

# Applying Basic Java

## CompletableFuture Features

**Douglas C. Schmidt**

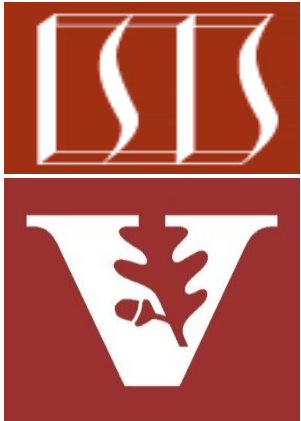
**[d.schmidt@vanderbilt.edu](mailto:d.schmidt@vanderbilt.edu)**

**[www.dre.vanderbilt.edu/~schmidt](http://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 features in the Java completable futures framework
- Know how to apply Java Completable Future basic features to asynchronously multiply BigFraction objects


<<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

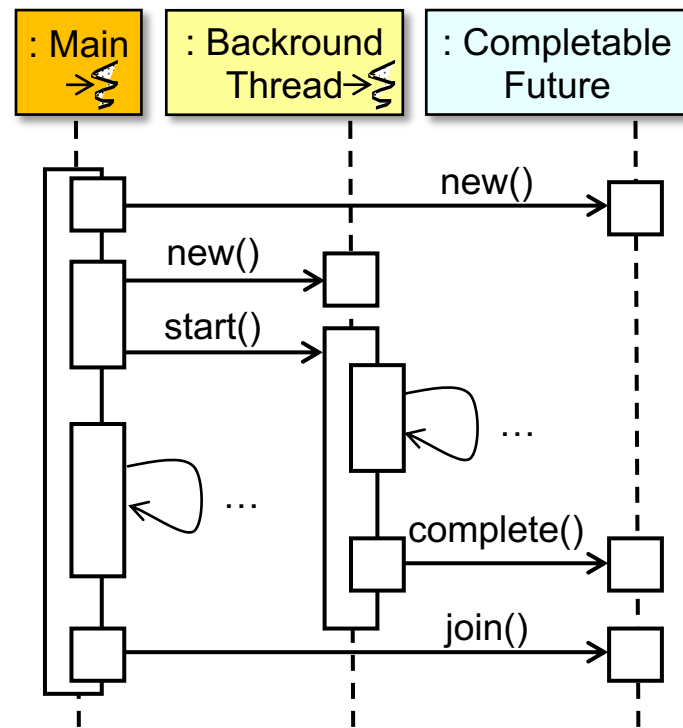
 toMixedString():String

See earlier lesson on "*Programming with Java Futures*"

# Learning Objectives in this Part of the Lesson

- Understand the basic features in the Java completable futures framework
- Know how to apply Java Completable Future basic features to asynchronously multiply BigFraction objects
- We examine source code & visualizations

```
new Thread (() -> {  
    BigFraction bf1 =  
        new BigFraction("626744/1566936");  
    BigFraction bf2 =  
        new BigFraction("609136/913704");  
  
    future.complete(bf1.multiply(bf2));  
}).start();
```



# Learning Objectives in this Part of the Lesson

- Understand the basic features in the Java completable futures framework
- Know how to apply Java Completable Future basic features to asynchronously multiply BigFraction objects
- Recognize limitations with these basic features

**LIMITED**

## 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 CompletableFuture Features

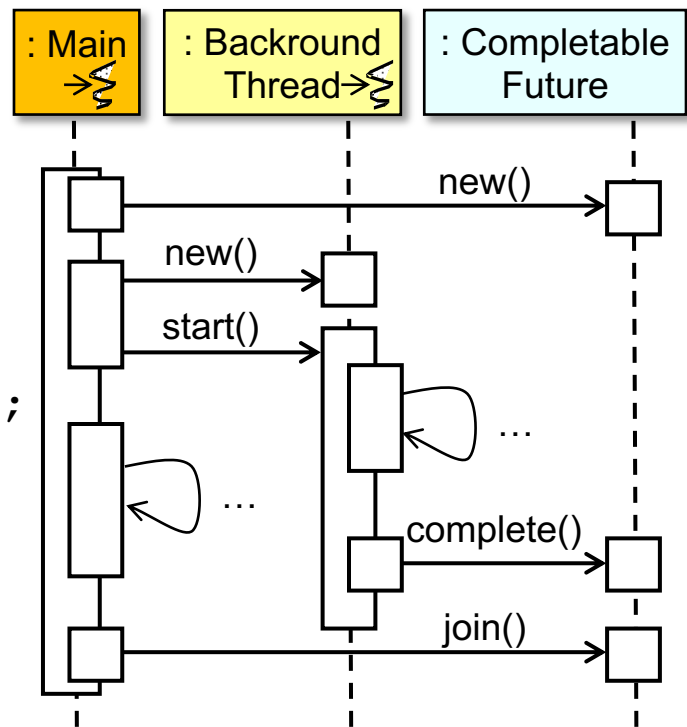
- BigFraction multiplication w/a completable future

```
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](https://github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex8)

# Applying Basic CompletableFuture Features

- BigFraction multiplication w/a completable future

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

