Overview of Java 8 CompletetableFutures

(Part 1)

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 the Java 8 completable futures framework provides an async concurrent programming model.

```java
/page\ = supplyAsync
     (getStartPage())

/imgNum\ = /page\ .thenApplyAsync
             (countImages(page))
             .thenApply(List::size)

/imgNum\ = /page\ .thenComposeAsync
            (crawlHyperLinks(page))

/imgNum\ .thenCombine(/imgNum\
            (imgNum, imgNum) ->
            Integer::sum)
```
Learning Objectives in this Part of the Lesson

• Know how the Java 8 completable futures framework provides an async concurrent programming model

• Recognize Java 8 completable futures overcome limitations with Java 5 futures
Overview of Completable Futures
Overview of Completable Futures

- Java’s 8 completable future framework provides an asynchronous & reactive concurrent programming model.

---

### Class CompletableFuture<T>

```java
java.lang.Object
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](docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html)
Overview of Completable Futures

- Java’s 8 completable future framework provides an asynchronous & reactive concurrent programming model
- Supports dependent actions that trigger upon completion of async operations

Task 1: Get start page asynchronously
Task 2: Count images on the page asynchronously
Task 3: Count images on all hyperlinked pages asynchronously
Task 4: Combine results to create the total asynchronously

These dependencies can be modeled via a data flow diagram

See en.wikipedia.org/wiki/Web_crawler
Overview of Completable Futures

- Java’s 8 completable future framework provides an asynchronous & reactive concurrent programming model
- Supports dependent actions that trigger upon completion of async operations

Async operations can be forked, chained, & joined

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletionStage.html
Overview of Completable Futures

- Java’s 8 completable future framework provides an asynchronous & reactive concurrent programming model
- Supports dependent actions that trigger upon completion of async operations
- Async operations can run concurrently in thread pools

See [www.nurkiewicz.com/2013/05/java-8-definitive-guide-to.html](http://www.nurkiewicz.com/2013/05/java-8-definitive-guide-to.html)
Overview of Completable Futures

- Java’s 8 completable future framework provides an asynchronous & reactive concurrent programming model
  - Supports dependent actions that trigger upon completion of async operations
  - Async operations can run concurrently in thread pools
    - Either the common fork-join pool or various types of user-designated thread pools
Overview of Completable Futures

- Java 8 completable futures, streams, & functional programming features can be combined to good effects!!

```
map(this::downloadImageAsync)
thenAccept(this::log)
flatMap(this::applyFiltersAsync)
collect(toFuture())
thenAccept(this::log)
```

Overview of Completable Futures

- Java 8 completable futures often need no explicit synchronization or threading when developing concurrent apps!

Alleviates many accidental & inherent complexities of concurrent programming.
Overview of Completable Futures

- Java 8 completable futures often need no explicit synchronization or threading when developing concurrent apps!

Java class libraries handle locking needed to protect shared mutable state.
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

```java
CompletableFuture<...> future = new CompletableFuture<>();
new Thread (() -> {
    ...
    future.complete(...);
}).start();
```

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

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

```java
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
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

```java
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

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

Completable futures can also be combined with Java 8 streams
End of Overview of Java 8 Completable Futures (Part 1)