

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

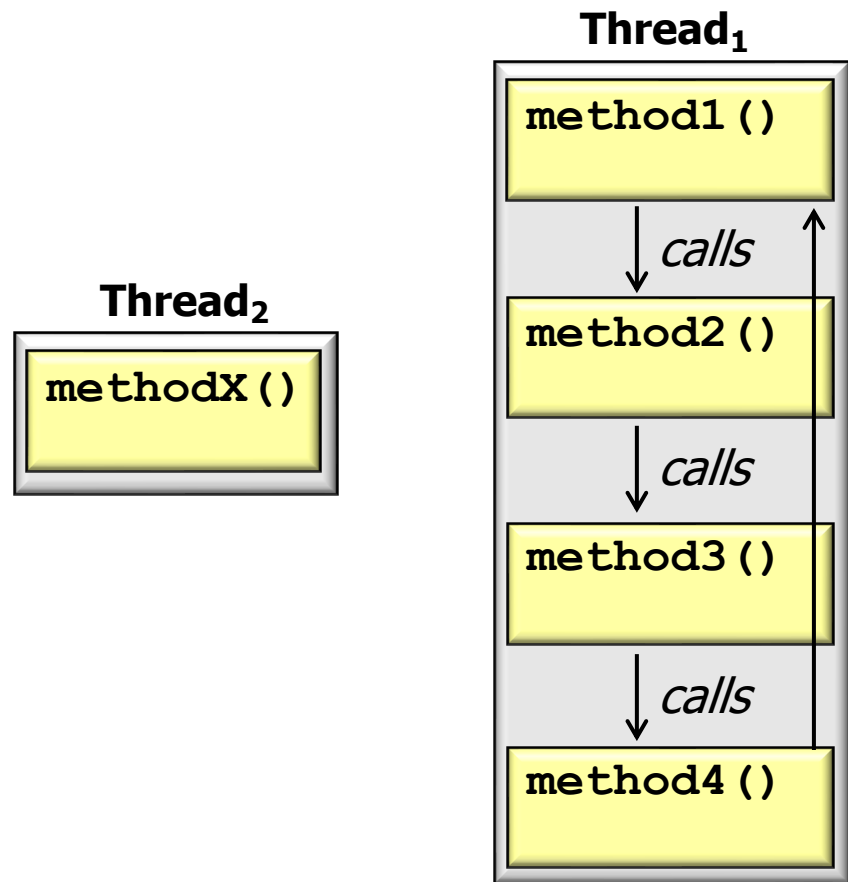
**Institute for Software
Integrated Systems**

**Vanderbilt University
Nashville, Tennessee, USA**



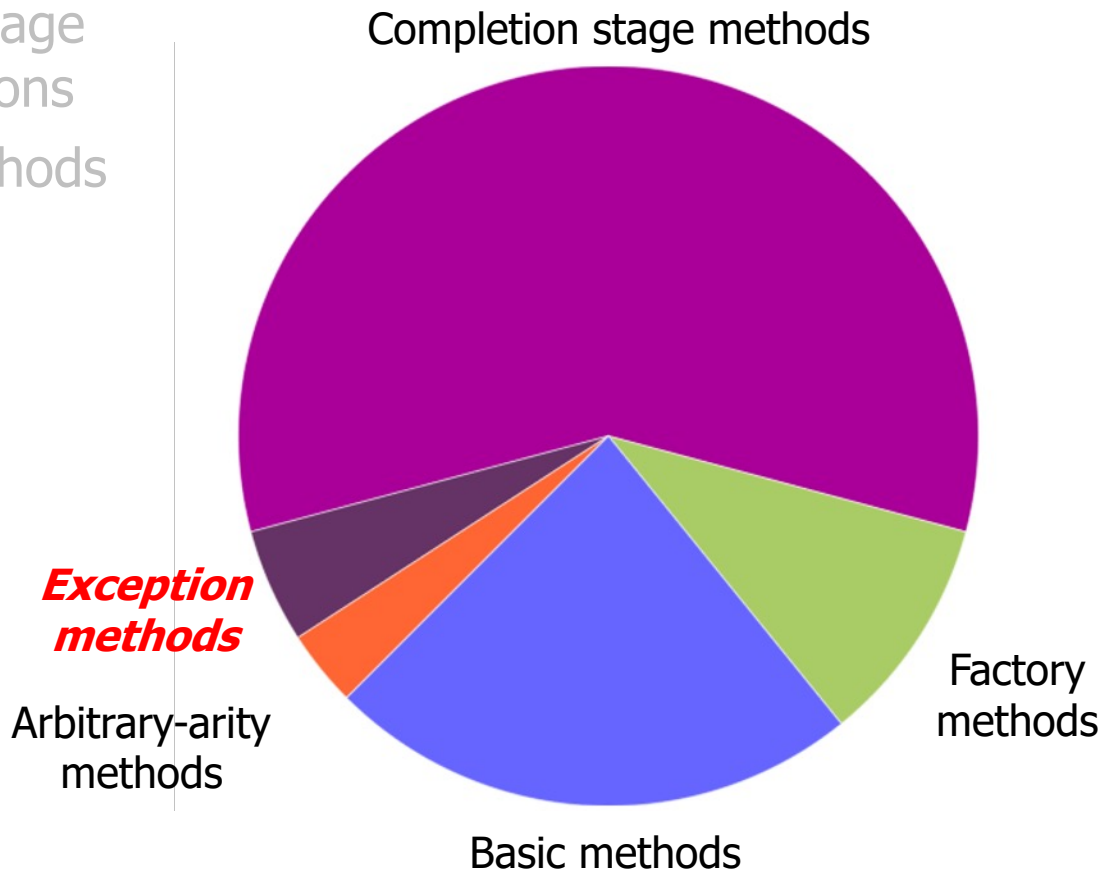
Learning Objectives in this Part of the Lesson

