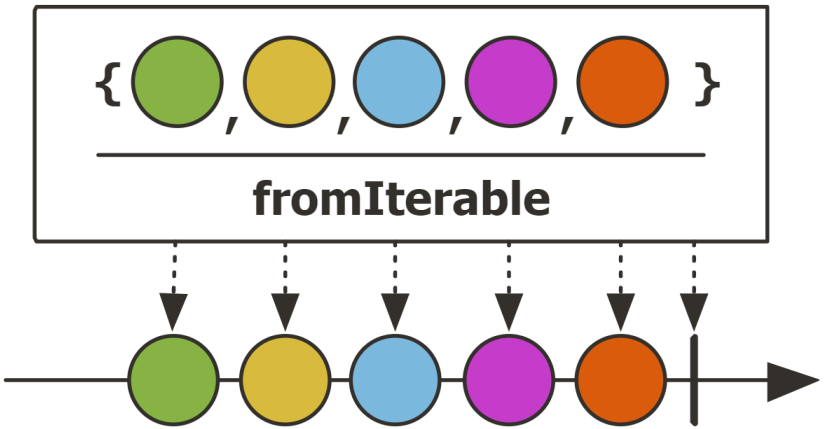

```
List<Integer> denominators =
  List.of(3, 4, 2, 0 1);

Observable
   .fromIterable(denominators)
   ...
```

See docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html

Applying Key Methods in the Observable Class to ex3

- The fromIterable() method
  - Create an Observable that emits the items contained in the given Iterable
  - This factory method adapts non-reactive input sources into the reactive model
  - Project Reactor's Flux.fromIterable() method works the same



See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#fromIterable

Applying Key Methods in the Observable Class to ex3

- The fromIterable() method
  - Create an Observable that emits the items contained in the given Iterable
  - This factory method adapts non-reactive input sources into the reactive model
  - Project Reactor's Flux.fromIterable() method works the same
  - Similar to the Collection.stream() method in Java Streams

stream

```
default Stream<E> stream()
```

Returns a sequential `Stream` with this collection as its source.

This method should be overridden when the `spliterator()` method cannot return a spliterator that is `IMMUTABLE`, `CONCURRENT`, or *late-binding*. (See `spliterator()` for details.)

Implementation Requirements:

The default implementation creates a sequential Stream from the collection's Spliterator.

Returns:

a sequential Stream over the elements in this collection

See docs.oracle.com/javase/8/docs/api/java/util/Collection.html#stream

Applying Key Methods in the Observable Class to ex3

- The flatMap() method
  - Transform the elements emitted by this Observable asynchronously

```
<R> Observable<R> flatMap
   (Function
      <? super T,
       ? extends ObservableSource
                        <? extends R>>
    mapper)
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#flatMap

Applying Key Methods in the Observable Class to ex3

- The flatMap() method
  - Transform the elements emitted by this Observable asynchronously
    - Items are emitted based on applying a function to each item emitted by this Observable

```
<R> Observable<R> flatMap
  (Function
     <? super T,
      ? extends ObservableSource
                    <? extends R>>
   mapper)
```

Applying Key Methods in the Observable Class to ex3

- The flatMap() method
  - Transform the elements emitted by this Observable asynchronously
    - Items are emitted based on applying a function to each item emitted by this Observable
    - That function returns an ObservableSource

```
<R> Observable<R> flatMap
   (Function
      <? super T,
       ? extends ObservableSource
                        <? extends R>>
    mapper)
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/ObservableSource.html

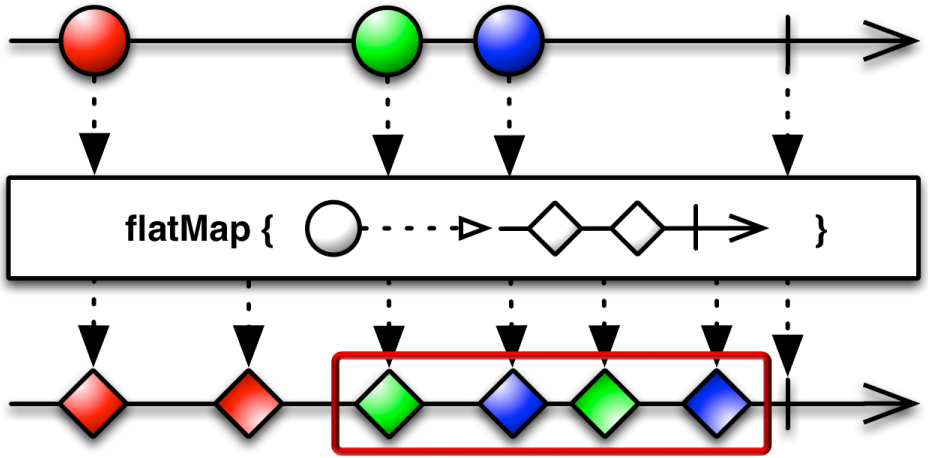
Applying Key Methods in the Observable Class to ex3

- The flatMap() method
  - Transform the elements emitted by this Observable asynchronously
    - Items are emitted based on applying a function to each item emitted by this Observable
    - That function returns an ObservableSource
    - The returned ObservableSources are merged & the results of this merger are emitted

```
<R> Observable<R> flatMap
   (Function
      <? super T,
       ? extends ObservableSource
                      <? extends R>>
    mapper)
```

