Applying Java Futures in Practice

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

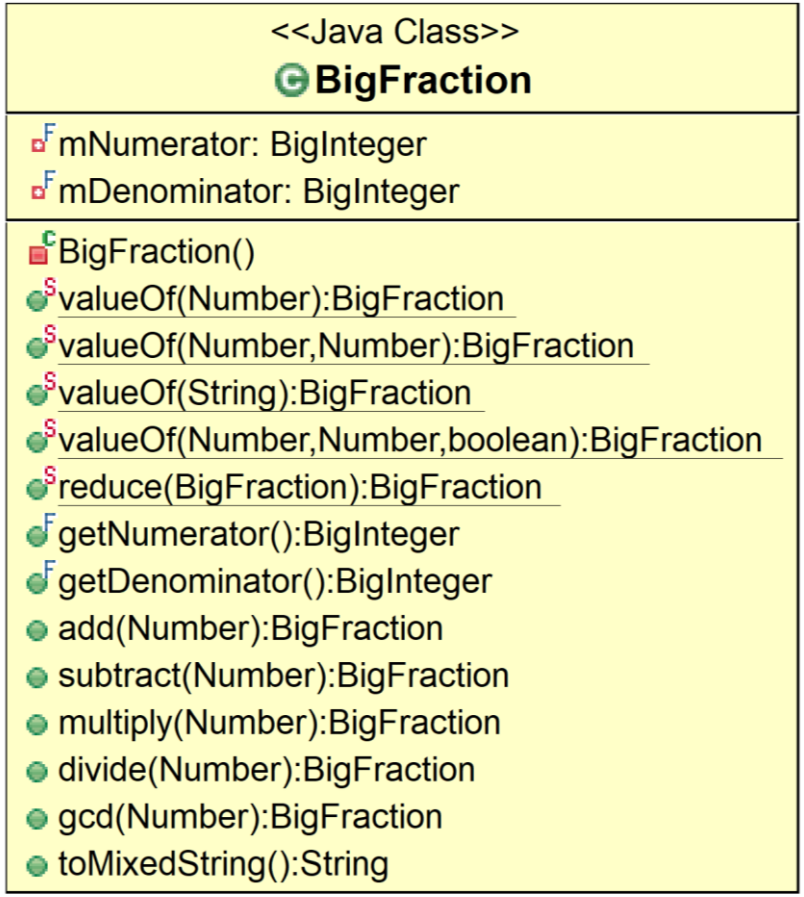
- Motivate the need for Java futures by understanding the pros & cons of synchrony & asynchrony
- Know how Java futures provide the foundation for completable futures in Java
- Understand how to multiply BigFraction objects concurrently via Java futures

```
String f1 = "62675744/15668936";
String f2 = "609136/913704";

Callable<BigFraction> task =
   () -> {
   BigFraction bf1 =
     new BigFraction(f1);
   BigFraction bf2 =
     new BigFraction(f2);
   return bf1.multiply(bf2); };

Future<BigFraction> future =
   commonPool().submit(task);
...
BigFraction res = future.get();
```

Overview of the BigFraction Class

Overview of the BigFraction Class

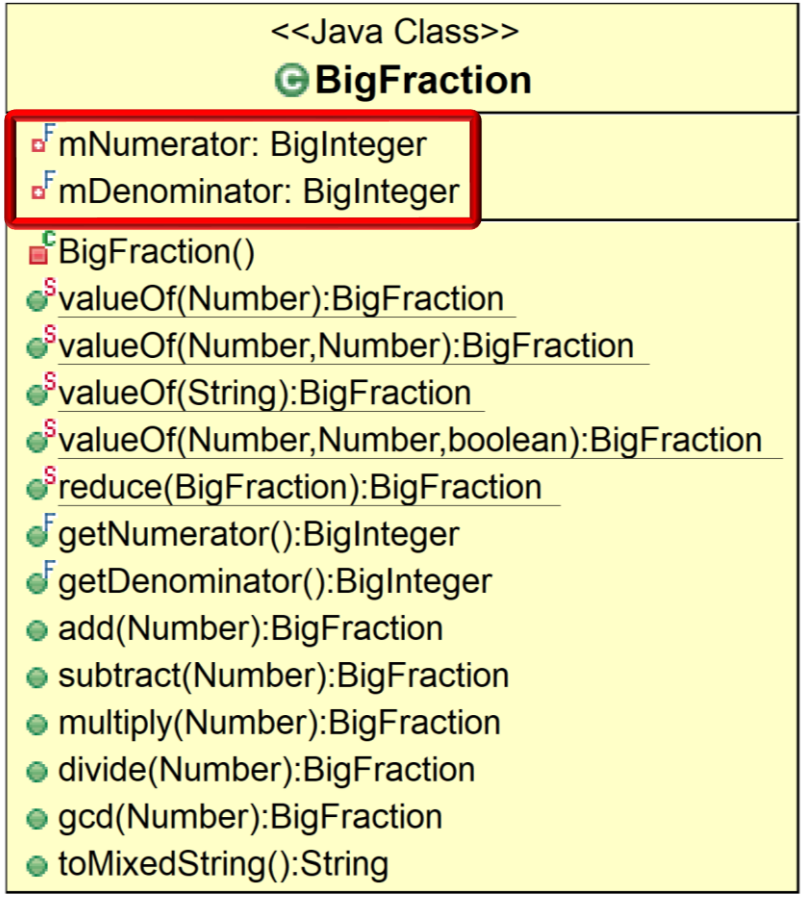
- We show how to apply Java futures in the context of a BigFraction class



