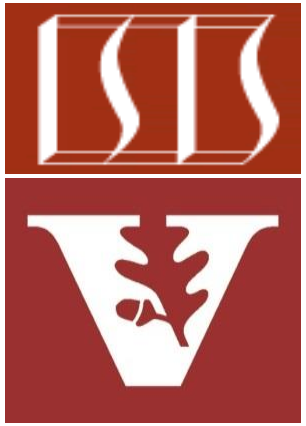
Advanced Java Completable Future Features: Arbitrary-Arity Methods

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt



Professor of Computer Science

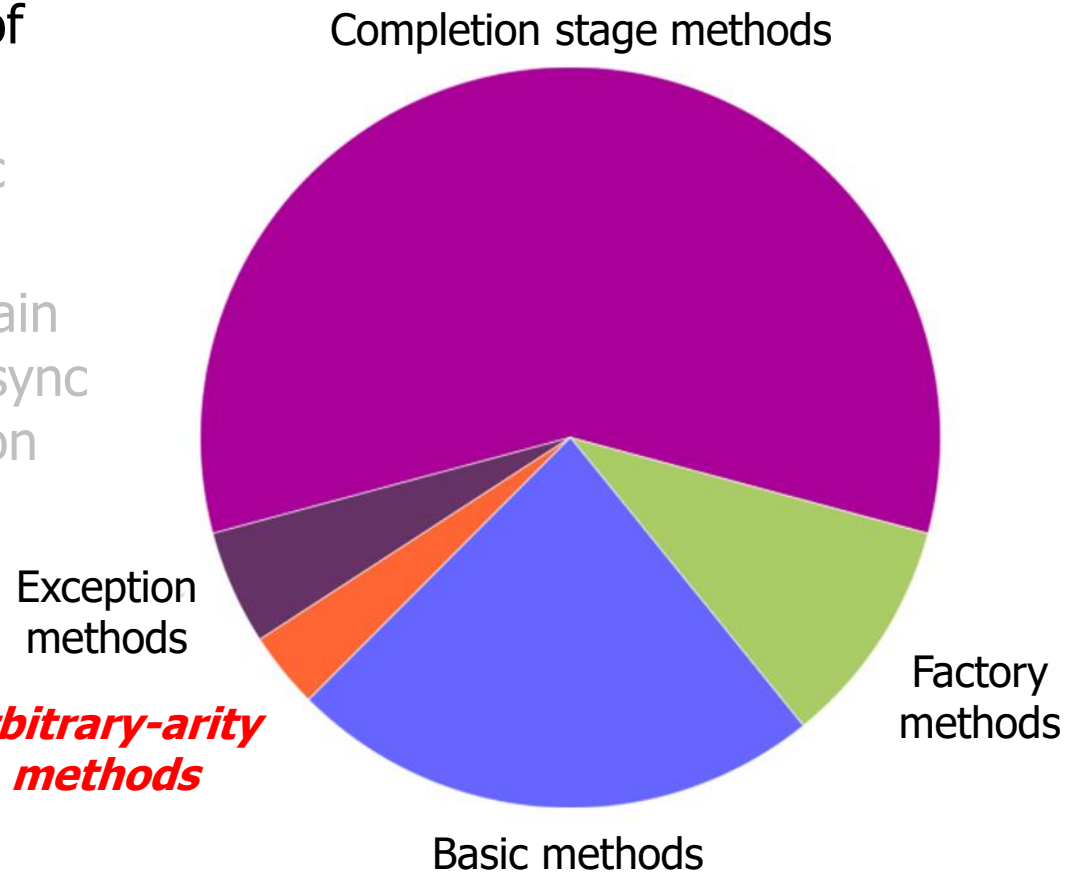
Institute for Software Integrated Systems

Vanderbilt University

Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand advanced features of completable futures, e.g.
  - Factory methods initiate async computations
  - Completion stage methods chain together actions to perform async result processing & composition
  - Aribitrary-arity methods that process futures in bulk



