

Overview of Basic Java 8

CompletableFuture Features (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

- Understand the basic completable futures features
- Know how to apply these basic features to operate on big fractions

<<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 github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex8

Learning Objectives in this Part of the Lesson

- Understand the basic completable futures features
- Know how to apply these basic features to operate on big fractions
- Recognize limitations with these basic features



Class `CompletableFuture<T>`

```
java.lang.Object  
    java.util.concurrent.CompletableFuture<T>
```

All Implemented Interfaces:

```
CompletionStage<T>, Future<T>
```

```
public class CompletableFuture<T>  
    extends Object  
    implements Future<T>, CompletionStage<T>
```

A `Future` that may be explicitly completed (setting its value and status), and may be used as a `CompletionStage`, supporting dependent functions and actions that trigger upon its completion.

When two or more threads attempt to `complete`, `completeExceptionally`, or `cancel` a `CompletableFuture`, only one of them succeeds.

In addition to these and related methods for directly manipulating status and results, `CompletableFuture` implements interface `CompletionStage` with the following policies:

Applying Basic Completable Future Features

Applying Basic Completable Future Features

- We show how to apply basic completable future features in the context of BigFraction

<<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)

Applying Basic Completable Future Features

- We show how to apply basic completable future features in the context of BigFraction
- Arbitrary-precision fraction, utilizing BigInteger for numerator & denominator

<<Java Class>>	
G 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

Applying Basic Completable Future Features

- We show how to apply basic completable future features in the context of BigFraction
 - Arbitrary-precision fraction, utilizing BigIntegers for numerator & denominator
 - Factory methods for creating “reduced” fractions, e.g.
 - $44/55 \rightarrow 4/5$
 - $12/24 \rightarrow 1/2$
 - $144/216 \rightarrow 2/3$

<<Java Class>>	
G 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

Applying Basic Completable Future Features

- We show how to apply basic completable future features in the context of BigFraction
 - 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 ($\rightarrow$ 1/2)

<<Java Class>>	
G 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

Applying Basic Completable Future Features

- We show how to apply basic completable future features in the context of BigFraction
 - 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., $18/4 \times 2/3 = 3$

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

Applying Basic Completable Future Features

- We show how to apply basic completable future features in the context of BigFraction
 - 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., $18/4 \rightarrow 4 \frac{1}{2}$

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

Applying Basic Completable Future Features

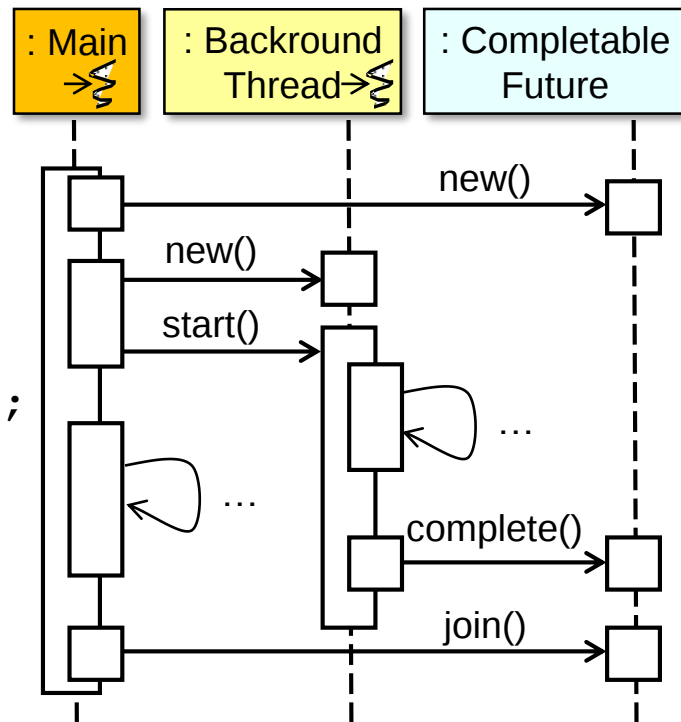
- Multiplying big fractions w/a completable future

```
CompletableFuture<BigFraction> future  
    = new CompletableFuture<>();
```

```
new Thread (() -> {  
    BigFraction bf1 =  
        new BigFraction("62675744/15668936");  
    BigFraction bf2 =  
        new BigFraction("609136/913704");  
  
    future.complete(bf1.multiply(bf2));  
}).start();
```

