Overcoming Limitations of Java Futures via Java Completable Futures

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

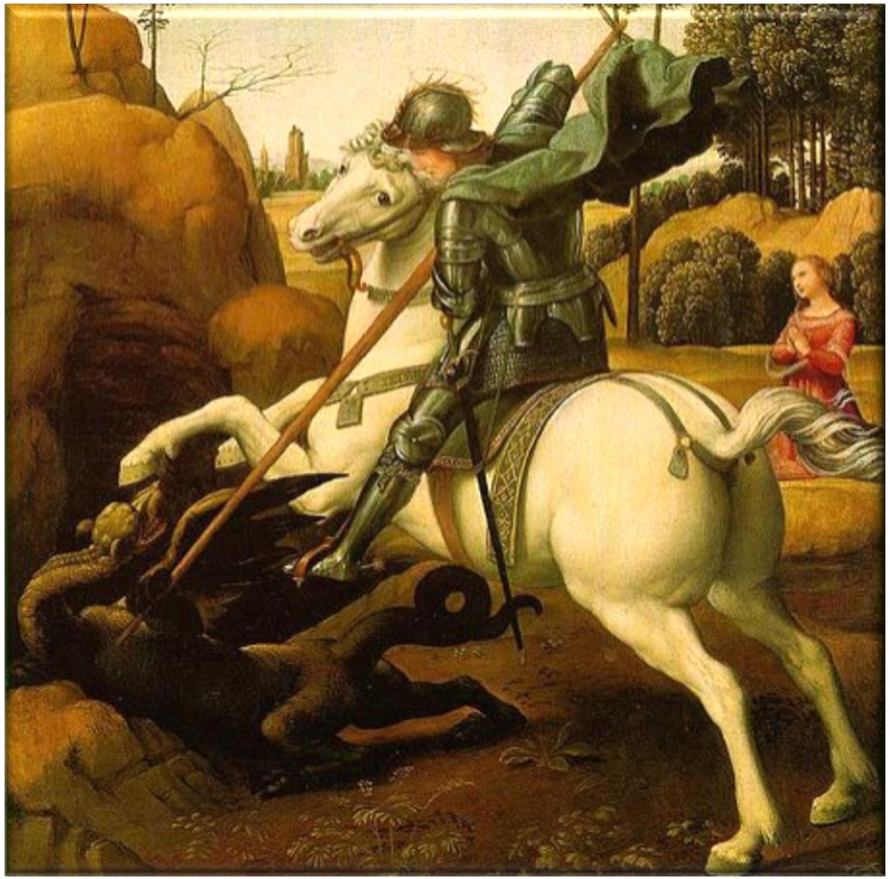
Institute for Software Integrated Systems

Vanderbilt University

Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Know how Java completable futures overcome limitations with Java futures

