

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**



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

```
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

```
return monos -> Mono
    .when(monos)

    .materialize()

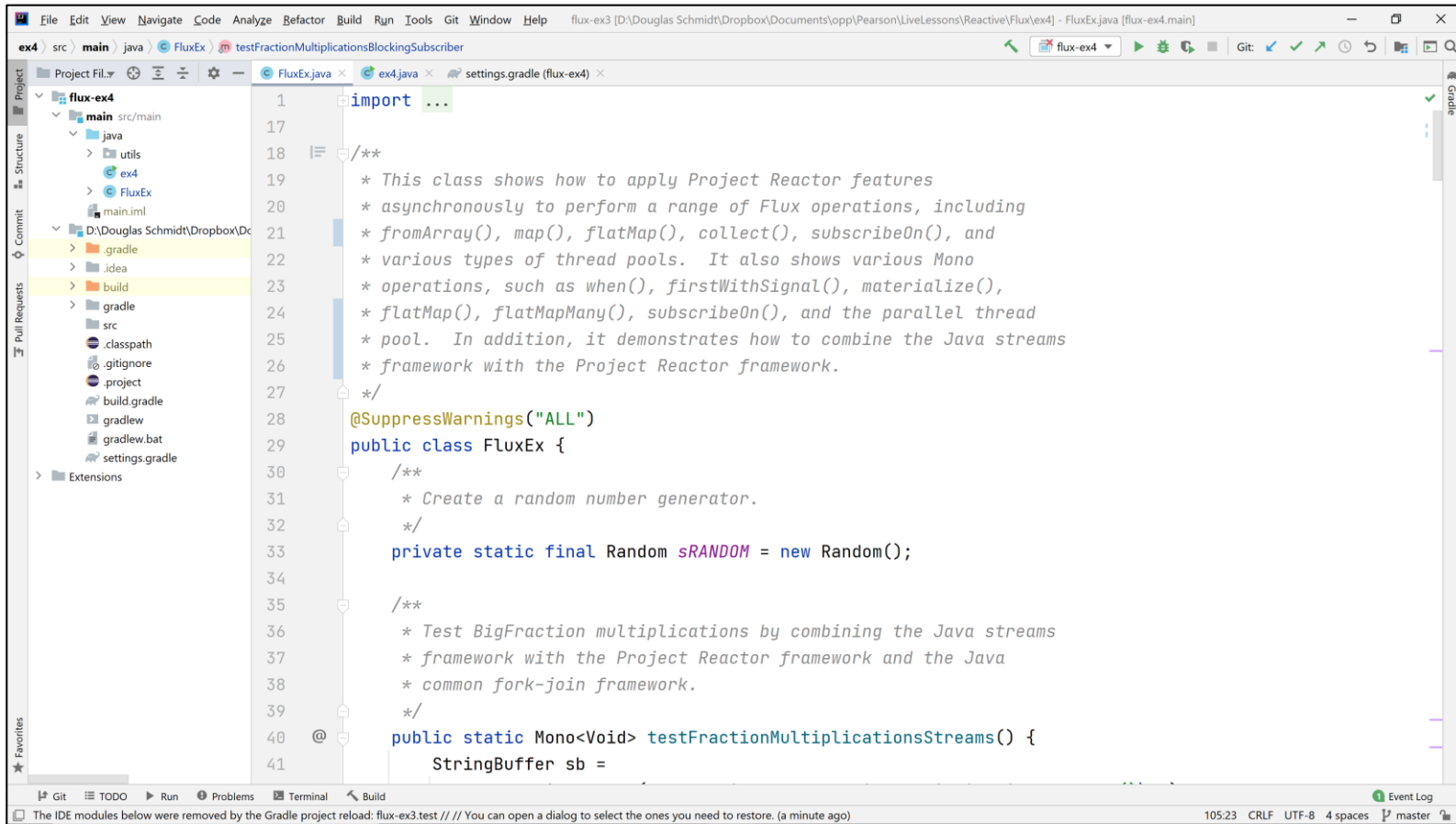
    .flatMap(v -> Flux
        .fromIterable(monos)

        .map(Mono::block)

        .collect(toList()));
```

Applying Key Operators in Project Reactor to ex4

Applying Key Operators in Project Reactor to ex4



```
1  import ...
17
18  /**
19   * This class shows how to apply Project Reactor features
20   * asynchronously to perform a range of Flux operations, including
21   * fromArray(), map(), flatMap(), collect(), subscribeOn(), and
22   * various types of thread pools. It also shows various Mono
23   * operations, such as when(), firstWithSignal(), materialize(),
24   * flatMap(), flatMapMany(), subscribeOn(), and the parallel thread
25   * pool. In addition, it demonstrates how to combine the Java streams
26   * framework with the Project Reactor framework.
27   */
28   @SuppressWarnings("ALL")
29   public class FluxEx {
30       /**
31        * Create a random number generator.
32        */
33       private static final Random sRANDOM = new Random();
34
35       /**
36        * Test BigFraction multiplications by combining the Java streams
37        * framework with the Project Reactor framework and the Java
38        * common fork-join framework.
39        */
40       public static Mono<Void> testFractionMultiplicationsStreams() {
41           StringBuffer sb =
```

See github.com/douglasraigschmidt/LiveLessons/tree/master/Reactive/flux/ex4

End of Applying Key Methods in Project Reactor: Case Study ex4 (Part 2)