Applying Key Methods in the Observable Class (Part 6)

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., collect(), take(), flatMap(), range(), flatMapCompletable(), subscribeOn(), generate(), & Schedulers.computation()

```
return Observable
   .generate(emitter)
   .take(sBIG_FRACTIONS)

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

   .collect(ArrayList<BigFraction>
            ::new, 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

- testFractionMultiplications1()
  - Use an asynchronous Observable stream & a pool of threads to perform BigFraction object reductions & multiplications

```
return Observable
   .generate(emitter)
   .take(sBIG_FRACTIONS)

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

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

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

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

Applying Key Methods in the Observable Class to ex2

- testFractionMultiplications1()
  - Use an asynchronous Observable stream & a pool of threads to perform BigFraction object reductions & multiplications
  - Demonstrates generate(), take(), flatMap(), collect(), filter(), flatMapCompletable(), range(), subscribeOn(), & Schedulers .computation()

```
return Observable
   .generate(emitter)
   .take(sBIG_FRACTIONS)

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

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

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

Applying Key Methods in the Observable Class to ex3

- The generate() method
  - Returns a cold, synchronous, & stateless generator of values

```
static <T> Observable<T> generate
  (Callable<Emitter<T> generator)
```

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

Applying Key Methods in the Observable Class to ex3

- The generate() method
  - Returns a cold, synchronous, & stateless generator of values
    - The param is called in a loop after a downstream Observer has subscribed
      - The callback should call onNext(), onError(), or onComplete() to signal a value or a terminal event

```
static <T> Observable<T> generate
  (Callable<Emitter<T> generator)
```

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

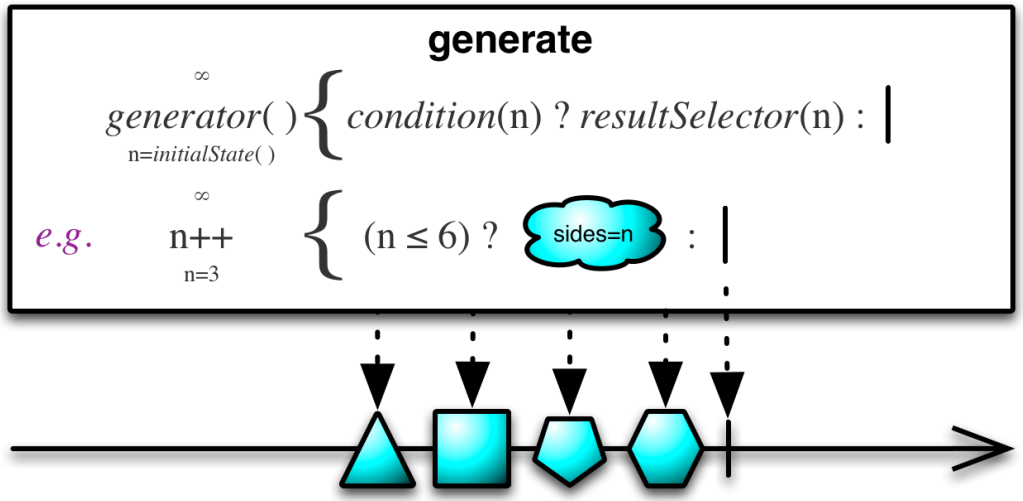
Applying Key Methods in the Observable Class to ex3

- The generate() method
  - Returns a cold, synchronous, & stateless generator of values
    - The param is called in a loop after a downstream Observer has subscribed
    - The new Observable instance is returned

