Key Transforming Operators in the Observable Class (Part 2)

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 Observable operators
  - Factory method operators
  - Transforming operators
    - Transform the values and/or types emitted by an Observable
      - e.g., flatMap() & flatMapCompletable()

Key Transforming Operators in the Observable Class

Key Transforming Operators in the Observable Class

- The flatMap() operator
  - 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

Key Transforming Operators in the Observable Class

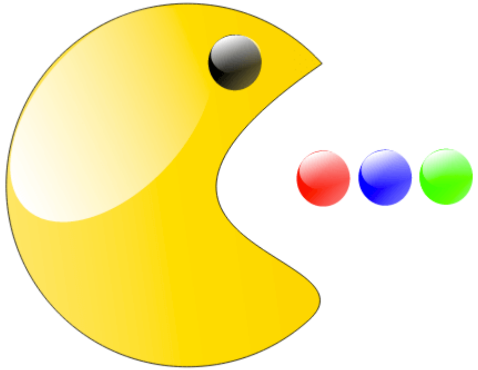
- The flatMap() operator
  - Transform the elements emitted by this Observable asynchronously
    - Items are emitted based on applying a function to each item emitted by this Observable

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

Key Transforming Operators in the Observable Class

- The flatMap() operator
  - 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
      - An ObservableSource can be consumed by an Observable

```
<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

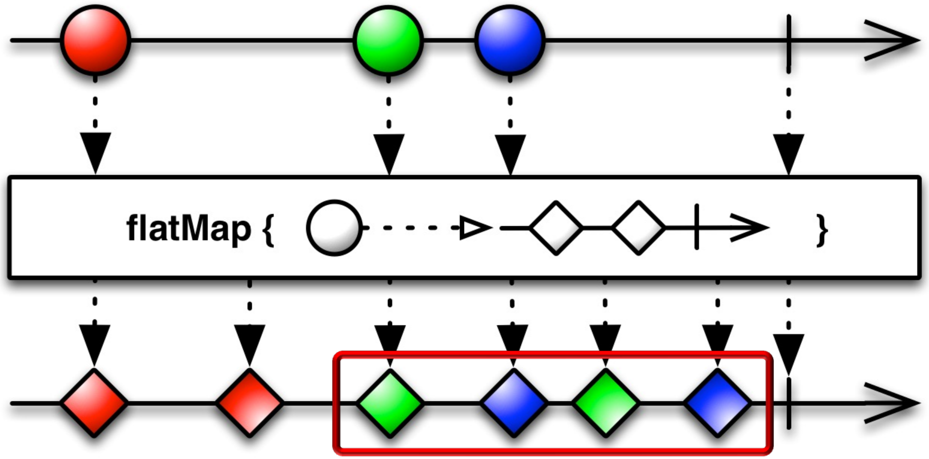
Key Transforming Operators in the Observable Class

- The flatMap() operator
  - 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 "flattened" & emitted

