The Java Fork-Join Pool: Overview of Example Applications

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

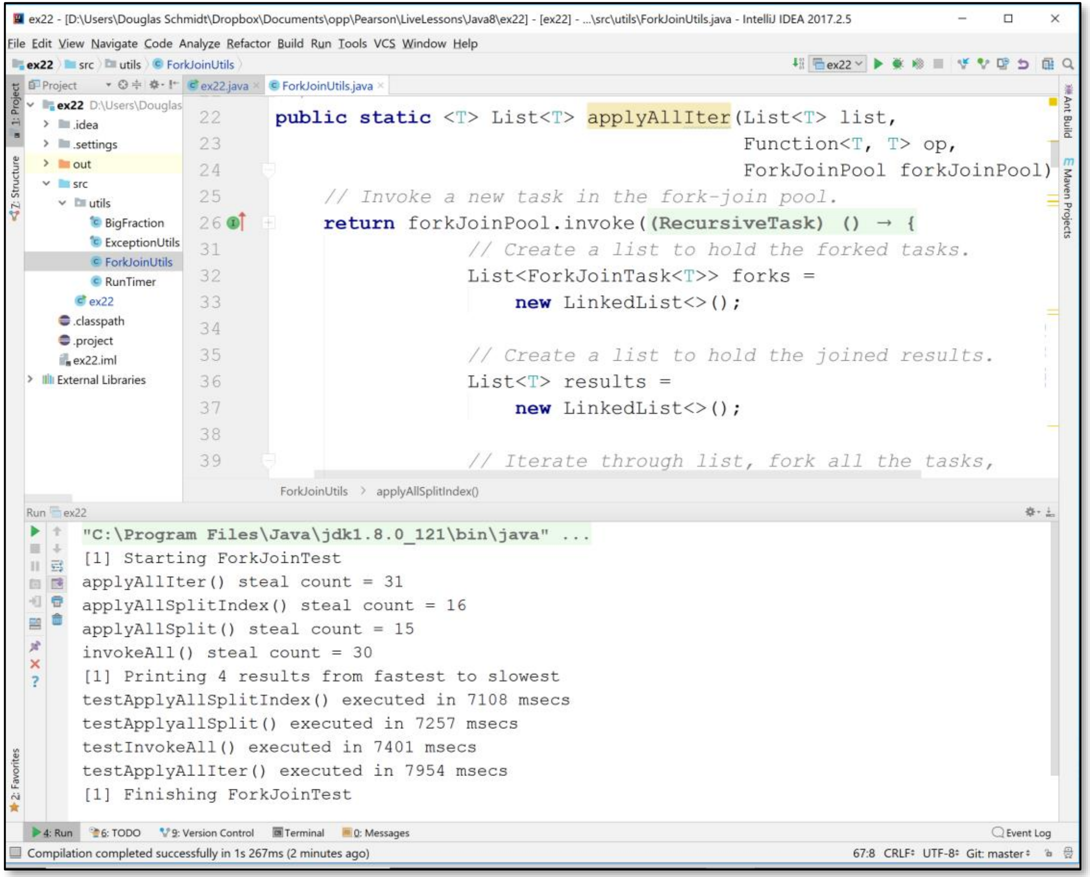
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Apply the fork-join framework


ex22 - [D:\Users\Douglas Schmidt\Dropbox\Documents\opp\Pearson\LiveLessons\Java8\ex22] - [ex22] - ...\src\utils\ForkJoinUtils.java - IntelliJ IDEA 2017.2.5
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
ex22 src utils ForkJoinUtils
Project
ex22 D:\Users\Douglas
.idea
.settings
out
src
utils
BigFraction
ExceptionUtils
ForkJoinUtils
RunTimer
ex22
.classpath
.project
ex22.iml
External Libraries
ex22.java
ForkJoinUtils.java
ForkJoinUtils > applyAllSplitIndex()
Run ex22
4: Run
6: TODO
9: Version Control
Terminal
0: Messages
Event Log
67:8 CRLF: UTF-8: Git: master
1: Project
7: Structure
2: Favorites
Ant Build
Maven Projects

