Applying Key Methods in the Observable Class (Part 7)

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 multiply & reduce Big Fraction objects
  - e.g., fromIterable(), map(), just(), flatMap(), reduce(), doOnSuccess(), ignoreElement(), subscribeOn(), & Schedulers.computation()

```
return Observable
        .fromIterable(bigFractions)

        .flatMap(bf -> Observable
                 .just(bf)
                 .subscribeOn
                   (Schedulers
                    .computation())
                 .map(multiplyFracs))

        .reduce(BigFraction::add)

        .doOnSuccess(displayResults)

        .ignoreElement();
```

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

- testFractionMultiplications2()
  - Use an asynchronous Observable stream & a pool of threads to multiply & add BigFractions

```
return Observable
        .fromIterable(bigFractions)

        .flatMap(bf -> Observable
                 .just(bf)
                 .subscribeOn
                   (Schedulers
                    .computation())
                 .map(multiplyFracs))

        .reduce(BigFraction::add)

        .doOnSuccess(displayResults)

        .ignoreElement();
```

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

Applying Key Methods in the Observable Class to ex3

- testFractionMultiplications2()
  - Use an asynchronous Observable stream & a pool of threads to multiply & add BigFractions
  - Demonstrates Observable methods
    - e.g., fromIterable(), map(), just(), flatMap(), reduce(), ignoreElement(), subscribeOn(), & Schedulers .computation() methods

```
return Observable
        .fromIterable(bigFractions)

        .flatMap(bf -> Observable
                 .just(bf)
                 .subscribeOn
                   (Schedulers
                    .computation())
                 .map(multiplyFracs))

        .reduce(BigFraction::add)

        .doOnSuccess(displayResults)

        .ignoreElement();
```

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

Applying Key Methods in the Observable Class to ex3

- testFractionMultiplications2()
  - Use an asynchronous Observable stream & a pool of threads to multiply & add BigFractions
  - Demonstrates Observable methods
    - e.g., fromIterable(), map(), just(), flatMap(), reduce(), ignoreElement(), subscribeOn(), & Schedulers .computation() methods

It also illustrates how to apply the flatMap() concurrency idiom

```
return Observable
        .fromIterable(bigFractions)

        .flatMap(bf -> Observable
                       .just(bf)
                       .subscribeOn
                         (Schedulers
                          .computation())
                       .map(multiplyFracs))

        .reduce(BigFraction::add)

        .doOnSuccess(displayResults)

        .ignoreElement();
```

Applying Key Methods in the Observable Class to ex3

- testFractionMultiplications2()
  - Use an asynchronous Observable stream & a pool of threads to multiply & add BigFractions
  - Demonstrates Observable methods
  - Also demonstrates a Maybe method
    - e.g., doOnSuccess()

```
return Observable
        .fromIterable(bigFractions)

        .flatMap(bf -> Observable
                  .just(bf)
                  .subscribeOn
                    (Schedulers
                     .computation())
                  .map(multiplyFracs))

        .reduce(BigFraction::add)

        .doOnSuccess(displayResults)

        .ignoreElement();
```

Applying Key Methods in the Observable Class to ex3

- The reduce() method
  - Returns a Maybe that applies an accumulator function to the 1st item emitted by current Observable

```
Maybe<U> reduce
   (BiFunction<T, T, T> reducer)
```

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

Applying Key Methods in the Observable Class to ex3

- The reduce() method
  - Returns a Maybe that applies an accumulator function to the 1st item emitted by current Observable
    - The result of that function is then fed along with the second item emitted by the current Observable into the same function

```
Maybe<U> reduce
   (BiFunction<T, T, T> reducer)
```

Applying Key Methods in the Observable Class to ex3

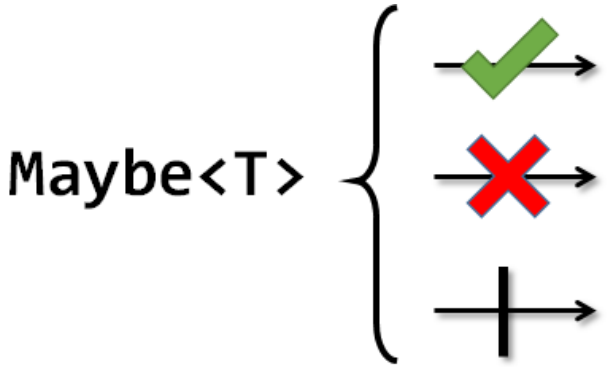
- The reduce() method
  - Returns a Maybe that applies an accumulator function to the 1st item emitted by current Observable
    - The result of that function is then fed along with the second item emitted by the current Observable into the same function
      - This continues until all items have been emitted by the current & finite Observable

```
Maybe<U> reduce
   (BiFunction<T, T, T> reducer)
```

