Applying Key Methods 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 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](reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html)
Learning Objectives in this Part of the Lesson

• Case study ex2 shows how to apply various RxJava operations \textit{asynchronously} to determine if randomly-generated BigInteger objects are prime or not
  • e.g., create(), interval(), map(), filter(), doOnNext(), take(), doOnComplete(), subscribe(), subscribeOn(), observeOn(), range(), ignoreElements(), count(), & various thread pools

See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/Observable/ex2
Applying Key Methods in the Observable Class to ex2
- **testIsPrimeTimed()**

  Use an asynchronous time-driven Observable stream that processes random BigInteger objects to determine which ones are prime.

  ```java
  Observable
  .create(ObservableEx::emitInterval)
  .map(bi ->
      ObservableEx.checkIfPrime
        (bi, sb))
  .doOnNext(bi -> ObservableEx
            .processResult(bi, sb))
  .doOnComplete(() ->
      BigFractionUtils
        .display
        (sb.toString()))
  .count()
  .ignoreElement();
  ```

See [Reactive/Observable/ex2/src/main/java/ObservableEx.java](Reactive/Observable/ex2/src/main/java/ObservableEx.java)
Applying Key Methods in the Observable Class to ex2

- testIsPrimeTimed()
- Use an asynchronous time-driven Observable stream that processes random BigInteger objects to determine which ones are prime
- Demonstrates create(), interval(), map(), filter(), doOnNext(), take(), doOnComplete(), subscribe(), & the Schedulers.computation() thread pool

```java
Observable
    .create(ObservableEx::emitInterval)
    .map(bi ->
        ObservableEx.checkIfPrime
            (bi, sb))
    .doOnNext(bi -> ObservableEx
        .processResult(bi, sb))
    .doOnComplete(() ->
        BigFractionUtils
            .display
            (sb.toString()))
    .count()
    .ignoreElement();
```
Applying Key Methods in the Observable Class to ex2

- The create() method
  - Create an Observable with the capability of emitting multiple elements in a synchronous or asynchronous manner

```java
static <T> Observable<T> create
    (ObservableOnSubscribe<T> source)
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#create
Applying Key Methods in the Observable Class to ex2

- The create() method
  - Create an Observable with the capability of emitting multiple elements in a synchronous or asynchronous manner
  - The ObservableOnSubscribe param has a subscribe() method that receives an instance of an ObservableEmitter instance

```java
static <T> Observable<T> create
(ObservableOnSubscribe<T> source)
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/ObservableOnSubscribe.html
Applying Key Methods in the Observable Class to ex2

- The `create()` method
  - Create an Observable with the capability of emitting multiple elements in a synchronous or asynchronous manner
- The `ObservableOnSubscribe` param has a `subscribe()` method that receives an instance of an `ObservableEmitter` instance
- `ObservableEmitter` can emit events via `onNext()`, `onError()`, & `onComplete()`

```java
static <T> Observable<T> create(
    ObservableOnSubscribe<T> source)
```

**Interface ObservableEmitter<T>**

Type Parameters:

- T - the value type to emit

All Superinterfaces:

- Emitter<T>

```java
public interface ObservableEmitter<T>
    extends Emitter<T>
```

Abstraction over an RxJava Observer that allows associating a resource with it.

The `Emitter.onNext(Object)`, `Emitter.onError(Throwable)`, `Emitter.onComplete()` methods should be called in a sequential manner, just like the Observer's methods should be. Use the `ObservableEmitter` the `serialize()` method returns instead of the original `ObservableEmitter` instance provided by the generator routine if you want to ensure this. The other methods are thread-safe.

