

Understanding Method Groupings in the Java Completable Futures API (Part 2)

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

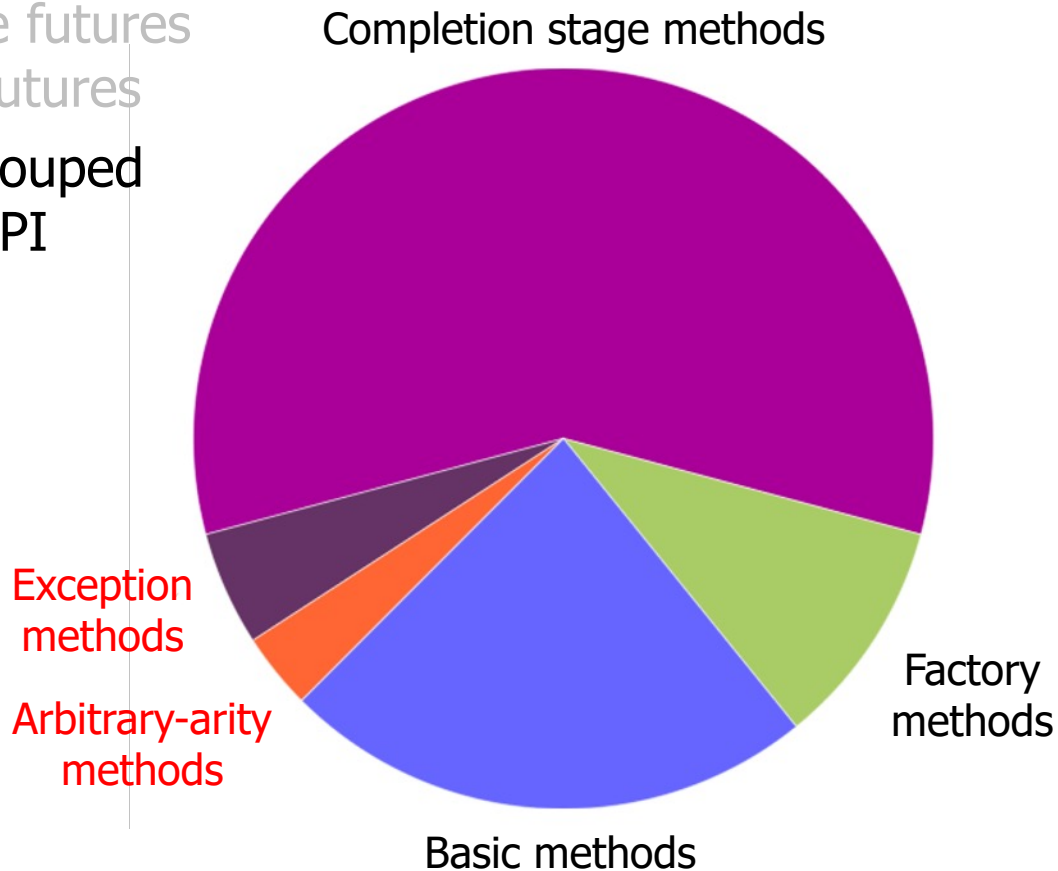
**Institute for Software
Integrated Systems**

**Vanderbilt University
Nashville, Tennessee, USA**



Learning Objectives in this Part of the Lesson

- Recognize how Java completable futures overcome limitations with Java futures
- Understand how methods are grouped in the Java completable future API



