

Overview of the BigFraction Case Studies

Douglas C. Schmidt

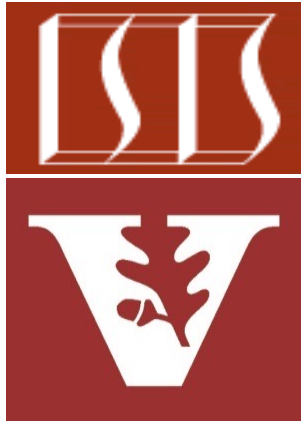
d.schmidt@vanderbilt.edu

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


- Understand key classes in the Project Reactor API
- Know how Project Reactor Flux supports backpressure
- Recognize the Flux overflow strategies
- Be aware of the structure & functionality of the BigFraction case studies
 - These case studies showcase many operators in the Project Reactor Mono, Flux, & ParallelFlux classes


<<Java Class>>


 **BigFraction**


 mNumerator: BigInteger


 mDenominator: BigInteger


 BigFraction()


 valueOf(Number):BigFraction


 valueOf(Number,Number):BigFraction


 valueOf(String):BigFraction


 valueOf(Number,Number,boolean):BigFraction


 reduce(BigFraction):BigFraction


 getNumerator():BigInteger


 getDenominator():BigInteger


 add(Number):BigFraction

 subtract(Number):BigFraction

 multiply(Number):BigFraction

 divide(Number):BigFraction

 gcd(Number):BigFraction

 toMixedString():String

Overview of the BigFraction Class

Overview of the BigFraction Class

- Upcoming lessons show how to apply Project Reactor features in the context of a BigFraction class
- Arbitrary-precision fraction, utilizing BigIntegers for numerator & denominator

```
<<Java Class>>
BigFraction
mNumerator: BigInteger
mDenominator: BigInteger
BigFraction()
valueOf(Number):BigFraction
valueOf(Number,Number):BigFraction
valueOf(String):BigFraction
valueOf(Number,Number,boolean):BigFraction
reduce(BigFraction):BigFraction
getNumerator():BigInteger
getDenominator():BigInteger
add(Number):BigFraction
subtract(Number):BigFraction
multiply(Number):BigFraction
divide(Number):BigFraction
gcd(Number):BigFraction
toMixedString():String
```

See [LiveLessons/blob/master/Java8/ex8/src/utils/BigFraction.java](https://livelessons.blob/master/Java8/ex8/src/utils/BigFraction.java)

Overview of the BigInteger Class

- Upcoming lessons show how to apply Project Reactor features in the context of a BigInteger class
- Arbitrary-precision fraction, utilizing BigIntegers for numerator & denominator
 - BigInteger provides arbitrary-precision integers & associated operators

Class BigInteger

```
java.lang.Object  
    java.lang.Number  
        java.math.BigInteger
```

All Implemented Interfaces:

```
Serializable, Comparable<BigInteger>
```

```
public class BigInteger  
    extends Number  
    implements Comparable<BigInteger>
```

Immutable arbitrary-precision integers. All operations behave as if BigIntegers were represented in two's-complement notation (like Java's primitive integer types). BigInteger provides analogues to all of Java's primitive integer operators, and all relevant methods from java.lang.Math. Additionally, BigInteger provides operations for modular arithmetic, GCD calculation, primality testing, prime generation, bit manipulation, and a few other miscellaneous operations.

See docs.oracle.com/javase/8/docs/api/java/math/BigInteger.html

Overview of the BigFraction Class

- Upcoming lessons show how to apply Project Reactor features in the context of a BigFraction class
 - Arbitrary-precision fraction, utilizing BigIntegers for numerator & denominator
- Factory methods to “reduce” fractions
 - $44/55 \rightarrow 4/5$
 - $12/24 \rightarrow 1/2$
 - $144/216 \rightarrow 2/3$

<<Java Class>>	
BigFraction	
F	mNumerator: BigInteger
F	mDenominator: BigInteger
C	BigFraction()
S	valueOf(Number):BigFraction
S	valueOf(Number,Number):BigFraction
S	valueOf(String):BigFraction
F	valueOf(Number,Number,boolean):BigFraction
S	reduce(BigFraction):BigFraction
F	getNumerator():BigInteger
F	getDenominator():BigInteger
C	add(Number):BigFraction
C	subtract(Number):BigFraction
C	multiply(Number):BigFraction
C	divide(Number):BigFraction
C	gcd(Number):BigFraction
C	toMixedString():String


















Overview of the BigFraction Class

- Upcoming lessons show how to apply Project Reactor features in the context of a BigFraction class
 - Arbitrary-precision fraction, utilizing BigIntegers for numerator & denominator
 - Factory methods to “reduce” fractions
 - Factory methods to create “non-reduced” fractions (& then reduce them)
 - e.g., 12/24 (\rightarrow 1/2)