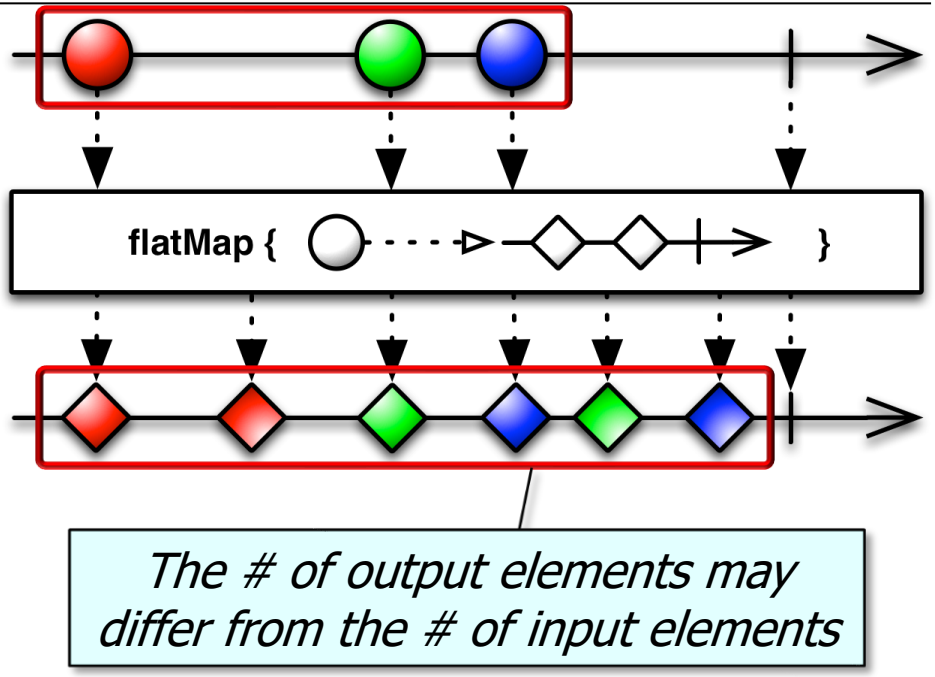
Applying Key Methods in the Observable Class to ex3

- The flatMap() method
  - Transform the elements emitted by this Observable asynchronously
    - Items are emitted based on applying a function to each item emitted by this Observable
    - That function returns an ObservableSource
    - The returned ObservableSources are merged & the results of this merger are emitted
      - They thus can interleave



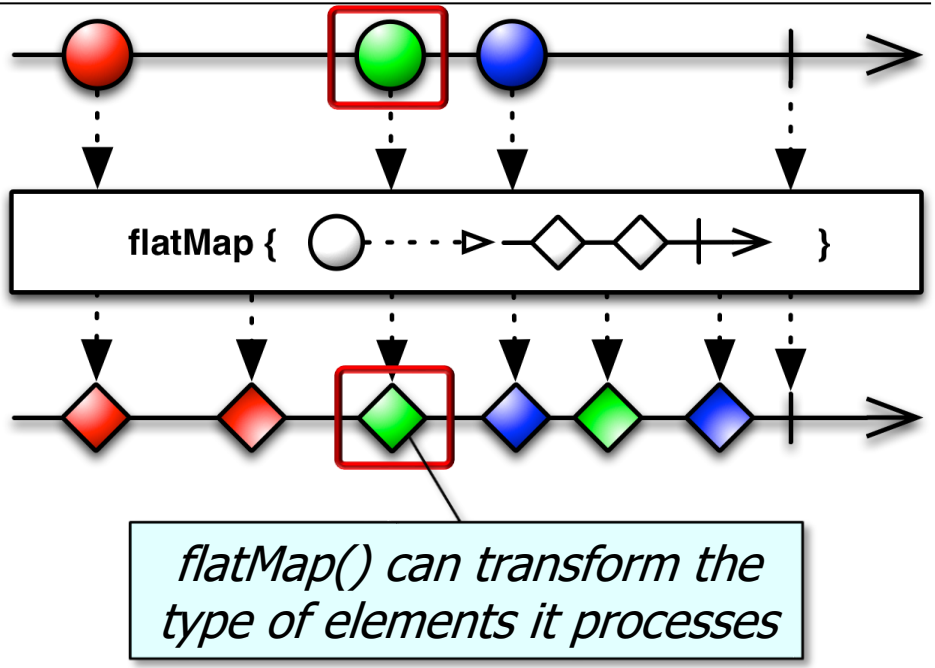
Applying Key Methods in the Observable Class to ex3

- The flatMap() method
  - Transform the elements emitted by this Observable asynchronously
    - Items are emitted based on applying a function to each item emitted by this Observable
    - That function returns an ObservableSource
  - The returned ObservableSources are merged & the results of this merger are emitted
    - They thus can interleave



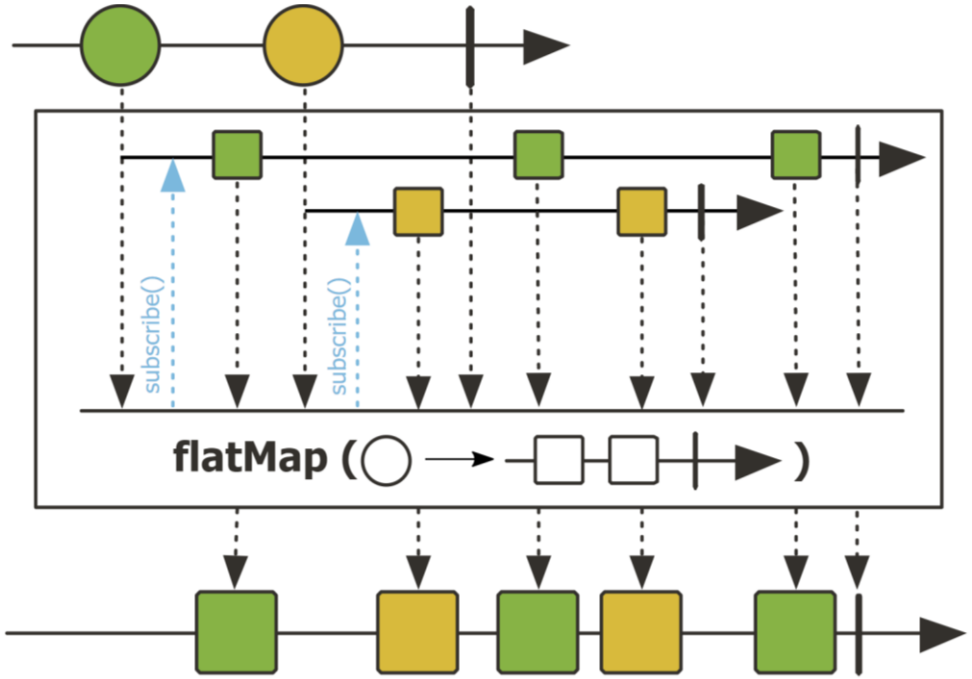
Applying Key Methods in the Observable Class to ex3

- The flatMap() method
  - Transform the elements emitted by this Observable asynchronously
    - Items are emitted based on applying a function to each item emitted by this Observable
    - That function returns an ObservableSource
  - The returned ObservableSources are merged & the results of this merger are emitted
    - They thus can interleave



Applying Key Methods in the Observable Class to ex3

- The flatMap() method
  - Transform the elements emitted by this Observable asynchronously
  - Project Reactor's Flux.flatMap() method works the same way



See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#flatMap

Applying Key Methods in the Observable Class to ex3

- The flatMap() method
  - Transform the elements emitted by this Observable asynchronously
  - Project Reactor's Flux.flatMap() method works the same way
  - Similar to the Stream.flatMap() method in Java Streams

flatMap

```
<R> Stream<R> flatMap(
Function<? super T,? extends Stream<? extends R>> mapper)
```

Returns a stream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is `closed` after its contents have been placed into this stream. (If a mapped stream is `null` an empty stream is used, instead.)

