

Applying Key Operators in the Flux Class: Case Study ex2 (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 ex2 shows how to use Flux operators `create()`, `map()`, `filter()`, `take()`, `subscribe()`, `subscribeOn()`, `publishOn()`, `then()`, `range()`, `doOnNext()`, & `doFinally()` to create large random `BigInteger` objects & asynchronously check if they are prime via publisher & subscriber threads created using `Schedulers.newParallel()`

```
Scheduler publisher = Schedulers
    .newParallel("publisher", 1));
```

```
Flux
```

```
    .range(1, sMAX_ITERATIONS)
    ...
    .subscribeOn(publisher)
    .map(__ -> BigInteger
        .valueOf(lowerBound + rand
            .nextInt(sMAX_ITERATIONS)))
    ...
    .doFinally(() -> publisher
        .dispose())
    .subscribe(sink::next,
        err -> ...,
        sink::complete);
```

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

Learning Objectives in this Part of the Lesson

- Part 2 of case study ex2 shows how to use Flux operators `create()`, `map()`, `filter()`, `take()`, `subscribe()`, `subscribeOn()`, `publishOn()`, `then()`, `range()`, `doOnNext()`, & `doFinally()` to create large random `BigInteger` objects & asynchronously check if they are prime via publisher & subscriber threads created using `Schedulers.newParallel()`
- The `Mono.fromRunnable()` operator is also shown

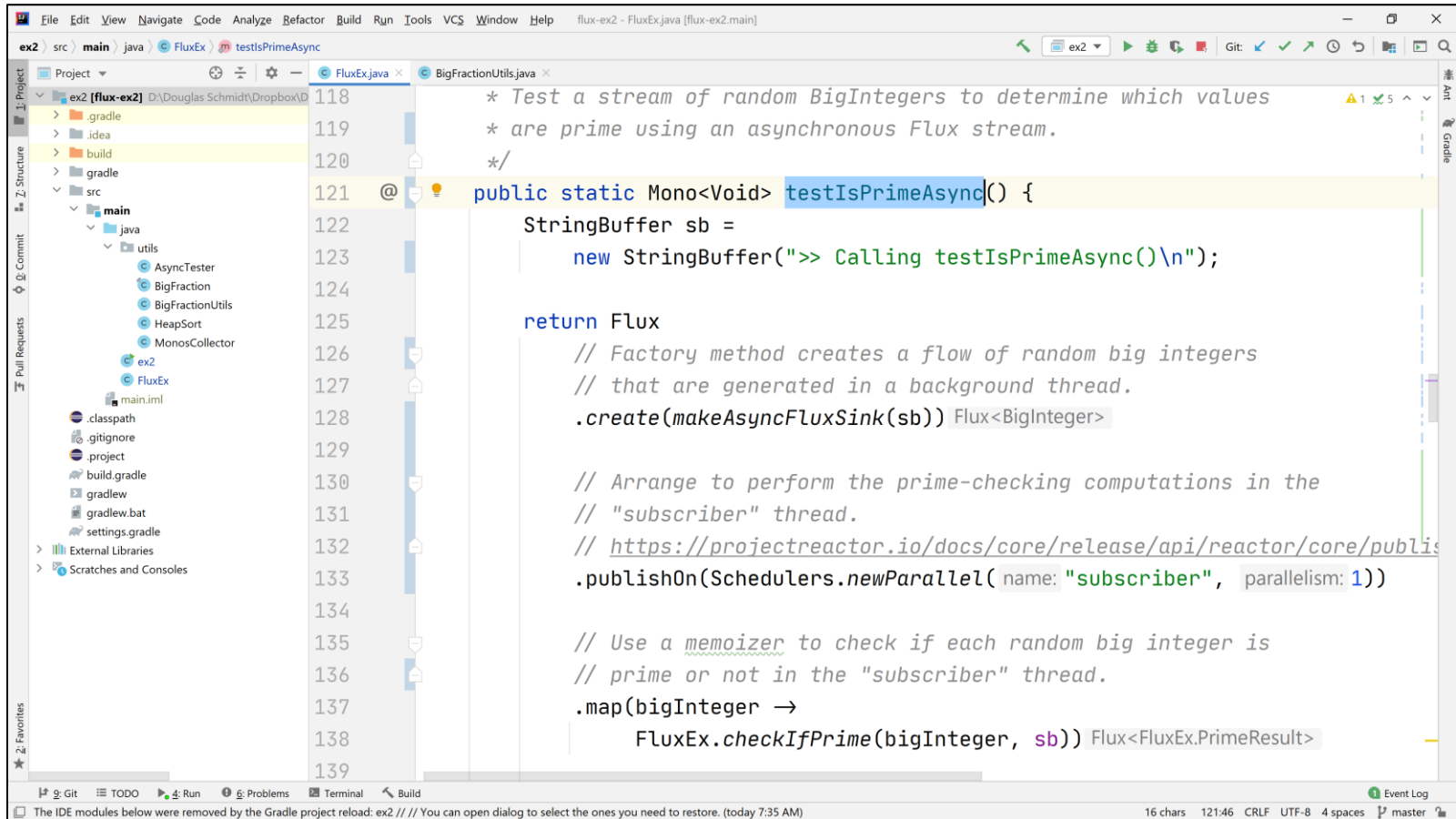
Flux

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

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

Applying Key Operators in the Flux Class to ex2

Applying Key Operators in the Flux Class to ex2



The screenshot shows an IDE window titled "flux-ex2 - FluxEx.java [flux-ex2.main]". The left sidebar displays a project structure for "ex2 [flux-ex2]" with folders like ".gradle", ".idea", "build", "gradle", "src", "main", "java", "utils", and files like "AsyncTester", "BigFraction", "BigFractionUtils", "HeapSort", "MonosCollector", "ex2", "FluxEx", "main.iml", ".classpath", ".gitignore", ".project", "build.gradle", "gradlew", "gradlew.bat", "settings.gradle", "External Libraries", and "Scratches and Consoles". The main editor shows the "testIsPrimeAsync" method in "FluxEx.java". The method is annotated with "@Test" and "*/" comments. It uses "StringBuffer sb" and "new StringBuffer(>> Calling testIsPrimeAsync()\n");". It returns "Flux" and uses "Flux.create(makeAsyncFluxSink(sb)) Flux<BigInteger>". It also uses "Flux.publishOn(Schedulers.newParallel(name: 'subscriber', parallelism: 1))" and "FluxEx.checkIfPrime(bigInteger, sb)) Flux<FluxEx.PrimeResult>".

```
118      * Test a stream of random BigIntegers to determine which values
119      * are prime using an asynchronous Flux stream.
120      */
121      @Test public static Mono<Void> testIsPrimeAsync() {
122          StringBuffer sb =
123              new StringBuffer(">> Calling testIsPrimeAsync()\n");
124
125          return Flux
126              // Factory method creates a flow of random big integers
127              // that are generated in a background thread.
128              .create(makeAsyncFluxSink(sb)) Flux<BigInteger>
129
130              // Arrange to perform the prime-checking computations in the
131              // "subscriber" thread.
132              // https://projectreactor.io/docs/core/release/api/reactor/core/publishOn
133              .publishOn(Schedulers.newParallel(name: "subscriber", parallelism: 1))
134
135              // Use a memoizer to check if each random big integer is
136              // prime or not in the "subscriber" thread.
137              .map(bigInteger ->
138                  FluxEx.checkIfPrime(bigInteger, sb)) Flux<FluxEx.PrimeResult>
139      }
```

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

End of Applying Key Methods in the Flux Class: Case Study ex2 (Part 2)