

How Java Completable Futures Overcome Limitations of Java Futures

Douglas C. Schmidt

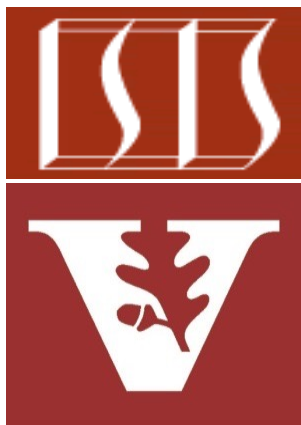
d.schmidt@vanderbilt.edu

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

- Recognize how Java completable futures overcome limitations with Java futures



See earlier lesson on "*Overview of the Java Completable Futures Framework*"

Overcoming Limitations with Java Futures

Overcoming Limitations with Java Futures

- The completable futures framework overcomes Java future limitations



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

Overcoming Limitations with Java Futures

- The completable futures framework overcomes Java future limitations
- *Can* be completed explicitly



you complete me

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

```
new Thread (() -> {  
    ...  
    future.complete(...);  
}).start();
```

*After complete() is done
calls to join() will unblock*

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

Overcoming Limitations with Java Futures

- The completable futures framework overcomes Java future limitations

- *Can* be completed explicitly
- *Can* be chained fluently to handle async results efficiently & cleanly

`CompletableFuture`

```
.supplyAsync(reduceFraction)
.thenApply(BigFraction
           :: toMixedString)
.thenAccept(System.out::println);
```

The action of each "completion stage" is triggered when the previous stage's future completes asynchronously

Overcoming Limitations with Java Futures

- The completable futures framework overcomes Java future limitations

- *Can* be completed explicitly
- *Can* be chained fluently to handle async results efficiently & cleanly
- Async programming thus looks more like sync programming!

`CompletableFuture`

```
.supplyAsync(reduceFraction)  
.thenApply(BigFraction  
           :: toMixedString)  
.thenAccept(System.out::println);
```



Overcoming Limitations with Java Futures

- The completable futures framework overcomes Java future limitations

- *Can* be completed explicitly
- *Can* be chained fluently to handle async results efficiently & cleanly
- *Can* be triggered reactively & efficiently as a *collection* of futures w/out undue overhead



```
CompletableFuture<List  
<BigFraction>> futureToList =  
Stream  
    .generate(generator)  
    .limit(sMAX_FRACTIONS)  
    .map(reduceFractions)  
    .collect(FuturesCollector  
        .toFuture());  
  
futureToList  
    .thenAccept(printList);
```

Create a single future that will be triggered when a group of other futures all complete

Overcoming Limitations with Java Futures

- The completable futures framework overcomes Java future limitations

- *Can* be completed explicitly
- *Can* be chained fluently to handle async results efficiently & cleanly
- *Can* be triggered reactively & efficiently as a *collection* of futures w/out undue overhead



```
CompletableFuture<List  
<BigFraction>> futureToList =  
Stream  
    .generate(generator)  
    .limit(sMAX_FRACTIONS)  
    .map(reduceFractions)  
    .collect(FuturesCollector  
        .toFuture());
```

```
futureToList  
    .thenAccept(printList);
```

*Print out the results after all async
fraction reductions have completed*

Overcoming Limitations with Java Futures

- The completable futures framework overcomes Java future limitations

- *Can* be completed explicitly
- *Can* be chained fluently to handle async results efficiently & cleanly
- *Can* be triggered reactively & efficiently as a *collection* of futures w/out undue overhead



```
CompletableFuture<List  
<BigFraction>> futureToList =  
    Stream  
        .generate(generator)  
        .limit(sMAX_FRACTIONS)  
        .map(reduceFractions)  
        .collect(FuturesCollector  
            .toFuture());
```

```
futureToList  
    .thenAccept(printList);
```

Java completable futures can also be combined with Java sequential streams

End of How Java Completable Futures Overcome Limitations of Java Futures