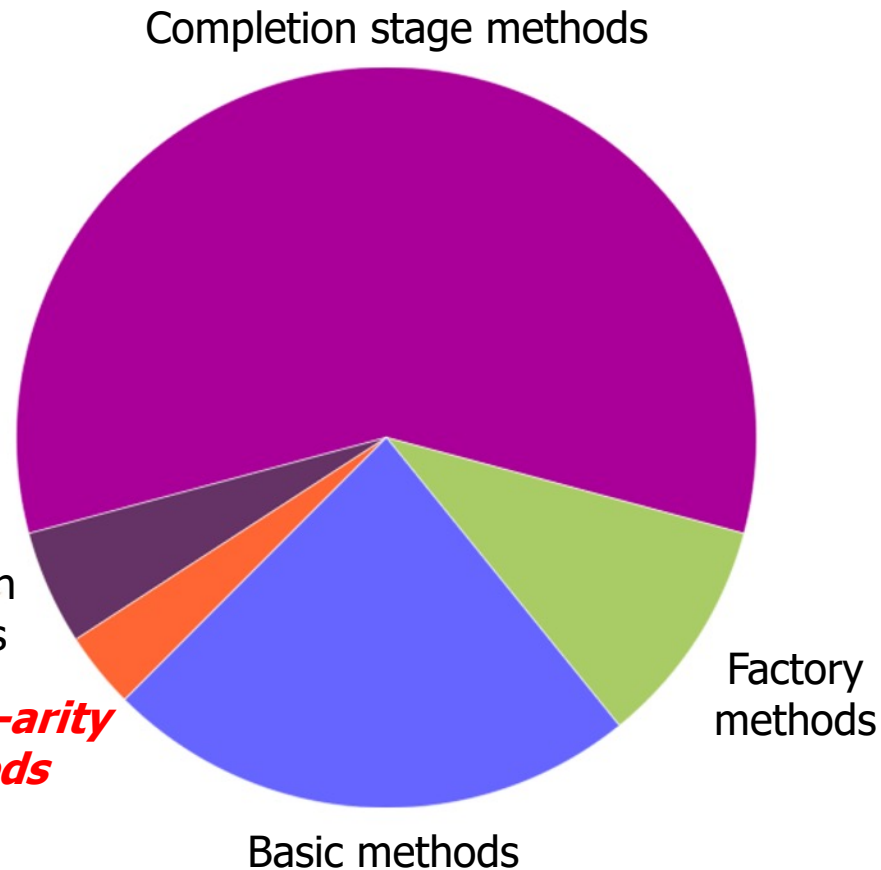
Grouping the Java Completable Future API

Grouping the Java Completable Future API

- Other completable future features are more advanced
 - Factory methods
 - Completion stage methods
 - "Arbitrary-arity" methods

Arity is the # of params accepted by a method

Arbitrary-arity methods



See en.wikipedia.org/wiki/Arity

Grouping the Java Completable Future API

- Other completable future features are more advanced
 - Factory methods
 - Completion stage methods
 - “Arbitrary-arity” methods
 - Process futures in bulk by combine multiple futures into a single future

```
<<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>
```

Grouping the Java Completable Future API

- Other completable future features are more advanced
 - Factory methods
 - Completion stage methods
 - “Arbitrary-arity” methods
 - Process futures in bulk by combine multiple futures into a single future
 - Single future triggered when *all* complete

```
<<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>
```

Grouping the Java Completable Future API

- Other completable future features are more advanced
 - Factory methods
 - Completion stage methods
 - “Arbitrary-arity” methods
 - Process futures in bulk by combine multiple futures into a single future
 - Single future triggered when *all* complete
 - Single future triggered when *first* completes

```
<<Java Class>>
CompletableFuture<T>

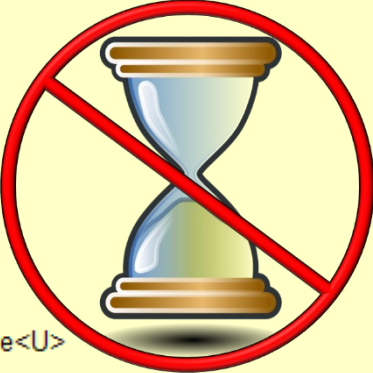
CompletableFuture()
cancel(boolean):boolean
isCancelled():boolean
isDone():boolean
get()
get(long,TimeUnit)
join()
complete(T):boolean
supplyAsync(Supplier<U>):CompletableFuture<U>
supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
runAsync(Runnable):CompletableFuture<Void>
runAsync(Runnable,Executor):CompletableFuture<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>
```