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

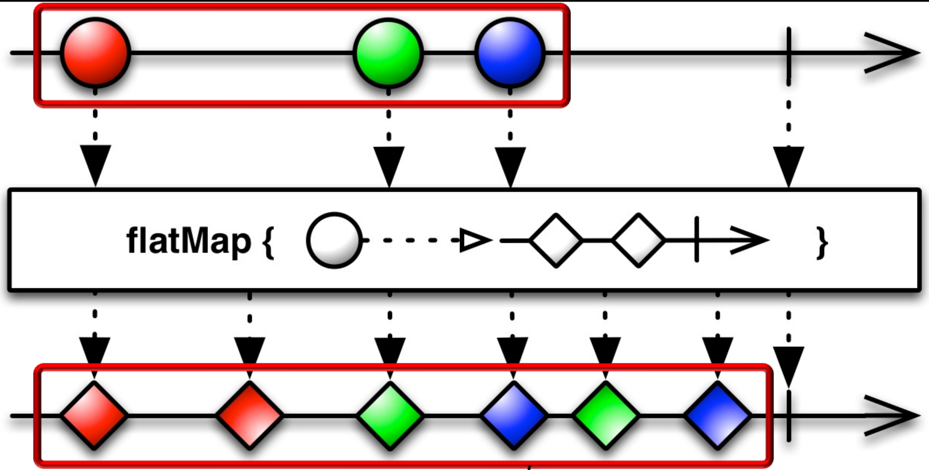
Key Transforming Operators in the Observable Class

- The flatMap() operator
  - 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 "flattened" & emitted
      - They thus can interleave



Key Transforming Operators in the Observable Class

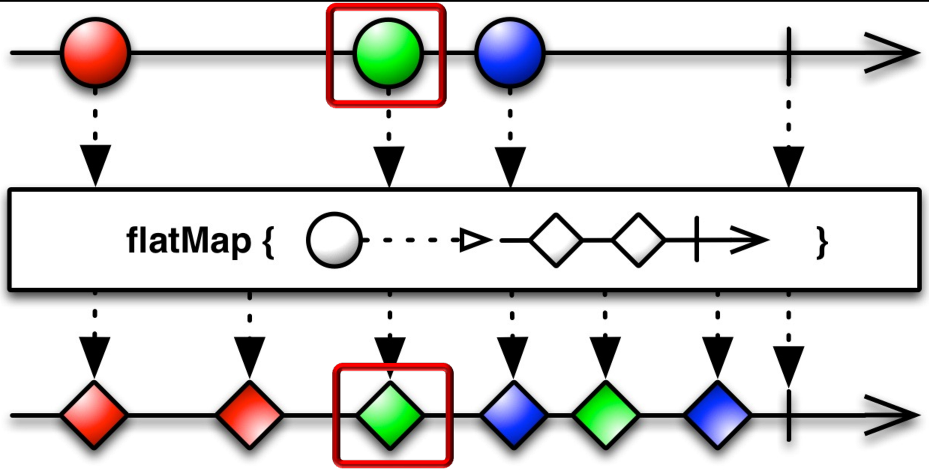
- The flatMap() operator
  - Transform the elements emitted by this Observable asynchronously
    - Items are emitted based on applying a function to each item emitted by this Observable
    - That function returns an ObservableSource
    - The returned ObservableSources are merged & the results of this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



The # of output elements may differ from the # of input elements

Key Transforming Operators in the Observable Class

- The flatMap() operator
  - 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 "flattened" & emitted
      - They thus can interleave



flatMap() can transform values and/ or types of elements it processes

Key Transforming Operators in the Observable Class

- The flatMap() operator
  - Transform the elements emitted by this Observable asynchronously
  - This operator is often used to trigger concurrent processing

