Advanced Java CompletableFuture Features:
Handling Runtime Exceptions (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

- Understand how completion stage methods chain dependent actions
- Know how to group these methods
- Single stage methods
- Two stage methods (and)
- Two stage methods (or)
- Apply these methods
- Handle runtime exceptions
  - Sync vs. async exceptions
Learning Objectives in this Part of the Lesson

- Understand how completion stage methods chain dependent actions
- Know how to group these methods
- Single stage methods
- Two stage methods (and)
- Two stage methods (or)
- Apply these methods
- Handle runtime exceptions
  - Sync vs. async exceptions
- Overview of methods
Synchronous vs. Asynchronous Exception Handling
Synchronous vs. Asynchronous Exception Handling

• Exception handling is more complex for asynchronous computations than for synchronous computations

See blog.lightstreamer.com/2014/07/exception-handling-in-asynchronous-java.html
The conventional exception handling model propagates exceptions up the runtime call stack synchronously.

1. Calls methods synchronously
2. Throws IOException
3. JVM searches up the call stack to find exception handler
4. Catches IOException

See en.wikipedia.org/wiki/Exception_handling
The conventional exception handling model propagates exceptions up the runtime call stack synchronously.

Therefore, the thread that calls a method is the same thread that can handle any exception that is thrown.

See [en.wikipedia.org/wiki/Exception_handling](en.wikipedia.org/wiki/Exception_handling)
In contrast, completable futures that run asynchronously don’t conform to the conventional call stack model.

Synchronous vs. Asynchronous Exception Handling

Task 1

\[
\text{supplyAsync (getPage())}
\]

Task 2

\[
\text{/imgNum1 = /page\ thenApplyAsync (countImages(page)) thenApply (List::size)}
\]

Task 3

\[
\text{/imgNum2 = /page\ thenComposeAsync (crawlHyperLinks (getPage()))}
\]

Task 4

\[
\text{/imgNum1 .thenCombineAsync (/imgNum2, (imgNum1, imgNum2) -> Integer::sum)}
\]
In contrast, completable futures that run asynchronously don’t conform to the conventional call stack model.

Completion stage methods can thus run in different worker threads than the thread where a method call originates!

See suryanarayanjena.wordpress.com/async-methods-in-completablefuture
• In contrast, completable futures that run asynchronously don’t conform to the conventional call stack model.

1. Initiates `methodX` asynchronously

2. Throws an Exception

3. Thread₁ is now in a completely different context & can’t handle `methodX`’s asynchronous exceptions!

4. Thread₁ can still handle its synchronous exceptions!

See suryanarayanjena.wordpress.com/async-methods-in-completablefuture
Overview of Handling Exceptions in Completion Stages
Overview of Handling Exceptions in Completion Stages

- Completion stage methods handle runtime exceptions that occur asynchronously.

### Overview of Handling Exceptions in Completion Stages

- Completion stage methods handle runtime exceptions that occur asynchronously.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Params</th>
<th>Returns</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>when Complete (Async)</code></td>
<td><code>Bi Consumer</code></td>
<td><code>Completable Future with result of earlier stage or throws exception</code></td>
<td>Handle outcome of a stage, whether a result value or an exception</td>
</tr>
<tr>
<td><code>handle (Async)</code></td>
<td><code>Bi Function</code></td>
<td><code>Completable Future with result of BiFunction</code></td>
<td>Handle outcome of a stage &amp; return new value</td>
</tr>
<tr>
<td><code>exceptionally (Async)</code></td>
<td><code>Function</code></td>
<td><code>Completable Future&lt;T&gt;</code></td>
<td>When exception occurs, replace exception with result value</td>
</tr>
</tbody>
</table>

Help make programs more *resilient* by handling erroneous computations gracefully.
Overview of Handling Exceptions in Completion Stages