```
new Thread Create an "incomplete" future
```



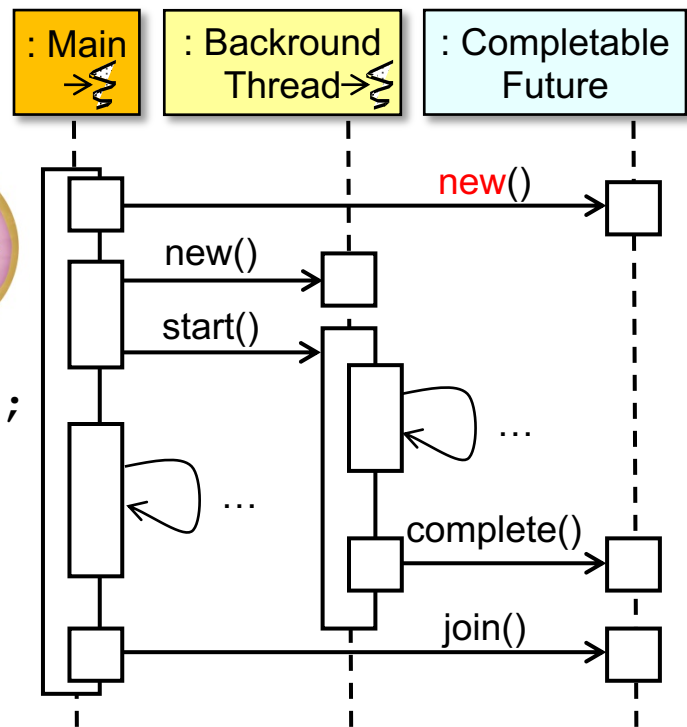
```
BigFraction bf1 =  
    new BigFraction("62675744/15668936");
```

```
BigFraction bf2 =  
    new BigFraction("609136/913704");
```

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

...

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



# Applying Basic CompletableFuture Features

- BigFraction multiplication w/a completable future

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

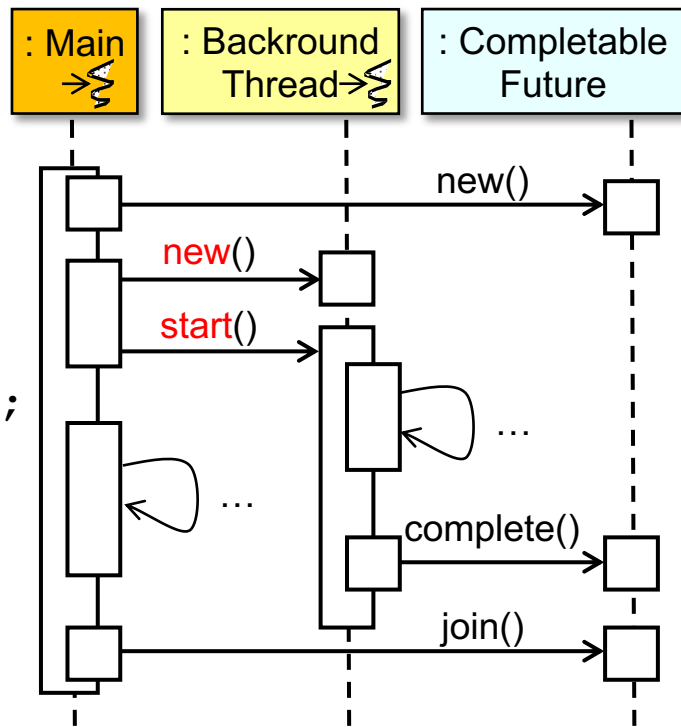
```
new Thread (() -> {  
    BigFraction bf1 =  
        new BigFraction("62675744/15668936");  
    BigFraction bf2 =  
        new BigFraction("609136/913704");
```

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

...

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

*Start computation in a background thread*





# Applying Basic CompletableFuture Features

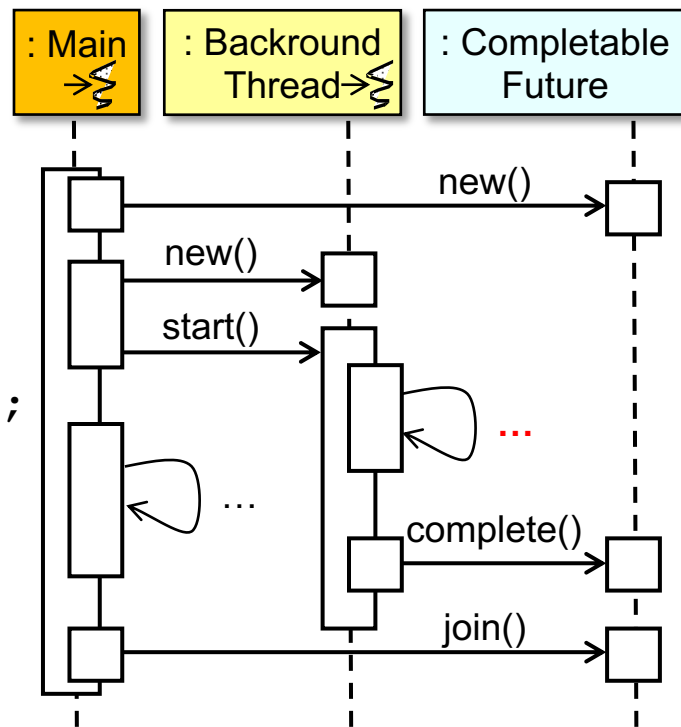
- BigFraction multiplication w/a completable future