```
static <T> Observable<T> generate
  (Callable<Emitter<T> generator)
```

Applying Key Methods in the Observable Class to ex3

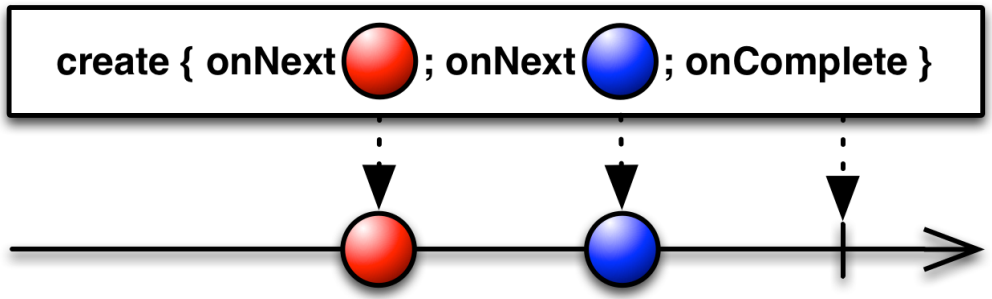
- The generate() method
  - Returns a cold, synchronous, & stateless generator of values
  - It is only allowed to generate one event at a time, which supports backpressure



See www.baeldung.com/rxjava-backpressure

Applying Key Methods in the Observable Class to ex3

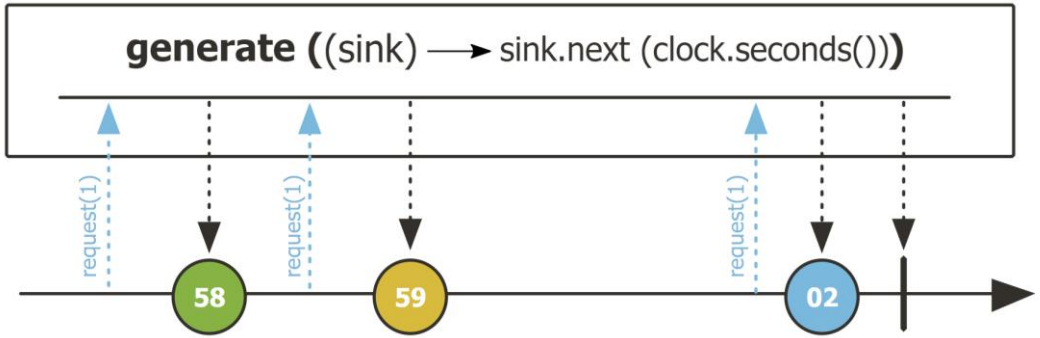
- The generate() method
  - Returns a cold, synchronous, & stateless generator of values
  - It is only allowed to generate one event at a time, which supports backpressure
    - In contrast, create() simply produces events whenever it wishes to do so
      - i.e., it ignores backpressure



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

Applying Key Methods in the Observable Class to ex3

- The generate() method
  - Returns a cold, synchronous, & stateless generator of values
  - It is only allowed to generate one event at a time, which supports backpressure
  - Project Reactor's Flux.generate() works the same



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

Applying Key Methods in the Observable Class to ex3

- The generate() method
  - Returns a cold, synchronous, & stateless generator of values
  - It is only allowed to generate one event at a time, which supports backpressure
  - Project Reactor's Flux.generate() works the same
  - Similar to the Stream.generate() method in Java Streams

generate

```
static <T> Stream<T> generate(Supplier<T> s)
```

Returns an infinite sequential unordered stream where each element is generated by the provided `Supplier`. This is suitable for generating constant streams, streams of random elements, etc.

Type Parameters:

`T - the type of stream elements`

Parameters:

`s - the Supplier of generated elements`

Returns:

`a new infinite sequential unordered Stream`

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

Applying Key Methods in the Observable Class to ex3

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

```
Single<U> collect
   (Supplier<? extends U>
      initialItemSupplier,
    BiConsumer<? super U, ? super T>
      collector)
```

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

Applying Key Methods in the Observable Class to ex3

- The collect() 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> collect
   (Supplier<? extends U>
      initialItemSupplier,
    BiConsumer<? super U, ? super T>
      collector)
...
.collect
   (ArrayList<BigFraction>::new,
    List::add)
...
```

Applying Key Methods in the Observable Class to ex3

- The collect() 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> collect
   (Supplier<? extends U>
      initialItemSupplier,
    BiConsumer<? super U, ? super T>
      collector)
...
.collect
   (ArrayList<BigFraction>::new,
    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