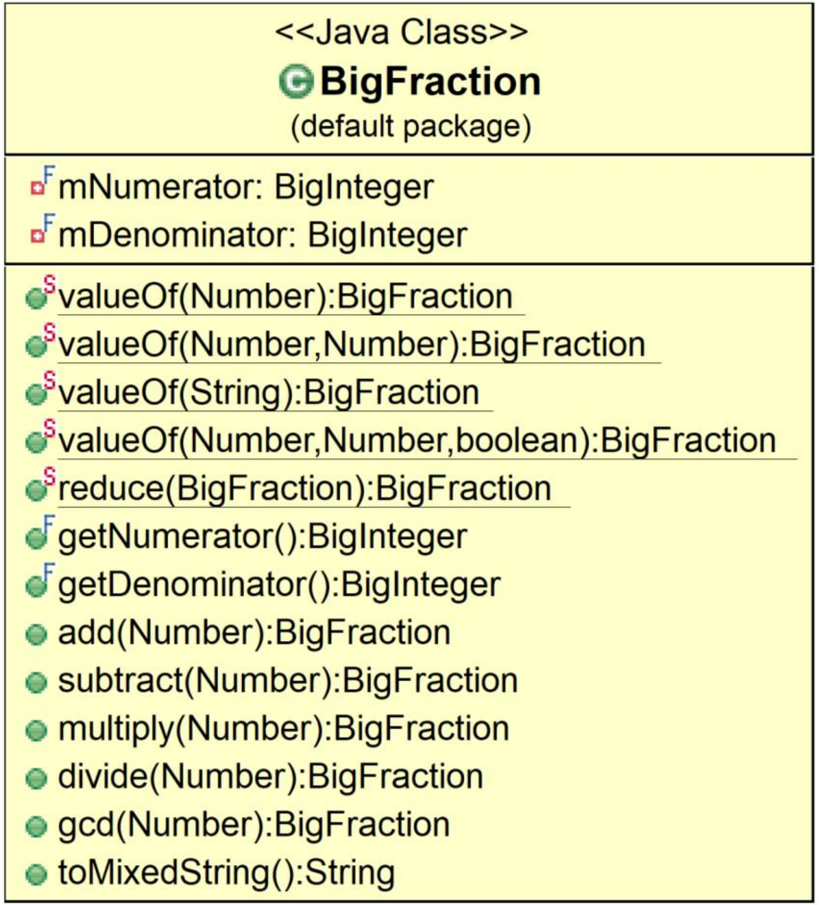
```
public static <T> List<T> applyAllIter(List<T> list,
                                       Function<T, T> op,
                                       ForkJoinPool forkJoinPool)
    // Invoke a new task in the fork-join pool.
    return forkJoinPool.invoke((RecursiveTask) () → {
            // Create a list to hold the forked tasks.
            List<ForkJoinTask<T>> forks =
                new LinkedList<>();

            // Create a list to hold the joined results.
            List<T> results =
                new LinkedList<>();

            // Iterate through list, fork all the tasks,
```

```
"C:\Program Files\Java\jdk1.8.0_121\bin\java" ...
[1] Starting ForkJoinTest
applyAllIter() steal count = 31
applyAllSplitIndex() steal count = 16
applyAllSplit() steal count = 15
invokeAll() steal count = 30
[1] Printing 4 results from fastest to slowest
testApplyAllSplitIndex() executed in 7108 msecs
testApplyallSplit() executed in 7257 msecs
testInvokeAll() executed in 7401 msecs
testApplyAllIter() executed in 7954 msecs
[1] Finishing ForkJoinTest
```

Compilation completed successfully in 1s 267ms (2 minutes ago)

Learning Objectives in this Part of the Lesson

- Apply the fork-join framework
  - Implement operations on BigFractions
    - Supports arbitrary-precision fractions, utilizing BigIntegers for numerator & denominator



<<Java Class>>

BigFraction

(default package)

mNumerator: BigInteger

mDenominator: BigInteger

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/ex22/src/utils/BigFraction.java

Learning Objectives in this Part of the Lesson

- Apply the fork-join framework
  - Implement operations on BigFractions
  - Use several different fork-join pool programming models

