

Understand Enhancements to the Java Completable Futures Framework

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

- Evaluate the pros of using the Java completable futures framework
- Evaluate the cons of using the Java completable futures framework
- Understand enhancements to the Java completable futures framework



Enhancements to the Java Completable Futures Framework

Enhancements to the Java Completable Futures Framework

- Java 9 provides enhancements to the Java 8 completable future framework

Methods	Params		
default Executor	()	Executor	Returns default <i>Executor</i> used for methods that don't specify an <i>Executor</i>
complete Async	Supplier<T>	Completable Future<T>	Complete <i>CompletableFuture</i> asynchronously using value given by the <i>Supplier</i>
orTimeout	long timeout, TimeUnit unit	Completable Future<T>	Resolves <i>CompletableFuture</i> exceptionally with <i>TimeoutException</i> , unless it is completed before the specified timeout
complete OnTimeout	T value, long timeout, TimeUnit unit	Completable Future<T>	Completes this <i>CompletableFuture</i> with the given value if not otherwise completed before the given timeout

See www.baeldung.com/java-9-completablefuture

Enhancements to the Java Completable Futures Framework

- Java 9 provides enhancements to the Java 8 completable future framework

Methods	Params		
default Executor	()	Executor	Returns default <i>Executor</i> used for methods that don't specify an <i>Executor</i>
complete Async	Supplier<T>	Completable Future<T>	Complete <i>CompletableFuture</i> asynchronously using value given by the <i>Supplier</i>
orTimeout	long timeout, TimeUnit unit	Completable Future<T>	Resolves <i>CompletableFuture</i> exceptionally with <i>TimeoutException</i> , unless it is completed before the specified timeout
complete OnTimeout	T value, long timeout, TimeUnit unit	Completable Future<T>	Completes this <i>CompletableFuture</i> with the given value if not otherwise completed before the given timeout

See docs.oracle.com/javase/9/docs/api/java/util/concurrent/CompletableFuture.html#defaultExecutor

Enhancements to the Java Completable Futures Framework

- Java 9 provides enhancements to the Java 8 completable future framework

Methods	Params		
default Executor	()	Executor	Returns default <i>Executor</i> used for methods that don't specify an <i>Executor</i>
complete Async	Supplier<T>	Completable Future<T>	Complete <i>CompletableFuture</i> asynchronously using value given by the <i>Supplier</i>
orTimeout	long timeout, TimeUnit unit	Completable Future<T>	Resolves <i>CompletableFuture</i> exceptionally with <i>TimeoutException</i> , unless it is completed before the specified timeout
complete OnTimeout	T value, long timeout, TimeUnit unit	Completable Future<T>	Completes this <i>CompletableFuture</i> with the given value if not otherwise completed before the given timeout

See docs.oracle.com/javase/9/docs/api/java/util/concurrent/CompletableFuture.html#completeAsync

Enhancements to the Java Completable Futures Framework

- Java 9 provides enhancements to the Java 8 completable future framework

Methods	Params		
default Executor	()	Executor	Returns default <i>Executor</i> used for methods that don't specify an <i>Executor</i>
complete Async	Supplier<T>	Completable Future<T>	Complete <i>CompletableFuture</i> asynchronously using value given by the <i>Supplier</i>
orTimeout	long timeout, TimeUnit unit	Completable Future<T>	Resolves <i>CompletableFuture</i> exceptionally with <i>TimeoutException</i> , unless it is completed before the specified timeout
complete OnTimeout	T value, long timeout, TimeUnit unit	Completable Future<T>	Completes this <i>CompletableFuture</i> with the given value if not otherwise completed before the given timeout

See docs.oracle.com/javase/9/docs/api/java/util/concurrent/CompletableFuture.html#orTimeout

Enhancements to the Java Completable Futures Framework

- Java 9 provides enhancements to the Java 8 completable future framework

Methods	Params		
default Executor	()	Executor	Returns default <i>Executor</i> used for methods that don't specify an <i>Executor</i>
complete Async	Supplier<T>	Completable Future<T>	Complete <i>CompletableFuture</i> asynchronously using value given by the <i>Supplier</i>
orTimeout	long timeout, TimeUnit unit	Completable Future<T>	Resolves <i>CompletableFuture</i> exceptionally with <i>TimeoutException</i> , unless it is completed before the specified timeout
complete OnTimeout	T value, long timeout, TimeUnit unit	Completable Future<T>	Completes this <i>CompletableFuture</i> with the given value if not otherwise completed before the given timeout