...

```
System.out.println(future.join().toMixedString());
```



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

Applying Basic Completable Future Features

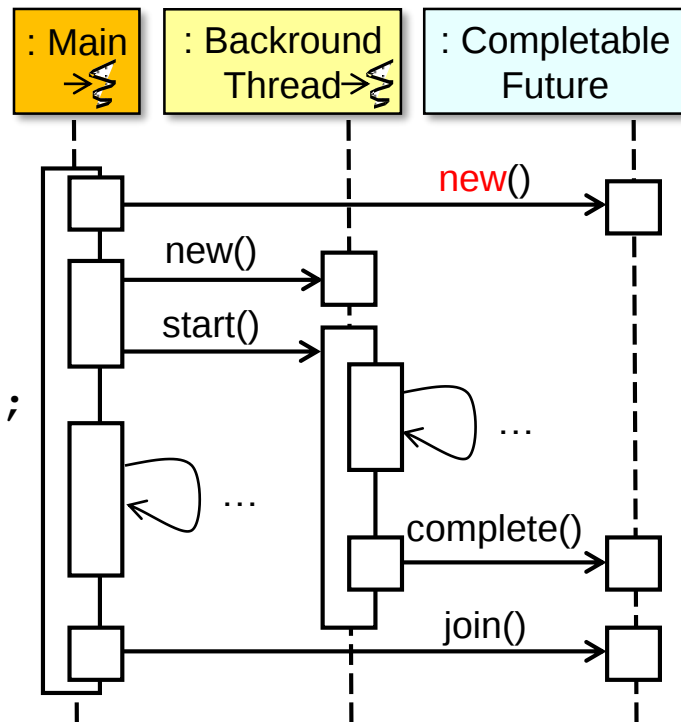
- Multiplying big fractions w/a completable future

```
CompletableFuture<BigFraction> future  
    = new CompletableFuture<>();
```

```
new Thread (() -> {  
    BigFraction bf1 =  
        new BigFraction("62675744/15668936");  
    BigFraction bf2 =  
        new BigFraction("609136/913704");  
  
    future.complete(bf1.multiply(bf2));  
}).start();
```

Make "empty" future

```
...  
System.out.println(future.join().toMixedString());
```



Applying Basic Completable Future Features

- Multiplying big fractions w/a completable future

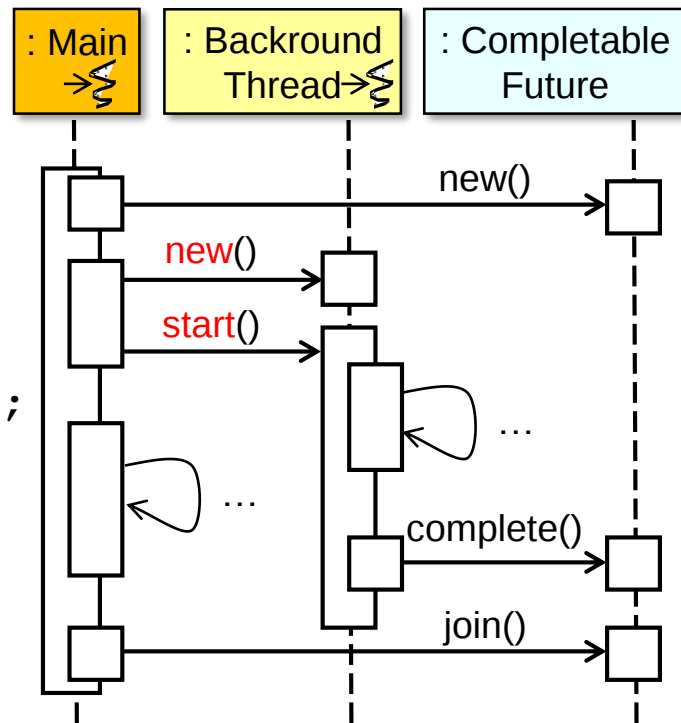
```
CompletableFuture<BigFraction> future  
    = new CompletableFuture<>();
```

```
new Thread (() -> {  
    BigFraction bf1 =  
        new BigFraction("62675744/15668936");  
    BigFraction bf2 =  
        new BigFraction("609136/913704");
```

```
    future.complete(bf1.multiply(bf2));  
}).start();
```