This is an intermediate operation.

API Note:

The `flatMap()` operation has the effect of applying a one-to-many transformation to the elements of the stream, and then flattening the resulting elements into a new stream.

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#flatMap

Applying Key Methods in the Observable Class to ex3

- flatMap() is often used when each item emitted by a stream needs to have its own threading operators applied to it
  - i.e., the "flatMap() concurrency idiom"

```
Observable
   .create(bigFractionEmitter)

   .flatMap(unreduced ->
         reduceAndMultiplyFraction
          (unreduced, Schedulers
                .computation()))

   .collectInto(new
      ArrayList<BigFraction>(),
      List::add)

   .flatMapCompletable(list ->
      BigFractionUtils
       .sortAndPrintList(list,sb));
```

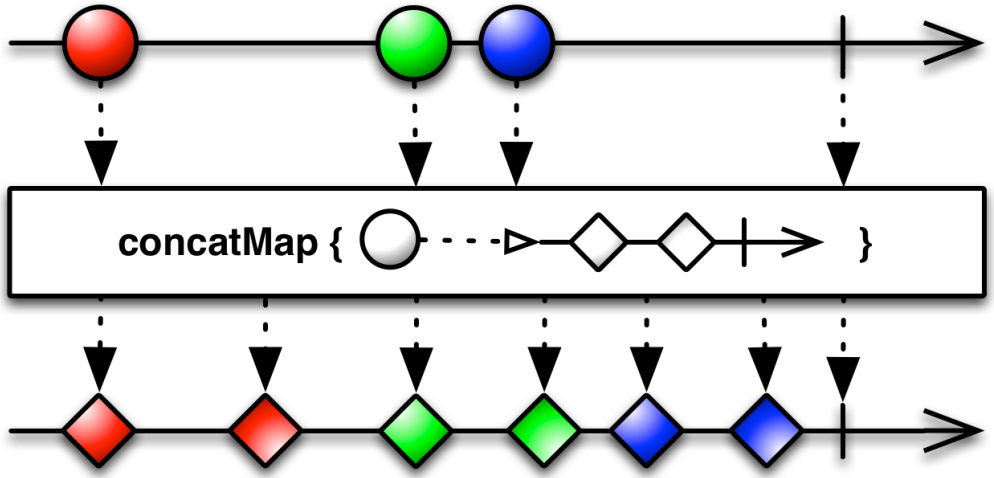
See www.nurkiewicz.com/2017/09/idiomatic-concurrency-flatmap-vs.html

Applying Key Methods in the Observable Class to ex3

- flatMap() doesn't ensure the order of the items in the resulting stream

Applying Key Methods in the Observable Class to ex3

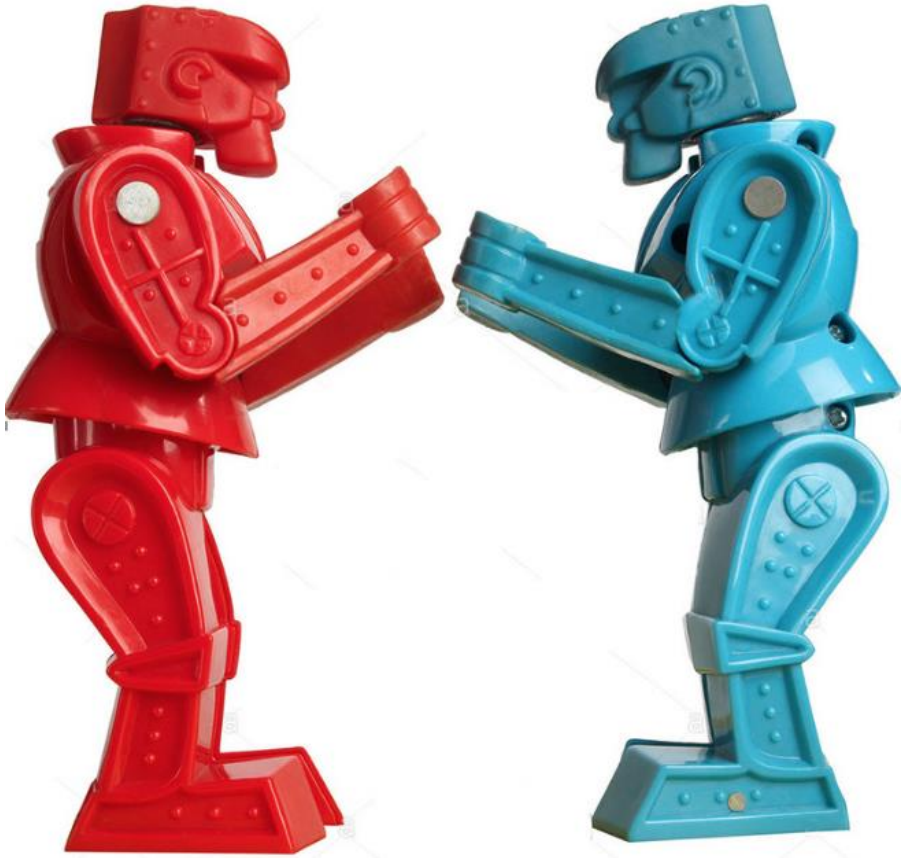
- flatMap() doesn't ensure the order of the items in the resulting stream
  - use concatMap() if order matters