See docs.oracle.com/javase/9/docs/api/java/util/concurrent/CompletableFuture.html#completeOnTimeout

Applying orTimeout() & completeOnTimeout()

Applying the `orTimeout()` & `completeOnTimeout()` Methods

- This example asynchronously performs web services within a bounded time

```
CompletableFuture
```

```
.supplyAsync(  
    () -> findBestPrice("LDN - NYC"))  
.thenCombine(CompletableFuture  
    .supplyAsync  
        (() -> queryExchangeRateFor("GBP"))  
        .completeOnTimeout(DEFAULT_RATE, 2, SECONDS),  
    this::convert)  
.orTimeout(3, SECONDS)  
.whenComplete((amount, ex) -> {  
    if (amount != null)  
        { System.out.println("The price is: " + amount + "GBP"); }  
    else { System.out.println(ex.toString()); }  
})) ...
```

See github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex27

Applying the `orTimeout()` & `completeOnTimeout()` Methods

- This example asynchronously performs web services within a bounded time

CompletableFuture

Asynchronously find the best price for a flight

```
.supplyAsync(  
    () -> findBestPrice("LDN - NYC"))
```

```
.thenCombine(CompletableFuture
```



```
.supplyAsync
```

```
    () -> queryExchangeRateFor("GBP"))
```

```
.completeOnTimeout(DEFAULT_RATE, 2, SECONDS),  
this::convert)
```

```
.orTimeout(3, SECONDS)
```

```
.whenComplete((amount, ex) -> {
```

```
    if (amount != null)
```

```
        { System.out.println("The price is: " + amount + "GBP"); }
```

```
    else { System.out.println(ex.toString()); }
```

```
}) ...
```

`supplyAsync()` uses the default executor (i.e., common fork-join pool)

Applying the orTimeout() & completeOnTimeout() Methods

- This example asynchronously performs web services within a bounded time

CompletableFuture

```
.supplyAsync(  
    () -> findBestPrice("LDN - NYC"))  
.thenCombine(CompletableFuture  
    .supplyAsync  
        (() -> queryExchangeRateFor("GBP"))  
        .completeOnTimeout(DEFAULT_RATE, 2, SECONDS),  
    this::convert)  
.orTimeout(3, SECONDS)  
.whenComplete((amount, ex) -> {  
    if (amount != null)  
        { System.out.println("The price is: " + amount + "GBP"); }  
    else { System.out.println(ex.toString()); }  
})) ...
```



Asynchronously find the latest exchange rate

Applying the `orTimeout()` & `completeOnTimeout()` Methods

- This example asynchronously performs web services within a bounded time

`CompletableFuture`

```
.supplyAsync(  
    () -> findBestPrice("LDN - NYC"))  
.thenCombine(CompletableFuture  
    .supplyAsync  
        (() -> queryExchangeRateFor("GBP"))  
        .completeOnTimeout(DEFAULT_RATE, 2, SECONDS),  
    this::convert)  
.orTimeout(3, SECONDS)  
.whenComplete((amount, ex) -> {  
    if (amount != null)  
        { System.out.println("The price is: " + amount + "GBP"); }  
    else { System.out.println(ex.toString()); }  
})) ...
```

Select the default rate if call runs longer than 2 seconds

Applying the `orTimeout()` & `completeOnTimeout()` Methods

- This example asynchronously performs web services within a bounded time

`CompletableFuture`

```
.supplyAsync(  
    () -> findBestPrice("LDN - NYC"))  
.thenCombine(CompletableFuture  
    .supplyAsync  
        (() -> queryExchangeRateFor("GBP"))  
    .completeOnTimeout(DEFAULT_RATE, 2, SECONDS),  
    this::convert)  
.orTimeout(3, SECONDS)  
.whenComplete((amount, ex) -> {  
    if (amount != null)  
        { System.out.println("The price is: " + amount + "GBP"); }  
    else { System.out.println(ex.toString()); }  
})) ...
```