Applying Key Methods in the Observable Class to ex3

- The reduce() method
  - Returns a Maybe that applies an accumulator function to the 1st item emitted by current Observable
    - The result of that function is then fed along with the second item emitted by the current Observable into the same function
  - The final result is emitted from the final call as the sole item of a Maybe

```
Maybe<U> reduce
   (BiFunction<T, T, T> reducer)
```



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

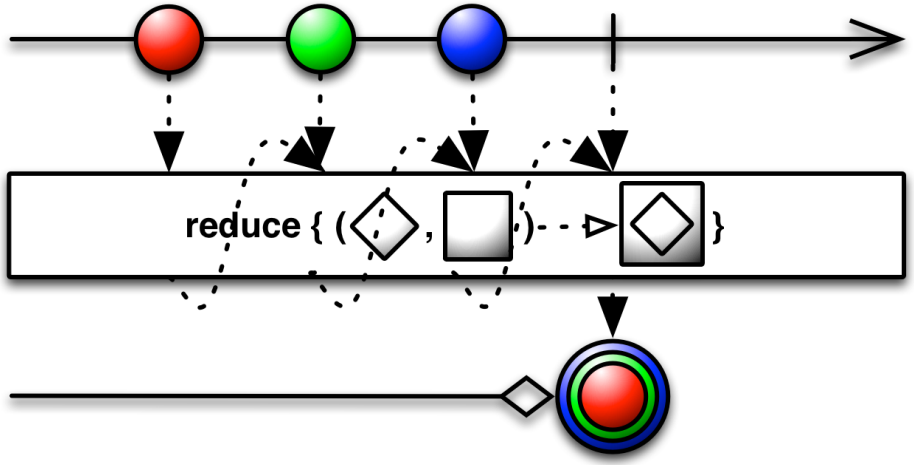
Applying Key Methods in the Observable Class to ex3

- The reduce() method
  - Returns a Maybe that applies an accumulator function to the 1st item emitted by current Observable
    - The result of that function is then fed along with the second item emitted by the current Observable into the same function
  - The final result is emitted from the final call as the sole item of a Maybe
    - If there are no items emitted by the Observable the Maybe will be empty

```
Maybe<U> reduce
   (BiFunction<T, T, T> reducer)
```

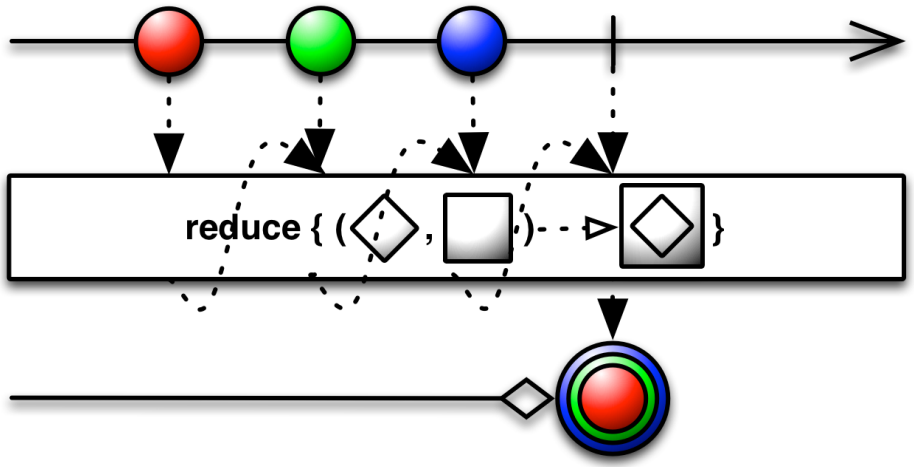
Applying Key Methods in the Observable Class to ex3

- The reduce() method
  - Returns a Maybe that applies an accumulator function to the 1st item emitted by current Observable
  - This operator requires the upstream to signal onComplete() before the accumulator object can be emitted