See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#concatMap

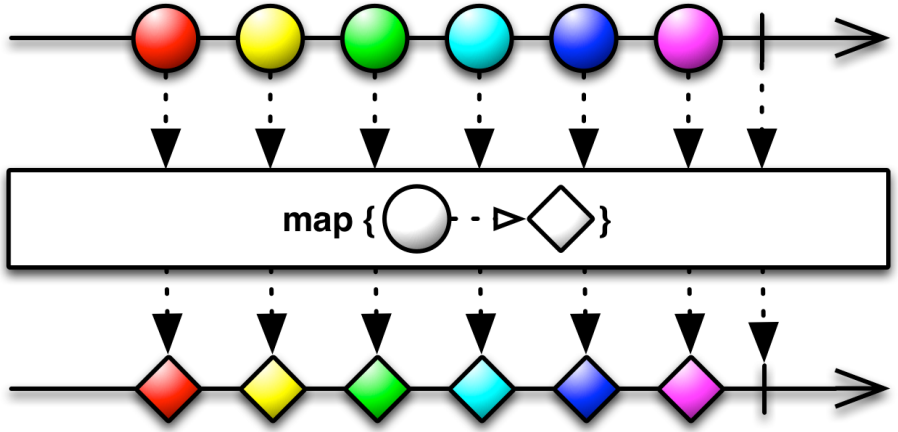
Applying Key Methods in the Observable Class to ex3

- The map() vs. flatMap() method



Applying Key Methods in the Observable Class to ex3

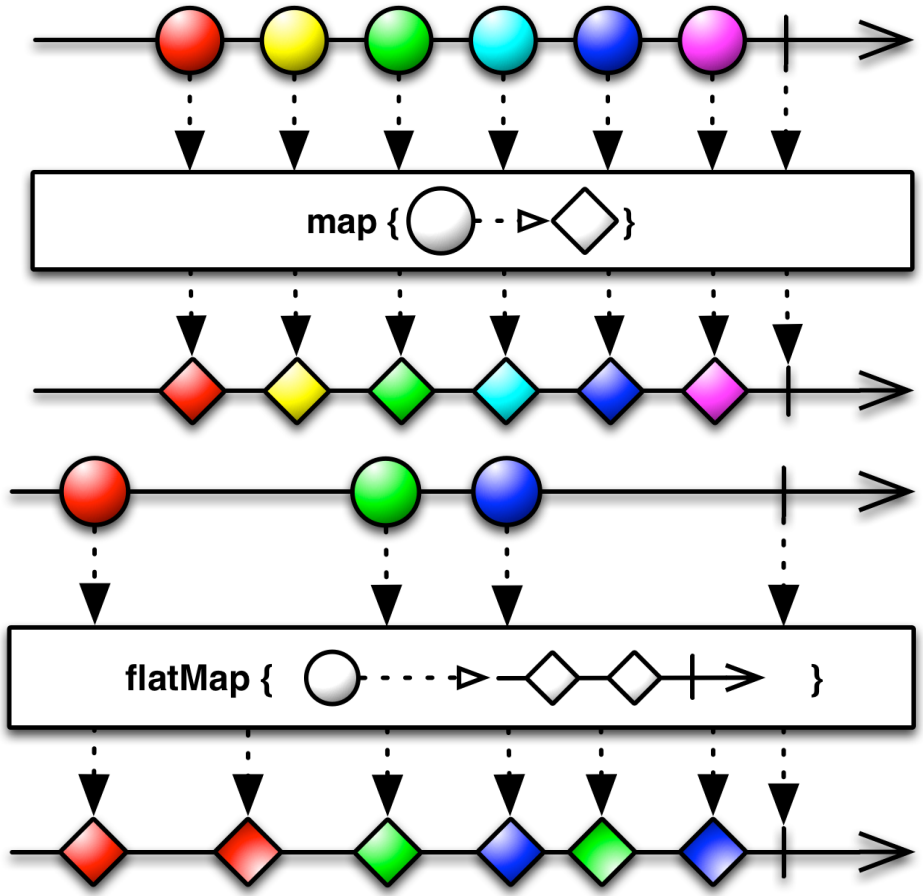
- The map() vs. flatMap() method
  - The map() operator transforms each value in a Observable stream into a single value
    - i.e., intended for synchronous, non-blocking, 1-to-1 transformations



See medium.com/mindorks/rxjava-operator-map-vs-flatmap-427c09678784

Applying Key Methods in the Observable Class to ex3

- The map() vs. flatMap() method
  - The map() operator transforms each value in a Observable stream into a single value
  - The flatMap() operator transforms each value in a Observable stream into an arbitrary number (zero or more) values
    - i.e., intended for asynchronous (often non-blocking) 1-to-N transformations



See medium.com/mindorks/rxjava-operator-map-vs-flatmap-427c09678784

Applying Key Methods in the Observable Class to ex3

- The collectInto() method
  - Collects items emitted by the finite source Observable into a single mutable data structure

```
Single<U> collectInto
   (U initialItem,
    BiConsumer<? super U, ? super T>
    collector)
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#collectInto

Applying Key Methods in the Observable Class to ex3

- The collectInto() method
  - Collects items emitted by the finite source Observable into a single mutable data structure
    - The 1st param is the mutable data structure that accumulates (collects) the items

```
Single<U> collectInto
   (U initialItem,
    BiConsumer<? super U, ? super T>
    collector)

...
.collectInto
   (new ArrayList<BigFraction>(),
    List::add)
...
```

Applying Key Methods in the Observable Class to ex3

- The collectInto() method
  - Collects items emitted by the finite source Observable into a single mutable data structure
    - The 1st param is the mutable data structure that accumulates (collects) the items
  - The 2nd param is a bi-consumer that accepts the accumulator & an emitted item
    - The accumulator is modified accordingly

```
Single<U> collectInto
   (U initialItem,
    BiConsumer<? super U, ? super T>
    collector)