*Combine & convert
the search results*

Applying the `orTimeout()` & `completeOnTimeout()` Methods

- This example asynchronously performs web services within a bounded time

`CompletableFuture`

```
.supplyAsync(  
    () -> findBestPrice("LDN - NYC"))  
.thenCombine(CompletableFuture  
    .supplyAsync  
        (() -> queryExchangeRateFor("GBP"))  
    .completeOnTimeout(DEFAULT_RATE, 2, SECONDS),  
    this::convert)  
.orTimeout(3, SECONDS)  
.whenComplete((amount, ex) -> {  
    if (amount != null)  
        { System.out.println("The price is: " + amount + "GBP"); }  
    else { System.out.println(ex.toString()); }  
})) ...
```

*Throws `TimeoutException`
if the time period elapses*

See 4comprehension.com/completablefuture-timeout

Applying the `orTimeout()` & `completeOnTimeout()` Methods

- This example asynchronously performs web services within a bounded time

`CompletableFuture`

```
.supplyAsync(  
    () -> findBestPrice("LDN - NYC"))  
.thenCombine(CompletableFuture  
    .supplyAsync  
        (() -> queryExchangeRateFor("GBP"))  
    .completeOnTimeout(DEFAULT_RATE, 2, SECONDS),  
    this::convert)  
.orTimeout(3, SECONDS)  
.whenComplete((amount, ex) -> {  
    if (amount != null)  
        { System.out.println("The price is: " + amount + "GBP"); }  
    else { System.out.println(ex.toString()); }  
}) ...
```

This method is always called, w/ or w/out an exception being thrown

Applying the orTimeout() & completeOnTimeout() Methods

- This example asynchronously performs web services within a bounded time

```
CompletableFuture
```

```
.supplyAsync(  
    () -> findBestPrice("LDN - NYC"))  
.thenCombine(CompletableFuture  
    .supplyAsync  
        (() -> queryExchangeRateFor("GBP"))  
    .completeOnTimeout(DEFAULT_RATE, 2, SECONDS),  
    this::convert)  
.orTimeout(3, SECONDS)  
.whenComplete((amount, ex) -> {  
    if (amount != null)  
        { System.out.println("The price is: " + amount + "GBP"); }  
    else { System.out.println(ex.toString()); }  
}) ...
```

*Print results if async
calls finished normally*

Applying the `orTimeout()` & `completeOnTimeout()` Methods

- This example asynchronously performs web services within a bounded time

`CompletableFuture`

```
.supplyAsync(  
    () -> findBestPrice("LDN - NYC"))  
.thenCombine(CompletableFuture  
    .supplyAsync  
        (() -> queryExchangeRateFor("GBP"))  
        .completeOnTimeout(DEFAULT_RATE, 2, SECONDS),  
    this::convert)  
.orTimeout(3, SECONDS)  
.whenComplete((amount, ex) -> {  
    if (amount != null)  
        { System.out.println("The price is: " + amount + "GBP"); }  
    else { System.out.println(ex.toString()); }  
}) ...
```

*Print the exception if
async call timed out*

Applying completeAsync() to Big Fractions

Applying completeAsync() to Big Fractions

- Using completeAsync() to multiply big fractions

```
void testFractionMultiplicationCompleteAsync() {
    StringBuilder sb = new StringBuilder(">> Starting test\n");
    String f1 = "62675744/15668936"; String f2 = "609136/913704";

    CompletableFuture<BigFraction> f = new CompletableFuture<>();
    f.thenRun(() -> sb.append("completeAsync() result = "));
    f.completeAsync(() -> {
        BigFraction bf1 = new BigFraction(f1);
        BigFraction bf2 = new BigFraction(f2);
        return bf1.multiply(bf2); });
    ...
    sb.append(f.join().toMixedString()); display(sb.toString());
}
```

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

Applying completeAsync() to Big Fractions

- Using completeAsync() to multiply big fractions

```
void testFractionMultiplicationCompleteAsync() {  
    StringBuilder sb = new StringBuilder(">> Starting test\n");  
    String f1 = "62675744/15668936"; String f2 = "609136/913704";
```

