Applying Key Operators in the Flux Class: Case Study ex3 (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 ex3 explores the use of Flux operators filter(), generate(), flatMap(), fromIterable(), reduce(), collect(), & the parallel thread pool to create, reduce, multiply, & display BigFraction objects asynchronously in a parallel thread pool.

```java
return Flux
 .generate(bigFractionEmitter)
 .take(sMAX_FRACTIONS)
 .flatMap(unreducedFraction ->
   reduceAndMultiplyFraction(unreducedFraction,
  Schedulers
   .parallel()))
 .collect(toList())
 .flatMap(list -> BigFractionUtils
   .sortAndPrintList(list,
   sb));
```
Learning Objectives in this Part of the Lesson

- Part 2 of case study ex3 explores the use of Flux operators filter(), generate(), flatMap(), fromIterable(), reduce(), collect(), & the parallel thread pool to create, reduce, multiply, & display BigFraction objects asynchronously in a parallel thread pool.
- It also shows the use of Mono operators like doOnNext(), map(), firstWithSignal(), subscribeOn(), flatMap(), fromCallable(), & then()

```java
return Mono
    .fromCallable(() -> BigFraction
        .reduce(unreducedFrac))
    .subscribeOn(scheduler)
    .doOnNext(result ->
        logBigFractionResult
            (unreducedFrac,
             sBigReducedFraction,
             result, sb))
    .map(reducedFraction ->
        reducedFraction
        .multiply
            (sBigReducedFraction));
```
Applying Key Operators in the Flux Class to ex3
Applying Key Operators in the Flux Class to ex3

```java
/**
 * Test an asynchronous Flux stream consisting of generate(),
 * take(), flatMap(), collect(), and a pool of threads to perform
 * BigFraction reductions and multiplications.
 */

@public static Mono<Void> testFractionMultiplications1() {

    StringBuffer sb =
        new StringBuffer(">> Calling testFractionMultiplications1()\n");

    sb.append("  Printing sorted results:");

    // Process the function in a flux stream.
    return Flux
        .generate((SynchronousSink<BigFraction> sink) -> sink
            // Emit a random big fraction every time a
            // request is made.
```
End of Applying Key Methods in the Flux Class: Case Study ex3 (Part 2)