Advanced Java CompletableFuture Features: Handling Runtime Exceptions (Part 1)

Douglas C. Schmidt

<u>d.schmidt@vanderbilt.edu</u>

www.dre.vanderbilt.edu/~schmidt



Professor of Computer Science

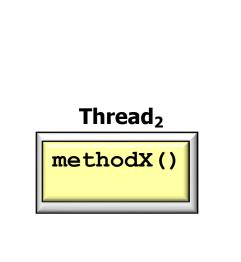
Institute for Software Integrated Systems

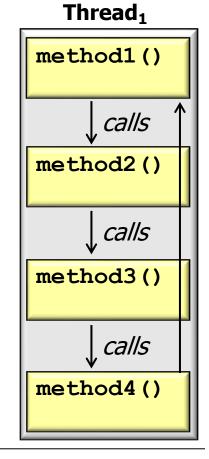
Vanderbilt University Nashville, Tennessee, USA



Learning Objectives in this Part of the Lesson

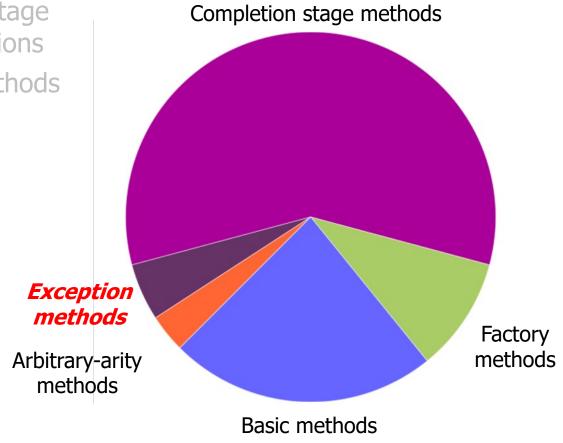
- Understand how completion stage methods chain dependent actions
- Know how to group these methods
- Single stage methods
- Two stage methods (and)
- Two stage methods (or)
- Apply these methods
- Handle runtime exceptions
 - Sync vs. async exceptions



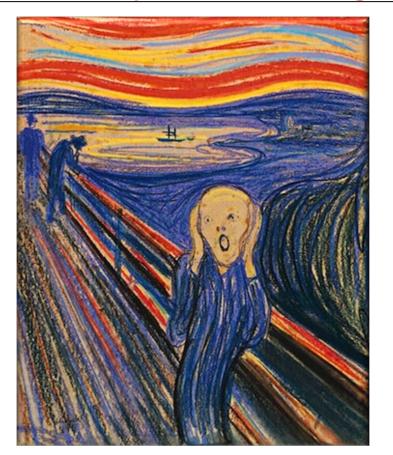


Learning Objectives in this Part of the Lesson

- 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



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



1. Calls methods synchronously

 The conventional exception handling model propagates exceptions up the runtime call stack synchronously

method1() calls method2() throws IOException {...} calls method3() throws IOException {...}

| calls

IOException {...}

method4()

Call Stack for Thread₁

4. Catches

IOException

2. Throws

IOException

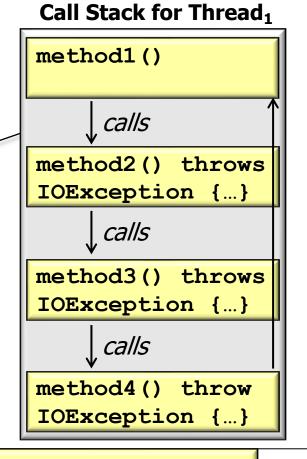
3. JVM searches up the call stack to find exception handler

See en.wikipedia.org/wiki/Exception handling

throw

 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

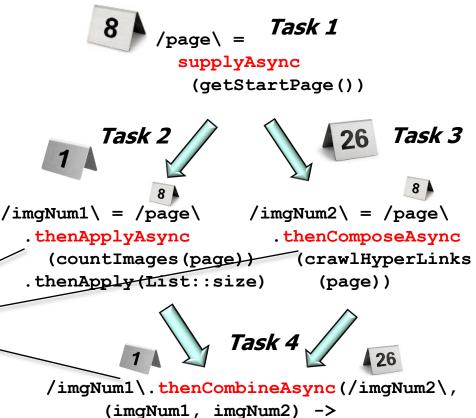


See en.wikipedia.org/wiki/Exception_handling

 In contrast, completable futures that run asynchronously don't conform to the conventional call stack model

```
Task 1
                supplyAsync
                 (getStartPage())
                      /imqNum2 = /page
/imgNum1 = /page
  .thenApplyAsync
                         .thenComposeAsync
    (countImages (page))
                           (crawlHyperLinks
  .thenApply(List::size)
                             (page))
                   Task 4
    /imgNum1\.thenCombineAsync(/imgNum2\,
         (imgNum1, imgNum2) ->
          Integer::sum)
```

 In contrast, completable futures that run asynchronously don't conform to the conventional call stack model



Integer::sum)

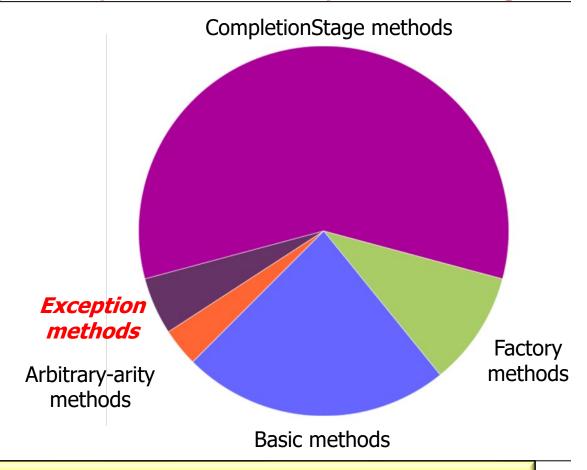
Completion stage methods can thus run in different worker threads than the thread where a method call originates!