See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html

Arbitrary-Arity Methods Process Futures in Bulk

Arbitrary-Arity Methods Process Futures in Bulk

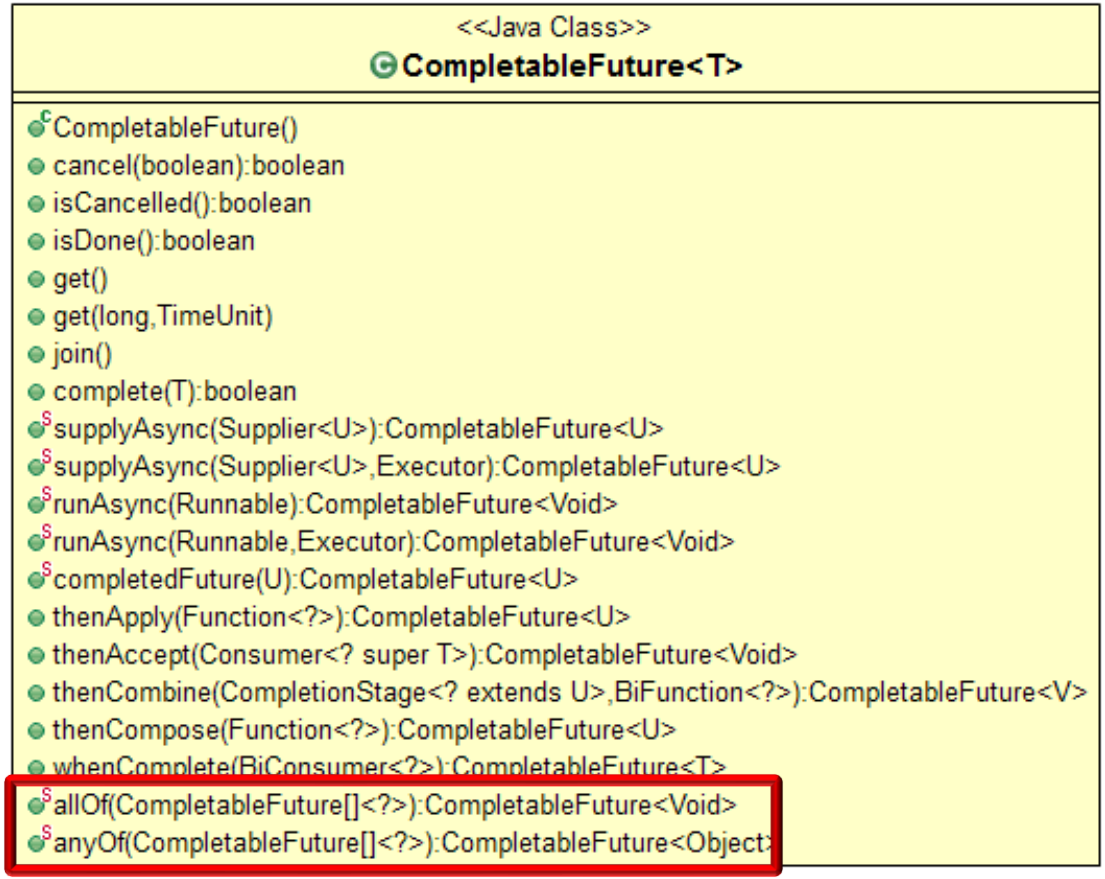
- Arbitrary-arity methods return futures that are triggered after completion of any/all futures

Methods	Params	Returns	Behavior
allOf	Varargs	CompletableFuture<Void>	Return a future that completes when all futures in params complete
anyOf	Varargs	CompletableFuture<Void>	Return a future that completes when any future in params complete

See en.wikipedia.org/wiki/Arity

Arbitrary-Arity Methods Process Futures in Bulk

- Arbitrary-arity methods return futures that are triggered after completion of any/all futures
  - The returned future can be used to wait for any or all of *N* completable futures in an array to complete



Arbitrary-Arity Methods Process Futures in Bulk

- Arbitrary-arity methods return futures that are triggered after completion of any/all futures
  - The returned future can be used to wait for any or all of *N* completable futures in an array to complete
    - This "wait" usually doesn't block, but instead uses completion stage methods