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/ObservableEmitter.html
The create() method

Create an Observable with the capability of emitting multiple elements in a synchronous or asynchronous manner

The ObservableOnSubscribe param has a subscribe() method that receives an instance of an ObservableEmitter instance

Elements can be emitted from different threads
Applying Key Methods in the Observable Class to ex2

- The create() method
  - Create an Observable with the capability of emitting multiple elements in a synchronous or asynchronous manner
- Project Reactor’s Flux.create() method works in a similar way
  - Though it is more complex

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#create
Applying Key Methods in the Observable Class to ex2

- The create() method
  - Create an Observable with the capability of emitting multiple elements in a synchronous or asynchronous manner
- Project Reactor’s Flux.create() method works in a similar way
- Also similar to the method Stream.generate() in Java Streams

```java
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](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#generate)
Applying Key Methods in the Observable Class to ex2

- The interval() method
- Create a Observable that emits long values starting with zero

static Observable<Long> interval
(long period, TimeUnit unit)

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#interval
Applying Key Methods in the Observable Class to ex2

- The interval() method
- Create a Observable that emits long values starting with zero
  - The value is incremented at the specified time intervals on the global timer

```java
static Observable<Long> interval(
  long period,
  TimeUnit unit)
```
Applying Key Methods in the Observable Class to ex2

- The interval() method
- Create a Observable that emits long values starting with zero
  - The value is incremented at the specified time intervals on the global timer
- This method emits long values on Schedulers.computation() method by default

---

```java
@NonNull
public static @NonNull Scheduler computation()

Returns a default, shared Scheduler instance intended for computational work.

This can be used for event-loops, processing callbacks and other computational work.

It is not recommended to perform blocking, IO-bound work on this scheduler. Use io() instead.
```

---

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/schedulers/Schedulers.html#computation](reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/schedulers/Schedulers.html#computation)
Applying Key Methods in the Observable Class to ex2

- The interval() method
  - Create a Observable that emits long values starting with zero
    - The value is incremented at the specified time intervals on the global timer
    - This method emits long values on Schedulers.computation() method by default
  - In normal conditions, the Observable never completes
Applying Key Methods in the Observable Class to ex2

- The interval() method
  - Create a Observable that emits long values starting with zero
    - The value is incremented at the specified time intervals on the global timer
    - This method emits long values on Schedulers.computation() method by default
  - In normal conditions, the Observable never completes
    - It’s therefore often used with take()

See upcoming discussion of the Observable.take() method
The interval() method

- Create a Observable that emits long values starting with zero
- Project Reactor’s Flux.interval() works the same

See [projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#interval](http://projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#interval)
Applying Key Methods in the Observable Class to ex2

- The filter() method
- Evaluate each source value against the given Predicate

```java
Observable<T> filter
(Predicate<? super T> p)
```

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#filter](reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#filter)
Applying Key Methods in the Observable Class to ex2

- The filter() method
- Evaluate each source value against the given Predicate
- If the predicate test succeeds, the value is emitted

\[
\text{Observable}\langle T \rangle \ \text{filter} \\
(\text{Predicate}\langle\text{? super } T \rangle \ p)
\]

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/functions/Predicate.html
Applying Key Methods in the Observable Class to ex2

- The filter() method
- Evaluate each source value against the given Predicate
  - If the predicate test succeeds, the value is emitted
  - If the predicate test fails, the value is ignored & a request of 1 is made upstream

```
Observable<T> filter
(Predicate<? super T> p)
```

**Interface Predicate<T>**

Type Parameters:
T - the type of the input to the predicate

Functional Interface:
This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.
Applying Key Methods in the Observable Class to ex2

- The filter() method
- Evaluate each source value against the given Predicate
  - If the predicate test succeeds, the value is emitted
  - If the predicate test fails, the value is ignored & a request of 1 is made upstream
- The # of output elements may be less than the # of input elements
Applying Key Methods in the Observable Class to ex2

- The filter() method
- Evaluate each source value against the given Predicate
  - If the predicate test succeeds, the value is emitted
  - If the predicate test fails, the value is ignored & a request of 1 is made upstream
- The # of output elements may be less than the # of input elements
- filter() can’t change the type or value of elements it processes
Applying Key Methods in the Observable Class to ex2

- The filter() method
  - Evaluate each source value against the given Predicate
- Project Reactor’s Flux.filter() method works the same

See [projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#filter](http://projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#filter)
Applying Key Methods in the Observable Class to ex2

- The filter() method
  - Evaluate each source value against the given Predicate
  - Project Reactor’s Flux.filter() method works the same
  - Similar to Stream.filter() method in Java Streams

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#filter](http://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#filter)
Applying Key Methods in the Observable Class to ex2

- The `doOnNext()` method
- Add a behavior