Learning Objectives in this Part of the Lesson

- Apply the fork-join framework
  - Implement operations on BigFractions
  - Use several different fork-join pool programming models, e.g.
    - **`applyAllIter()`**
      - Uses "work-stealing" to disperse tasks to worker threads

```
<T> List<T> applyAllIter
  (List<T> list,
   Function<T, T> op,
   ForkJoinPool fjPool) {
  return fjPool
    .invoke(new
       RecursiveTask
         <List<T>>() {
       protected List<T>
         compute() {
                ...
         }
       });
}
```

See upcoming lesson on "*Java Fork-Join Pool: Implementing applyAllIter()*"

Learning Objectives in this Part of the Lesson

- Apply the fork-join framework
  - Implement operations on BigFractions
  - Use several different fork-join pool programming models, e.g.
    - `applyAllIter()`
    - **`applyAllSplit()`**
      - Uses recursive decomposition to disperse tasks to worker threads

```
<T> List<T> applyAllSplit
   (List<T> list,
    Function<T, T> op,
    ForkJoinPool fjPool) {
    class SplitterTask
    extends RecursiveTask
      <List<T>> { ... }

  return fjPool
    .invoke(new
       SplitterTask(list));
}
```

See upcoming lesson on "*Java Fork-Join Pool: Implementing applyAllSplit()*"

Learning Objectives in this Part of the Lesson

- Apply the fork-join framework
  - Implement operations on BigFractions
  - Use several different fork-join pool programming models, e.g.
    - `applyAllIter()`
    - `applyAllSplit()`
    - **`applyAllSplitIndex()`**
      - Uses optimized recursive decomposition to disperse tasks to worker threads

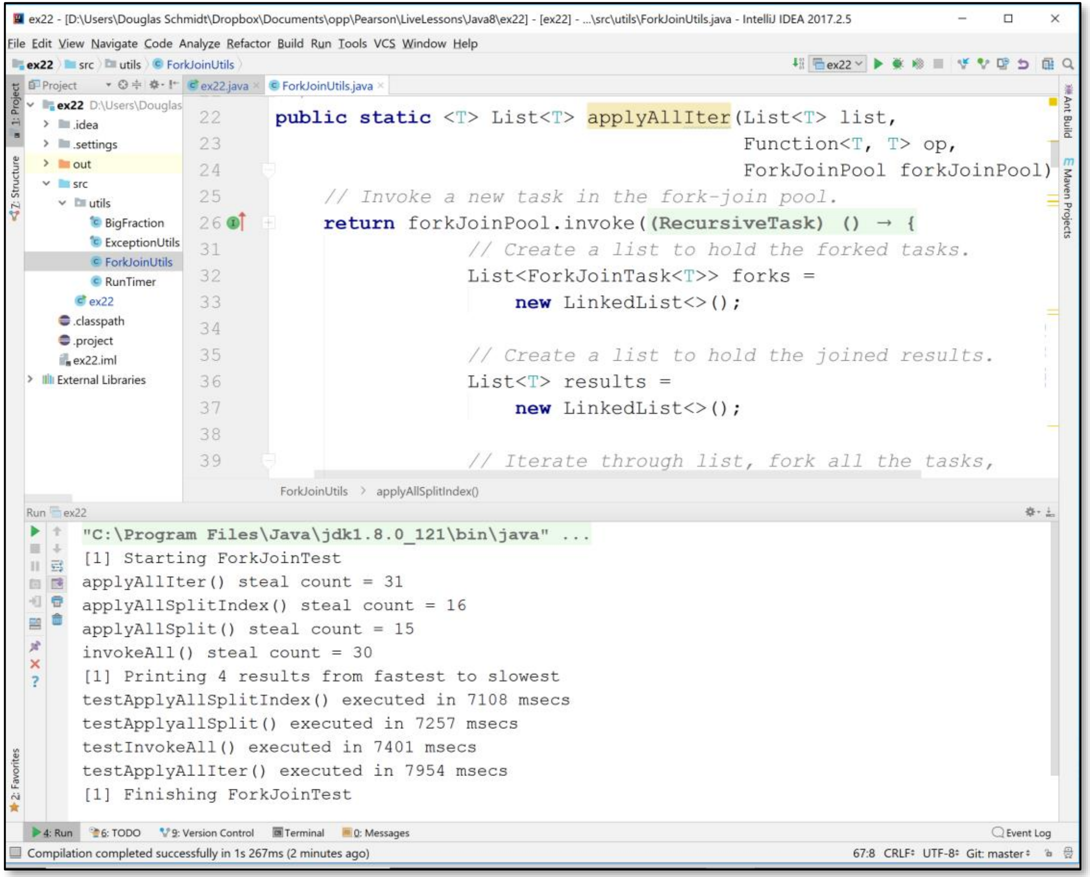
```
<T> List<T> applyAllSplitIndex
  (List<T> list,
   Function<T, T> op,
   ForkJoinPool fjPool) {
  ...
  class SplitterTask
  extends RecursiveAction
  { ... }

  fjPool.invoke(new
    SplitterTask
    (0, list.size()));

  return Arrays.asList(res);
}
```

