Applying Key Operators in Project Reactor: Case Study ex4 (Part 3)

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 3 of case study ex4 applies Flux operators create(), flatMap(), & subscribe(), as well as FluxSink to create, multiply, & display BigFraction objects asynchronously

```
Flux
  .create(makeEmitter(count,
                      sb),
          FluxSink
          .OverflowStrategy
          .ERROR)

  .flatMap(bf1 ->
     multiplyFraction(bf1,
        sBigReducedFraction,
        Schedulers.parallel(),
        sb))

  .subscribe
     (blockingSubscriber);
```

Learning Objectives in this Part of the Lesson

- Part 3 of case study ex4 applies Flux operators create(), flatMap(), & subscribe(), as well as FluxSink to create, multiply, & display BigFraction objects asynchronously

```
Flux
   .create(makeEmitter(count,
                       sb),
           FluxSink
            .OverflowStrategy
            .ERROR)

   .flatMap(bf1 ->
      multiplyFraction(bf1,
         sBigReducedFraction,
         Schedulers.parallel(),
         sb))

   .subscribe
      (blockingSubscriber);
```

This example applies an overflow strategy

Learning Objectives in this Part of the Lesson

- Part 3 of case study ex4 applies Flux operators create(), flatMap(), & subscribe(), as well as FluxSink to create, multiply, & display BigFraction objects asynchronously
  - It also shows how to use Mono operators fromSupplier() & subscribeOn()

```
Mono<BigFraction>
multiplyFraction(BigFraction bf1,
                 BigFraction bf2,
               Scheduler scheduler,
                StringBuffer sb) {
  return Mono
    .fromSupplier(() -> bf1
                     .multiply(bf2))

    .subscribeOn(scheduler);
}
```

Learning Objectives in this Part of the Lesson

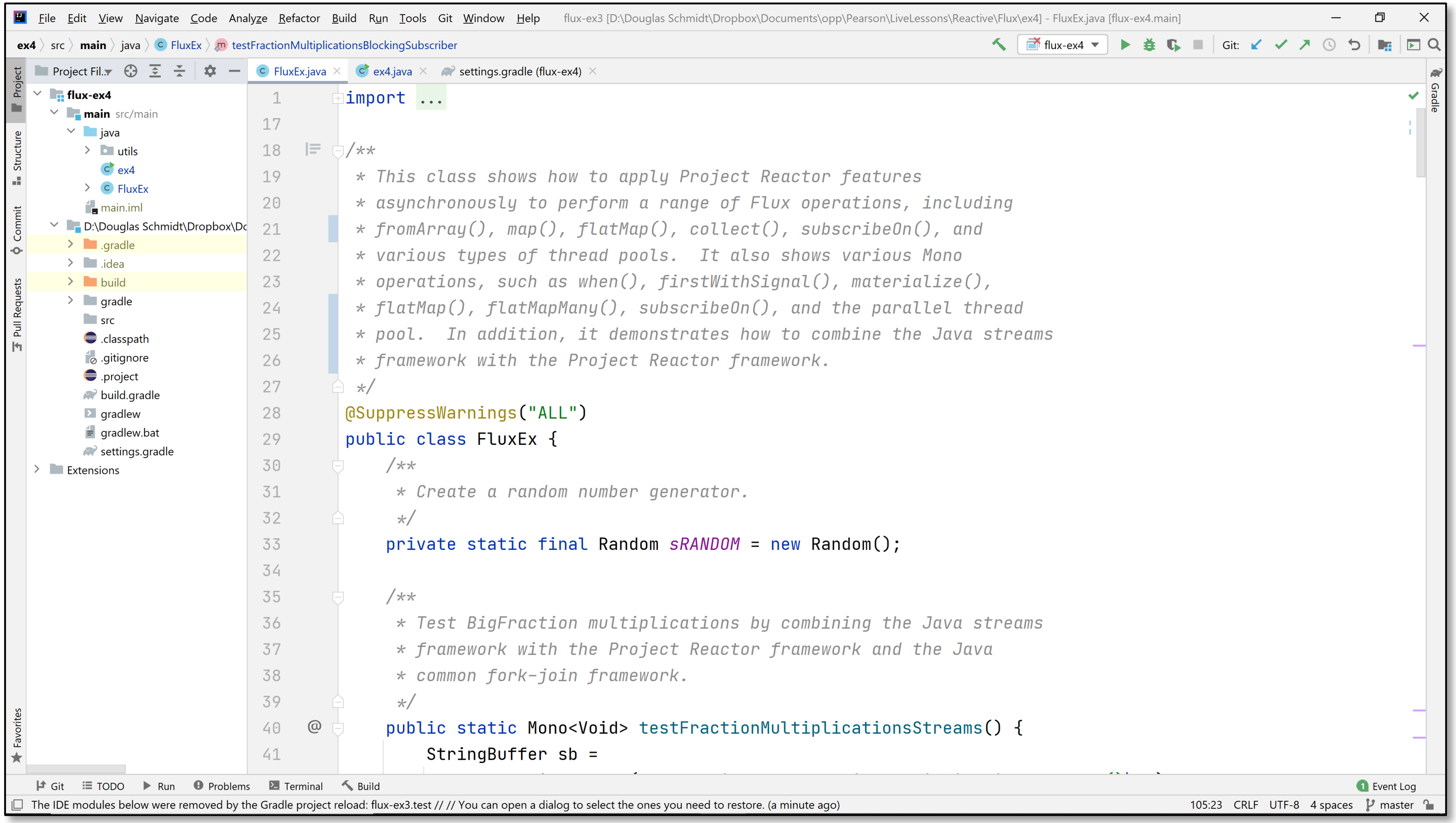
- Part 3 of case study ex4 applies Flux operators create(), flatMap(), & subscribe(), as well as FluxSink to create, multiply, & display BigFraction objects asynchronously
  - It also shows how to use Mono operators fromSupplier() & subscribeOn()
  - In addition, it shows how to create & use a generic blocking Subscriber
    - Can be applied to workaround the lack of a blockingSubscribe() operator

```
class BlockingSubscriber<T>
      implements Subscriber<T> {
  ...
  final CountDownLatch mLatch;
  ...
  @Override
  public void onComplete() {
    ...
    mLatch.countDown();
  }
  ...
}
```

However, this subscriber is "backpressure unaware"

Applying Key Operators in Project Reactor to ex4

Applying Key Operators in Project Reactor to ex4



File Edit View Navigate Code Analyze Refactor Build Run Tools Git Window Help
flux-ex3 [D:\Douglas Schmidt\Dropbox\Documents\opp\Pearson\LiveLessons\Reactive\Flux\ex4] - FluxEx.java [flux-ex4.main]
ex4 src main java FluxEx testFractionMultiplicationsBlockingSubscriber
flux-ex4
Git:
Project Fil.
FluxEx.java
ex4.java
settings.gradle (flux-ex4)
flux-ex4
main src/main
java
utils
ex4
FluxEx
main.iml
D:\Douglas Schmidt\Dropbox\Dc
.gradle
.idea
build
gradle
src
.classpath
.gitignore
.project
build.gradle
gradlew
gradlew.bat
settings.gradle
Extensions
Project
Structure
Commit
Pull Requests
Favorites
Gradle
Git
TODO
Run
Problems
Terminal
Build
Event Log
105:23 CRLF UTF-8 4 spaces master

```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 =
```

The IDE modules below were removed by the Gradle project reload: flux-ex3.test // // You can open a dialog to select the ones you need to restore. (a minute ago)

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

End of Applying Key Methods in Project Reactor: Case Study ex4 (Part 3)