```java
Observable<T> doOnNext
(Consumer<? super T> onNext)
```

Applying Key Methods in the Observable Class to ex2

- The `doOnNext()` method
- Add a behavior
  - This behavior is triggered when the Observable emits an item

```
Observable<T> doOnNext
(Consumer<? super T> onNext)
```

**Interface Consumer<T>**

**Type Parameters:**
T - the type of the input to the operation

**All Known Subinterfaces:**
Stream.Builder<T>

**Functional Interface:**
This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

Applying Key Methods in the Observable Class to ex2

• The `doOnNext()` method

• Add a behavior
  • This behavior is triggered when the Observable emits an item
  • i.e., it is a “callback”

```
Observable<T> doOnNext
(Consumer<? super T> onNext)
```

See en.wikipedia.org/wiki/Callback_(computer_programming)
Applying Key Methods in the Observable Class to ex2

- The `doOnNext()` method
  - Add a behavior
  - Can’t change the type or value of elements it processes
Applying Key Methods in the Observable Class to ex2

- The `doOnNext()` method
  - Add a behavior
- Project Reactor’s method Flux.
  - `doOnNext()` works the same

See [projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#doOnNext](projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#doOnNext)
Applying Key Methods in the Observable Class to ex2

• The doOnNext() method
  • Add a behavior
  • Project Reactor’s method Flux .doOnNext() works the same
• Similar to Stream.peek() method in Java Streams

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#peek
Applying Key Methods in the Observable Class to ex2

- The take() method  
  \( \text{Observable}\langle T\rangle \ \text{take}(\text{long} \ n) \)
- Take only the first N values  
  from this Observable, if available

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#take](http://reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#take)
Applying Key Methods in the Observable Class to ex2

- The `take()` method
  - Take only the first N values from this Observable, if available
  - Used to limit otherwise “infinite” streams

See earlier discussion of the `Observable.interval()` method
Applying Key Methods in the Observable Class to ex2

• The take() method
  • Take only the first N values from this Observable, if available
  • Used to limit otherwise “infinite” streams
  • Project Reactor’s Flux.take() method works the same

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#take
Applying Key Methods in the Observable Class to ex2

• The take() method
  • Take only the first N values from this Observable, if available
  • Used to limit otherwise “infinite” streams
  • Project Reactor’s Flux.take() method works the same
  • Similar to Stream.limit() in Java Streams

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#limit
Applying Key Methods in the Observable Class to ex2

• The count() method
• Returns a Single that counts the total # of items emitted by the current Observable & emits this count as a 64-bit Long

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#count
Applying Key Methods in the Observable Class to ex2

- The count() method
- Returns a Single that counts the total # of items emitted by the current Observable & emits this count as a 64-bit Long
- count() doesn’t operate on a particular Scheduler by default
Applying Key Methods in the Observable Class to ex2

- The `count()` method
  - Returns a Single that counts the total # of items emitted by the current Observable & emits this count as a 64-bit Long
  - Project Reactor’s Flux.count() method works the same

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#count
Applying Key Methods in the Observable Class to ex2

- The `count()` method
  - Returns a Single that counts the total # of items emitted by the current Observable & emits this count as a 64-bit Long
  - Project Reactor’s `Flux.count()` method works the same
  - Similar to the `Stream.count()` method in Java Streams

```java
long count()