...
.collectInto
   (new ArrayList<BigFraction>(),
    List::add)
...
```

Interface BiConsumer<T1,T2>

Type Parameters:

T1 - the first value type

T2 - the second value type

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/functions/BiConsumer.html

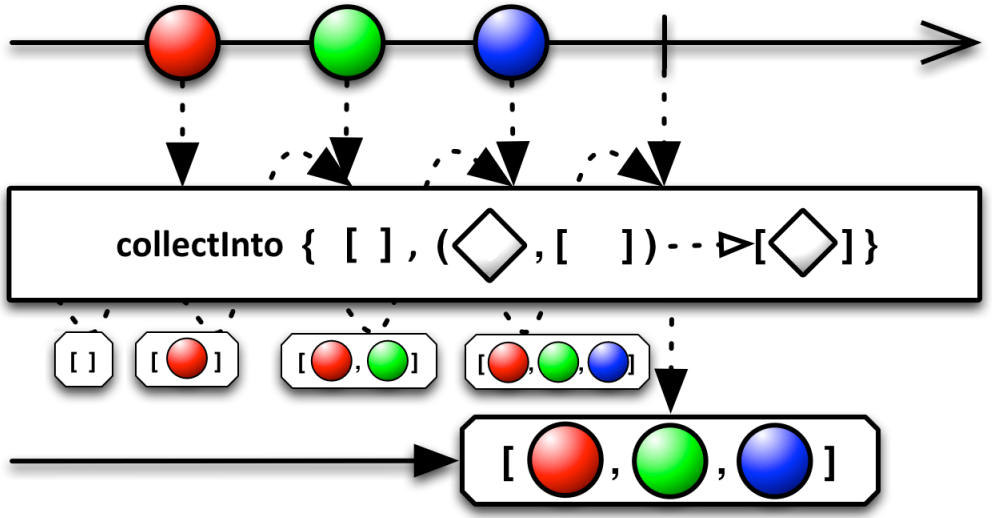
Applying Key Methods in the Observable Class to ex3

- The collectInto() method
  - Collects items emitted by the finite source Observable into a single mutable data structure
    - The 1st param is the mutable data structure that accumulates (collects) the items
    - The 2nd param is a bi-consumer that accepts the accumulator & an emitted item
  - Returns a Single that emits this structure

```
Single<U> collectInto
   (U initialItem,
    BiConsumer<? super U, ? super T>
    collector)
```

Applying Key Methods in the Observable Class to ex3

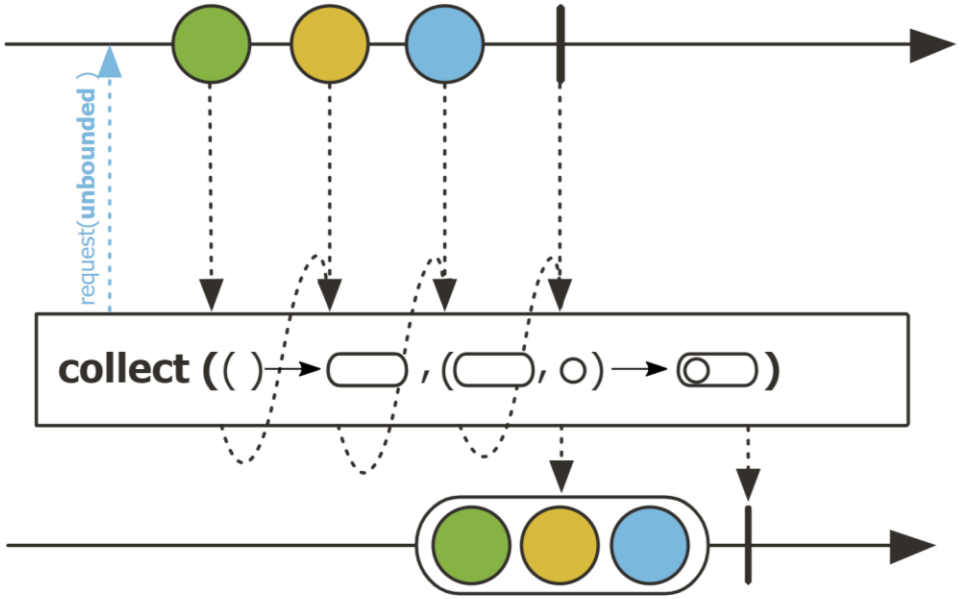
- The collectInto() method
  - Collects items emitted by the finite source Observable into a single mutable data structure
  - This method is a simplified version of reduce() that does not need to return the state on each pass



See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#reduce

Applying Key Methods in the Observable Class to ex3

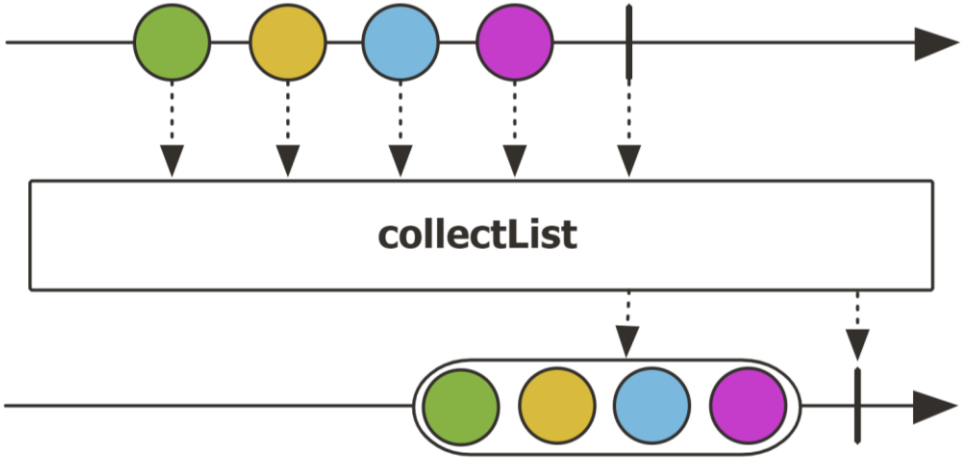
- The collectInto() method
  - Collects items emitted by the finite source Observable into a single mutable data structure
  - This method is a simplified version of reduce() that does not need to return the state on each pass
  - Project Reactor's Flux.collect() method works the same way



See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#collect

Applying Key Methods in the Observable Class to ex3

- The collectInto() method
  - Collects items emitted by the finite source Observable into a single mutable data structure
  - This method is a simplified version of reduce() that does not need to return the state on each pass
  - Project Reactor's Flux.collect() method works the same way
    - Flux.collectList() is an a more concise (albeit limited) option



See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#collectList

Applying Key Methods in the Observable Class to ex3

- The collectInto() method
  - Collects items emitted by the finite source Observable into a single mutable data structure
  - This method is a simplified version of reduce() that does not need to return the state on each pass
  - Project Reactor's Flux.collect() method works the same
  - Similar to the Stream.collect() method in Java Streams

collect

```
<R> R collect(Supplier<R> supplier,
              BiConsumer<R,? super T> accumulator,
              BiConsumer<R,R> combiner)
```

Performs a mutable reduction operation on the elements of this stream. A mutable reduction is one in which the reduced value is a mutable result container, such as an `ArrayList`, and elements are incorporated by updating the state of the result rather than by replacing the result. This produces a result equivalent to:

