Applying Key Methods in the Observable Class (Part 3)

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

```
Observable
  .create(ObservableEx::emitAsync)
  .doOnNext(s -> ObservableEx.print(s, sb))
  .observeOn(Schedulers.newThread())
  .map(bi ->
      ObservableEx.checkIfPrime(bi, sb))
  .doOnNext(bi -> ObservableEx.processResult(bi, sb))
  .doOnComplete(() -> BigFractionUtils.display(sb.toString()))
  .ignoreElements();
```

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/Observable/ex2](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/Observable/ex2)
Applying Key Methods in the Observable Class to ex2
• testIsPrimeAsync()  
• Use an asynchronous Observable stream that processes random BigInteger objects to determine which ones are prime

```java
Observable.create(ObservableEx::emitAsync)
  .doOnNext(s -> ObservableEx
    .print(s, sb))
  .observeOn(Schedulers.newThread())
  .map(bi ->
    ObservableEx.checkIfPrime
      (bi, sb))
  .doOnNext(bi -> ObservableEx
    .processResult(bi, sb))
  .doOnComplete(() -> BigFractionUtils
    .display(sb.toString()))
  .ignoreElements();
```

See Reactive/Observable/ex2/src/main/java/ObservableEx.java
• testIsPrimeAsync()
• Use an asynchronous Observable stream that processes random BigInteger objects to determine which ones are prime
• Demonstrates create(), map(), filter(), doOnNext(), take(), doOnComplete(), subscribe(), subscribeOn(), observeOn(), range(), ignoreElements() & Schedulers.newThread()

```java
Applying Key Methods in the Observable Class to ex2

Observable
  .create(ObservableEx::emitAsync)
  .doOnNext(s -> ObservableEx.print(s, sb))
  .observeOn(Schedulers.newThread())
  .map(bi ->
      ObservableEx.checkIfPrime
      (bi, sb))
  .doOnNext(bi -> ObservableEx
      .processResult(bi, sb))
  .doOnComplete(() -> BigFractionUtils
      .display(sb.toString()))
  .ignoreElements();
```
Applying Key Methods in the Observable Class to ex2

- The range() method
  
- Build a Observable that only emits a sequence of incrementing integers

```java
static Observable<Integer> range(int start, int count)
```

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

• The range() method
  • Build a Observable that only emits a sequence of incrementing integers
  • range() emits integers between start & start + count & then completes

```java
static Observable<Integer> range(int start, int count)
```
Applying Key Methods in the Observable Class to ex2

- The range() method
  - Build a Observable that only emits a sequence of incrementing integers
  - range() emits integers between start & start + count & then completes
    - start is included in the range, up to & including count
Applying Key Methods in the Observable Class to ex2

- The range() method
  - Build a Observable that only emits a sequence of incrementing integers
  - range() emits integers between start & start + count & then completes
- Project Reactor’s Flux.range() method works the same
Applying Key Methods in the Observable Class to ex2

- The range() method
  - Build a Observable that only emits a sequence of incrementing integers
  - range() emits integers between start & start + count & then completes
  - Project Reactor’s Flux.range() method works the same
  - Similar to the IntStream.rangeClosed() method in Java Streams

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

- The subscribeOn() method
- Run subscribe(), onSubscribe(), & request() on the specified Scheduler worker

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

- The `subscribeOn()` method
- Run `subscribe()`, `onSubscribe()`, & `request()` on the specified Scheduler worker
- The scheduler param indicates what thread to perform the operation on

```java
Observable<T> subscribeOn(Scheduler scheduler)
```

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

- The subscribeOn() method
  - Run subscribe(), onSubscribe(), & request() on the specified Scheduler worker
    - The scheduler param indicates what thread to perform the operation on
    - Returns the Observable requesting async processing
Applying Key Methods in the Observable Class to ex2

- The `subscribeOn()` method
  - Run `subscribe()`, `onSubscribe()`, & `request()` on the specified Scheduler worker
- The `subscribeOn()` semantics are a bit unusual
The subscribeOn() method