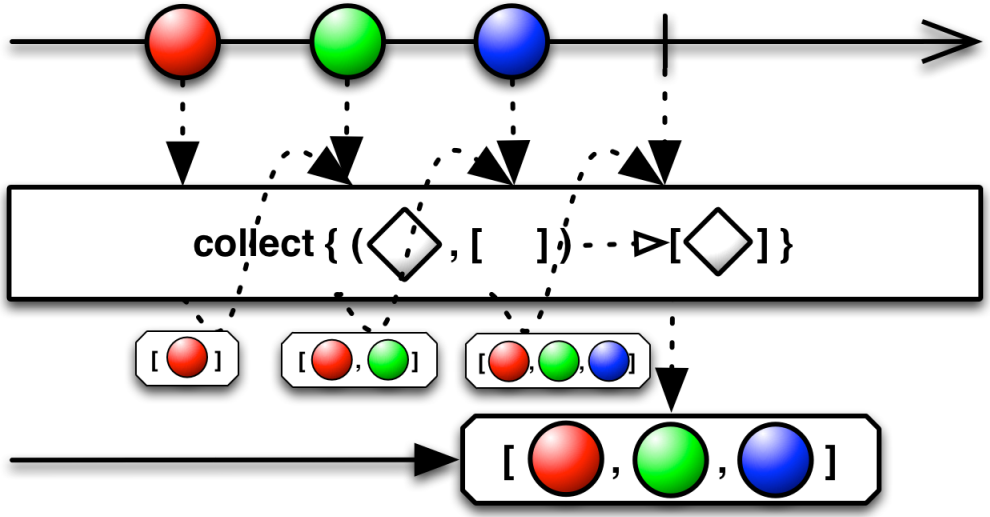
- The collect() method
  - Collects items emitted by the finite source Observable into a single mutable data structure
    - The 1st param is the mutable data structure that accumulates (collects) the items
    - The 2nd param is a bi-consumer that accepts the accumulator & an emitted item
    - Returns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

```
Single<U> collect
   (Supplier<? extends U>
      initialItemSupplier,
    BiConsumer<? super U, ? super T>
      collector)
```

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

Applying Key Methods in the Observable Class to ex3

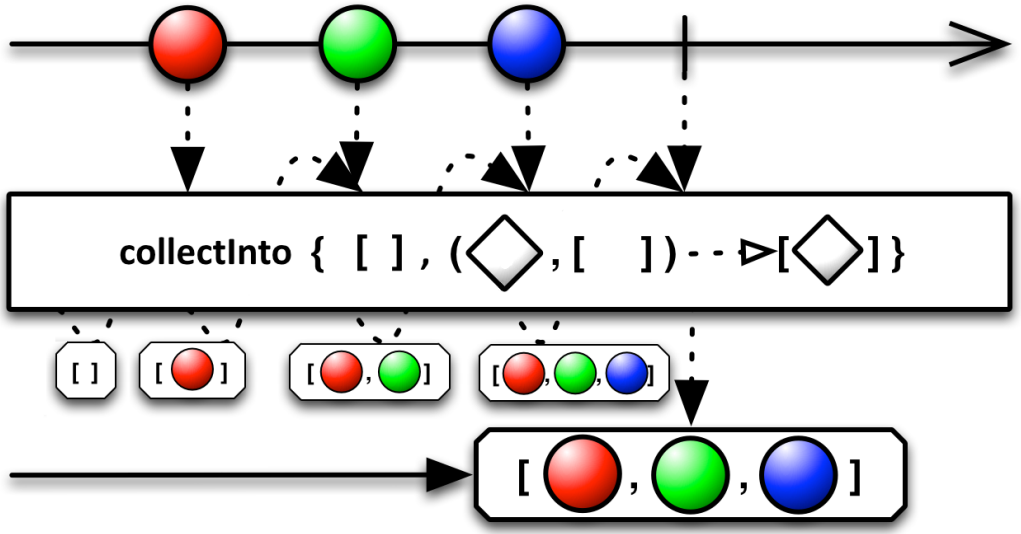
- The collect() method
  - Collects items emitted by the finite source Observable into a single mutable data structure
  - This method is a simplified version of reduce() that does not need to return the state on each pass