See upcoming lesson on "*Java Fork-Join Pool: Implementing applyAllSplitIndex()*"

Applying the Java Fork-Join Framework

Applying the Java Fork-Join Framework

- Reduce & multiply big fractions using several different models of programming the Java fork-join framework



ex22 - [D:\Users\Douglas Schmidt\Dropbox\Documents\opp\Pearson\LiveLessons\Java8\ex22] - [ex22] - ...\src\utils\ForkJoinUtils.java - IntelliJ IDEA 2017.2.5
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
ex22 src utils ForkJoinUtils
Project
ex22 D:\Users\Douglas
.idea
.settings
out
src
utils
BigFraction
ExceptionUtils
ForkJoinUtils
RunTimer
ex22
.classpath
.project
ex22.iml
External Libraries
ex22.java
ForkJoinUtils.java
ForkJoinUtils > applyAllSplitIndex()
Run ex22
4: Run
6: TODO
9: Version Control
Terminal
0: Messages
Event Log
67:8 CRLF UTF-8 Git: master

```
public static <T> List<T> applyAllIter(List<T> list,
                                       Function<T, T> op,
                                       ForkJoinPool forkJoinPool)
    // Invoke a new task in the fork-join pool.
    return forkJoinPool.invoke((RecursiveTask) () → {
            // Create a list to hold the forked tasks.
            List<ForkJoinTask<T>> forks =
                new LinkedList<>();

            // Create a list to hold the joined results.
            List<T> results =
                new LinkedList<>();

            // Iterate through list, fork all the tasks,
```

```
"C:\Program Files\Java\jdk1.8.0_121\bin\java" ...
[1] Starting ForkJoinTest
applyAllIter() steal count = 31
applyAllSplitIndex() steal count = 16
applyAllSplit() steal count = 15
invokeAll() steal count = 30
[1] Printing 4 results from fastest to slowest
testApplyAllSplitIndex() executed in 7108 msecs
testApplyallSplit() executed in 7257 msecs
testInvokeAll() executed in 7401 msecs
testApplyAllIter() executed in 7954 msecs
[1] Finishing ForkJoinTest
```