Grouping the Java Completable Future API

- Other completable future features are more advanced
 - Factory methods
 - Completion stage methods
 - “Arbitrary-arity” methods
 - Process futures in bulk by combine multiple futures into a single future
 - Single future triggered when *all* complete
 - Single future triggered when *first* completes

<<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>
```

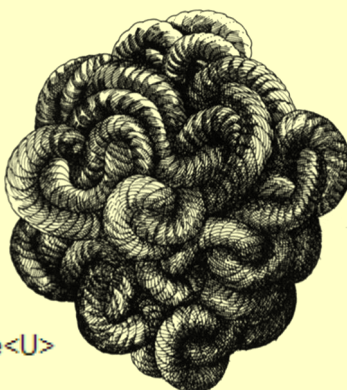


Help make programs more *responsive* by not blocking caller code

Grouping the Java Completable Future API

- Other completable future features are more advanced
 - Factory methods
 - Completion stage methods
 - “Arbitrary-arity” methods
 - Process futures in bulk by combine multiple futures into a single future
 - Single future triggered when *all* complete
 - Single future triggered when *first* completes

<<Java Class>>
CompletableFuture<T>

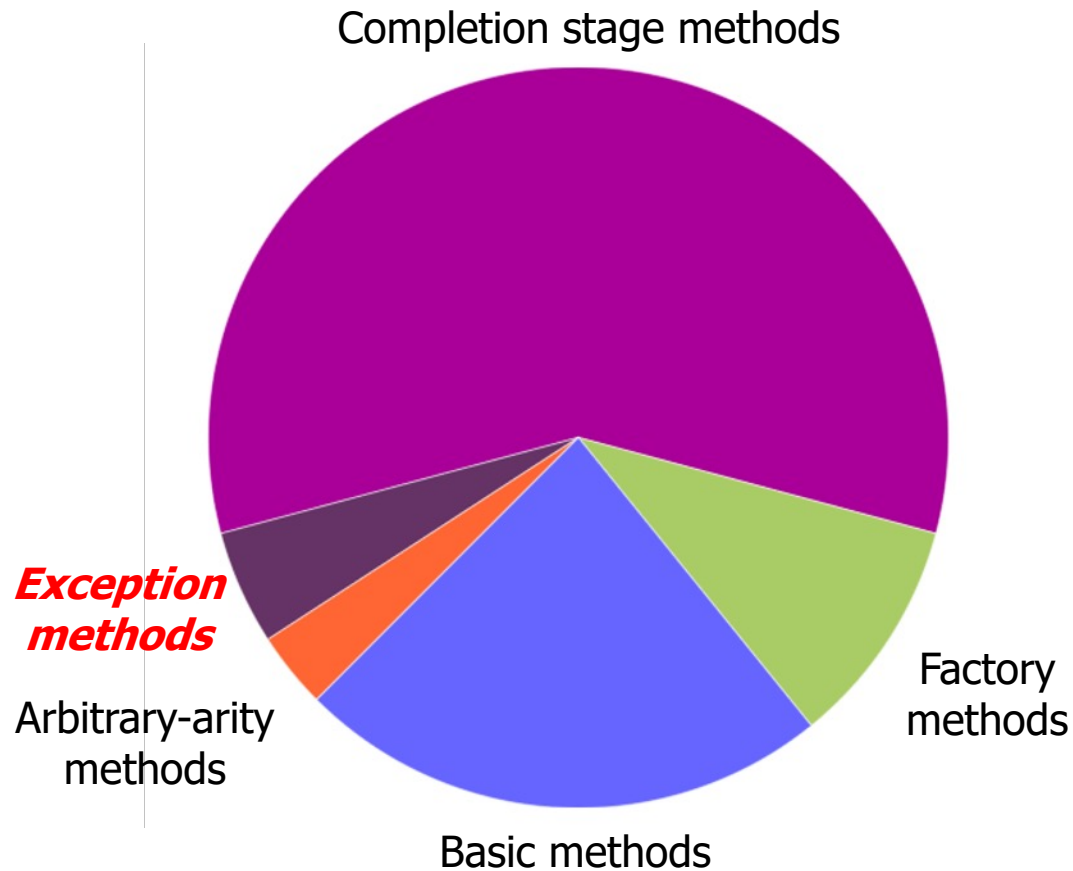