```
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 BigInteger)*

See [docs.oracle.com/javase/8/docs/api/java/math/BigInteger.html](https://docs.oracle.com/javase/8/docs/api/java/math/BigInteger.html)

# Applying Basic CompletableFuture Features

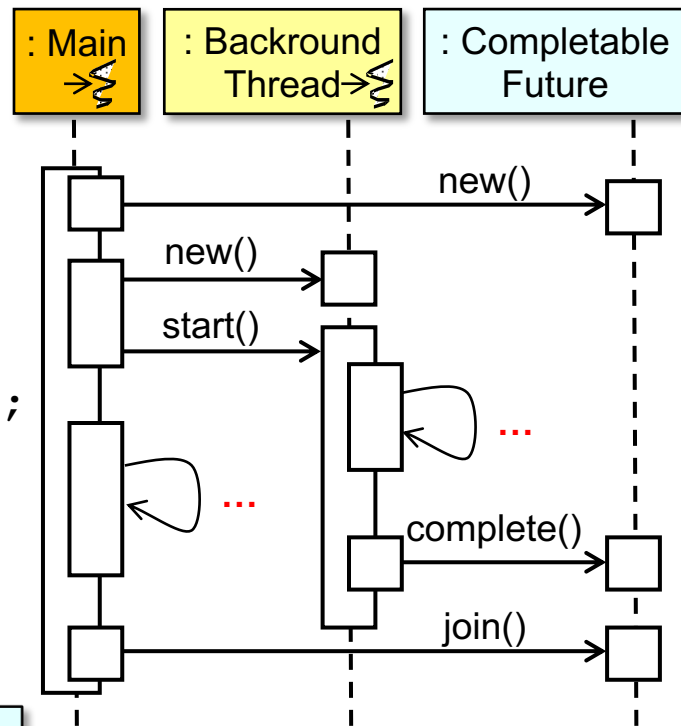
- BigFraction multiplication w/a completable future

```
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 CompletableFuture Features

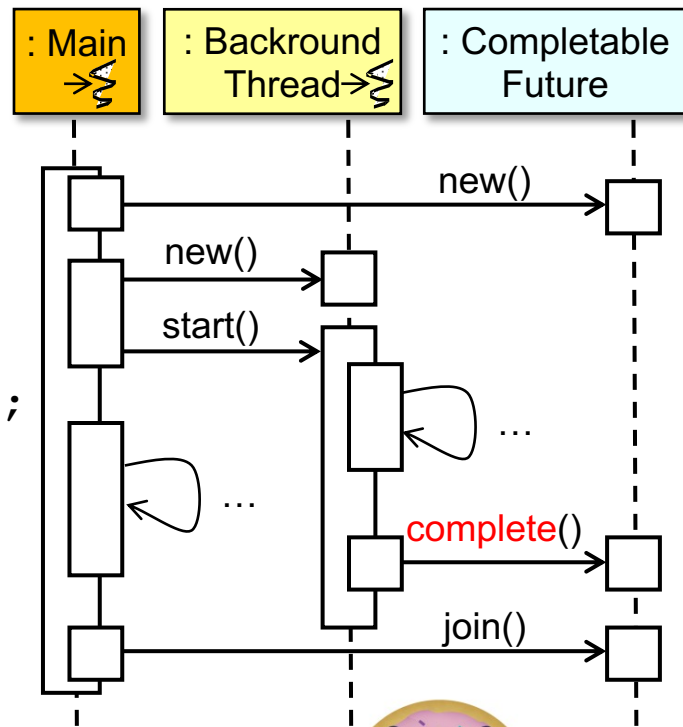
- BigFraction multiplication w/a completable future

