

Understanding Method Groupings in the Java Completable Futures API (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**

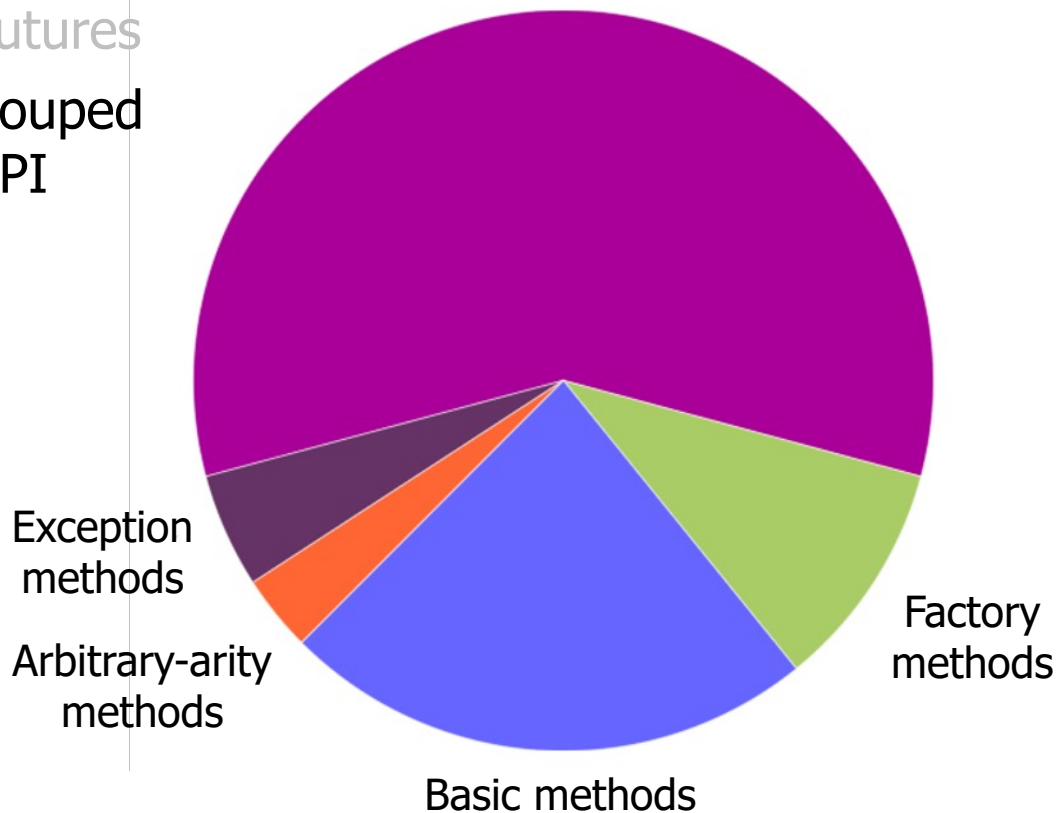


Learning Objectives in this Part of the Lesson

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

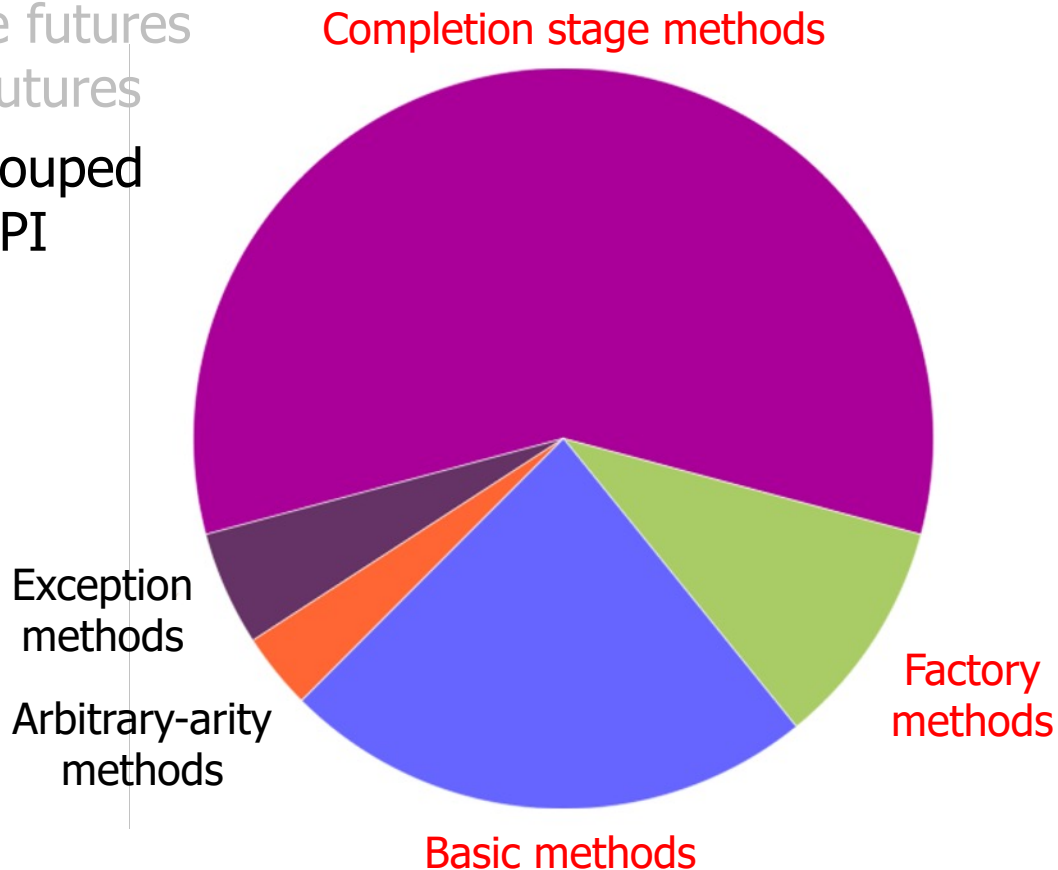


Completion stage methods



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



Birds-Eye View of the Java Completable Future API

Birds-Eye View of the Java Completable Future API

- The entire completable futures framework resides in 1 public class with 60+ methods!!!

