Applying Key Operators in Project Reactor: Case Study ex4 (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
Part 2 of case study ex4 shows how to integrate Java Streams operations `generate()`, `limit()`, `map()`, & `collect()` with Project Reactor Mono operators `fromCallable()`, `then()`, `materialize()`, `firstWithSignal()`, `map()`, `flatMap()`, `subscribeOn()`, & `when()` to create, reduce, multiply, & display BigFraction objects asynchronously.

```java
return Stream
  .generate(() ->
    makeBigFraction(sRANDOM, false))
  .limit(sMAX_FRACTIONS)
  .map(unreducedBigFraction ->
    reduceAndMultiplyFraction
    (unreducedBigFraction,
    Schedulers.fromExecutor
      (ForkJoinPool
        .commonPool())))
  .collect(toMono())
  .flatMap(list ->
    BigFractionUtils
    .sortAndPrintList(list, sb));
```

Learning Objectives in this Part of the Lesson

- Part 2 of case study ex4 shows how to integrate Java Streams operations `generate()`, `limit()`, `map()`, & `collect()` with Project Reactor Mono operators `fromCallable()`, `then()`, `materialize()`, `firstWithSignal()`, `map()`, `flatMap()`, `subscribeOn()`, & `when()` to create, reduce, multiply, & display BigFraction objects asynchronously.

- It also shows how to implement a Java Streams Collector for asynchronous Mono objects.

```java
return monos -> Mono
    .when(monas)
    .materialize()
    .flatMap(v -> Flux
        .fromIterable(monas)
        .map(Mono::block)
        .collect(toList()));
```

Applying Key Operators in Project Reactor to ex4
Applying Key Operators in Project Reactor to ex4

```java
import ...

/**
 * This class shows how to apply Project Reactor features
 * asynchronously to perform a range of Flux operations, including
 * fromArray(), map(), flatMap(), collect(), subscribeOn(), and
 * various types of thread pools. It also shows various Mono
 * operations, such as when(), firstWithSignal(), materialize(),
 * flatMap(), flatMapMany(), subscribeOn(), and the parallel thread
 * pool. In addition, it demonstrates how to combine the Java streams
 * framework with the Project Reactor framework.
 */

@SuppressWarnings("ALL")
public class FluxEx {

    /**
     * Create a random number generator.
     */
    private static final Random sRANDOM = new Random();

    /**
     * Test BigFraction multiplications by combining the Java streams
     * framework with the Project Reactor framework and the Java
     * common fork-join framework.
     */
    public static Mono<Void> testFractionMultiplicationsStreams() {
        StringBuffer sb =
```
End of Applying Key Methods in Project Reactor: Case Study ex4 (Part 2)