The Structure & Functionality of 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

• Understand the key principles underlying reactive programming

• Recognize the Java completable futures framework’s structure & functionality

Task 1

8

//page\ = supplyAsync
  (getStartPage())

Task 2

/imgNum1\ = /page
  .thenApplyAsync
    (countImages(page))
  .thenApply(List::size)

Task 3

/imgNum2\ = /page
  .thenComposeAsync
    (crawlHyperLinks (page))

Task 4

/imgNum1/.thenCombine(/imgNum2, 
  (imgNum1, imgNum2) -> 
  Integer::sum)

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex19
Overview of the Java Completable Future Framework
Overview of the Java Completable Futures Framework

- Java's completable futures framework provides an asynchronous & reactive parallel programming model.

```
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:
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html
Overview of the Java Completable Futures Framework

- Java's completable futures framework provides an asynchronous & reactive parallel programming model
- As a baseline, consider a web crawler implementation that’s synchronous

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

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

- Java's completable futures framework provides an asynchronous & reactive parallel programming model
- As a baseline, consider a web crawler implementation that's synchronous
  - The time needed to perform all these steps is the sum of each step sequentially

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

See [en.wikipedia.org/wiki/Sequential_algorithm](en.wikipedia.org/wiki/Sequential_algorithm)
Overview of the Java Completable Futures Framework

- In contrast, Java's completable futures framework 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/Data-flow_diagram](en.wikipedia.org/wiki/Data-flow_diagram)
In contrast, Java's completable futures framework supports dependent actions that trigger upon completion of async operations.

Overview of the Java Completable Futures Framework

Async operations can be forked, chained, & joined

Task 1
```
Task 1
= supplyAsync
  (getStartPage())
```

Task 2
```
Task 2
= page
  .thenApplyAsync
    (countImages(page))
  .thenApply
    (List::size)
```

Task 3
```
Task 3
= page
  .thenComposeAsync
    (crawlHyperLinks(page))
```

Task 4
```
Task 4
= page
  .thenCombine
    (/imgNum2,
     (imgNum1, imgNum2) ->
      Integer::sum)
```
Overview of the Java Completable Futures Framework

- In contrast, Java's completable futures framework supports dependent actions that trigger upon completion of async operations.

Enables async programming to resemble sync programming via "completion stages"

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

- In contrast, Java's completable futures framework supports dependent actions that trigger upon completion of async operations.
- These async operations can run in a thread pool.

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

- In contrast, Java's completable futures framework supports dependent actions that trigger upon completion of async operations.
- These async operations can run in a thread pool.
  - Either a (common) fork-join pool or various types of pre- or user-defined thread pools.

```java
Task 1
supplyAsync
  (getStartPage())

Task 2
/imgNum1\ = /page\ 8
  .thenApplyAsync
    (countImages(page))
  .thenApply(List::size)

Task 3
/imgNum2\ = /page\ 8
  .thenComposeAsync
    (crawlHyperLinks(page))

Task 4
/imgNum1\ .thenCombine(/imgNum2\, (imgNum1, imgNum2) ->
  Integer::sum)
```
Overview of the Java Completable Futures Framework

- In contrast, Java's completable futures framework supports dependent actions that trigger upon completion of async operations.
- These async operations can run in a thread pool.
- The time needed to perform these tasks depends on how well tasks can be parallelized.

\[
\text{Speedup}(N) = \frac{1}{(1-P)+\frac{P}{N}}
\]

Serial part of job = 1 (100%) - Parallel part

Parallel part is divided up by N workers.

See en.wikipedia.org/wiki/Amdahl's_law
Overview of the Java Completable Futures Framework

- The entire Java completable futures framework resides in one public class with 60+ methods

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html
Overview of the Java Completable Futures Framework

- Java completable futures, sequential streams, & functional programming features can be combined nicely!!

HTML page to download

```html
<HTML><BODY>…</BODY></HTML>
```

A pool of worker threads

- `page.select("a[href"]")`
- `stream()`
- `map(url -> countImages(…, url, …))`
- `collect(toFuture())`
- `thenApply(IntStream::sum)`

See [github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex19](https://github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex19)
Overview of the Java Completable Futures Framework

- Java completable futures often need no explicit synchronization or threading when developing parallel programs!

Alleviates many accidental & inherent complexities of parallel programming.
Overview of the Java Completable Futures Framework

- Java completable futures often need no explicit synchronization or threading when developing parallel programs!

Java class libraries handle locking needed to protect shared mutable state

---

HTML page to download

```html
<HTML><BODY>...</BODY></HTML>
```

```java
page.select("a[href]")
```

```java
stream()
```

```java
map(url -> countImages(..., url, ...))
```

```java
collect(toFuture())
```

```java
thenApply(IntStream::sum)
```
End of the Structure & Functionality of the Java Completable Futures Framework