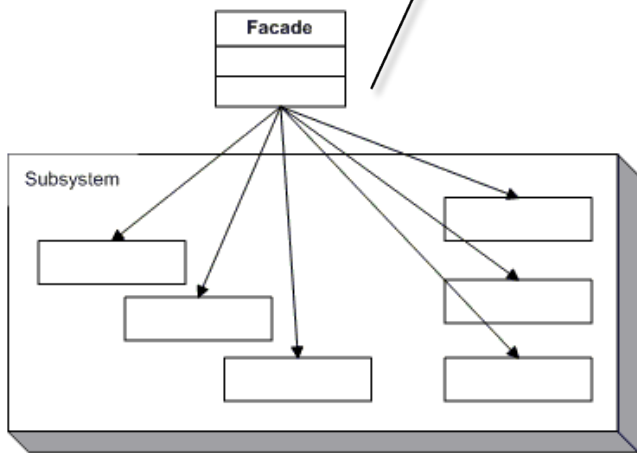
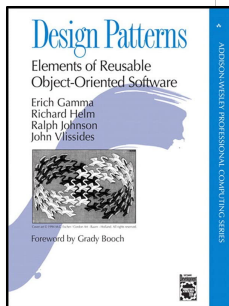
<<Java Class>>	
📄 CompletableFuture<T>	
📄	CompletableFuture()
📄	cancel(boolean):boolean
📄	isCancelled():boolean
📄	isDone():boolean
📄	get()
📄	get(long,TimeUnit)
📄	join()
📄	complete(T):boolean
📄	📄 supplyAsync(Supplier<U>):CompletableFuture<U>
📄	📄 supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
📄	📄 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>

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

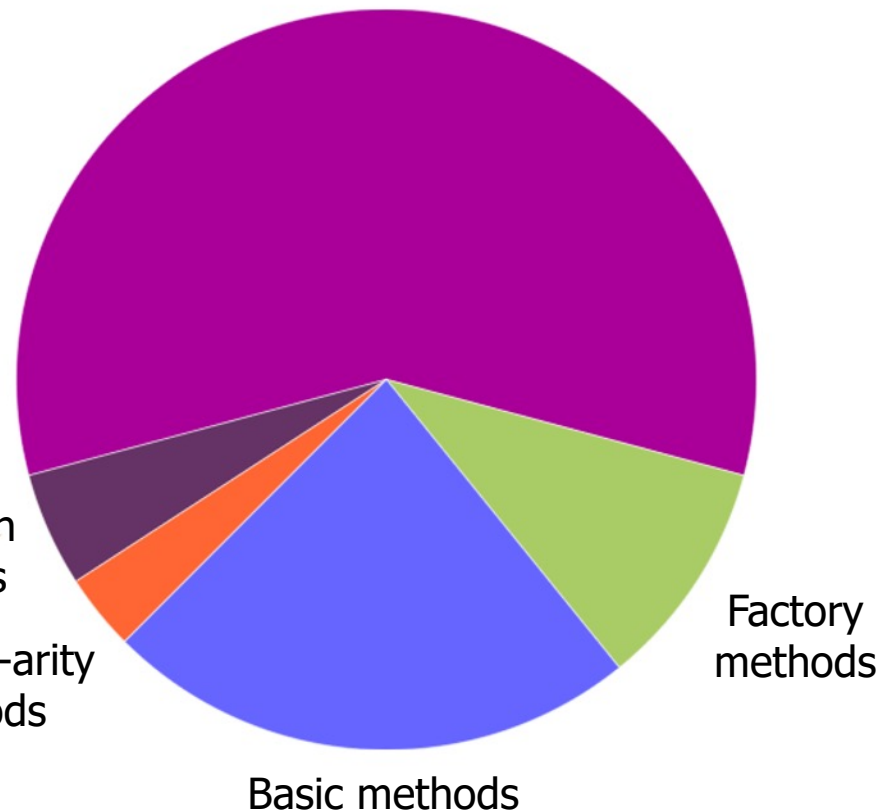
Birds-Eye View of the Java Completable Future API