See LiveLessons/blob/master/Java8/ex8/src/utils/BigFraction.java

Overview of the BigFraction Class

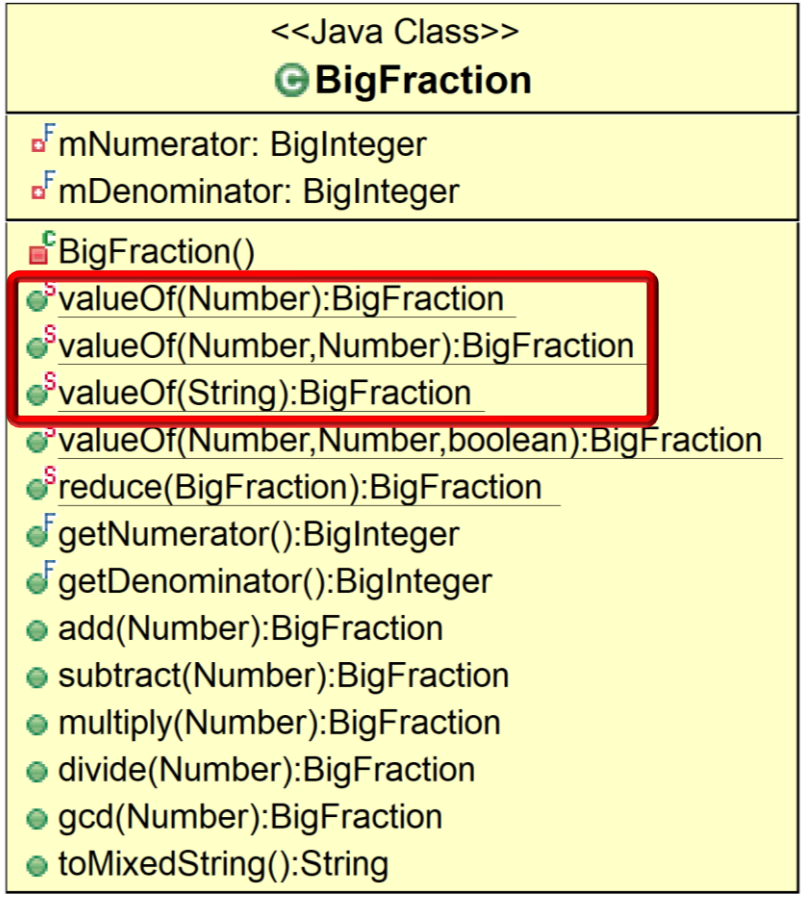
- We show how to apply Java futures in the context of a BigFraction class
  - Arbitrary-precision fraction, utilizing BigIntegers for numerator & denominator