- Completion stage methods handle runtime exceptions that occur asynchronously.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Params</th>
<th>Returns</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>when Complete</td>
<td>Bi</td>
<td>Completable Future with result of earlier stage or throws exception</td>
<td>Handle outcome of a stage, whether a result value or an exception</td>
</tr>
<tr>
<td>(Async)</td>
<td>Consumer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>handle</td>
<td>Bi</td>
<td>Completable Future with result of BiFunction</td>
<td>Handle outcome of a stage &amp; return new value</td>
</tr>
<tr>
<td>(Async)</td>
<td>Function</td>
<td></td>
<td></td>
</tr>
<tr>
<td>exceptionally</td>
<td>Function</td>
<td>Completable Future&lt;T&gt;</td>
<td>When exception occurs, replace exception with result value</td>
</tr>
<tr>
<td>(Async)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*These methods run in the invoking thread or the same thread as previous stage*

*The thread that executes these methods depends on various runtime factors*
### Overview of Handling Exceptions in Completion Stages

- Completion stage methods handle runtime exceptions that occur asynchronously.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Params</th>
<th>Returns</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>when Complete (Async)</td>
<td>Bi Consumer</td>
<td>Completable Future with result of earlier stage or throws exception</td>
<td>Handle outcome of a stage, whether a result value or an exception</td>
</tr>
<tr>
<td>handle (Async)</td>
<td>Bi Function</td>
<td>Completable Future with result of BiFunction</td>
<td>Handle outcome of a stage &amp; return new value</td>
</tr>
<tr>
<td>exceptionally (Async)</td>
<td>Function</td>
<td>Completable Future&lt;T&gt;</td>
<td>When exception occurs, replace exception with result value</td>
</tr>
</tbody>
</table>

*Async() variants run in some thread pool*

See [blog.krecan.net/2013/12/25/completablefutures-why-to-use-async-methods](blog.krecan.net/2013/12/25/completablefutures-why-to-use-async-methods)
### Overview of Handling Exceptions in Completion Stages

- Completion stage methods handle runtime exceptions that occur asynchronously.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Params</th>
<th>Returns</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>when Complete</td>
<td>Bi Consumer</td>
<td>Completable Future with result of earlier stage or throws exception</td>
<td>Handle outcome of a stage, whether a result value or an exception</td>
</tr>
<tr>
<td>(Async)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>handle</td>
<td>Bi Function</td>
<td>Completable Future with result of BiFunction</td>
<td>Handle outcome of a stage &amp; return new value</td>
</tr>
<tr>
<td>(Async)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>exceptionally</td>
<td>Function</td>
<td>Completable Future&lt; T&gt;</td>
<td>When exception occurs, replace exception with result value</td>
</tr>
<tr>
<td>(Async)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Overview of Handling Exceptions in Completion Stages**

- Completion stage methods handle runtime exceptions that occur asynchronously.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Params</th>
<th>Returns</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>when Complete (Async)</td>
<td>Bi Consumer</td>
<td>Completable Future with result of earlier stage or throws exception</td>
<td>Handle outcome of a stage, whether a result value or an exception</td>
</tr>
<tr>
<td>handle (Async)</td>
<td>Bi Function</td>
<td>Completable Future with result of BiFunction</td>
<td>Handle outcome of a stage &amp; return new value</td>
</tr>
<tr>
<td>exceptionally (Async)</td>
<td>Function</td>
<td>Completable Future&lt;T&gt;</td>
<td>When exception occurs, replace exception with result value</td>
</tr>
</tbody>
</table>

*Added in Java 12*

See [www.logicbig.com/tutorials/core-java-tutorial/java-12-changes/completion-stage-new-methods.html](http://www.logicbig.com/tutorials/core-java-tutorial/java-12-changes/completion-stage-new-methods.html)
Overview of Handling Exceptions in Completion Stages

- Completion stage methods handle runtime exceptions that occur asynchronously
- Summary of capabilities

<table>
<thead>
<tr>
<th>Item</th>
<th>handle()</th>
<th>whenComplete()</th>
<th>exceptionally()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access to success?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Access to failure?</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Can recover from failure?</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Can transform result from $T$ to $U$?</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Trigger when success?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Trigger when failure?</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Has an async version?</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes (Java 12)</td>
</tr>
</tbody>
</table>
End of Advanced Java
CompletableFuture Features:
Handling Runtime Exceptions
(Part 1)