```
R result = supplier.get();
for (T element : this stream)
    accumulator.accept(result, element);
return result;
```

Like `reduce(Object, BinaryOperator)`, `collect` operations can be parallelized without requiring additional synchronization.

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#collect

Applying Key Methods in the Observable Class to ex3

- The flatMapCompletable() method
  - "flatMaps" an Observable into a Completable

```
Completable flatMapCompletable
   (Function<? super T,
             ? extends
               CompletableSource>
    mapper))
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#flatMapCompletable

Applying Key Methods in the Observable Class to ex3

- The flatMapCompletable() method
  - "flatMaps" an Observable into a Completable, e.g.,
    - Maps each element of the current Observable into CompletableSources

```
Completable flatMapCompletable
  (Function<? super T,
            ? extends
              CompletableSource>
   mapper))
```

Applying Key Methods in the Observable Class to ex3

- The flatMapCompletable() method
  - "flatMaps" an Observable into a Completable, e.g.,
    - Maps each element of the current Observable into CompletableSources
    - Subscribes to them & waits for the completion of the upstream & all CompletableSources

```
Completable flatMapCompletable
   (Function<? super T,
             ? extends
               CompletableSource>
    mapper))
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/CompletableSource.html

Applying Key Methods in the Observable Class to ex3

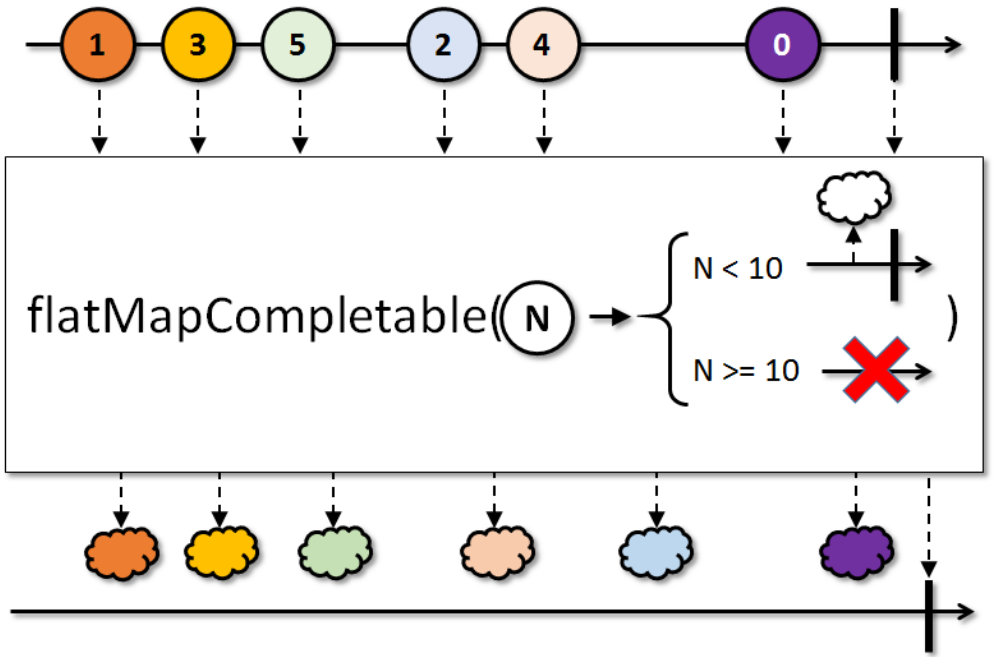
- The flatMapCompletable() method
  - "flatMaps" an Observable into a Completable, e.g.,
    - Maps each element of the current Observable into CompletableSources
    - Subscribes to them & waits for the completion of the upstream & all CompletableSources
    - Returns the new Completable instance

```
Completable flatMapCompletable
  (Function<? super T,
            ? extends
              CompletableSource>
   mapper))
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Completable.html

Applying Key Methods in the Observable Class to ex3

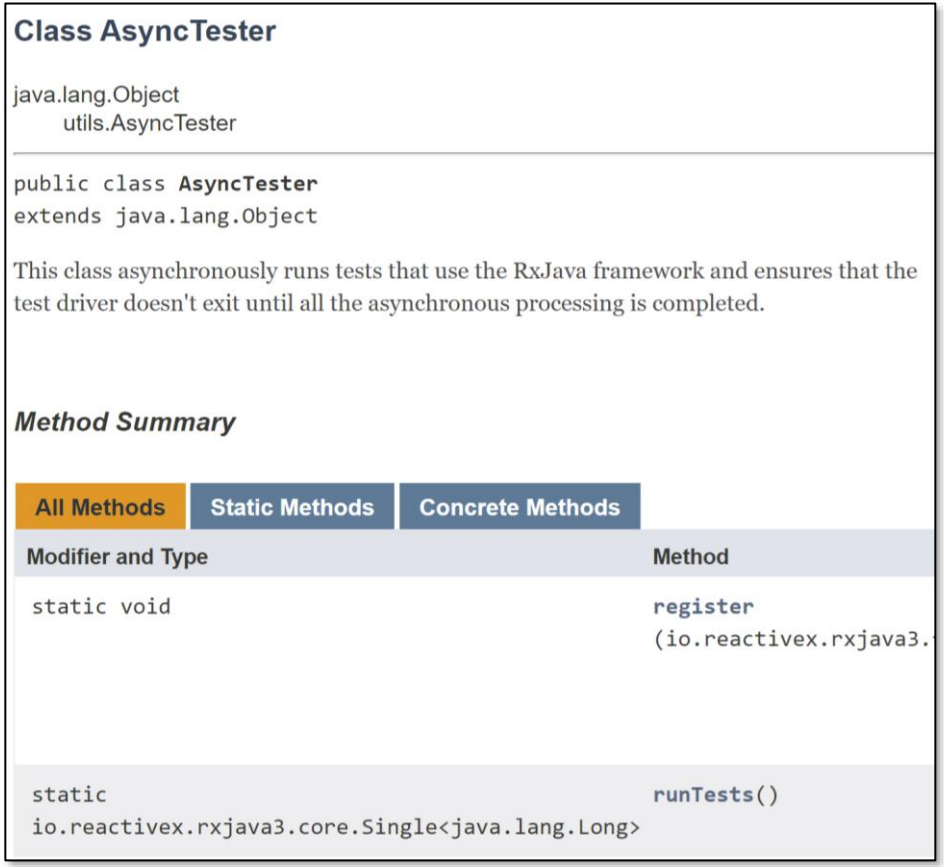
- The flatMapCompletable() method
  - "flatMaps" an Observable into a Completable
  - The Completable returned waits for the upstream's Observable terminal event (onComplete())



See medium.com/@daniel.rodak/combining-rxjava2-completable-with-observable-6dda410a3c83

Applying Key Methods in the Observable Class to ex3

- The flatMapCompletable() method
  - "flatMaps" an Observable into a Completable
  - The Completable returned waits for the upstream's Observable terminal event (onComplete())
    - Used to integrate with the AsyncTester framework



Class AsyncTester

java.lang.Object
    utils.AsyncTester

```
public class AsyncTester
extends java.lang.Object
```

This class asynchronously runs tests that use the RxJava framework and ensures that the test driver doesn't exit until all the asynchronous processing is completed.