```
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());
```



*Explicitly complete the future w/result*



you complete me

# Applying Basic CompletableFuture Features

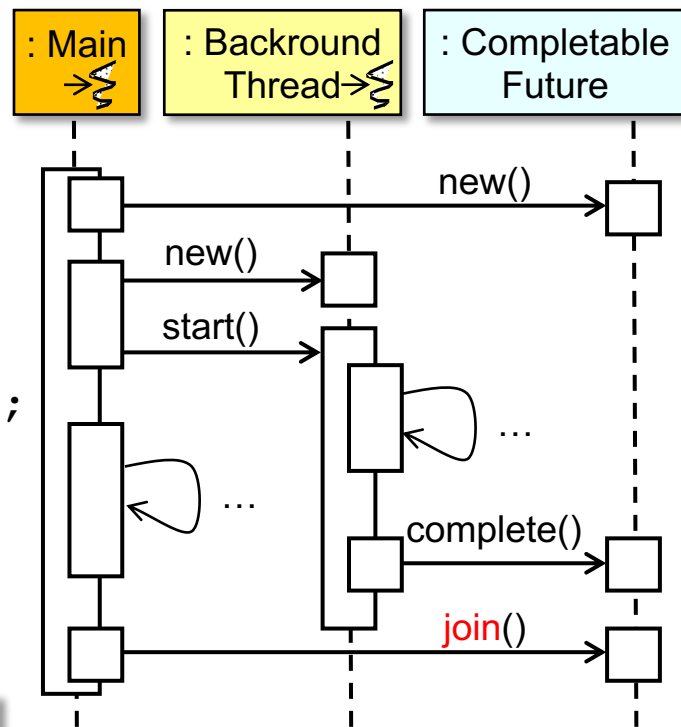
- BigFraction multiplication w/a completable future

```
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());
```



# Applying Basic CompletableFuture Features

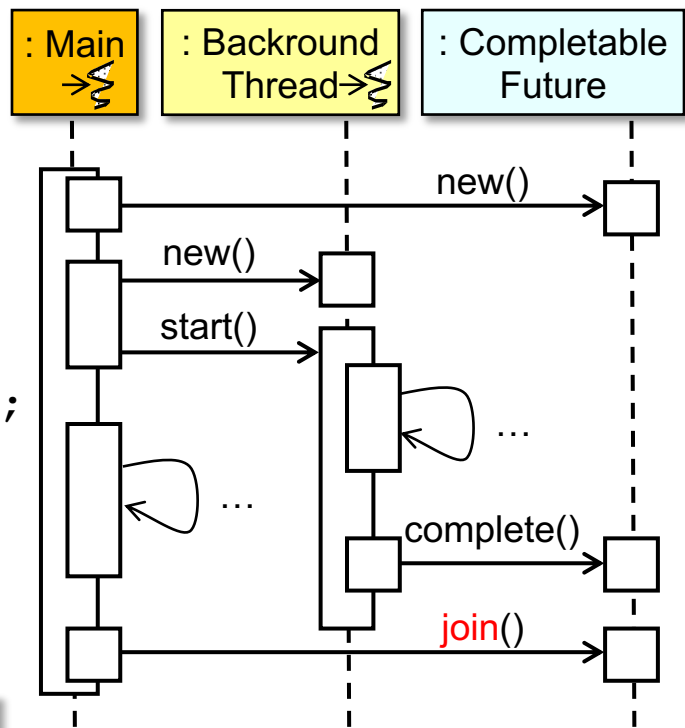
- BigFraction multiplication w/a completable future