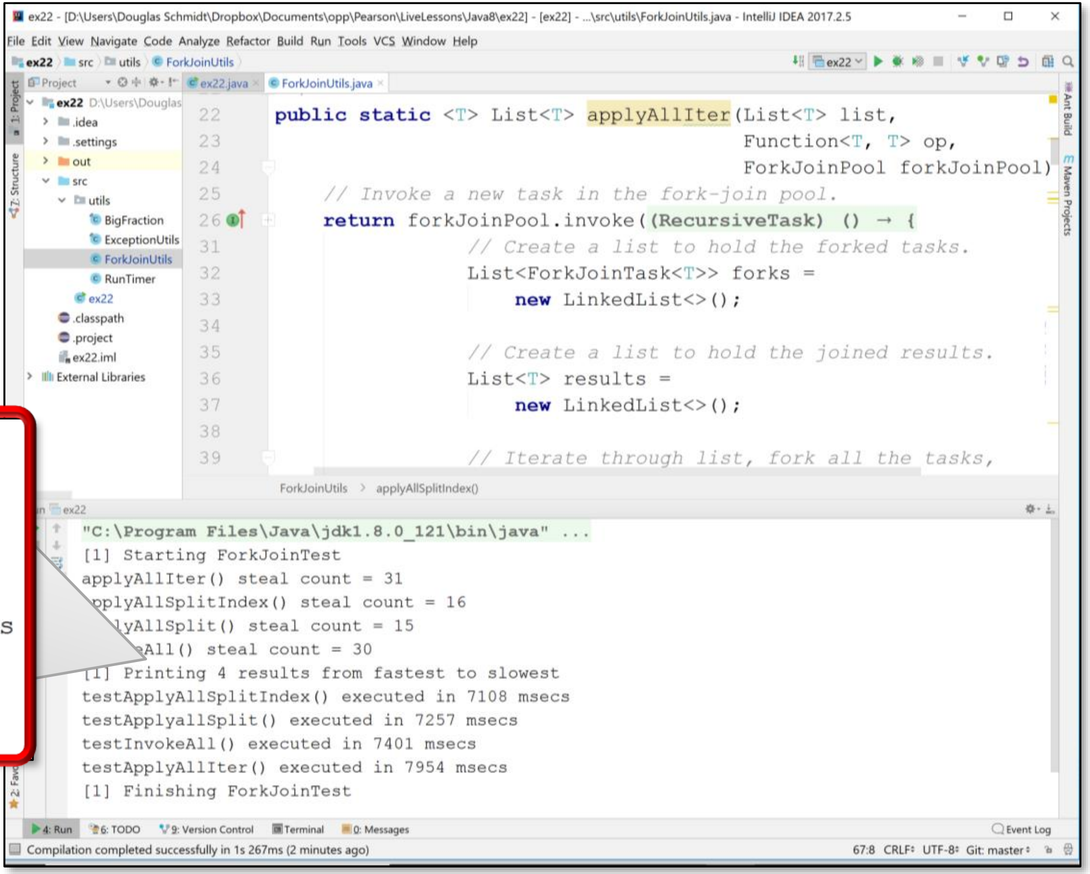
Compilation completed successfully in 1s 267ms (2 minutes ago)

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex22

Applying the Java Fork-Join Framework

- Reduce & multiply big fractions using several different models of programming the Java fork-join framework
  - These model have different performance pros & cons

```
applyAllIter() steal count = 31
applyAllSplitIndex() steal count = 16
applyAllSplit() steal count = 15
invokeAll() steal count = 30
[1] Printing 4 results from fastest to slowest
testApplyAllSplitIndex() executed in 7108 msecs
testApplyallSplit() executed in 7257 msecs
testInvokeAll() executed in 7401 msecs
testApplyAllIter() executed in 7954 msecs
```


ex22 - [D:\Users\Douglas Schmidt\Dropbox\Documents\opp\Pearson\LiveLessons\Java8\ex22] - [ex22] - ...\src\utils\ForkJoinUtils.java - IntelliJ IDEA 2017.2.5
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
ex22 src utils ForkJoinUtils
Project
ex22 D:\Users\Douglas
.idea
.settings
out
src
utils
BigFraction
ExceptionUtils
ForkJoinUtils
RunTimer
ex22
.classpath
.project
ex22.iml
External Libraries
ex22.java
ForkJoinUtils.java
ForkJoinUtils > applyAllSplitIndex()
4: Run
6: TODO
9: Version Control
Terminal
0: Messages
Event Log
Compilation completed successfully in 1s 267ms (2 minutes ago)
67:8 CRLF: UTF-8: Git: master:

```java
public static <T> List<T> applyAllIter(List<T> list,
                                       Function<T, T> op,
                                       ForkJoinPool forkJoinPool)
    // Invoke a new task in the fork-join pool.
    return forkJoinPool.invoke((RecursiveTask) () → {
            // Create a list to hold the forked tasks.
            List<ForkJoinTask<T>> forks =
                new LinkedList<>();

            // Create a list to hold the joined results.
            List<T> results =
                new LinkedList<>();

            // Iterate through list, fork all the tasks,
```

```
"C:\Program Files\Java\jdk1.8.0_121\bin\java" ...
[1] Starting ForkJoinTest
applyAllIter() steal count = 31
pplyAllSplitIndex() steal count = 16
lyAllSplit() steal count = 15
All() steal count = 30
[1] Printing 4 results from fastest to slowest
testApplyAllSplitIndex() executed in 7108 msecs
testApplyallSplit() executed in 7257 msecs
testInvokeAll() executed in 7401 msecs
testApplyAllIter() executed in 7954 msecs
[1] Finishing ForkJoinTest
```