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method
static void	register (io.reactivex.rxjava3.
static io.reactivex.rxjava3.core.Single<java.lang.Long>	runTests()

See Reactive/Single/ex3/src/main/java/utils/AsyncTester.java

Applying Key Methods in the Observable Class to ex3

- The flatMapCompletable() method
  - "flatMaps" an Observable into a Completable
  - The Completable returned waits for the upstream's Observable terminal event (onComplete())
    - Used to integrate with the AsyncTester framework
      - i.e., the Completable isn't triggered until all async processing is finished

```
return Observable
   .create(ObservableEx::bFEmitter)

   .flatMap(unreducedFraction ->
      reduceAndMultiplyFraction
        (unreducedFraction,
         Schedulers.computation()))

   .collectInto(new ArrayList
      <BigFraction>(), List::add)

   .flatMapCompletable(list ->
         BigFractionUtils
         .sortAndPrintList(list,
                           sb));
```

See Reactive/Single/ex3/src/main/java/utils/AsyncTester.java

Applying Key Methods in the Single Class to ex3

Applying Key Methods in the Single Class to ex3

- The ambArray() method
  - Runs multiple SingleSources & signals the events of the first one that signals

```
static <T> Single<T> ambArray
   (SingleSource<? extends T>...
    sources)
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Single.html#ambArray

Applying Key Methods in the Single Class to ex3

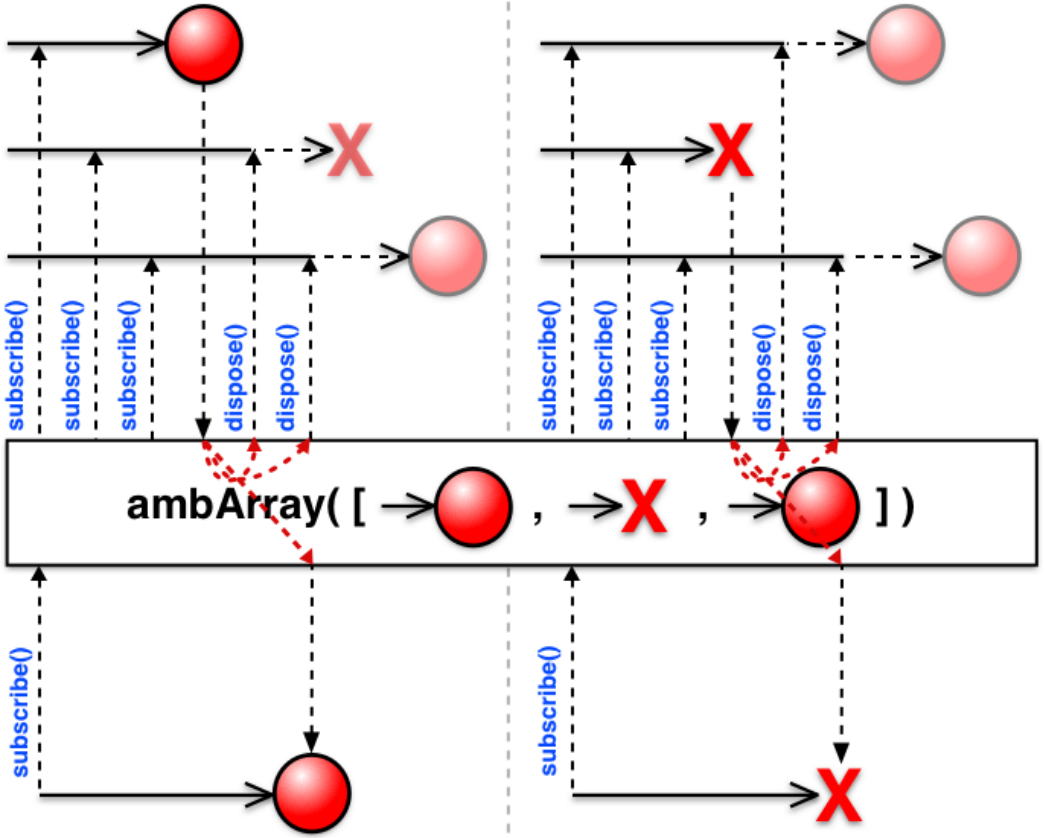
- The ambArray() method
  - Runs multiple SingleSources & signals the events of the first one that signals
  - This method picks the fastest of competing Single sources

```
static <T> Single<T> ambArray
   (SingleSource<? extends T>...
    sources)
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/SingleSource.html

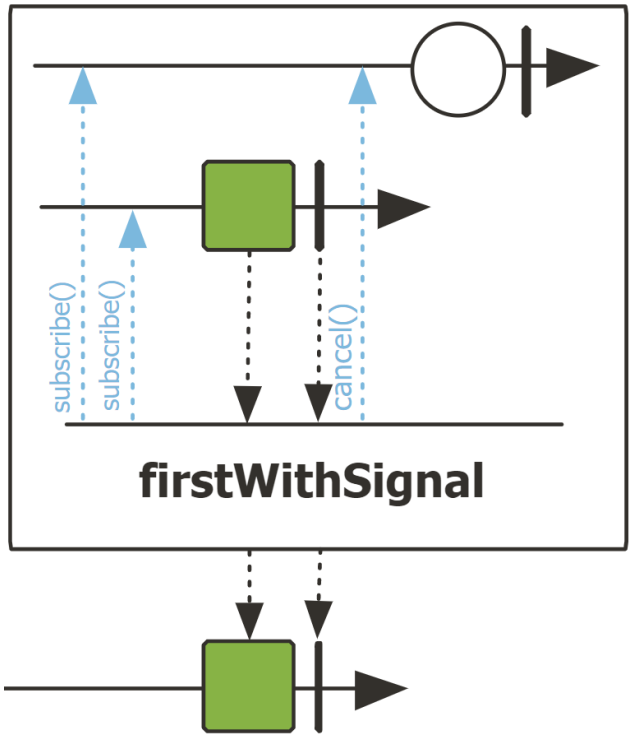
Applying Key Methods in the Single Class to ex3