- This framework implements the Façade pattern

Provide a unified interface to a set of interfaces in a sub-system that makes the subsystem easier to use



Completion stage methods



Exception methods

Arbitrary-arity methods

Basic methods

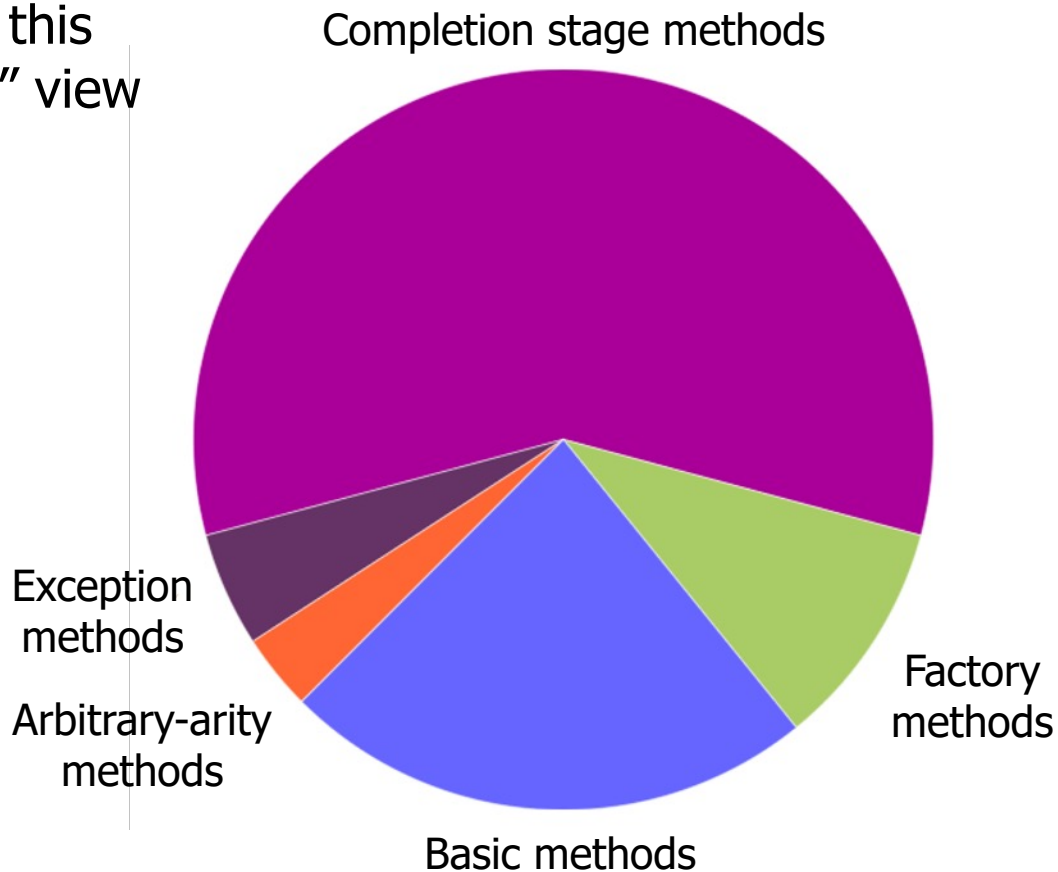
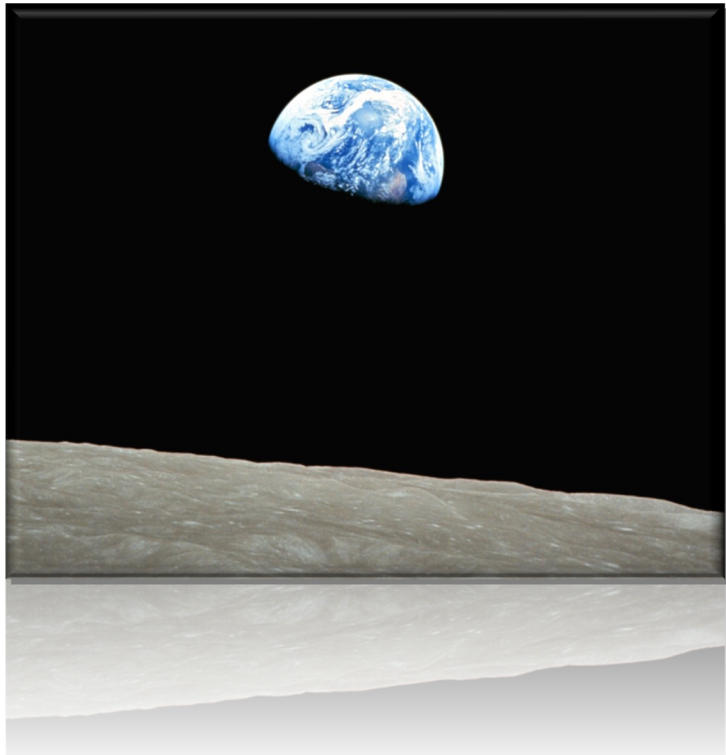
Factory methods