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

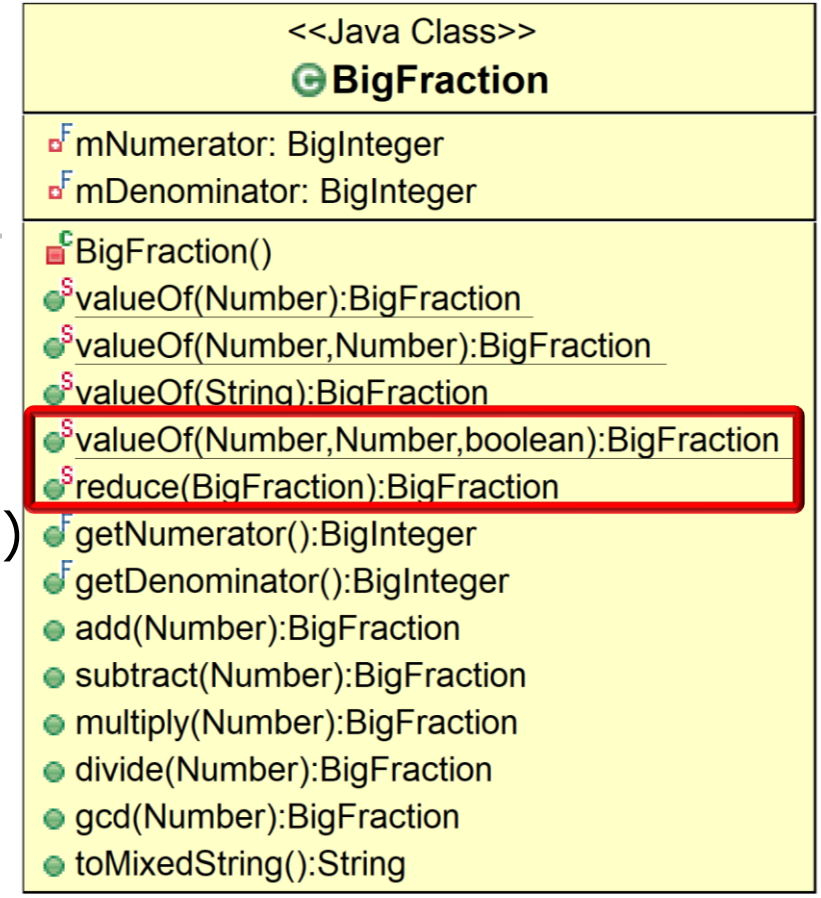
Overview of the BigFraction Class

- We show how to apply Java futures in the context of a BigFraction class
  - Arbitrary-precision fraction, utilizing BigIntegers for numerator & denominator
  - Factory methods for creating "reduced" fractions, e.g.
    - 44/55 → 4/5
    - 12/24 → 1/2
    - 144/216 → 2/3



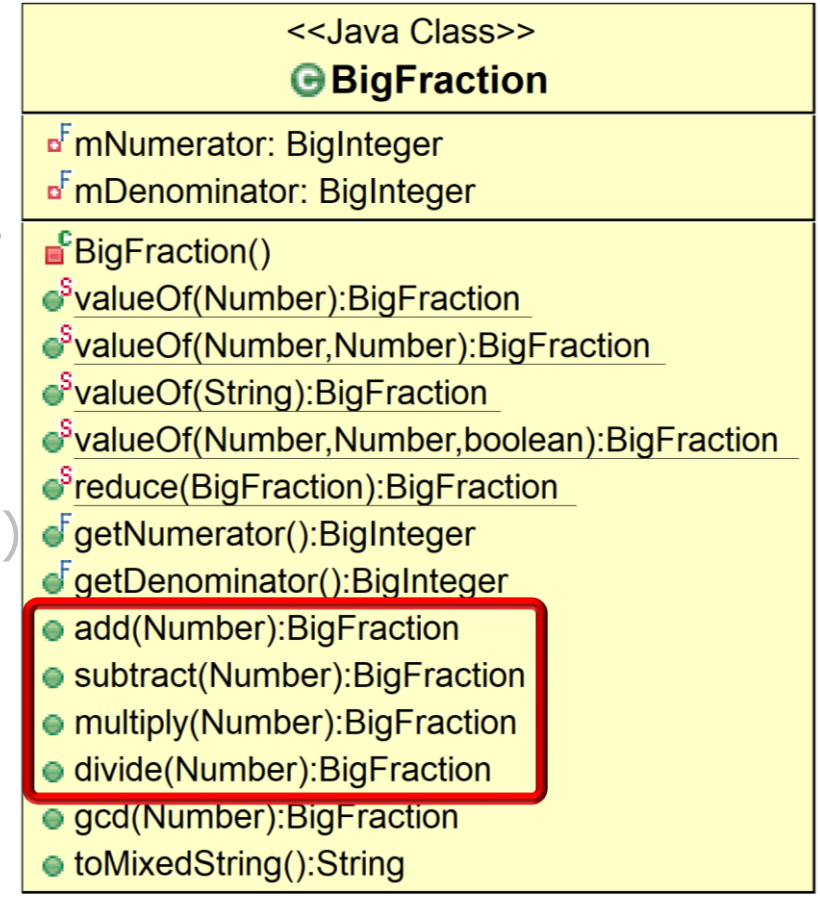
Overview of the BigFraction Class

- We show how to apply Java futures in the context of a BigFraction class
  - Arbitrary-precision fraction, utilizing BigIntegers for numerator & denominator
  - Factory methods for creating "reduced" fractions
  - Factory methods for creating "non-reduced" fractions (& then reducing them)
    - e.g., 12/24 (→ 1/2)