<<Java Class>>	
BigFraction	
F mNumerator: BigInteger	
F mDenominator: BigInteger	
C BigFraction()	
S <u>valueOf(Number):BigFraction</u>	
S <u>valueOf(Number,Number):BigFraction</u>	
S <u>valueOf(String):BigFraction</u>	
S <u>valueOf(Number,Number,boolean):BigFraction</u>	
S <u>reduce(BigFraction):BigFraction</u>	
F getNumerator():BigInteger	
F getDenominator():BigInteger	
C add(Number):BigFraction	
C subtract(Number):BigFraction	
C multiply(Number):BigFraction	
C divide(Number):BigFraction	
C gcd(Number):BigFraction	
C toMixedString():String	

Overview of the BigFraction Class

- Upcoming lessons show how to apply Project Reactor features in the context of a BigFraction class
 - Arbitrary-precision fraction, utilizing BigIntegers for numerator & denominator
 - Factory methods to “reduce” fractions
 - Factory methods to create “non-reduced” fractions (& then reduce them)
- Arbitrary-precision fraction arithmetic
 - e.g., $18/4 \times 2/3 = 3$

<<Java Class>>	
	BigFraction
	mNumerator: BigInteger
	mDenominator: BigInteger
	BigFraction()
	valueOf(Number):BigFraction
	valueOf(Number,Number):BigFraction
	valueOf(String):BigFraction
	valueOf(Number,Number,boolean):BigFraction
	reduce(BigFraction):BigFraction
	getNumerator():BigInteger
	getDenominator():BigInteger
	add(Number):BigFraction
	subtract(Number):BigFraction
	multiply(Number):BigFraction
	divide(Number):BigFraction
	gcd(Number):BigFraction
	toMixedString():String

Overview of the BigFraction Class

- Upcoming lessons show how to apply Project Reactor features in the context of a BigFraction class
 - Arbitrary-precision fraction, utilizing BigIntegers for numerator & denominator
 - Factory methods to “reduce” fractions
 - Factory methods to create “non-reduced” fractions (& then reduce them)
 - Arbitrary-precision fraction arithmetic
 - Create a mixed fraction from an improper fraction
 - e.g., $18/4 \rightarrow 4 \frac{1}{2}$

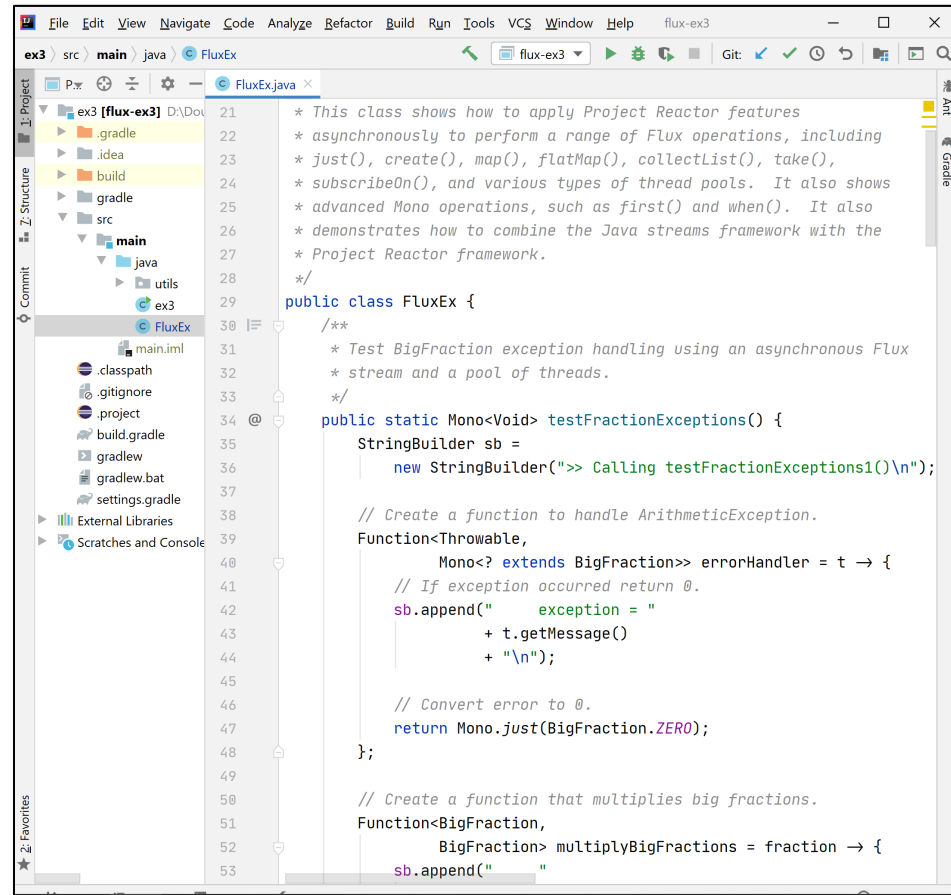
<<Java Class>>	
BigFraction	
F	mNumerator: BigInteger
F	mDenominator: BigInteger
C	BigFraction()
S	valueOf(Number):BigFraction
S	valueOf(Number,Number):BigFraction
S	valueOf(String):BigFraction
S	valueOf(Number,Number,boolean):BigFraction
S	reduce(BigFraction):BigFraction
F	getNumerator():BigInteger
F	getDenominator():BigInteger
	add(Number):BigFraction
	subtract(Number):BigFraction
	multiply(Number):BigFraction
	divide(Number):BigFraction
	gcd(Number):BigFraction
	toMixedString():String