- 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

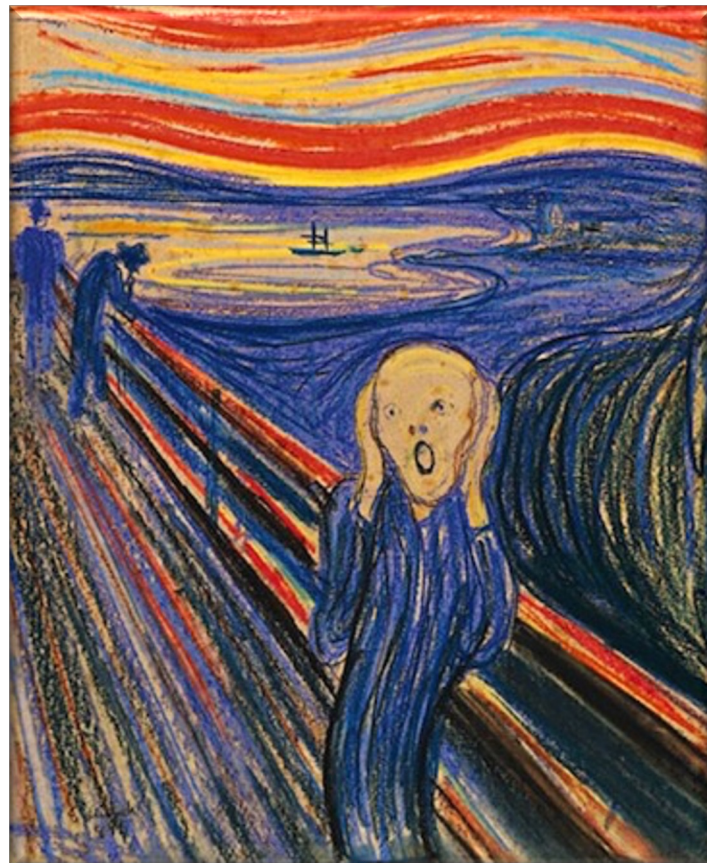
- Understand how completion stage methods chain dependent actions
- Know how to group these methods
- Single stage methods
- Two stage methods (and)
- Two stage methods (or)
- Apply these methods
- Handle runtime exceptions
 - Sync vs. async exceptions
- Overview of methods