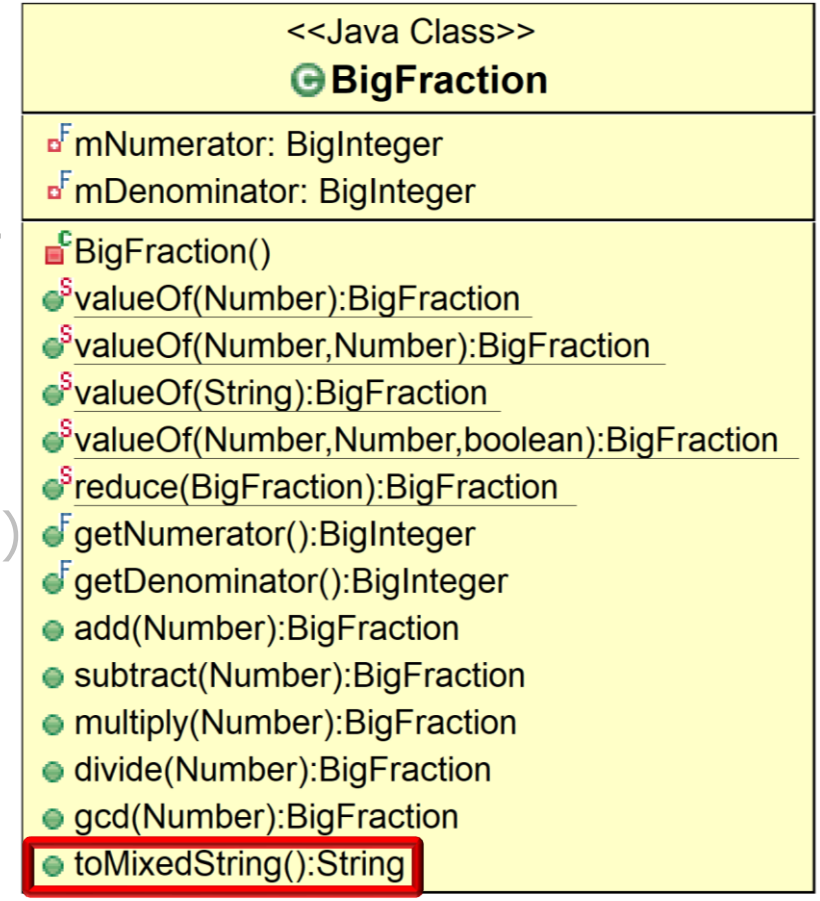
Overview of the BigFraction Class

- We show how to apply Java futures in the context of a BigFraction class
  - Arbitrary-precision fraction, utilizing BigIntegers for numerator & denominator
  - Factory methods for creating "reduced" fractions
  - Factory methods for creating "non-reduced" fractions (& then reducing them)
  - Arbitrary-precision fraction arithmetic
    - e.g., 18/4 ×2/3 = 3



Overview of the BigFraction Class

- We show how to apply Java futures in the context of a BigFraction class
  - Arbitrary-precision fraction, utilizing BigIntegers for numerator & denominator
  - Factory methods for creating "reduced" fractions
  - Factory methods for creating "non-reduced" fractions (& then reducing them)
  - Arbitrary-precision fraction arithmetic
  - Create a mixed fraction from an improper fraction
    - e.g., 18/4 → 4 1/2



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

Programming BigFraction Objects with Java Futures

Programming BigFraction Objects with Java Futures

- Example of using Java Future via a Callable & the common fork-join pool

```
String f1 = "62675744/15668936";
String f2 = "609136/913704";

Callable<BigFraction> task = () -> {
   BigFraction bf1 =
     new BigFraction(f1);
   BigFraction bf2 =
     new BigFraction(f2);
   return bf1.multiply(bf2); };

Future<BigFraction> future =
   commonPool().submit(task);
...
BigFraction result =
   future.get();
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8

Programming BigFraction Objects with Java Futures

- Example of using Java Future via a Callable & the common fork-join pool