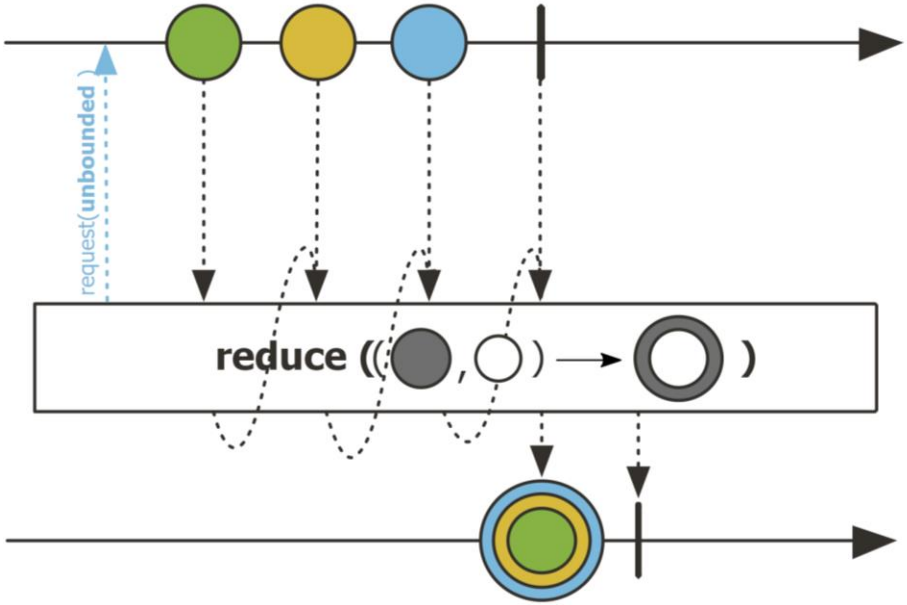
Applying Key Methods in the Observable Class to ex3

- The reduce() method
  - Returns a Maybe that applies an accumulator function to the 1st item emitted by current Observable
  - This operator requires the upstream to signal onComplete() before the accumulator object can be emitted
    - Sources that are infinite & never complete will never emit anything through this operator
      - An infinite source may lead to a fatal OutOfMemoryError



Applying Key Methods in the Observable Class to ex3

- The reduce() method
  - Returns a Maybe that applies an accumulator function to the 1st item emitted by current Observable
  - This operator requires the upstream to signal onComplete() before the accumulator object can be emitted
  - Project Reactor's Flux.reduce() method works the same



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

Applying Key Methods in the Observable Class to ex3

- The reduce() method
  - Returns a Maybe that applies an accumulator function to the 1st item emitted by current Observable
  - This operator requires the upstream to signal onComplete() before the accumulator object can be emitted
  - Project Reactor's Flux.reduce() method works the same
  - Similar to the Stream.reduce() method in Java Streams

reduce

`Optional<T> reduce(BinaryOperator<T> accumulator)`

Performs a reduction on the elements of this stream, using an associative accumulation function, and returns an `Optional` describing the reduced value, if any. This is equivalent to:

```
boolean foundAny = false;
T result = null;
for (T element : this stream) {
    if (!foundAny) {
        foundAny = true;
        result = element;
    }
    else
        result = accumulator.apply(result, element);
}
return foundAny ? Optional.of(result) : Optional.empty();
```

but is not constrained to execute sequentially.

The `accumulator` function must be an associative function.

This is a terminal operation.

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

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"

```
return Observable
        .fromIterable(bigFractions)

        .flatMap(bf -> Observable
                       .just(bf)
                       .subscribeOn
                         (Schedulers
                          .computation())
                       .map(multiplyFracs))

        .reduce(BigFraction::add)

        .doOnSuccess(displayResults)

        .ignoreElement();
```