*Start computation in
a background thread*

```
...  
System.out.println(future.join().toMixedString());
```



Applying Basic Completable Future Features

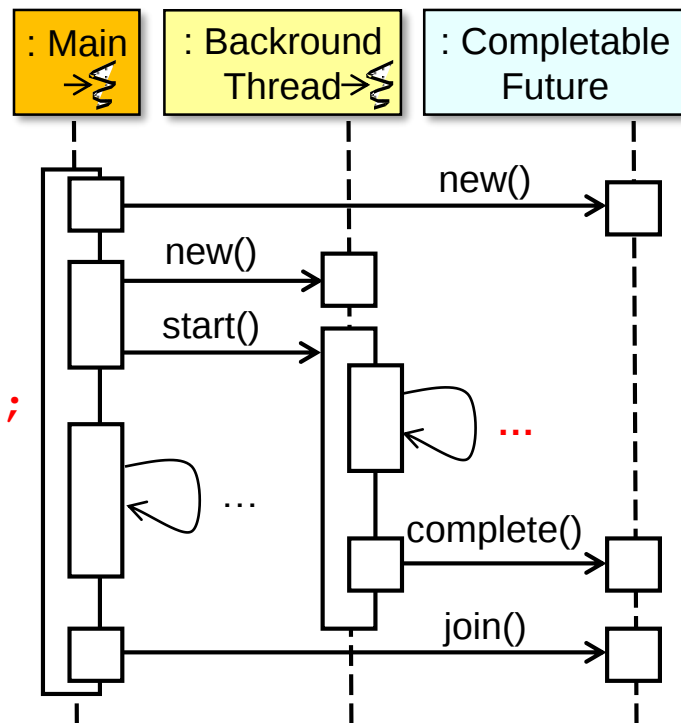
- Multiplying big fractions w/a completable future

```
CompletableFuture<BigFraction> future  
    = new CompletableFuture<>();
```

```
new Thread (() -> {  
    BigFraction bf1 =  
        new BigFraction("62675744/15668936");  
    BigFraction bf2 =  
        new BigFraction("609136/913704");  
  
    future.complete(bf1.multiply(bf2));  
}).start();
```

...

```
System.out.println(future.join().toMixedString());
```



The computation multiplies BigFractions (via BigIntegers)

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

Applying Basic Completable Future Features

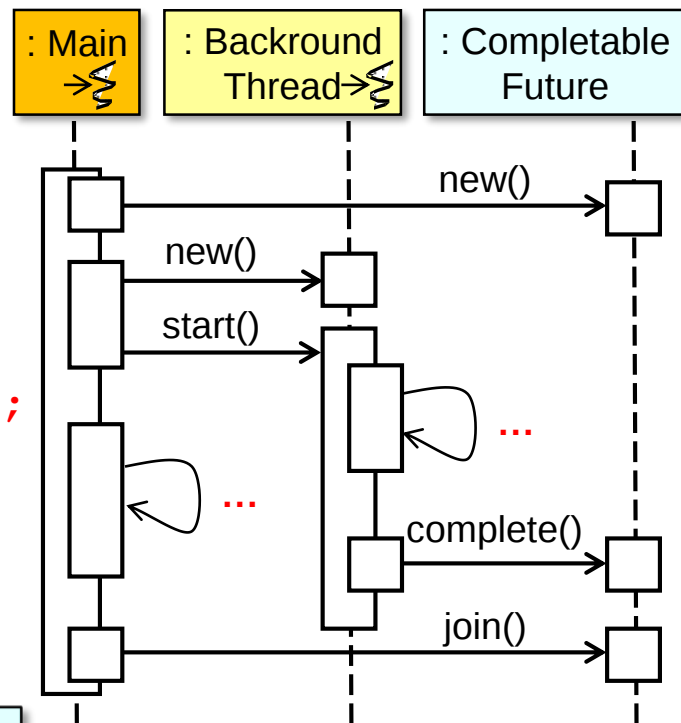
- Multiplying big fractions w/a completable future

```
CompletableFuture<BigFraction> future  
    = new CompletableFuture<>();
```

```
new Thread (() -> {  
    BigFraction bf1 =  
        new BigFraction("62675744/15668936");  
    BigFraction bf2 =  
        new BigFraction("609136/913704");  
  
    future.complete(bf1.multiply(bf2));  
}).start();
```

These computations run concurrently

```
...  
System.out.println(future.join().toMixedString());
```



