Applying Key Operators in the Flux Class: Case Study ex2 (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

- Part 2 of case study ex2 shows how to use Flux operators `create()`, `map()`, `filter()`, `take()`, `subscribe()`, `subscribeOn()`, `publishOn()`, `then()`, `range()`, `doOnNext()`, & `doFinally()` to create large random `BigInteger` objects & asynchronously check if they are prime via publisher & subscriber threads created using `Schedulers.newParallel()`.

```
Scheduler publisher = Schedulers.newParallel("publisher", 1));

Flux
 .range(1, sMAX_ITERATIONS)
 ... 
 .subscribeOn(publisher)
 .map(__ -> BigInteger
   .valueOf(lowerBound + rand
   .nextInt(sMAX_ITERATIONS)))
 ... 
 .doFinally(() -> publisher
   .dispose())
 .subscribe(sink::next,
   err -> ...,
   sink::complete);
```

See [github.com/douglas craig schmidt/LiveLessons/tree/master/Reactive/Flux/ex2](https://github.com/douglas craig schmidt/LiveLessons/tree/master/Reactive/Flux/ex2)
Learning Objectives in this Part of the Lesson

- Part 2 of case study ex2 shows how to use Flux operators `create()`, `map()`, `filter()`, `take()`, `subscribe()`, `subscribeOn()`, `publishOn()`, `then()`, `range()`, `doOnNext()`, & `doFinally()` to create large random `BigInteger` objects & asynchronously check if they are prime via publisher & subscriber threads created using `Schedulers.newParallel()`.

- The `Mono.fromRunnable()` operator is also shown.

```java
Flux
  .create(makeAsyncFluxSink(sb))
  ...
  .map(bigInteger ->
      FluxEx.checkIfPrime
        (bigInteger, sb))
  .doOnNext(bigInteger -> FluxEx
            .processResult
              (bigInteger, sb))
  ...
  .then(Mono.fromRunnable(() ->
        BigFractionUtils
          .display
            (sb.toString())))
```

See [github.com/douglasraigslist/LiveLessons/tree/master/Reactive/Flux/ex2](https://github.com/douglasraigslist/LiveLessons/tree/master/Reactive/Flux/ex2)
Applying Key Operators in the Flux Class to ex2
Applying Key Operators in the Flux Class to ex2

```java
@charset "UTF-8";

public static Mono<Void> testIsPrimeAsync() {
    StringBuffer sb =
        new StringBuffer(">> Calling testIsPrimeAsync()
"");

    return Flux
        .create(makeAsyncFluxSink(sb))
        .map(bigInteger ->
            FluxEx.checkIfPrime(bigInteger, sb)
        );

    // Factory method creates a flow of random big integers
    // that are generated in a background thread.

    // Arrange to perform the prime-checking computations in the
    // "subscriber" thread.
    .publishOn(Schedulers.newParallel(
        name: "subscriber", parallelism: 1)
    );

    // Use a memoizer to check if each random big integer is
    // prime or not in the "subscriber" thread.
}
```

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/flux/ex2](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/flux/ex2)
End of Applying Key Methods in the Flux Class: Case Study ex2 (Part 2)