<<Java Class>>

CompletableFuture<T>

- CompletableFuture()
- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long,TimeUnit)
- join()
- complete(T):boolean
- supplyAsync(Supplier<U>):CompletableFuture<U>
- supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
- runAsync(Runnable):CompletableFuture<Void>
- runAsync(Runnable,Executor):CompletableFuture<Void>
- completedFuture(U):CompletableFuture<U>
- thenApply(Function<?>):CompletableFuture<U>
- thenAccept(Consumer<? super T>):CompletableFuture<Void>
- thenCombine(CompletionStage<? extends U>,BiFunction<?>):CompletableFuture<V>
- thenCompose(Function<?>):CompletableFuture<U>
- whenComplete(BiConsumer<?>):CompletableFuture<T>
- allOf(CompletableFuture[]<?>):CompletableFuture<Void>
- anyOf(CompletableFuture[]<?>):CompletableFuture<Object>

Help make programs more *responsive* by not blocking caller code

Arbitrary-Arity Methods Process Futures in Bulk

- Arbitrary-arity methods return futures that are triggered after completion of any/all futures
  - The returned future can be used to wait for any or all of *N* completable futures in an array to complete
  - We focus on allOf(), which is like thenCombine() on steroids!

<<Java Class>>

CompletableFuture<T>

- CompletableFuture()
- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long,TimeUnit)
- join()
- complete(T):boolean
- supplyAsync(Supplier<U>):CompletableFuture<U>
- supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
- runAsync(Runnable):CompletableFuture<Void>
- runAsync(Runnable,Executor):CompletableFuture<Void>
- completedFuture(U):CompletableFuture<U>
- thenApply(Function<?>):CompletableFuture<U>
- thenAccept(Consumer<? super T>):CompletableFuture<Void>
- thenCombine(CompletionStage<? extends U>,BiFunction<?>):CompletableFuture<V>
- thenCompose(Function<?>):CompletableFuture<U>
- whenComplete(BiConsumer<?>):CompletableFuture<T>
- **allOf(CompletableFuture[]<?>):CompletableFuture<Void>**
- anyOf(CompletableFuture[]<?>):CompletableFuture<Object>

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#allOf

Arbitrary-Arity Methods Process Futures in Bulk

- These arbitrary-arity methods are hard to program directly

<<Java Class>>

CompletableFuture<T>

- CompletableFuture()
- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long,TimeUnit)
- join()
- complete(T):boolean
- supplyAsync(Supplier<U>):CompletableFuture<U>
- supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
- runAsync(Runnable):CompletableFuture<Void>
- runAsync(Runnable,Executor):CompletableFuture<Void>
- completedFuture(U):CompletableFuture<U>
- thenApply(Function<?>):CompletableFuture<U>
- thenAccept(Consumer<? super T>):CompletableFuture<Void>
- thenCombine(CompletionStage<? extends U>,BiFunction<?>):CompletableFuture<V>
- thenCompose(Function<?>):CompletableFuture<U>
- whenComplete(BiConsumer<?>):CompletableFuture<T>
- allOf(CompletableFuture[]<?>):CompletableFuture<Void>
- anyOf(CompletableFuture[]<?>):CompletableFuture<Object>

See en.wikipedia.org/wiki/Gordian_Knot

Arbitrary-Arity Methods Process Futures in Bulk

- These arbitrary-arity methods are hard to program directly
  - Instead, program them via wrappers

<<Java Class>>

CompletableFuture<T>

- CompletableFuture()
- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long,TimeUnit)
- join()
- complete(T):boolean
- supplyAsync(Supplier<U>):CompletableFuture<U>
- supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
- runAsync(Runnable):CompletableFuture<Void>
- runAsync(Runnable,Executor):CompletableFuture<Void>
- completedFuture(U):CompletableFuture<U>
- thenApply(Function<?>):CompletableFuture<U>
- thenAccept(Consumer<? super T>):CompletableFuture<Void>
- thenCombine(CompletionStage<? extends U>,BiFunction<?>):CompletableFuture<V>
- thenCompose(Function<?>):CompletableFuture<U>
- whenComplete(BiConsumer<?>):CompletableFuture<T>
- allOf(CompletableFuture[]<?>):CompletableFuture<Void>
- anyOf(CompletableFuture[]<?>):CompletableFuture<Object>

See next lesson on *"Advanced Java CompletableFuture Features: Arbitrary-Arity Methods"*

End of Advanced Java Completable Future Features: Arbitrary-Arity Methods