```
return Observable
  .fromCallable(() -> BigFraction
       .reduce(unreducedFraction))

  .subscribeOn(scheduler)

  .flatMap(reducedFraction ->
           Observable
            .fromCallable(() ->
               reducedFraction
               .multiply
                 (sBigReducedFrac))
            .subscribeOn
              (scheduler));
```

See next part of the lesson on the RxJava flatMap() concurrency idiom

Key Transforming Operators in the Observable Class

- The flatMap() operator
  - Transform the elements emitted by this Observable asynchronously
  - This operator is often used to trigger concurrent processing

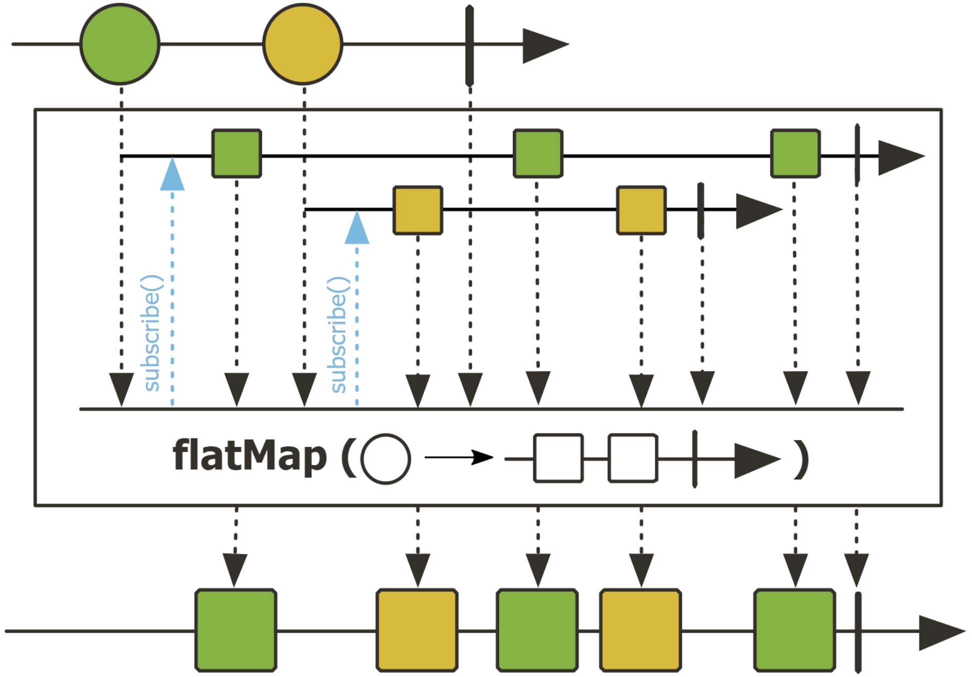
```
return Observable
   .fromIterable(bigFractionList)

   .flatMap(bf -> Observable
      .fromCallable(() -> bf
         .multiply(sBigFraction))

      .subscribeOn
         (Schedulers
          .computation()))

   .reduce(BigFraction::add)
```

Return an Observable that emits multiplied BigFraction objects via the RxJava flatMap() concurrency idiom

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

Key Transforming Operators in the Observable Class

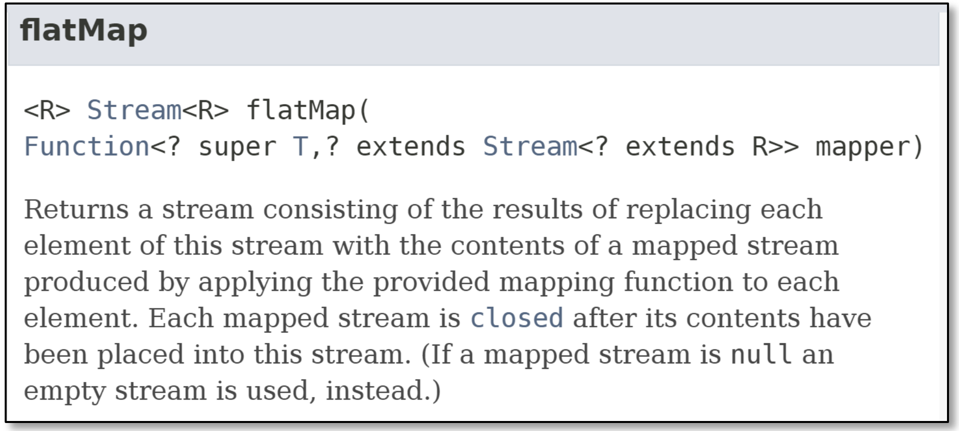
- The flatMap() operator
  - Transform the elements emitted by this Observable asynchronously