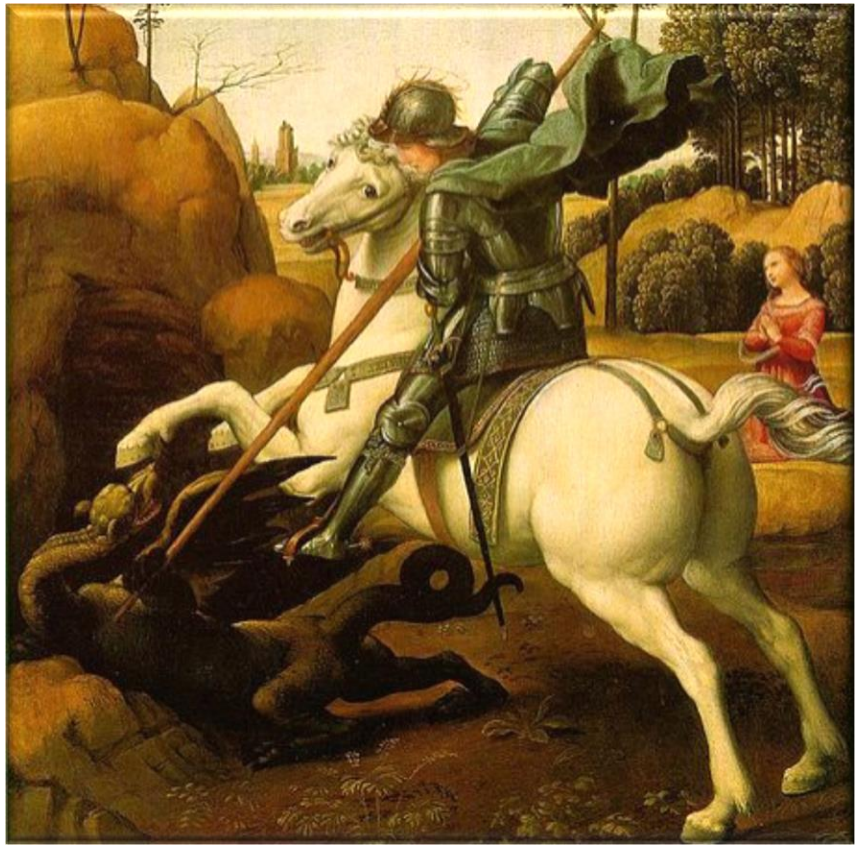


See en.wikipedia.org/wiki/Java_version_history

Overcoming Limitations with Java Futures

Overcoming Limitations with Java Futures

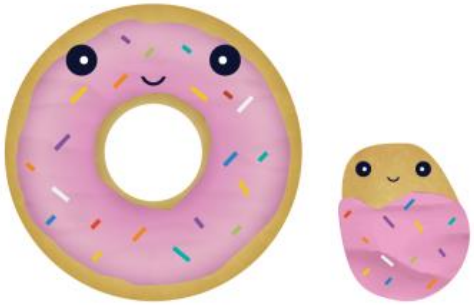
- The completable future framework overcomes Java future limitations



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

Overcoming Limitations with Java Futures

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




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

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

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

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

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

Overcoming Limitations with Java Futures

- The completable future framework overcomes Java future limitations
  - *Can* be completed explicitly
  - *Can* be chained fluently to handle async results efficiently & cleanly

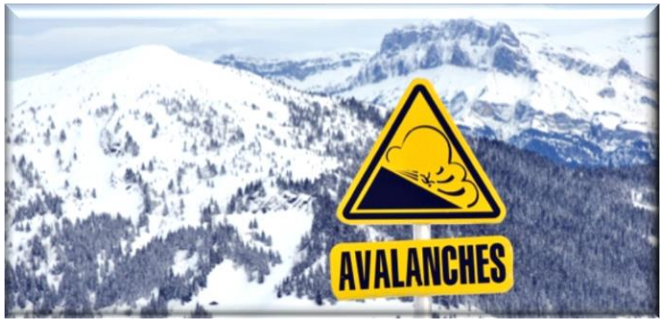
```
CompletableFuture
    .supplyAsync(reduceFraction)
    .thenApply(BigFraction
                    ::toMixedString)
    .thenAccept(System.out::println);
```

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

See en.wikipedia.org/wiki/Fluent_interface

Overcoming Limitations with Java Futures

- The completable future 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



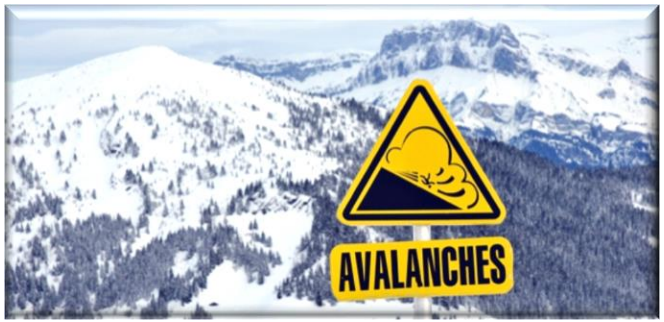

```
CompletableFuture<List
   <BigFraction>> futureToList =
   Stream
      .generate(generator)
      .limit(sMAX_FRACTIONS)
      .map(reduceFractions)
      .collect(FuturesCollector
                  .toFutures());
futureToList
   .thenAccept(printList);
```

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

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

Overcoming Limitations with Java Futures

- The completable future 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
                  .toFutures());
futureToList
   .thenAccept(printList);
```

Print out the results after all async fraction reductions have completed

Overcoming Limitations with Java Futures

- The completable future 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
                  .toFutures());
futureToList
   .thenAccept(printList);
```

Java completable futures can also be combined with Java sequential streams

End of Overcoming Limitations of Java Futures via Java Completable Futures