Synchronous vs. Asynchronous Exception Handling

Synchronous vs. Asynchronous Exception Handling

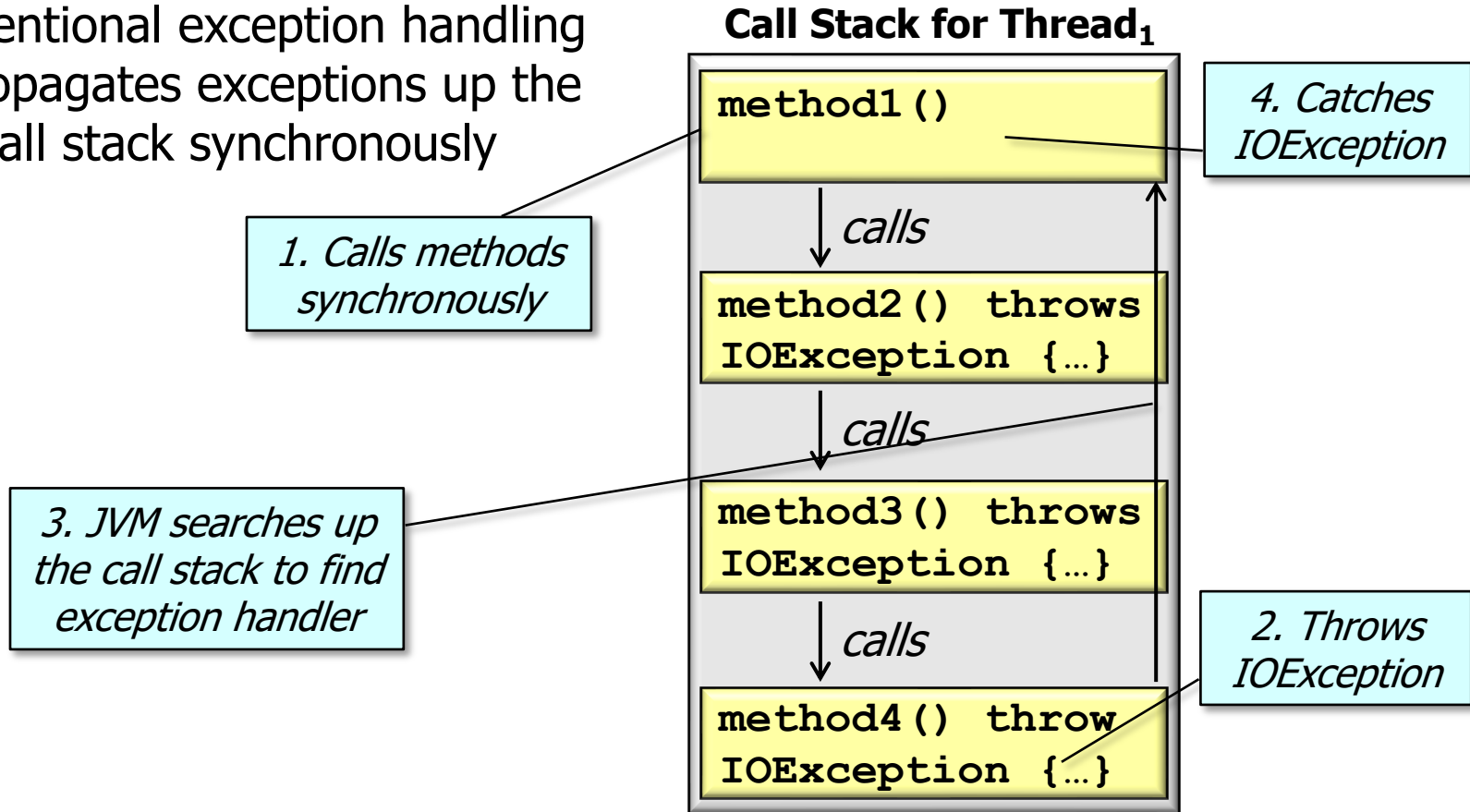
- Exception handling is more complex for asynchronous computations than for synchronous computations