This string builder holds intermediate results

```
    CompletableFuture<BigFraction> f = new CompletableFuture<>();  
    f.thenRun(() -> sb.append("completeAsync() result = "));  
    f.completeAsync(() -> {  
        BigFraction bf1 = new BigFraction(f1);  
        BigFraction bf2 = new BigFraction(f2);  
        return bf1.multiply(bf2); });  
    ...  
    sb.append(f.join().toMixedString()); display(sb.toString());  
}
```

Applying completeAsync() to Big Fractions

- Using completeAsync() to multiply big fractions

```
void testFractionMultiplicationCompleteAsync() {  
    StringBuilder sb = new StringBuilder(">> Starting test\n");  
    String f1 = "62675744/15668936"; String f2 = "609136/913704";
```

Define a pair of big fractions

```
    CompletableFuture<BigFraction> f = new CompletableFuture<>();  
    f.thenRun(() -> sb.append("completeAsync() result = "));  
    f.completeAsync(() -> {  
        BigFraction bf1 = new BigFraction(f1);  
        BigFraction bf2 = new BigFraction(f2);  
        return bf1.multiply(bf2); });  
    ...  
    sb.append(f.join().toMixedString()); display(sb.toString());  
}
```

Applying completeAsync() to Big Fractions

- Using completeAsync() to multiply big fractions

```
void testFractionMultiplicationCompleteAsync() {  
    StringBuilder sb = new StringBuilder(">> Starting test\n");  
    String f1 = "62675744/15668936"; String f2 = "609136/913704";
```

Create a new empty completable future

```
    CompletableFuture<BigFraction> f = new CompletableFuture<>();  
    f.thenRun(() -> sb.append("completeAsync() result = "));  
    f.completeAsync(() -> {  
        BigFraction bf1 = new BigFraction(f1);  
        BigFraction bf2 = new BigFraction(f2);  
        return bf1.multiply(bf2); });  
    ...  
    sb.append(f.join().toMixedString()); display(sb.toString());  
}
```

Applying completeAsync() to Big Fractions

- Using completeAsync() to multiply big fractions

```
void testFractionMultiplicationCompleteAsync() {  
    StringBuilder sb = new StringBuilder(">> Starting test\n");  
    String f1 = "62675744/15668936"; String f2 = "609136/913704";
```

Register an action that appends a string when run (in calling thread)

```
    CompletableFuture<BigFraction> f = new CompletableFuture<>();  
    f.thenRun(() -> sb.append("completeAsync() result = "));  
    f.completeAsync(() -> {  
        BigFraction bf1 = new BigFraction(f1);  
        BigFraction bf2 = new BigFraction(f2);  
        return bf1.multiply(bf2); });  
    ...  
    sb.append(f.join().toMixedString()); display(sb.toString());  
}
```

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

Applying completeAsync() to Big Fractions

- Using completeAsync() to multiply big fractions

```
void testFractionMultiplicationCompleteAsync() {  
    StringBuilder sb = new StringBuilder(">> Starting test\n");  
    String f1 = "62675744/15668936"; String f2 = "609136/913704";
```

Arrange to execute a supplier lambda in common fork-join pool

```
    CompletableFuture<BigFraction> f = new CompletableFuture<>();  
    f.thenRun(() -> sb.append("completeAsync() result = "));  
    f.completeAsync(() -> {  
        BigFraction bf1 = new BigFraction(f1);  
        BigFraction bf2 = new BigFraction(f2);  
        return bf1.multiply(bf2); });  
    ...  
    sb.append(f.join().toMixedString()); display(sb.toString());  
}
```

See docs.oracle.com/javase/9/docs/api/java/util/concurrent/CompletableFuture.html#completeAsync

Applying completeAsync() to Big Fractions

- Using completeAsync() to multiply big fractions

```
void testFractionMultiplicationCompleteAsync() {  
    StringBuilder sb = new StringBuilder(">> Starting test\n");  
    String f1 = "62675744/15668936"; String f2 = "609136/913704";  
  
    CompletableFuture<BigFraction> f = new CompletableFuture<>();  
    f.thenRun(() -> sb.append("completeAsync() result = "));  
    f.completeAsync(() -> {  
        BigFraction bf1 = new BigFraction(f1);  
        BigFraction bf2 = new BigFraction(f2);  
        return bf1.multiply(bf2); });  
    ...  
    sb.append(f.join().toMixedString()); display(sb.toString());  
}
```