Returns the count of elements in this stream. This is a special case of a reduction and is equivalent to:

    return mapToLong(e -> 1L).sum();
```

This is a terminal operation.

Returns:
the count of elements in this stream

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#count](docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#count)
Applying Key Methods in the Observable Class to ex2

- The subscribe() method
- Subscribe a Consumer to this Observable

```java
Disposable subscribe
(Consumer<? super T> consumer,
 Consumer<? super Throwable>
   errorConsumer,
   Runnable completeConsumer)
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#subscribe
Applying Key Methods in the Observable Class to ex2

- The subscribe() method
- Subscribe a Consumer to this Observable
- This method consumes all elements in the sequence, handles errors, & reacts to completion

```java
Disposable subscribe(
    Consumer<? super T> consumer,
    Consumer<? super Throwable> errorConsumer,
    Runnable completeConsumer)
```

**Interface Consumer<T>**

Type Parameters:
T - the type of the input to the operation

All Known Subinterfaces:
Stream.Builder<T>

Functional Interface:
This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

Applying Key Methods in the Observable Class to ex2

- The `subscribe()` method
  - Subscribe a Consumer to this Observable
  - This method consumes all elements in the sequence, handles errors, & reacts to completion
  - This subscription requests unbounded demand
    - i.e., `Long.MAX_VALUE`

```java
Disposable subscribe(
    Consumer<? super T> consumer,
    Consumer<? super Throwable> errorConsumer,
    Runnable completeConsumer)
```
The `subscribe()` method

Subscribe a Consumer to this Observable

This method consumes all elements in the sequence, handles errors, & reacts to completion

This subscription requests unbounded demand

Signals emitted to this method are represented by the following regular expression:

```plaintext
onNext()*(onComplete()|onError())?
```
Applying Key Methods in the Observable Class to ex2

- The subscribe() method
  - Subscribe a Consumer to this Observable
    - This method consumes all elements in the sequence, handles errors, & reacts to completion
    - A Disposable is returned, which indicates a task or resource that can be cancelled/disposed

```java
Disposable subscribe(Consumer<? super T> consumer,
                     Consumer<? super Throwable> errorConsumer,
                     Runnable completeConsumer)
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/disposables/Disposable.html
Applying Key Methods in the Observable Class to ex2

- The subscribe() method
- Subscribe a Consumer to this Observable
  - This method consumes all elements in the sequence, handles errors, & reacts to completion
  - A Disposable is returned, which indicates a task or resource that can be cancelled/disposed
  - Disposables can be accumulated & disposed in one fell swoop!

```java
CompositeDisposable mDisposables
    (mPublisherScheduler,
     mSubscriberScheduler,
     mSubscriber);

...  

mDisposables.dispose();
```

See: reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/disposables/CompositeDisposable.html
Applying Key Methods in the Observable Class to ex2

• The subscribe() method
  • Subscribe a Consumer to this Observable
  • Calling this method will *not* block the caller thread until the upstream terminates normally or with an error
Applying Key Methods in the Observable Class to ex2

- The subscribe() method
  - Subscribe a Consumer to this Observable
  - Calling this method will *not* block the caller thread until the upstream terminates normally or with an error
  - These semantics motivate the need for the AsyncTester framework!

See Reactive/Observable/ex2/src/main/java/utils/AsyncTester.java
Applying Key Methods in the Observable Class to ex2

- The subscribe() method
  - Subscribe a Consumer to this Observable
  - Calling this method will not block the caller thread until the upstream terminates normally or with an error
  - Project Reactor’s method Flux.subscribe() works the same

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#subscribe
Applying Key Methods in the Observable Class to ex2

```java
@ public static Mono<Void> testIsPrimeTimed() {
    StringBuffer sb =
        new StringBuffer(">> Calling testIsPrimeTimed()\n");

    return Flux
        .create(makeTimedFluxSink(sb))
        .flatMap(FluxEx::checkIfPrime)
        .doOnNext(bigInteger ->
            FluxEx.processResult(bigInteger,
```
End of Applying Key Methods in the Observable Class (Part 2)