e.g., some incur more “stealing”, copy more data, make more method calls, etc.

Applying the Java Fork-Join Framework

- Reduce & multiply big fractions using several different models of programming the Java fork-join framework
  - These model have different performance pros & cons
  - Java functional programming & sequential streams features are applied to simplify the code

```
List<BigFraction> fractionList =
  Stream
    .generate(() ->
       makeBigFraction(new Random(),
                       false))
    .limit(sMAX_FRACTIONS)
    .collect(toList());

  Function<BigFraction,
           BigFraction> op =
    bigFraction -> BigFraction
      .reduce(bigFraction)
      .multiply(sBigReducedFraction);
```

Applying the Java Fork-Join Framework

- Reduce & multiply big fractions using the Java fork-join framework

```
public static void main(String[] argv) throws IOException {
  List<BigFraction> fractionList = Stream
    .generate(() -> makeBigFraction(new Random(), false))
    .limit(sMAX_FRACTIONS)
    .collect(toList());

  Function<BigFraction, BigFraction> op = bigFraction ->
    BigFraction
      .reduce(bigFraction)
      .multiply(sBigReducedFraction);

  testApplyAllIter(fractionList, op);
  testApplyAllSplit(fractionList, op);
  testApplyAllSplitIndex(fractionList, op); ...
```

See LiveLessons/blob/master/Java8/ex22/src/ex22.java

Applying the Java Fork-Join Framework

- Reduce & multiply big fractions using the Java fork-join framework

```
public static void main(String[] argv) throws IOException {
  List<BigFraction> fractionList = Stream
    .generate(() -> makeBigFraction(new Random(), false))
    .limit(sMAX_FRACTIONS)
    .collect(toList());

  Function<BigFraction, BigFraction> op = bigFraction ->
    BigFraction
      .reduce(bigFraction)
      .multiply(sBigReducedFraction);

  testApplyAllIter(fractionList, op);
  testApplyAllSplit(fractionList, op);
  testApplyAllSplitIndex(fractionList, op); ...
```

Use a Java stream to generate random BigFractions up to sMAX_FRACTIONS

This is the primary use of Java streams in this example

Applying the Java Fork-Join Framework

- Reduce & multiply big fractions using the Java fork-join framework

```
BigFraction makeBigFraction(Random random, boolean reduced) {
  BigInteger numerator =
    new BigInteger(150000, random);

  BigInteger denominator =
    numerator.divide(BigInteger
                     .valueOf(random.nextInt(10) + 1));

  return BigFraction.valueOf(numerator,
                             denominator,
                             reduced);
}
```

Factory method that creates a large & random big fraction

Applying the Java Fork-Join Framework

- Reduce & multiply big fractions using the Java fork-join framework

```
BigFraction makeBigFraction(Random random, boolean reduced) {
  BigInteger numerator =
    new BigInteger(150000, random);

  BigInteger denominator =
    numerator.divide(BigInteger
                     .valueOf(random.nextInt(10) + 1));

  return BigFraction.valueOf(numerator,
                             denominator,
                             reduced);
}
```

Make a random numerator uniformly distributed over range 0 to $(2^{150000} - 1)$

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

Applying the Java Fork-Join Framework

- Reduce & multiply big fractions using the Java fork-join framework

```
BigFraction makeBigFraction(Random random, boolean reduced) {
  BigInteger numerator =
    new BigInteger(150000, random);

  BigInteger denominator =
    numerator.divide(BigInteger
                     .valueOf(random.nextInt(10) + 1));

  return BigFraction.valueOf(numerator,
                             denominator,
                             reduced);
}
```