- Run subscribe(), onSubscribe(), & request() on the specified Scheduler worker

The subscribeOn() semantics are a bit unusual

- Placing this operator in a chain impacts the execution context of the onNext(), onError(), & onComplete() signals
  - i.e., from beginning of chain up to the next occurrence of an observeOn() (if any)

```java
Observable
  .range(1, sMAX_ITERATIONS)
  .subscribeOn(Schedulers.newThread())
  .map(__ -> BigInteger
            .valueOf(lowerBound + rand
                  .nextInt(sMAX_ITERATIONS)))
  .doOnNext(s ->
              ObservableEx.print(s, sb))
  .subscribe(emitter::next,
             error ->
             emitter.complete(),
             emitter::complete);
```
Applying Key Methods in the Observable Class to ex2

- The `subscribeOn()` method
  - Run `subscribe()`, `onSubscribe()`, & `request()` on the specified Scheduler worker
  - The `subscribeOn()` semantics are a bit unusual
  - Project Reactor’s method Flux. `subscribeOn()` works the same

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

- The `observeOn()` method
- Run `onNext()`, `onComplete()`, & `onError()` on a supplied Scheduler worker

```java
Observable<T> observeOn(Scheduler scheduler)
```
Applying Key Methods in the Observable Class to ex2

- The `observeOn()` method
- Run `onNext()`, `onComplete()`, & `onError()` on a supplied Scheduler worker
- The scheduler param indicates what thread to perform the operation on

```java
Observable<T> observeOn(Scheduler scheduler)
```

*Class Scheduler*

```java
java.lang.Object
io.reactivex.rxjava3.core.Scheduler

Direct Known Subclasses:
TestScheduler
```

```java
public abstract class Scheduler
extends Object

A Scheduler is an object that specifies an API for scheduling units of work provided in the form of Runnables to be executed without delay (effectively as soon as possible), after a specified time delay or periodically and represents an abstraction over an asynchronous boundary that ensures these units of work get executed by some underlying task-execution scheme (such as custom Threads, event loop, Executor or Actor system) with some uniform properties and guarantees regardless of the particular underlying scheme.
```

Applying Key Methods in the Observable Class to ex2

- The `observeOn()` method
- Run `onNext()`, `onComplete()`, & `onError()` on a supplied Scheduler worker
  - The scheduler param indicates what thread to perform the operation on
- Returns the Observable requesting async processing

```java
Observable<T> observeOn(Scheduler scheduler)
```
Applying Key Methods in the Observable Class to ex2

- The `observeOn()` method
  - Run `onNext()`, `onComplete()`, & `onError()` on a supplied Scheduler worker
- The `observeOn()` semantics are fairly straightforward
Applying Key Methods in the Observable Class to ex2

return Observable
  .create(ObservableEx::emitAsync)
  .observeOn(Schedulers.newThread())
  .map(bi -> ObservableEx
       .checkIfPrime(bi, sb))
  .doOnNext(bi -> ObservableEx
       .processResult(bi, sb))
  .doOnComplete(() ->
       BigFractionUtils
       .display(sb.toString()))
  .count()
  .ignoreElement();

• The observeOn() method
• Run onNext(), onComplete(), & onError() on a supplied Scheduler worker
• The observeOn() semantics are fairly straightforward
• It influences the threading context where the rest of the operators in the chain below it execute
• Up to a new occurrence of observeOn() in a chain (if any)
Applying Key Methods in the Observable Class to ex2

- The `observeOn()` method
  - Run `onNext()`, `onComplete()`, & `onError()` on a supplied Scheduler worker
- The `observeOn()` semantics are fairly straightforward
- Project Reactor’s method `Flux.publishOn()` works the same

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

```java
public static Mono<Void> testIsPrimeAsync() {
    StringBuffer sb = new StringBuffer(">> Calling testIsPrimeAsync()\n");
    return Flux
        .create(makeAsyncFluxSink(sb))
        .publishOn(Schedulers.newParallel("subscriber", parallelism: 1))
        .map(bigInteger -> FluxEx.checkIfPrime(bigInteger, sb))
        .filter(PrimeResult::isPrime)
        .subscribe(System.out::println);
}
```

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/Observable/ex2](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/Observable/ex2)
End of Applying Key Methods in the Observable Class (Part 3)