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

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/douglas-craigschmidt/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

```java
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

```java
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
• 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();

It also illustrates how to apply the flatMap() concurrency idiom
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 1\textsuperscript{st} item emitted by current Observable

\[
\text{Maybe}\langle U \rangle \ \text{reduce} \\
\quad (\text{BiFunction}\langle T, T, T \rangle \ \text{reducer})
\]

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

```java
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 1\textsuperscript{st} 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
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`

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Maybe.html](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 1\textsuperscript{st} 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

\[
\text{Maybe}<U> \text{ reduce } (\text{BiFunction}<T, T, T> \text{ reducer})
\]
Applying Key Methods in the Observable Class to ex3

- The `reduce()` method
  - Returns a Maybe that applies an accumulator function to the 1\textsuperscript{st} 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

```java
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:

```java
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](docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#reduce)
• 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”

```java
return Observable.fromIterable(bigFractions)
    .flatMap(bf -> Observable.just(bf)
        .subscribeOn(Schedulers.computation())
        .map(multiplyFracs))
    .reduce(BigFraction::add)
    .doOnSuccess(displayResults)
    .ignoreElement();
```

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 Reactive/Observable/ex3/src/main/java/ObservableEx.java
Applying Key Methods in the Observable Class to ex3

```java
/**
 * 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
        .create(ObservableEx::bigFractionEmitter)
        // Emit a stream of random unreduced big fractions.
        .map(bigDecimal -> reduceAndMultiplyFraction(bigDecimal, Schedulers.computation()))
        // Iterate thru the elements using RxJava's flatMap()
        .flatMap(unreducedFraction ->
            reduceAndMultiplyFraction(unreducedFraction, Schedulers.computation()))
        // Collect the results into an ArrayList.
        .collectInto(new ArrayList<>(), List::add)
        // Process the ArrayList and return a Completable that
```
End of Applying Key Methods in the Observable Class (Part 7)