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

Applying Key Methods in the Observable Class to ex3

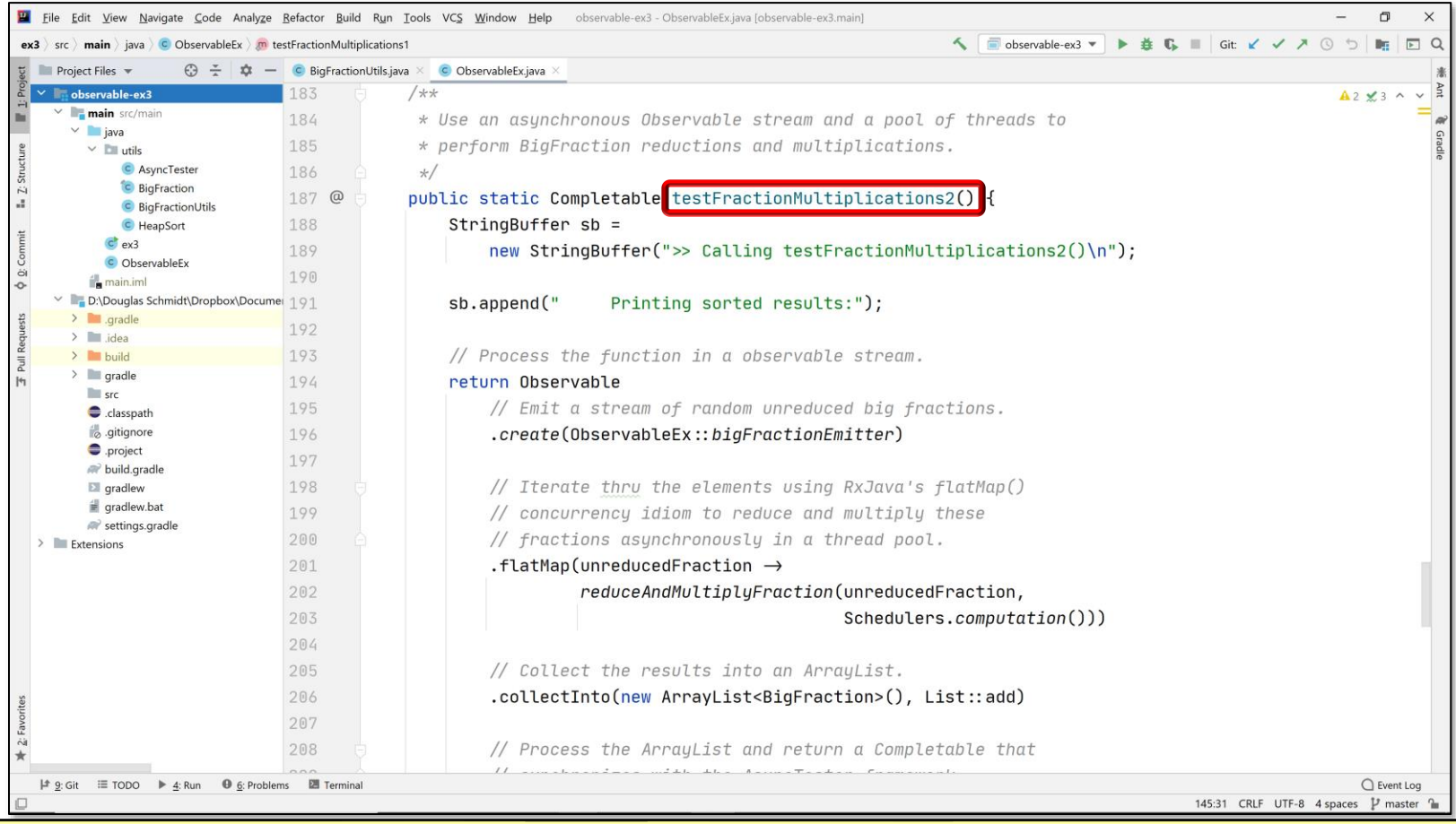
- flatMap() is often used when each item emitted by a stream needs to have its own threading operators applied to it
  - i.e., the "flatMap() concurrency idiom"

```
return Observable
        .fromIterable(bigFractions)

        .flatMap(bf -> Observable
                       .just(bf)
                       .subscribeOn
                         (Schedulers
                          .computation())
                       .map(multiplyFracs))

        .reduce(BigFraction::add)

        .doOnSuccess(displayResults)

        .ignoreElement();
```

Each instance of this inner chain runs in a background thread in the computation thread pool

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

Applying Key Methods in the Observable Class to ex3



```
/**
 * Use an asynchronous Observable stream and a pool of threads to
 * perform BigFraction reductions and multiplications.
 */
public static Completable testFractionMultiplications2() {
    StringBuffer sb =
        new StringBuffer(">> Calling testFractionMultiplications2()\n");

    sb.append("     Printing sorted results:");

    // Process the function in a observable stream.
    return Observable
        // Emit a stream of random unreduced big fractions.
        .create(ObservableEx::bigFractionEmitter)

        // Iterate thru the elements using RxJava's flatMap()
        // concurrency idiom to reduce and multiply these
        // fractions asynchronously in a thread pool.
        .flatMap(unreducedFraction ->
                reduceAndMultiplyFraction(unreducedFraction,
                                          Schedulers.computation()))

        // Collect the results into an ArrayList.
        .collectInto(new ArrayList<BigFraction>(), List::add)

        // Process the ArrayList and return a Completable that
```

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

End of Applying Key Methods in the Observable Class (Part 7)