  - This operator is often used to trigger concurrent processing
  - Project Reactor's Flux.flatMap() operator works the same way



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

Key Transforming Operators in the Observable Class

- The flatMap() operator
  - Transform the elements emitted by this Observable asynchronously
  - This operator is often used to trigger concurrent processing
  - Project Reactor's Flux.flatMap() operator 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.)

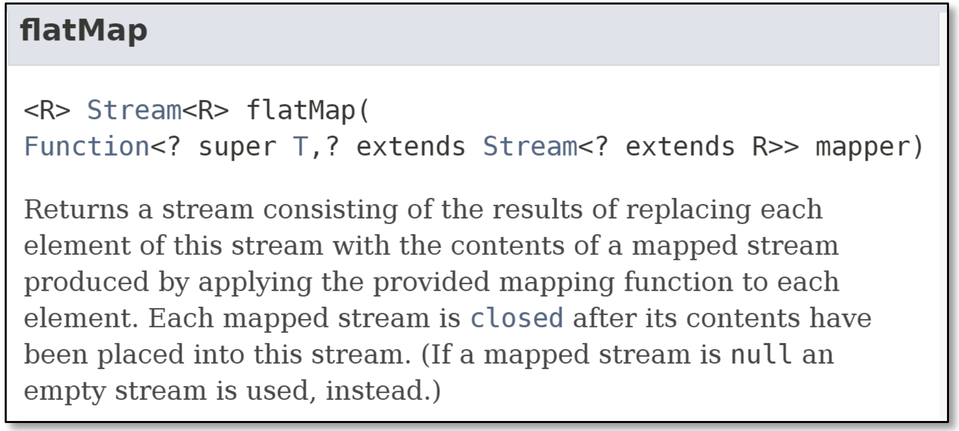
```
List<String> a = List.of("d", "g");
List<String> b = List.of("a", "c");
Stream
     .of(a, b)
     .flatMap(List::stream)
     .sorted()
     .forEach(System.out::println);
```

Flatten, sort, & print two lists of strings

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

Key Transforming Operators in the Observable Class

- The flatMap() operator
  - Transform the elements emitted by this Observable asynchronously
  - This operator is often used to trigger concurrent processing
  - Project Reactor's Flux.flatMap() operator works the same way
  - Similar to the Stream.flatMap() method in Java Streams
    - However, Stream.flatMap() doesn't support parallelism..



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.)