See blog.lightstreamer.com/2014/07/exception-handling-in-asynchronous-java.html

Synchronous vs. Asynchronous Exception Handling

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

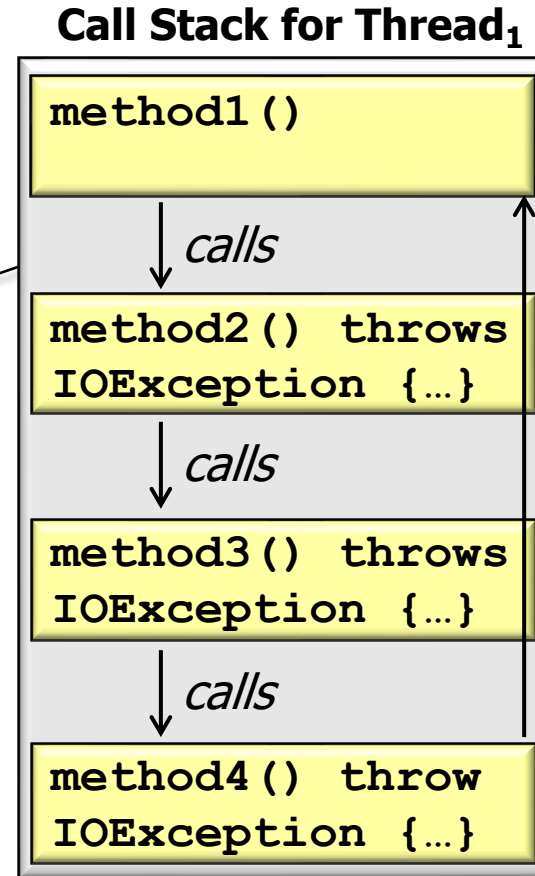


See en.wikipedia.org/wiki/Exception_handling

Synchronous vs. Asynchronous Exception Handling

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

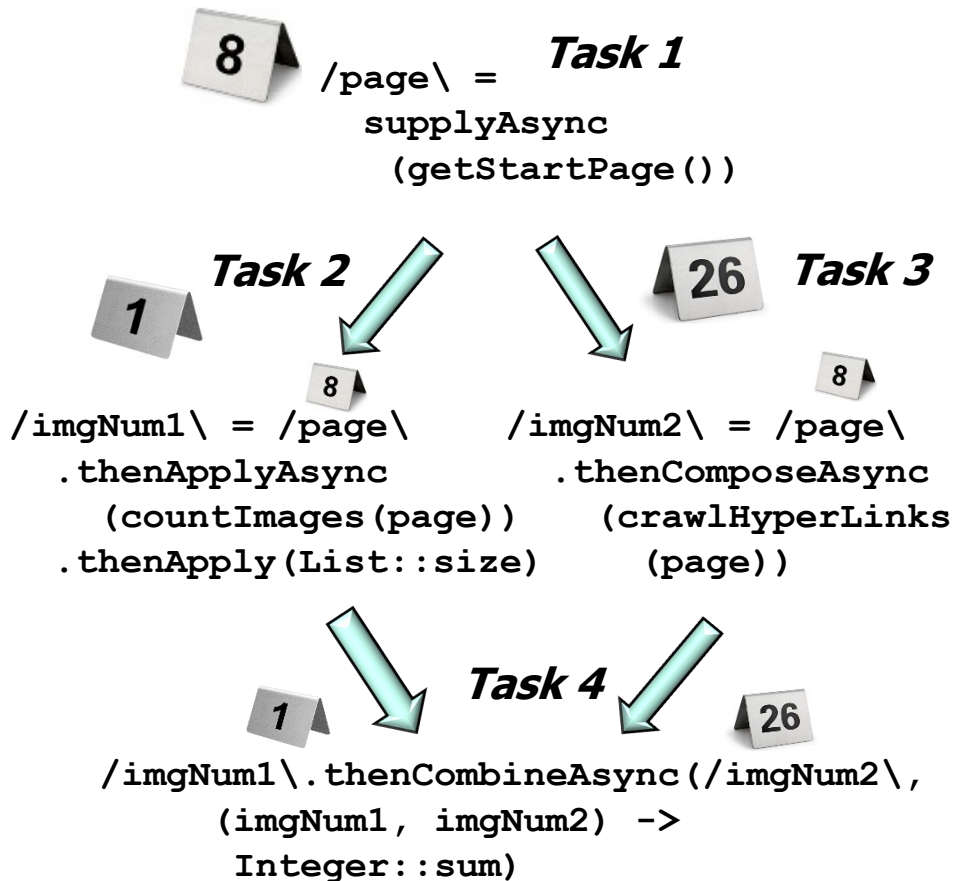
Therefore, the thread that calls a method is the same thread that can handle any exception that is thrown