See suryanarayanjena.wordpress.com/async-methods-in-completable-future

 In contrast, completable futures that **Thread**₁ 1. Initiates methodX run asynchronously don't conform asynchronously method1() to the conventional call stack model Thread₂ calls methodX() method2() 2. Throws an Exception calls 3. Thread₁ is now in a completely different context method3() & can't handle methodX's asynchronous exceptions! calls 4. Thread₁ can still handle method4() its synchronous exceptions!

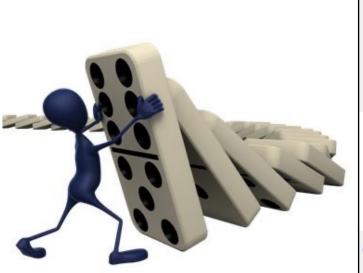
See suryanarayanjena.wordpress.com/async-methods-in-completablefuture

 Completion stage methods handle runtime exceptions that occur asynchronously



See www.jesperdj.com/2015/09/26/the-future-is-completable-in-java-8

Completion stage methods handle runtime exceptions
 Methods Params Returns Behavior
 When Bi Completable Handle outcom



that occur asynchronously

when Complete (Async)	Bi Consumer	Completable Future with result of earlier stage or throws exception	whether a result
handle (Async)	Bi Function	Future with	
exceptionally	Function	Completable	When exception

Future<T>

occurs, replace

exception with

Help make programs more *resilient* by handling erroneous computations gracefully

(Async)

 Completion stage methods Methods Returns **Behavior Params** handle runtime exceptions Completable Handle outcome when that occur asynchronously Consumer Future with Complete of a stage, (Async) whether a result result of earlier stage value or an or throws exception

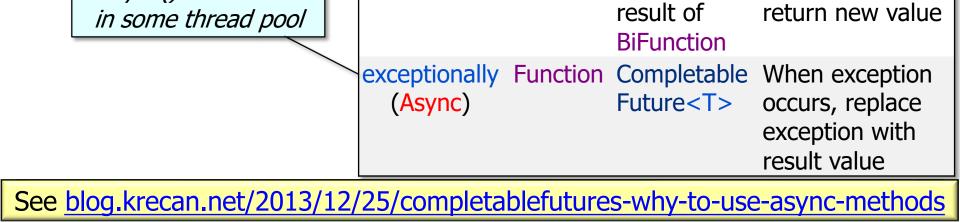
exception handle Bi Completable Handle outcome These methods run in the Function Future with (Async) of a stage & invoking thread or the same return new value result of thread as previous stage **BiFunction** exceptionally Function Completable When exception Future<T> (Async) occurs, replace exception with result value

The thread that executes these methods depends on various runtime factors

 Completion stage methods Methods **Params** Returns **Behavior** handle runtime exceptions Completable Handle outcome when Bi **Consumer Future with** that occur asynchronously Complete of a stage, (Async) whether a result result of earlier stage value or an or throws exception exception handle Bi Completable Handle outcome

(Async)

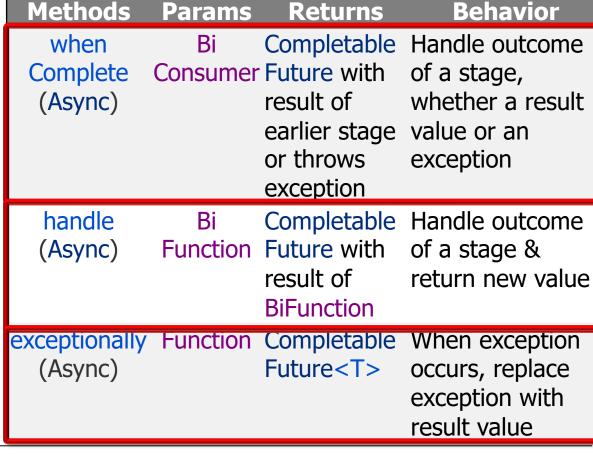
*Async() variants run



Function Future with

of a stage &

 Completion stage methods handle runtime exceptions that occur asynchronously



Methods Completion stage methods Returns **Behavior Params** handle runtime exceptions Completable Handle outcome when Bi that occur asynchronously Consumer Future with Complete of a stage, (Async) result of earlier stage value or an

whether a result or throws exception exception handle Bi Completable Handle outcome Function Future with (Async) of a stage & return new value result of **BiFunction** Added in Java 12 exceptionally Function Completable When exception Future<T> (Async) occurs, replace exception with

result value See www.logicbig.com/tutorials/core-java-tutorial/java-12-changes/completion-stage-new-methods.html

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

Item	handle()	whenComplete()	exceptionally()
Access to success?	Yes	Yes	No
Access to failure?	Yes	Yes	Yes
Can recover from failure?	Yes	No	Yes
Can transform result from T to U?	Yes	No	No
Trigger when success?	Yes	Yes	No
Trigger when failure?	Yes	Yes	Yes
Has an async version?	Yes	Yes	Yes (Java 12)

End of Advanced Java CompletableFuture Features: Handling Runtime Exceptions (Part 1)