Applying Basic Completable Future Features

- Multiplying big fractions w/a completable future

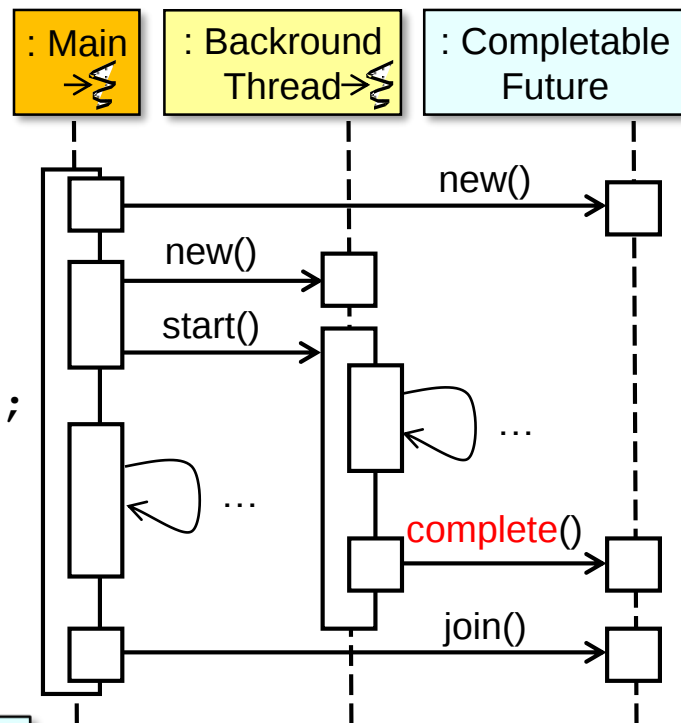
```
CompletableFuture<BigFraction> future  
    = new CompletableFuture<>();
```

```
new Thread (() -> {  
    BigFraction bf1 =  
        new BigFraction("62675744/15668936");  
    BigFraction bf2 =  
        new BigFraction("609136/913704");
```

```
    future.complete(bf1.multiply(bf2));  
}).start();
```

Explicitly complete the future w/result

```
...  
System.out.println(future.join().toMixedString());
```



Applying Basic Completable Future Features

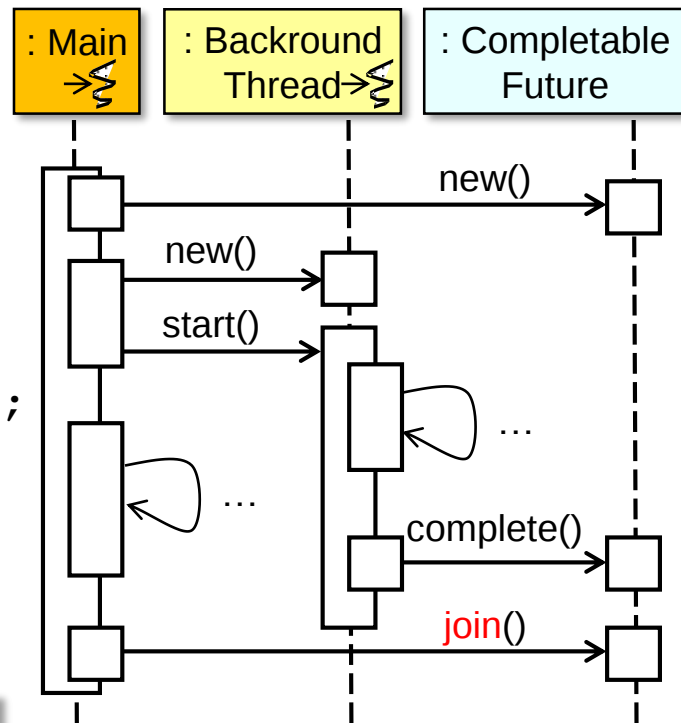
- Multiplying big fractions w/a completable future

```
CompletableFuture<BigFraction> future  
    = new CompletableFuture<>();
```

```
new Thread (() -> {  
    BigFraction bf1 =  
        new BigFraction("62675744/15668936");  
    BigFraction bf2 =  
        new BigFraction("609136/913704");  
  
    future.complete(bf1.multiply(bf2));  
}).start();
```

join() blocks until result is computed

```
...  
System.out.println(future.join() .toMixedString());
```