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

Applying Key Methods in the Observable Class to ex3

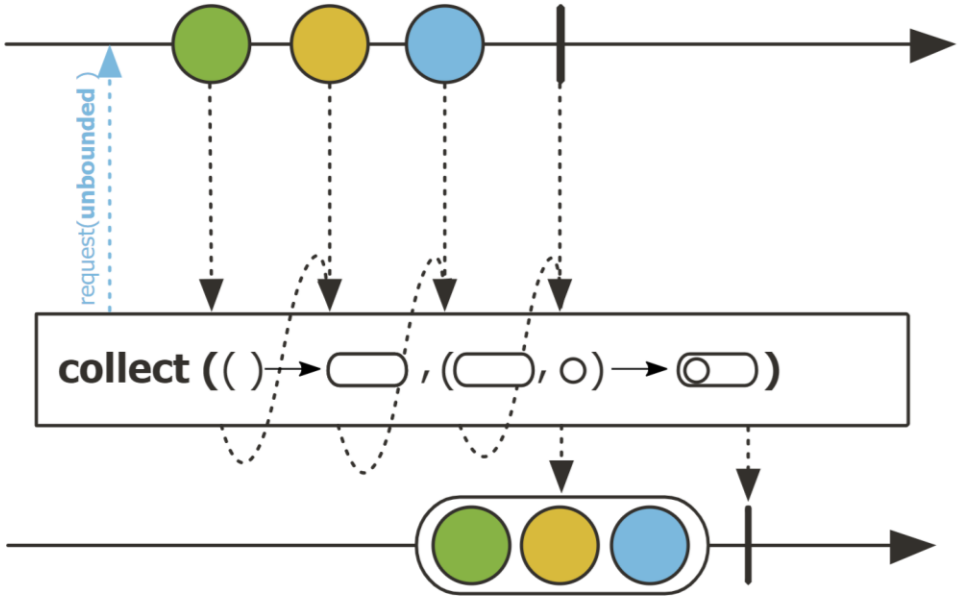
- The collect() method
  - Collects items emitted by the finite source Observable into a single mutable data structure
  - This method is a simplified version of reduce() that does not need to return the state on each pass
    - It's also essentially identical to Observable.collectInto()



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

Applying Key Methods in the Observable Class to ex3

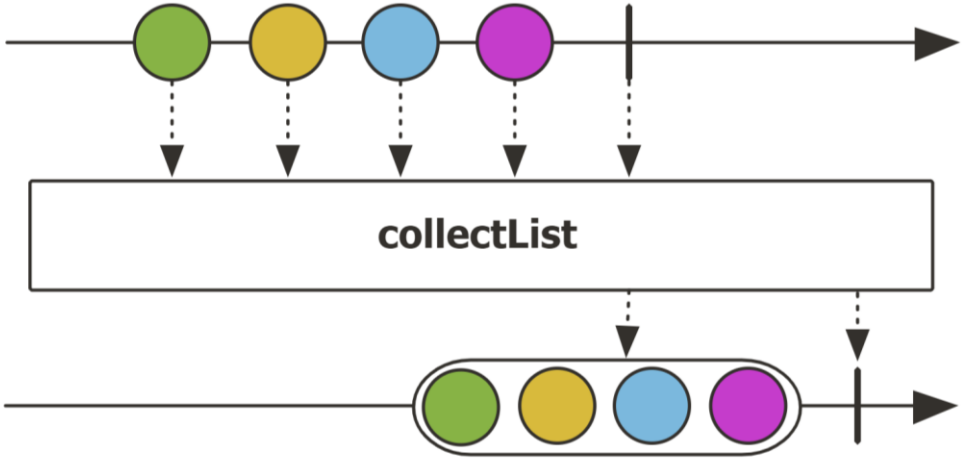
- The collect() method
  - Collects items emitted by the finite source Observable into a single mutable data structure
  - This method is a simplified version of reduce() that does not need to return the state on each pass
  - Project Reactor's Flux.collect() method works the same way