```
List<String> a = List.of("d", "g");
List<String> b = List.of("a", "c");
Stream
    .of(a, b).parallel()
    .flatMap(List::stream)
    .sorted()
    .forEach(System.out::println);
```

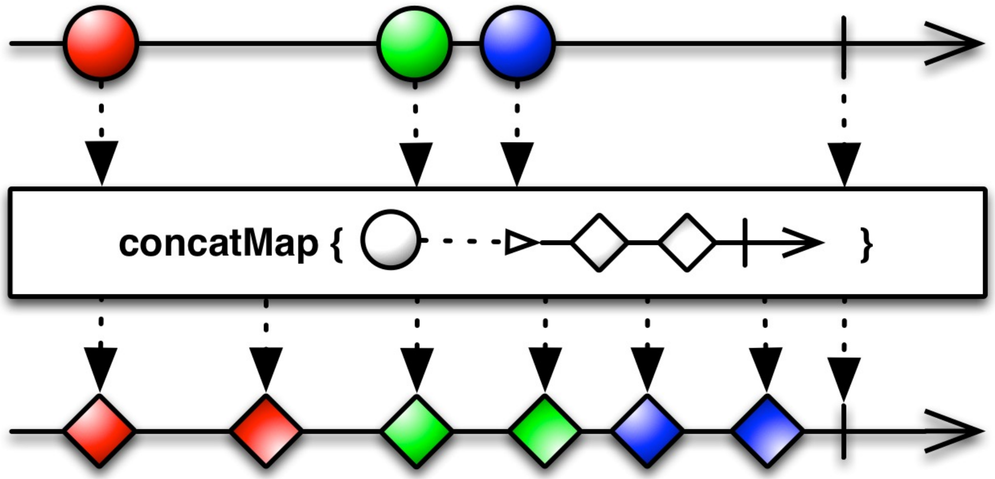
See stackoverflow.com/questions/45038120/parallel-flatmap-always-sequential/66386078

Key Transforming Operators in the Observable Class

- The flatMap() operator
  - Transform the elements emitted by this Observable asynchronously
  - This operator is often used to trigger concurrent processing
  - Project Reactor's Flux.flatMap() operator works the same way
  - Similar to the Stream.flatMap() method in Java Streams
  - flatMap() doesn't ensure the order of the items in the resulting stream

Key Transforming Operators in the Observable Class

- The flatMap() operator
  - Transform the elements emitted by this Observable asynchronous
  - This operator is often used to trigger concurrent processing
  - Project Reactor's Flux.flatMap() operator works the same way
  - Similar to the Stream.flatMap() method in Java Streams
  - 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

Key Transforming Operators in the Observable Class

- The flatMapCompletable() operator
  - "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

Key Transforming Operators in the Observable Class

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

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

Key Transforming Operators in the Observable Class

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

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

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