```
String f1 = "62675744/15668936";
String f2 = "609136/913704";

Callable<BigFraction> task = () -> {
  BigFraction bf1 =
    new BigFraction(f1);
  BigFraction bf2 =
    new BigFraction(f2);
  return bf1.multiply(bf2); };

Future<BigFraction> future =
  commonPool().submit(task);
...
BigFraction result =
  future.get();
```

Callable is a two-way task that returns a result via a single method with "no" arguments

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Callable.html

Programming BigFraction Objects with Java Futures

- Example of using Java Future via a Callable & the common fork-join pool

```
String f1 = "62675744/15668936";
String f2 = "609136/913704";

Callable<BigFraction> task = () -> {
   BigFraction bf1 =
     new BigFraction(f1);
   BigFraction bf2 =
     new BigFraction(f2);
   return bf1.multiply(bf2); };

Future<BigFraction> future =
   commonPool().submit(task);
...
BigFraction result =
   future.get();
```

Java enables the initialization of a callable via a supplier lambda

See lesson on "*Overview of Java Lambda Expressions and Method References*"

Programming BigFraction Objects with Java Futures

- Example of using Java Future via a Callable & the common fork-join pool

```
String f1 = "62675744/15668936";
String f2 = "609136/913704";

Callable<BigFraction> task = () -> {
   BigFraction bf1 =
     new BigFraction(f1);
   BigFraction bf2 =
     new BigFraction(f2);
   return bf1.multiply(bf2); };

Future<BigFraction> future =
   commonPool().submit(task);
...
BigFraction result =
   future.get();
```

Can pass values to a callable via effectively final variables

See javarevisited.blogspot.com/2015/03/what-is-effectively-final-variable-of.html

Programming BigFraction Objects with Java Futures

- Example of using Java Future via a Callable & the common fork-join pool

```
String f1 = "62675744/15668936";
String f2 = "609136/913704";

Callable<BigFraction> task = () -> {
   BigFraction bf1 =
     new BigFraction(f1);
   BigFraction bf2 =
     new BigFraction(f2);
   return bf1.multiply(bf2); };

Future<BigFraction> future =
   commonPool().submit(task);
...
BigFraction result =
   future.get();
```

Submit a two-way task to run in a thread pool (in this case the common fork-join pool)

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html

Programming BigFraction Objects with Java Futures

- Example of using Java Future via a Callable & the common fork-join pool

```
String f1 = "62675744/15668936";
String f2 = "609136/913704";

Callable<BigFraction> task = () -> {
   BigFraction bf1 =
     new BigFraction(f1);
   BigFraction bf2 =
     new BigFraction(f2);
   return bf1.multiply(bf2); };

Future<BigFraction> future =
   commonPool().submit(task);
...
BigFraction result =
   future.get();
```

submit() returns a future representing the pending results of the task

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html

Programming BigFraction Objects with Java Futures

- Example of using Java Future via a Callable & the common fork-join pool

```
String f1 = "62675744/15668936";
String f2 = "609136/913704";

Callable<BigFraction> task = () -> {
   BigFraction bf1 =
     new BigFraction(f1);
   BigFraction bf2 =
     new BigFraction(f2);
   return bf1.multiply(bf2); };

Future<BigFraction> future =
   commonPool().submit(task);
...
BigFraction result =
   future.get();
```

Other code can run here concurrently wrt the task running in the background

Programming BigFraction Objects with Java Futures

- Example of using Java Future via a Callable & the common fork-join pool

```
String f1 = "62675744/15668936";
String f2 = "609136/913704";

Callable<BigFraction> task = () -> {
   BigFraction bf1 =
     new BigFraction(f1);
   BigFraction bf2 =
     new BigFraction(f2);
   return bf1.multiply(bf2); };

Future<BigFraction> future =
   commonPool().submit(task);
...
BigFraction result =
   future.get();
```

get() blocks if necessary for the computation to complete & then retrieves its result

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html#get

Programming BigFraction Objects with Java Futures

- Example of using Java Future via a Callable & the common fork-join pool

```
String f1 = "62675744/15668936";
String f2 = "609136/913704";

Callable<BigFraction> task = () -> {
   BigFraction bf1 =
     new BigFraction(f1);
   BigFraction bf2 =
     new BigFraction(f2);
   return bf1.multiply(bf2); };

Future<BigFraction> future =
   commonPool().submit(task);
...
BigFraction result =
   future.get(n, SECONDS);
```

get() can also perform polling & timed-waits

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html#get

End of Applying Java Futures in Practice