This method sets all the processing in motion

Applying completeAsync() to Big Fractions

- Using completeAsync() to multiply big fractions

```
void testFractionMultiplicationCompleteAsync() {
    StringBuilder sb = new StringBuilder(">> Starting test\n");
    String f1 = "62675744/15668936"; String f2 = "609136/913704";

    CompletableFuture<BigFraction> f = new CompletableFuture<>();
    f.thenRun(() -> sb.append("completeAsync() result = "));
    f.completeAsync(() -> {
        BigFraction bf1 = new BigFraction(f1);
        BigFraction bf2 = new BigFraction(f2);
        return bf1.multiply(bf2); });
    ...
    sb.append(f.join().toMixedString()); display(sb.toString());
}
```

*This supplier lambda
is used to multiply two
BigFraction objects*

Applying completeAsync() to Big Fractions

- Using completeAsync() to multiply big fractions

```
void testFractionMultiplicationCompleteAsync() {  
    StringBuilder sb = new StringBuilder(">> Starting test\n");  
    String f1 = "62675744/15668936"; String f2 = "609136/913704";  
  
    CompletableFuture<BigFraction> f = new CompletableFuture<>();  
    f.thenRun(() -> sb.append("completeAsync() result = "));  
    f.completeAsync(() -> {  
        BigFraction bf1 = new BigFraction(f1);  
        BigFraction bf2 = new BigFraction(f2);  
        return bf1.multiply(bf2); });  
    ...  
    sb.append(f.join().toMixedString()); display(sb.toString());  
}
```

*These computations
run concurrently*

Applying completeAsync() to Big Fractions

- Using completeAsync() to multiply big fractions

```
void testFractionMultiplicationCompleteAsync() {
    StringBuilder sb = new StringBuilder(">> Starting test\n");
    String f1 = "62675744/15668936"; String f2 = "609136/913704";

    CompletableFuture<BigFraction> f = new CompletableFuture<>();
    f.thenRun(() -> sb.append("completeAsync() result = "));
    f.completeAsync(() -> {
        BigFraction bf1 = new BigFraction(f1);
        BigFraction bf2 = new BigFraction(f2);
        return bf1.multiply(bf2); });
    ...
    sb.append(f.join().toMixedString()); display(sb.toString());
}
```

join() blocks until result is complete

Applying completeAsync() to Big Fractions

- Using completeAsync() to multiply big fractions

```
void testFractionMultiplicationCompleteAsync() {
    StringBuilder sb = new StringBuilder(">> Starting test\n");
    String f1 = "62675744/15668936"; String f2 = "609136/913704";

    CompletableFuture<BigFraction> f = new CompletableFuture<>();
    f.thenRun(() -> sb.append("completeAsync() result = "));
    f.completeAsync(() -> {
        BigFraction bf1 = new BigFraction(f1);
        BigFraction bf2 = new BigFraction(f2);
        return bf1.multiply(bf2); });
    ...
    sb.append(f.join().toMixedString()); display(sb.toString());
}
```

Append the mixed fraction to sb

Applying completeAsync() to Big Fractions

- Using completeAsync() to multiply big fractions

```
void testFractionMultiplicationCompleteAsync() {
    StringBuilder sb = new StringBuilder(">> Starting test\n");
    String f1 = "62675744/15668936"; String f2 = "609136/913704";

    CompletableFuture<BigFraction> f = new CompletableFuture<>();
    f.thenRun(() -> sb.append("completeAsync() result = "));
    f.completeAsync(() -> {
        BigFraction bf1 = new BigFraction(f1);
        BigFraction bf2 = new BigFraction(f2);
        return bf1.multiply(bf2); });
    ...
    sb.append(f.join().toMixedString()); display(sb.toString());
}
```

*Display output
as a string*

End of Understand Enhancements to the Java Completable Futures Framework