Key Transforming Operators in the Observable Class

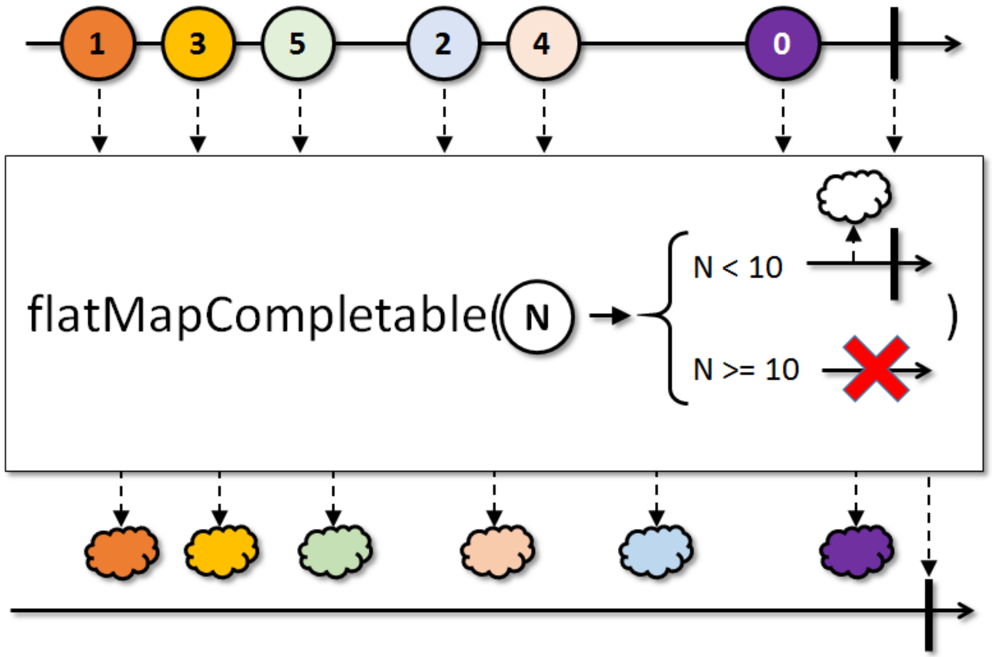
- The flatMapCompletable() operator
  - "flatMaps" an Observable into a Completable, e.g.,
    - Maps each element of the current Observable into CompletableSource objects
    - Subscribes to them & waits for the completion of the upstream & all CompletableSource objects
    - Returns the new Completable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

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

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

Key Transforming Operators in the Observable Class

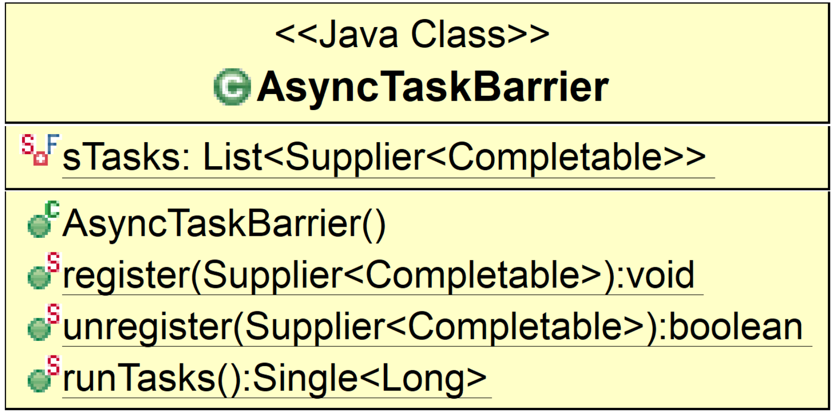
- The flatMapCompletable() operator
  - "flatMaps" an Observable into a Completable
  - The Completable returned waits for the upstream's Observable terminal event (onComplete())



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

Key Transforming Operators in the Observable Class

- The flatMapCompletable() operator
  - "flatMaps" an Observable into a Completable
  - The Completable returned waits for the upstream's Observable terminal event (onComplete())
    - Used to integrate w/the RxJava AsyncTaskBarrier framework



<<Java Class>>
AsyncTaskBarrier

sTasks: List<Supplier<Completable>>

AsyncTaskBarrier()
register(Supplier<Completable>):void
unregister(Supplier<Completable>):boolean
runTasks():Single<Long>

See Reactive/Observable/ex3/src/main/java/utils/AsyncTaskBarrier.java

Key Transforming Operators in the Observable Class

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

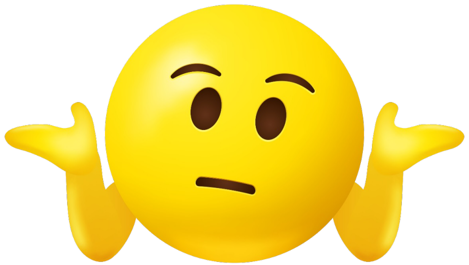
```
Observable
    .fromIterable(sTasks)