See en.wikipedia.org/wiki/Exception_handling

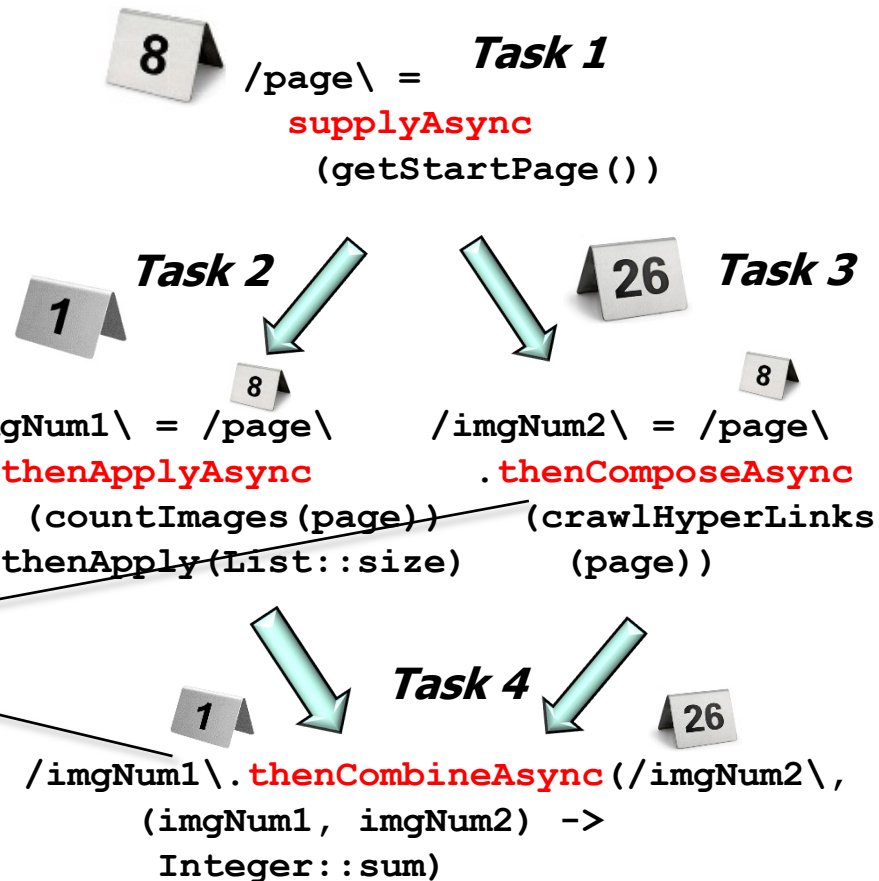
Synchronous vs. Asynchronous Exception Handling