See www.mathsisfun.com/improper-fractions.html

Overview of the BigFraction Case Studies

Overview of the BigFraction Case Studies

- These case studies show how to create, reduce, multiply, & display BigFraction objects synchronously, asynchronously, & concurrently using Project Reactor framework features



```
21  * This class shows how to apply Project Reactor features
22  * asynchronously to perform a range of FLUX operations, including
23  * just(), create(), map(), flatMap(), collectList(), take(),
24  * subscribeOn(), and various types of thread pools. It also shows
25  * advanced Mono operations, such as first() and when(). It also
26  * demonstrates how to combine the Java streams framework with the
27  * Project Reactor framework.
28  */
29
30  public class FluxEx {
31
32      /**
33       * Test BigFraction exception handling using an asynchronous Flux
34       * stream and a pool of threads.
35       */
36      @Test
37      public static Mono<Void> testFractionExceptions() {
38          StringBuilder sb =
39              new StringBuilder(">> Calling testFractionExceptions1()\n");
40
41          // Create a function to handle ArithmeticException.
42          Function<Throwable,
43              Mono<? extends BigFraction>> errorHandler = t -> {
44              // If exception occurred return 0.
45              sb.append("    exception = "
46                  + t.getMessage()
47                  + "\n");
48
49              // Convert error to 0.
50              return Mono.just(BigFraction.ZERO);
51          };
52
53          // Create a function that multiplies big fractions.
54          Function<BigFraction,
55              BigFraction> multiplyBigFractions = fraction -> {
56              sb.append("    ")

```

Overview of the BigFraction Case Studies

- The Project Reactor Mono case studies show how to create, reduce, multiply, & display BigFraction objects using many Mono features
- e.g., fromCallable(), just(), zip(), zipWith(), doOnSuccess(), first(), when(), then(), subscribeOn(), & various thread pools

```
BigFraction unreducedFraction =
    makeBigFraction(...);

return Mono
    .fromCallable(() -> BigFraction
        .reduce(unreducedFraction))
    .subscribeOn
        (Schedulers.single())
    .map(result ->
        result.toMixedString())
    .doOnSuccess(result ->
        System.out.println
            ("big fraction = "
            + result + "\n"))
    .then();
```

Overview of the BigFraction Case Studies

- The Project Reactor Flux case studies show how to create, reduce, multiply, & display BigFraction objects using many Flux features
- e.g., `fromIterable()`, `just()`, `map()`, `create()`, `doOnNext()`, `flatMap()`, `take()`, `interval()`, `subscribeOn()`, `collectList()`, `subscribe()`, & various thread pools

Flux

```
.create  
    (bigFractionEmitter)  
.take (sMAX_FRACTIONS)  
.flatMap (unreducedFraction ->  
          reduceAndMultiplyFraction  
          (unreducedFraction,  
           Schedulers.parallel ()))  
.collectList ()  
.flatMap (list ->  
          BigFractionUtils  
          .sortAndPrintList  
          (list, sb));
```

Overview of the BigFraction Case Studies

- The Project Reactor Flux case studies show how to create, reduce, multiply, & display BigFraction objects using many Flux features
 - e.g., `fromIterable()`, `just()`, `map()`, `create()`, `doOnNext()`, `flatMap()`, `take()`, `interval()`, `subscribeOn()`, `collectList()`, `subscribe()`, & various thread pools
- They also demonstrate how the Java streams framework can be used together with the Project Reactor framework

Class Flux<T>

```
java.lang.Object  
    reactor.core.publisher.Flux<T>
```

Type Parameters:

T - the element type of this Reactive Streams `Publisher`

All Implemented Interfaces:

`Publisher<T>`, `CorePublisher<T>`

Direct Known Subclasses:

`ConnectableFlux`, `FluxOperator`, `FluxProcessor`, `GroupedFlux`

Interface Stream<T>

Type Parameters:

T - the type of the stream elements

All Superinterfaces:

`AutoCloseable`, `BaseStream<T,Stream<T>>`

```
public interface Stream<T>  
    extends BaseStream<T,Stream<T>>
```

A sequence of elements supporting sequential and parallel aggregate operations. The following example illustrates an aggregate operation using `Stream` and `IntStream`:

End of Overview of the BigFraction Case Studies