```
CompletableFuture()
cancel(boolean):boolean
isCancelled():boolean
isDone():boolean
get()
get(long,TimeUnit)
join()
complete(T):boolean
supplyAsync(Supplier<U>):CompletableFuture<U>
supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
runAsync(Runnable):CompletableFuture<Void>
runAsync(Runnable,Executor):CompletableFuture<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>
```

They are complicated to program directly & best served via defining a wrapper!

Grouping the Java Completable Future API

- Other completable future features are more advanced
 - Factory methods
 - Completion stage methods
 - "Arbitrary-arity" methods
 - Exception methods



Grouping the Java Completable Future API

- Other completable future features are more advanced
 - Factory methods
 - Completion stage methods
 - “Arbitrary-arity” methods
- Exception methods
 - Handle exceptional conditions at runtime

```
<<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>
```

Grouping the Java Completable Future API

- Other completable future features are more advanced
 - Factory methods
 - Completion stage methods
 - “Arbitrary-arity” methods
- Exception methods
 - Handle exceptional conditions at runtime
 - These methods are essential since async exceptions are different than sync exceptions

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

Grouping the Java Completable Future API

- Other completable future features are more advanced
 - Factory methods
 - Completion stage methods
 - “Arbitrary-arity” methods
- Exception methods
 - Handle exceptional conditions at runtime



<<Java Class>>	
CompletableFuture<T>	
•	CompletableFuture()
•	cancel(boolean):boolean
•	isCancelled():boolean
•	isDone():boolean
•	get()
•	get(long,TimeUnit)
•	join()
•	complete(T):boolean
• ^S	supplyAsync(Supplier<U>):CompletableFuture<U>
• ^S	supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
• ^S	runAsync(Runnable):CompletableFuture<Void>
• ^S	runAsync(Runnable,Executor):CompletableFuture<Void>
• ^S	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>
• ^S	anyOf(CompletableFuture[]<?>):CompletableFuture<Object>

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

Grouping the Java Completable Future API

- All methods are implemented internally via a message-passing framework



<<Java Class>>	
CompletableFuture<T>	
●	CompletableFuture()
●	cancel(boolean):boolean
●	isCancelled():boolean
●	isDone():boolean
●	get()
●	get(long,TimeUnit)
●	join()
●	complete(T):boolean
● ^S	supplyAsync(Supplier<U>):CompletableFuture<U>
● ^S	supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
● ^S	runAsync(Runnable):CompletableFuture<Void>
● ^S	runAsync(Runnable,Executor):CompletableFuture<Void>
● ^S	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>
● ^S	allOf(CompletableFuture[]<?>):CompletableFuture<Void>
● ^S	anyOf(CompletableFuture[]<?>):CompletableFuture<Object>

Ensures loose coupling, isolation, & location transparency between components

Grouping the Java Completable Future API

- All methods are implemented internally via a message-passing framework
- Various Java thread pools are used to process the messages

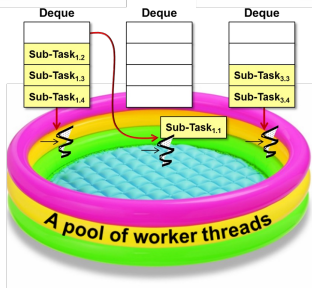
*Cached (Variable-sized)
Thread Pool*



*Fixed-sized
Thread Pool*



*Work-stealing
Thread Pool*



<<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 www.baeldung.com/thread-pool-java-and-guava

End of Understanding Method Groupings in the Java Completable Futures API (Part 2)