- The ambArray() method
  - Runs multiple SingleSources & signals the events of the first one that signals
  - This method picks the fastest of competing Single sources
    - The rest are disposed of



Applying Key Methods in the Single Class to ex3

- The ambArray() method
  - Runs multiple SingleSources & signals the events of the first one that signals
  - This method picks the fastest of competing Single sources
  - Project Reactor's method Mono. firstWithSignal() works the same



See projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html#firstWithSignal

Applying Key Methods in the Single Class to ex3

- The ambArray() method
  - Runs multiple SingleSources & signals the events of the first one that signals
  - This method picks the fastest of competing Single sources
  - Project Reactor's method Mono. firstWithSignal() works the same
  - Similar to the Java Completable Future.anyOf() method

anyOf

```
public
static CompletableFuture<Object> anyOf(
CompletableFuture<?>... cfs)
```

Returns a new CompletableFuture that is completed when any of the given CompletableFutures complete, with the same result. Otherwise, if it completed exceptionally, the returned CompletableFuture also does so, with a CompletionException holding this exception as its cause. If no CompletableFutures are provided, returns an incomplete CompletableFuture.

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#anyOf

Applying Key Methods in the Single Class to ex3

- The ambArray() method
  - Runs multiple SingleSources & signals the events of the first one that signals
  - This method picks the fastest of competing Single sources
  - Project Reactor's method Mono. firstWithSignal() works the same
  - Similar to the Java Completable Future.anyOf() method
    - Also a generalization of Completable Future.applyToEither()

applyToEither

```
<U> CompletionStage<U> applyToEither(
CompletionStage<? extends T> other,
Function<? super T,U> fn)
```

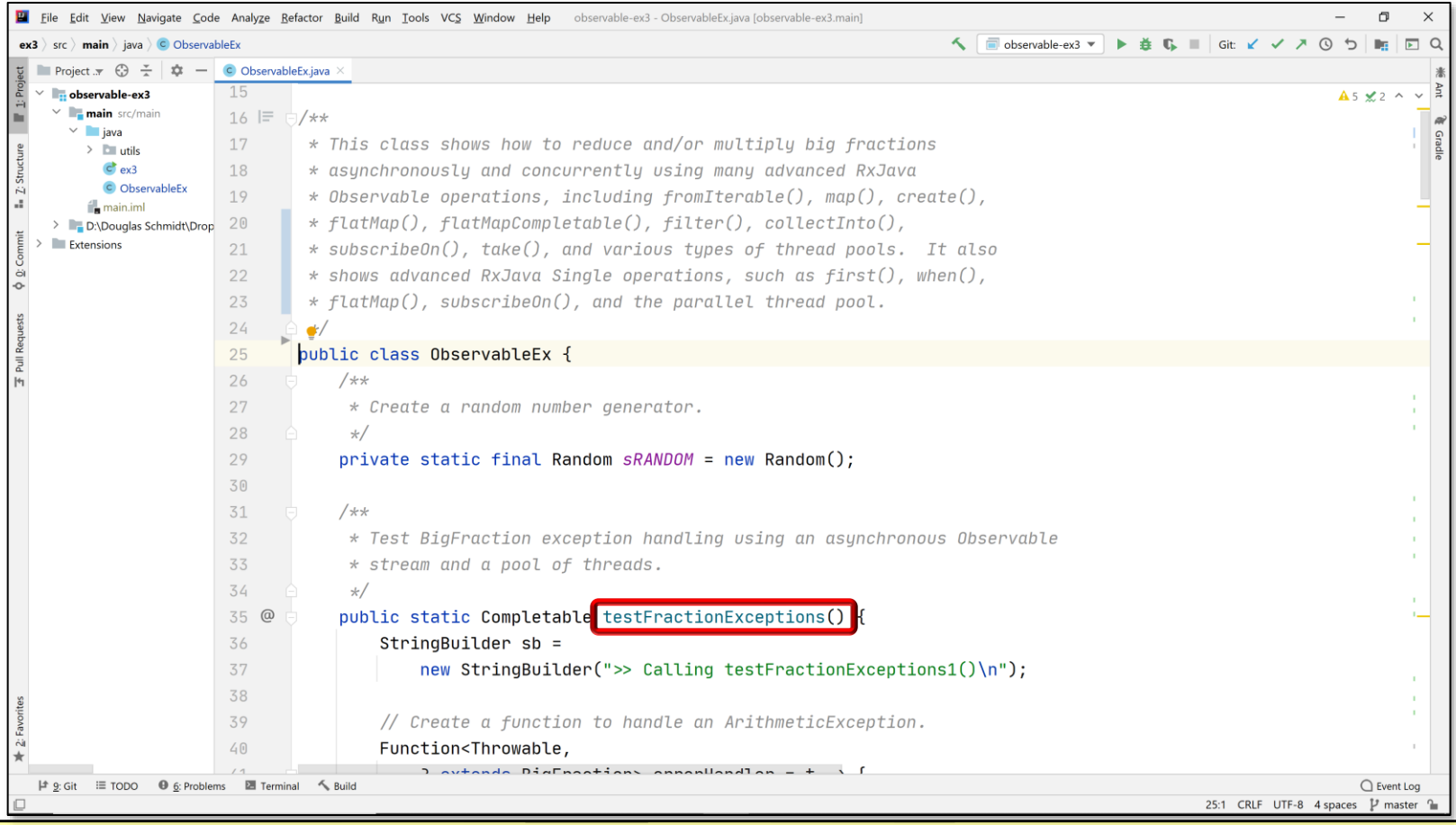
Returns a new CompletionStage that, when either this or the other given stage complete normally, is executed with the corresponding result as argument to the supplied function. See the CompletionStage documentation for rules covering exceptional completion.

Type Parameters:

U - the function's return type

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#applyToEither

Applying Key Methods in the Observable Class to ex3



See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/Observable/ex3

End of Applying Key Methods in the Observable Class (Part 4)