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

```java
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.

```
<<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

- 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.
  - \( \frac{44}{55} \rightarrow \frac{4}{5} \)
  - \( \frac{12}{24} \rightarrow \frac{1}{2} \)
  - \( \frac{144}{216} \rightarrow \frac{2}{3} \)

```
<<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

- 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., $\frac{18}{4} \times \frac{2}{3} = 3$
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

```java
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](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8)
Example of using Java Future via a Callable & the common fork-join pool

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

```
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 docs.oracle.com/javase/8/docs/api/java/util/concurrent/Callable.html
Java enables the initialization of a callable via a supplier lambda

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 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

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

```java
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](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html)
Example of using Java Future via a Callable & the common fork-join pool

```java
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 [docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html)
• 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();
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#execute
Example of using Java Future via a Callable & the common fork-join pool:

```java
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);
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html#get
End of Applying Java Futures in Practice