    .map(Supplier::get)

    .flatMapCompletable(c -> c)

    .toSingleDefault((long)
                         sTasks.size());
```

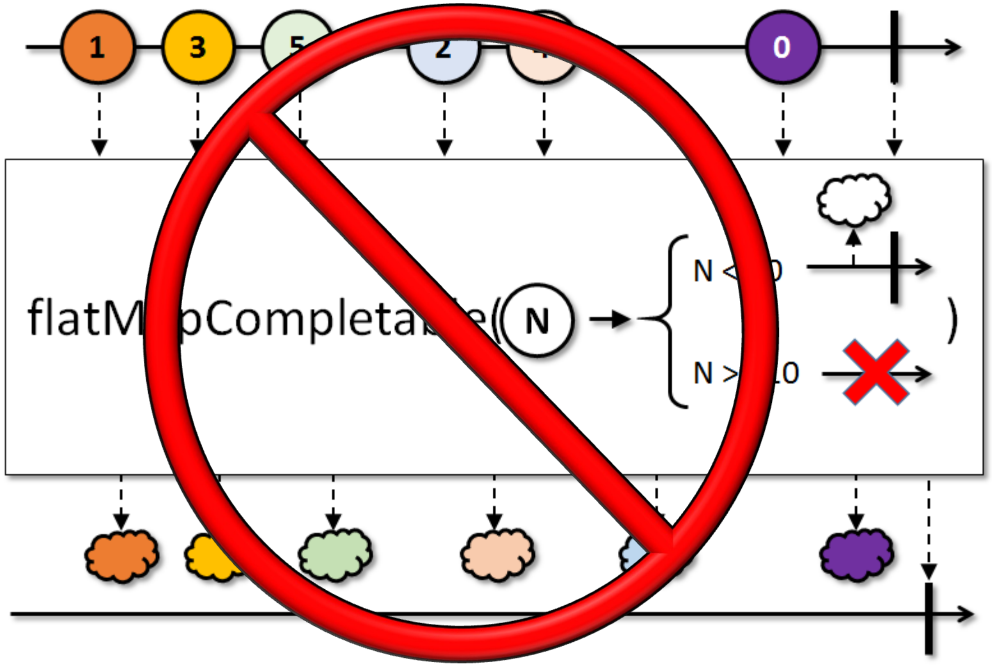
Map each Observable element into a CompletableSource, subscribes to them, & wait until the upstream & all CompletableSource objects complete

See Reactive/Observable/ex3/src/main/java/utils/AsyncTaskBarrier.java

Key Transforming Operators in the Observable Class

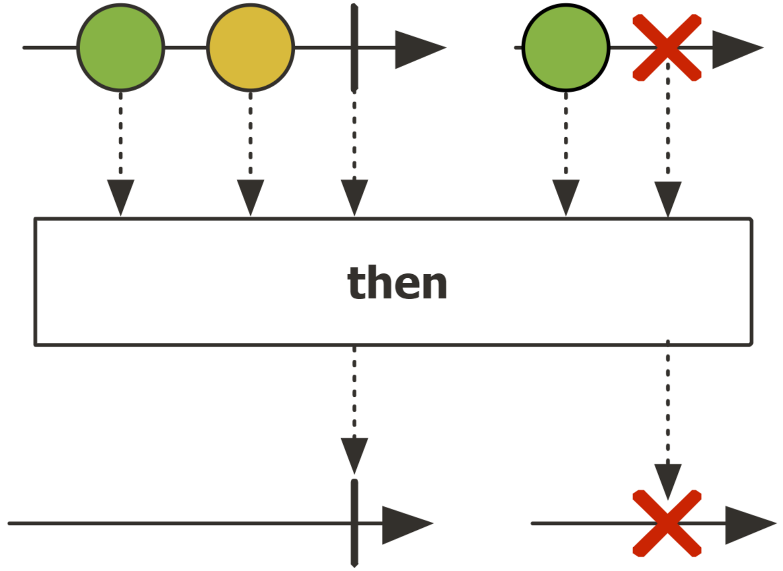
- The flatMapCompletable() operator
  - "flatMaps" an Observable into a Completable
  - The Completable returned waits for the upstream's Observable terminal event (onComplete())
  - Project Reactor has no operator like flatMapCompletable()





Key Transforming Operators in the Observable Class

- The flatMapCompletable() operator
  - "flatMaps" an Observable into a Completable
  - The Completable returned waits for the upstream's Observable terminal event (onComplete())
  - Project Reactor has no operator like flatMapCompletable()
    - However, Project Reactor's Flux. then() & Mono.then() operators provide a similar capability when used in conjunction with flatMap()



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

Key Transforming Operators in the Observable Class

- The flatMapCompletable() operator
  - "flatMaps" an Observable into a Completable
  - The Completable returned waits for the upstream's Observable terminal event (onComplete())
  - Project Reactor has no operator like flatMapCompletable()
    - However, Project Reactor's Flux. then() & Mono.then() operators provide a similar capability when used in conjunction with flatMap()
      - Used to integrate w/the Project Reactor AsyncTaskBarrier framework

```
Flux
    .fromIterable(sTasks)

    .flatMap(Supplier::get)

    .collectList()

    .onErrorContinue(errorHandler)

    .flatMap(__ -> ...);
```

See Reactive/flux/ex3/src/main/java/utils/AsyncTaskBarrier.java

Key Transforming Operators in the Observable Class

- The flatMapCompletable() operator
  - "flatMaps" an Observable into a Completable
  - The Completable returned waits for the upstream's Observable terminal event (onComplete())
  - Project Reactor has no operator like flatMapCompletable()
  - The CompletableFuture.allOf() method can be combined with the Java Streams collector framework for a similar effect

```
Stream
  .generate(() ->
     makeBigFraction
       (new Random(), false))

  .limit(sMAX_FRACTIONS)

  .map(reduceAndMultiplyFraction)

  .collect(FuturesCollector
             .toFuture())

  .thenAccept
     (this::sortAndPrintList);
```

See Java8/ex19/src/main/java/utils/FuturesCollector.java

End of Key Transforming Operators in the Observable Class (Part 2)