Applying Key Operators in the Flux Class: Case Study ex2 (Part 1)

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 1 of case study ex2 shows how to use Flux operators create(), interval(), map(), filter(), take(), subscribe(), subscribeOn(), then(), publishOn(), & doOnNext() to create large random BigInteger objects & asynchronously check if they are prime in a background thread from the default parallel thread pool

```
Flux
   .interval(sSLEEP_DURATION)

   .subscribeOn(publisher)

   .map(sGenerateRandomBigInteger)

   .filter(sOnlyOdd)

   .take(sMAX_ITERATIONS)

   .subscribe(sink::next,
              err -> sink
                     .complete(),
              sink::complete);
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/Flux/ex2

Learning Objectives in this Part of the Lesson

- Part 1 of case study ex2 shows how to use Flux operators create(), interval(), map(), filter(), take(), subscribe(), subscribeOn(), then(), publishOn(), & doOnNext() to create large random BigInteger objects & asynchronously check if they are prime in a background thread from the default parallel thread pool
  - The Mono.fromRunnable() operator is also shown

```
Flux
   .create(makeTimedFluxSink(sb))
   ...
   .map(bigInteger ->
        FluxEx.checkIfPrime
          (bigInteger, sb))

   .doOnNext(bigInteger -> FluxEx
          .processResult
            (bigInteger, sb))
   ...
   .then(Mono.fromRunnable(() ->
         BigFractionUtils
           .display
             (sb.toString())));
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/Flux/ex2

Applying Key Operators in the Flux Class to ex2

Applying Key Operators in the Flux Class to ex2

```
     * Test a stream of random BigIntegers to determine which values
     * are prime using an asynchronous time-driven Flux stream.
     */
    public static Mono<Void> testIsPrimeTimed() {
        // We use a StringBuffer because it is thread-safe!
        StringBuffer sb =
            new StringBuffer(">> Calling testIsPrimeTimed()\n");

        // Callback that writes the BigInteger to the StringBuffer.
        Consumer<BigInteger> logBigInteger =
            s -> FluxEx.print(s, sb);

        return Flux
            // Factory method creates a flow of random big integers
            // that are generated at a periodic interval in a
            // background thread.
            .create(makeTimedFluxSink()) Flux<BigInteger>

            // Print the big integer as a debugging aid.
            .doOnNext(logBigInteger)
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/flux/ex2

End of Applying Key Methods in the Flux Class: Case Study ex2 (Part 1)