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



Synchronous vs. Asynchronous Exception Handling

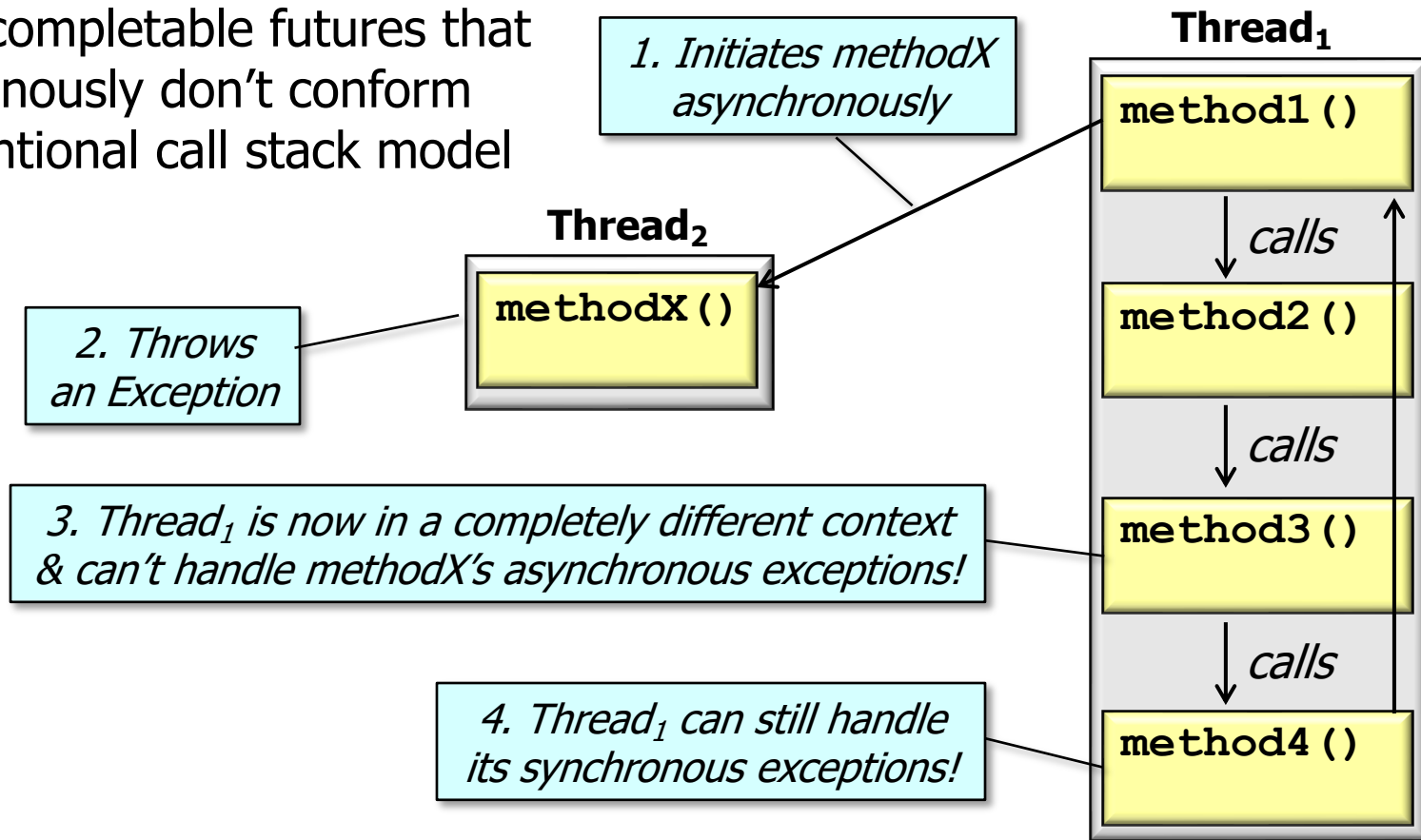
- In contrast, completable futures that run asynchronously don't conform to the conventional call stack model



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

Synchronous vs. Asynchronous Exception Handling

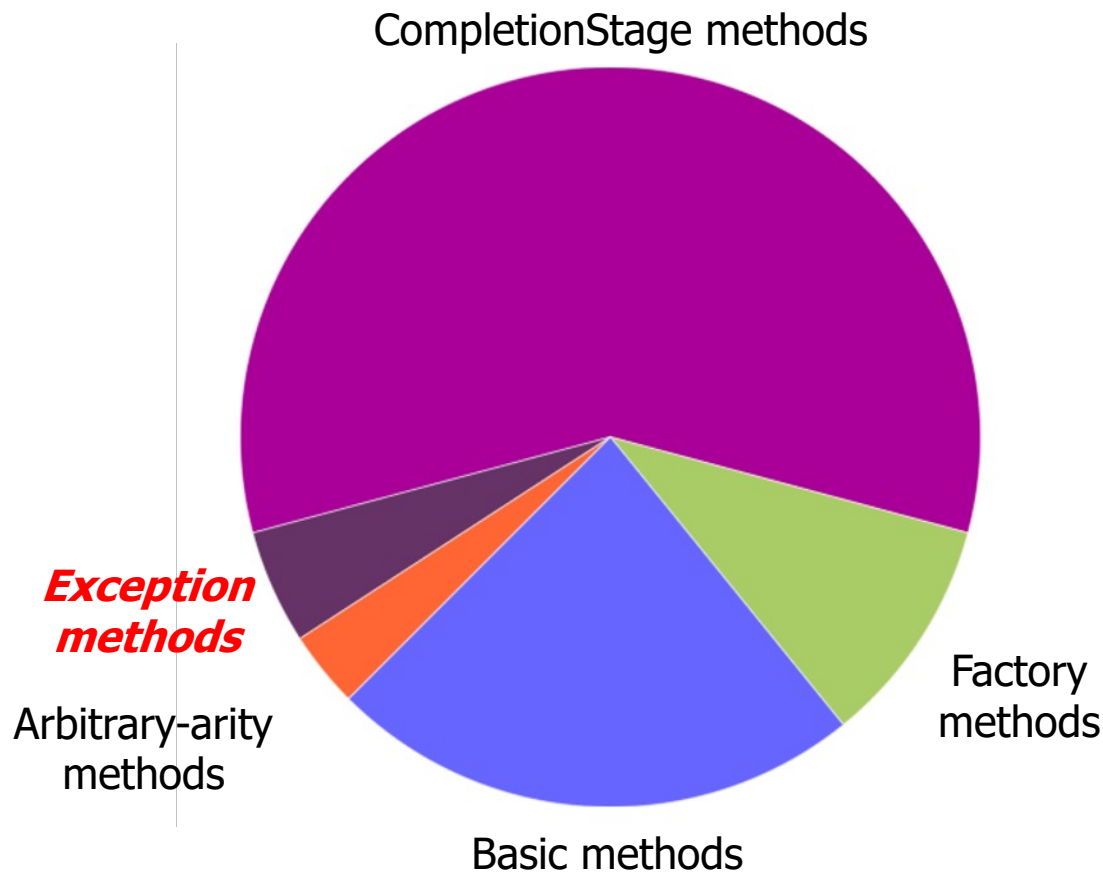
- In contrast, completable futures that run asynchronously don't conform to the conventional call stack model