Limitations with Basic Completable Futures Features

Limitations with Basic Completable Futures Features

- Basic completable future features have similar limitations as futures
 - Cannot* be chained fluently to handle async results
 - Cannot* be triggered reactively
 - Cannot* be treated efficiently as a *collection* of futures

LIMITED

<<Java Interface>>

Future<V>

- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long,TimeUnit)

<<Java Class>>

CompletableFuture<T>

- CompletableFuture()
- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long,TimeUnit)
- join()
- complete(T):boolean
- supplyAsync(Supplier<U>):CompletableFuture<U>
- supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
- runAsync(Runnable):CompletableFuture<Void>
- runAsync(Runnable,Executor):CompletableFuture<Void>
- completedFuture(U):CompletableFuture<U>
- thenApply(Function<?>):CompletableFuture<U>
- thenAccept(Consumer<? super T>):CompletableFuture<Void>
- thenCombine(CompletionStage<? extends U>,BiFunction<?>):CompletableFuture<V>
- thenCompose(Function<?>):CompletableFuture<U>
- whenComplete(BiConsumer<?>):CompletableFuture<T>
- allOf(CompletableFuture[]<?>):CompletableFuture<Void>
- anyOf(CompletableFuture[]<?>):CompletableFuture<Object>

Limitations with Basic Completable Futures Features

- e.g., `join()` blocks until the future is completed..

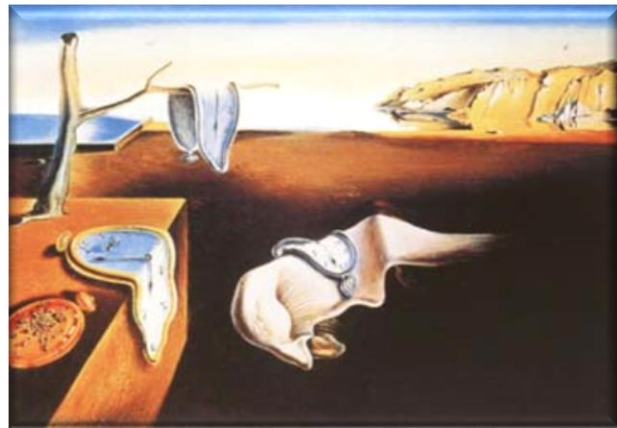
```
CompletableFuture<BigFraction> future  
    = new CompletableFuture<>();
```

```
new Thread (() -> {  
    BigFraction bf1 =  
        new BigFraction("62675744/15668936");  
    BigFraction bf2 =  
        new BigFraction("609136/913704");
```

```
    future.complete(bf1.multiply(bf2));  
}).start();
```

```
...
```

```
System.out.println(future.join().toMixedString());
```



*This blocking call underutilizes
cores & increases overhead*

Limitations with Basic Completable Futures Features

- e.g., `join()` blocks until the future is completed..

```
CompletableFuture<BigFraction> future  
    = new CompletableFuture<>();
```

```
new Thread (() -> {  
    BigFraction bf1 =  
        new BigFraction("62675744/15668936");  
    BigFraction bf2 =  
        new BigFraction("609136/913704");  
  
    future.complete(bf1.multiply(bf2));  
}).start();
```



Using a timeout to bound the blocking duration is inefficient & error-prone

```
...  
System.out.println(future.join(1, SECONDS).toMixedString());
```

See crondev.blog/2017/01/23/timeouts-with-java-8-completablefuture-youre-probably-doing-it-wrong

Limitations with Basic Completable Futures Features

- We therefore need to leverage the advanced features of completable futures



Class `CompletableFuture<T>`

```
java.lang.Object  
    java.util.concurrent.CompletableFuture<T>
```

All Implemented Interfaces:

```
CompletionStage<T>, Future<T>
```

```
public class CompletableFuture<T>  
    extends Object  
    implements Future<T>, CompletionStage<T>
```

A `Future` that may be explicitly completed (setting its value and status), and may be used as a `CompletionStage`, supporting dependent functions and actions that trigger upon its completion.

When two or more threads attempt to `complete`, `completeExceptionally`, or `cancel` a `CompletableFuture`, only one of them succeeds.

In addition to these and related methods for directly manipulating status and results, `CompletableFuture` implements interface `CompletionStage` with the following policies:

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

End of Overview of Basic Java 8 Completable Future Features (Part 2)