Make a denominator by dividing the numerator by random # between 1 & 10

Applying the Java Fork-Join Framework

- Reduce & multiply big fractions using the Java fork-join framework

```
BigFraction makeBigFraction(Random random, boolean reduced) {
  BigInteger numerator =
    new BigInteger(150000, random);

  BigInteger denominator =
    numerator.divide(BigInteger
                     .valueOf(random.nextInt(10) + 1));

  return BigFraction.valueOf(numerator,
                             denominator,
                             reduced);
}
```

Return a BigFraction w/the numerator & denominator

Applying the Java Fork-Join Framework

- Reduce & multiply big fractions using the Java fork-join framework

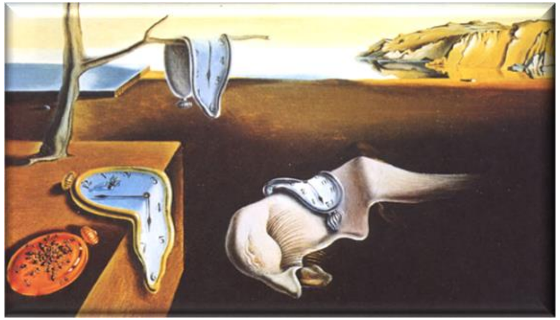
```
public static void main(String[] argv) throws IOException {
  List<BigFraction> fractionList = Stream
    .generate(() -> makeBigFraction(new Random(), false))
    .limit(sMAX_FRACTIONS)
    .collect(toList());

  Function<BigFraction, BigFraction> op = bigFraction ->
    BigFraction
      .reduce(bigFraction)
      .multiply(sBigReducedFraction);

  testApplyAllIter(fractionList, op);
  testApplyAllSplit(fractionList, op);
  testApplyAllSplitIndex(fractionList, op); ...
```

A function that reduces & multiplies a big fraction

Applying the Java Fork-Join Framework

- Reduce & multiply big fractions using the Java fork-join framework

```
public static void main(String[] argv) throws IOException {
  List<BigFraction> fractionList = Stream
    .generate(() -> makeBigFraction(new Random(), false))
    .limit(sMAX_FRACTIONS)
    .collect(toList());

  Function<BigFraction, BigFraction> op = bigFraction ->
    BigFraction
      .reduce(bigFraction)
      .multiply(sBigReducedFraction);

  testApplyAllIter(fractionList, op);
  testApplyAllSplit(fractionList, op);
  testApplyAllSplitIndex(fractionList, op); ...
```



This function takes a surprisingly long time to run!

Applying the Java Fork-Join Framework

- Reduce & multiply big fractions using the Java fork-join framework

```
public static void main(String[] argv) throws IOException {
  List<BigFraction> fractionList = Stream
    .generate(() -> makeBigFraction(new Random(), false))
    .limit(sMAX_FRACTIONS)
    .collect(toList());

  Function<BigFraction, BigFraction> op = bigFraction ->
    BigFraction
      .reduce(bigFraction)
      .multiply(sBigReducedFraction);

  testApplyAllIter(fractionList, op);
  testApplyAllSplit(fractionList, op);
  testApplyAllSplitIndex(fractionList, op); ...
```

Run various fork-join tests

Applying the Java Fork-Join Framework

- Test the applyAllIter(), applyAllSplit(), & applyAllSplitIndex() helper methods

```
void testApplyAllIter(List<BigFraction> fractionList,
                      Function<BigFraction, BigFraction> op)
{ applyAllIter(fractionList, op, new ForkJoinPool()); }


void testApplyAllSplit(List<BigFraction> fractionList,
                       Function<BigFraction, BigFraction> op)
{ applyAllSplit(fractionList, op, new ForkJoinPool()); }


void testApplyAllSplitIndex
                     (List<BigFraction> fractionList,
                      Function<BigFraction, BigFraction> op)
{ applyAllSplitIndex(fractionList, op, new ForkJoinPool()); }
```