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



Methods	Params	Returns	Behavior
<code>whenComplete(Async)</code>	<code>BiConsumer</code>	<code>CompletableFuture</code> with result of earlier stage or throws exception	Handle outcome of a stage, whether a result value or an exception
<code>handle(Async)</code>	<code>BiFunction</code>	<code>CompletableFuture</code> with result of <code>BiFunction</code>	Handle outcome of a stage & return new value
<code>exceptionally(Async)</code>	<code>Function</code>	<code>CompletableFuture<T></code>	When exception occurs, replace exception with result value

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

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

Methods	Params	Returns	Behavior
<code>whenComplete(Async)</code>	<code>BiConsumer</code>	<code>CompletableFuture</code> with result of earlier stage or throws exception	Handle outcome of a stage, whether a result value or an exception
<code>handle(Async)</code>	<code>BiFunction</code>	<code>CompletableFuture</code> with result of <code>BiFunction</code>	Handle outcome of a stage & return new value
<code>exceptionally(Async)</code>	<code>Function</code>	<code>CompletableFuture<T></code>	When exception occurs, replace exception with result value

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

Methods	Params	Returns	Behavior
<code>whenComplete(Async)</code>	<code>BiConsumer</code>	<code>CompletableFuture</code> with result of earlier stage or throws exception	Handle outcome of a stage, whether a result value or an exception
<code>handle(Async)</code>	<code>BiFunction</code>	<code>CompletableFuture</code> with result of <code>BiFunction</code>	Handle outcome of a stage & return new value
<code>exceptionally(Async)</code>	<code>Function</code>	<code>CompletableFuture<T></code>	When exception occurs, replace exception with result value

**Async() variants run in some thread pool*

See blog.krean.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

Methods	Params	Returns	Behavior
<code>whenComplete(Async)</code>	<code>BiConsumer</code>	<code>CompletableFuture</code> with result of earlier stage or throws exception	Handle outcome of a stage, whether a result value or an exception
<code>handle(Async)</code>	<code>BiFunction</code>	<code>CompletableFuture</code> with result of <code>BiFunction</code>	Handle outcome of a stage & return new value
<code>exceptionally(Async)</code>	<code>Function</code>	<code>CompletableFuture<T></code>	When exception occurs, replace exception with result value

Overview of Handling Exceptions in Completion Stages

- Completion stage methods handle runtime exceptions that occur asynchronously

Methods	Params	Returns	Behavior
<code>whenComplete(Async)</code>	<code>BiConsumer</code>	<code>CompletableFuture</code> with result of earlier stage or throws exception	Handle outcome of a stage, whether a result value or an exception
<code>handle(Async)</code>	<code>BiFunction</code>	<code>CompletableFuture</code> with result of <code>BiFunction</code>	Handle outcome of a stage & return new value
<code>exceptionally(Async)</code>	<code>Function</code>	<code>CompletableFuture<T></code>	When exception occurs, replace exception with result value

Added in Java 12

Overview of Handling Exceptions in Completion Stages

- 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)

See mincong.io/2020/05/30/exception-handling-in-completable-future

End of Advanced Java

CompletableFuture Features: Handling Runtime Exceptions (Part 1)