```
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();
```

*Convert result to a mixed fraction*

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



See [www.mathsisfun.com/mixed-fractions.html](http://www.mathsisfun.com/mixed-fractions.html)

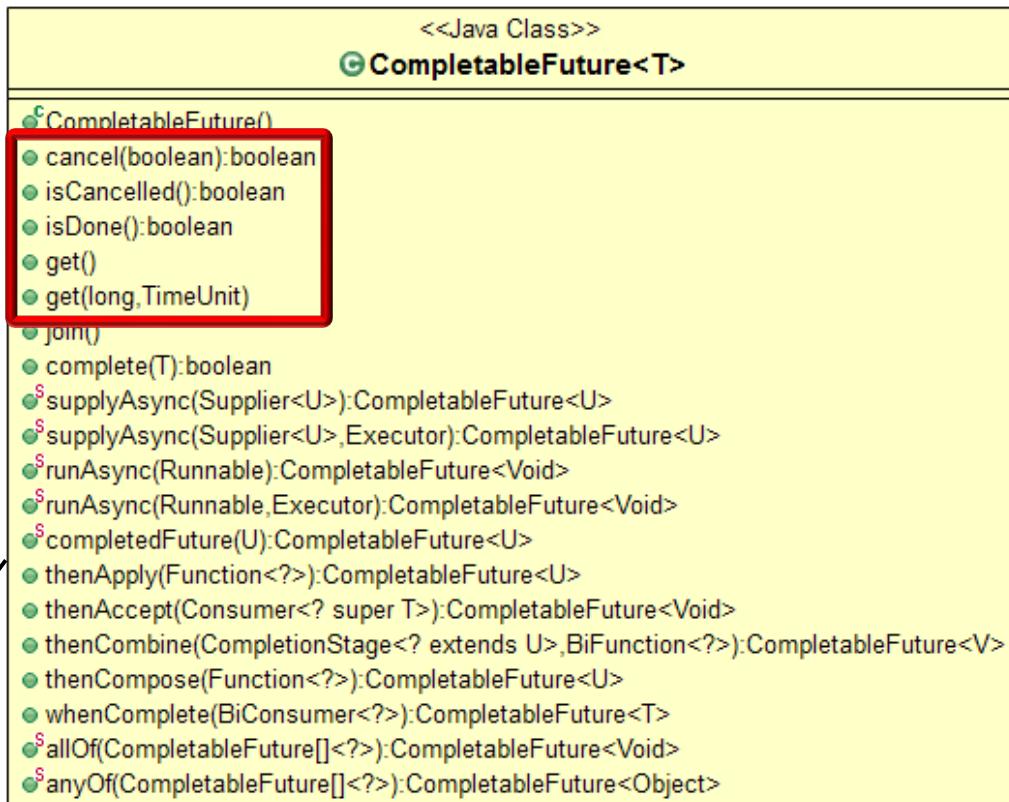
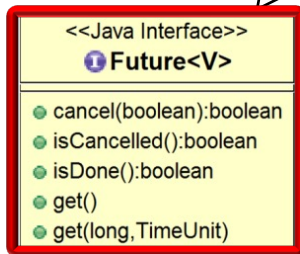
---

# Limitations with Basic CompletableFutures Features

# Limitations with Basic CompletableFuture Features

- Basic CompletableFuture 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**



See earlier lesson on "Evaluating the Pros & Cons of Java Futures"

# Limitations with Basic CompletableFutures Features

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

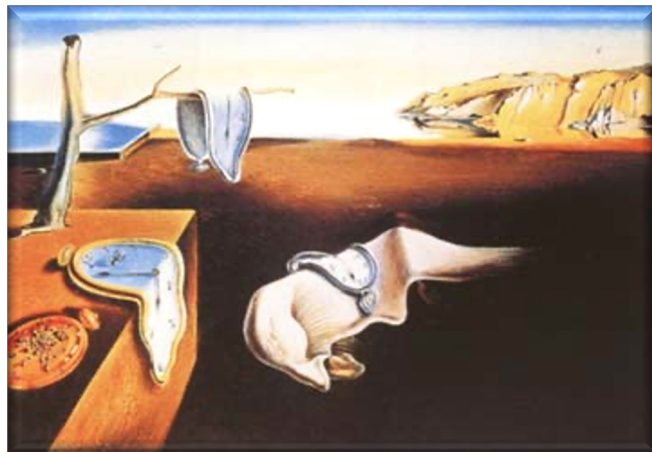
```
CompletableFuture<BigFraction> future  
    = new CompletableFuture<>();
```

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

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

*Blocking underutilizes cores & increases overhead*

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





# Limitations with Basic CompletableFuture Features

- ..using timed get() here is also problematic..

```
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.get(1, SECONDS).toMixedString());
```

See [crondev.blog/2017/01/23/timeouts-with-java-8-completablefuture-youre-probably-doing-it-wrong](https://crondev.blog/2017/01/23/timeouts-with-java-8-completablefuture-youre-probably-doing-it-wrong)

# Limitations with Basic CompletableFuture 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:

---

# End of Applying Basic Java CompletableFuture Features