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

Applying Key Methods in the Observable Class to ex3

- The collect() 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 collect() 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)
```

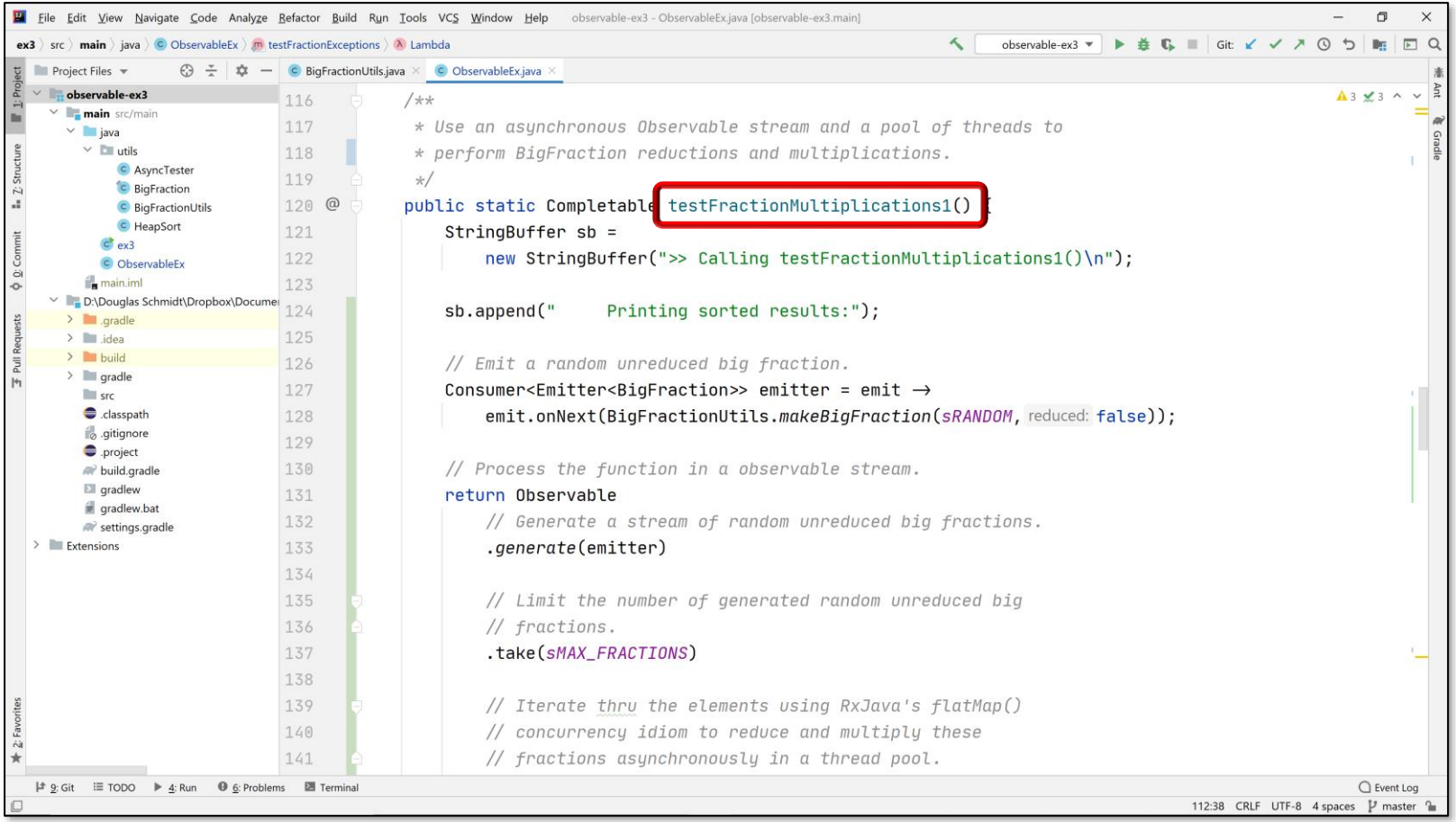
Performs a mutable reduction operation on the elements of this stream. A mutable reduction is one in which the reduced value is a mutable result container, such as an `ArrayList`, and elements are incorporated by updating the state of the result rather than by replacing the result. This produces a result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



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

End of Applying Key Methods in the Observable Class (Part 6)