Applying the Java Fork-Join Framework

- Test the applyAllIter(), applyAllSplit(), & applyAllSplitIndex() helper methods

```
void testApplyAllIter(List<BigFraction> fractionList,
                      Function<BigFraction, BigFraction> op)
{ applyAllIter(fractionList, op, new ForkJoinPool()); }


void testApplyAllSplit(List<BigFraction> fractionList,
                       Function<BigFraction, BigFraction> op)
{ applyAllSplit(fractionList, op, new ForkJoinPool()); }


void testApplyAllSplitIndex
                     (List<BigFraction> fractionList,
                      Function<BigFraction, BigFraction> op)
{ applyAllSplitIndex(fractionList, op, new ForkJoinPool()); }
```

Each helper method uses a different means of applying the fork-join framework

Applying the Java Fork-Join Framework

- Test the applyAllIter(), applyAllSplit(), & applyAllSplitIndex() helper methods

```
void testApplyAllIter(List<BigFraction> fractionList,
                      Function<BigFraction, BigFraction> op)
{ applyAllIter(fractionList, op, new ForkJoinPool()); }
```

Uses "work-stealing" to disperse tasks to worker threads

```
void testApplyAllSplit(List<BigFraction> fractionList,
                       Function<BigFraction, BigFraction> op)
{ applyAllSplit(fractionList, op, new ForkJoinPool()); }


void testApplyAllSplitIndex
                     (List<BigFraction> fractionList,
                      Function<BigFraction, BigFraction> op)
{ applyAllSplitIndex(fractionList, op, new ForkJoinPool()); }
```

Applying the Java Fork-Join Framework

- Test the applyAllIter(), applyAllSplit(), & applyAllSplitIndex() helper methods

```
void testApplyAllIter(List<BigFraction> fractionList,
                      Function<BigFraction, BigFraction> op)
{ applyAllIter(fractionList, op, new ForkJoinPool()); }
```

```
void testApplyAllSplit(List<BigFraction> fractionList,
                       Function<BigFraction, BigFraction> op)
{ applyAllSplit(fractionList, op, new ForkJoinPool()); }
```

Uses recursive decomposition to disperse tasks to worker threads

```
void testApplyAllSplitIndex
                     (List<BigFraction> fractionList,
                      Function<BigFraction, BigFraction> op)
{ applyAllSplitIndex(fractionList, op, new ForkJoinPool()); }
```

Applying the Java Fork-Join Framework

- Test the applyAllIter(), applyAllSplit(), & applyAllSplitIndex() helper methods

```
void testApplyAllIter(List<BigFraction> fractionList,
                      Function<BigFraction, BigFraction> op)
{ applyAllIter(fractionList, op, new ForkJoinPool()); }


void testApplyAllSplit(List<BigFraction> fractionList,
                       Function<BigFraction, BigFraction> op)
{ applyAllSplit(fractionList, op, new ForkJoinPool()); }


void testApplyAllSplitIndex
                     (List<BigFraction> fractionList,
                      Function<BigFraction, BigFraction> op)
{ applyAllSplitIndex(fractionList, op, new ForkJoinPool()); }
```

Uses optimized recursive decomposition to disperse tasks to worker threads

Applying the Java Fork-Join Framework

- Test the applyAllIter(), applyAllSplit(), & applyAllSplitIndex() helper methods

```
void testApplyAllIter(List<BigFraction> fractionList,
                      Function<BigFraction, BigFraction> op)
{ applyAllIter(fractionList, op, new ForkJoinPool()); }


void testApplyAllSplit(List<BigFraction> fractionList,
                       Function<BigFraction, BigFraction> op)
{ applyAllSplit(fractionList, op, new ForkJoinPool()); }


void testApplyAllSplitIndex
                     (List<BigFraction> fractionList,
                      Function<BigFraction, BigFraction> op)
{ applyAllSplitIndex(fractionList, op, new ForkJoinPool()); }
```

Each helper method gets its own fork-join pool sized to the # of processor cores

End of the Java Fork-Join Pool: Overview of Example Applications