See en.wikipedia.org/wiki/Facade_pattern

Grouping the Java Completable Future API

Grouping the Java Completable Future API

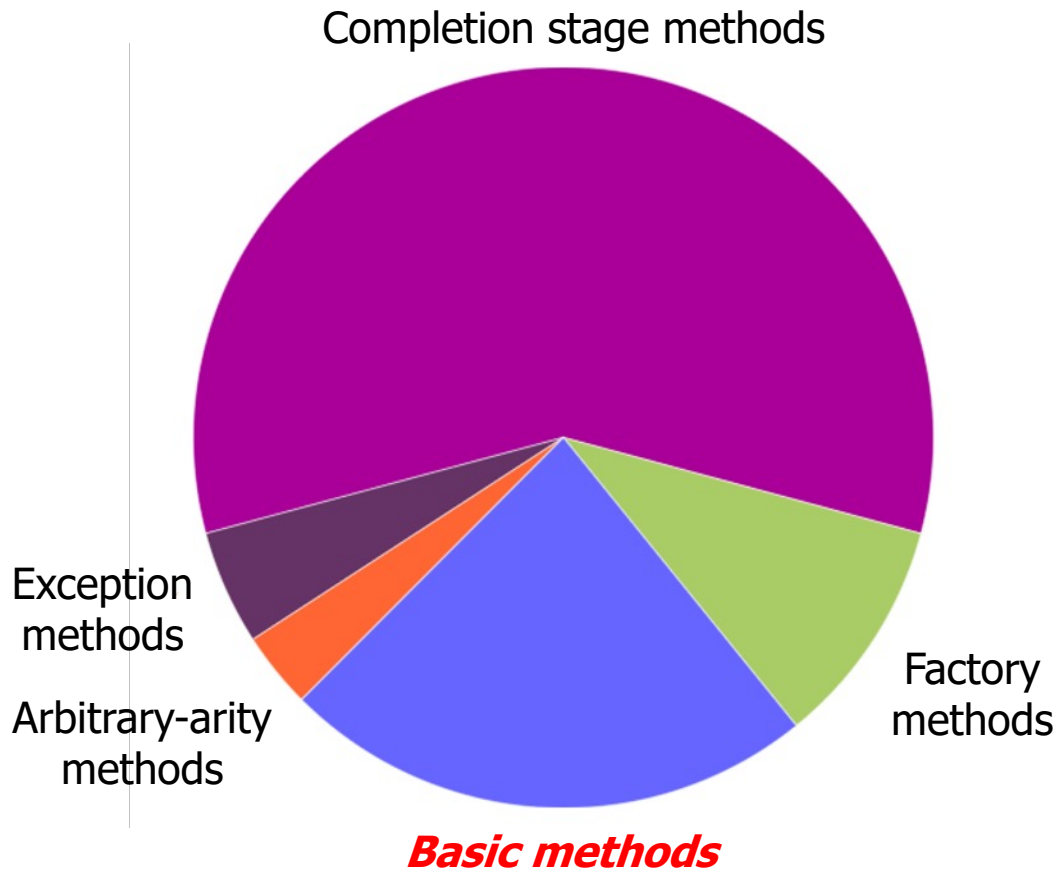
- Given the large # of methods in this API it helps to have a "birds-eye" view



See en.wikipedia.org/wiki/Earthrise

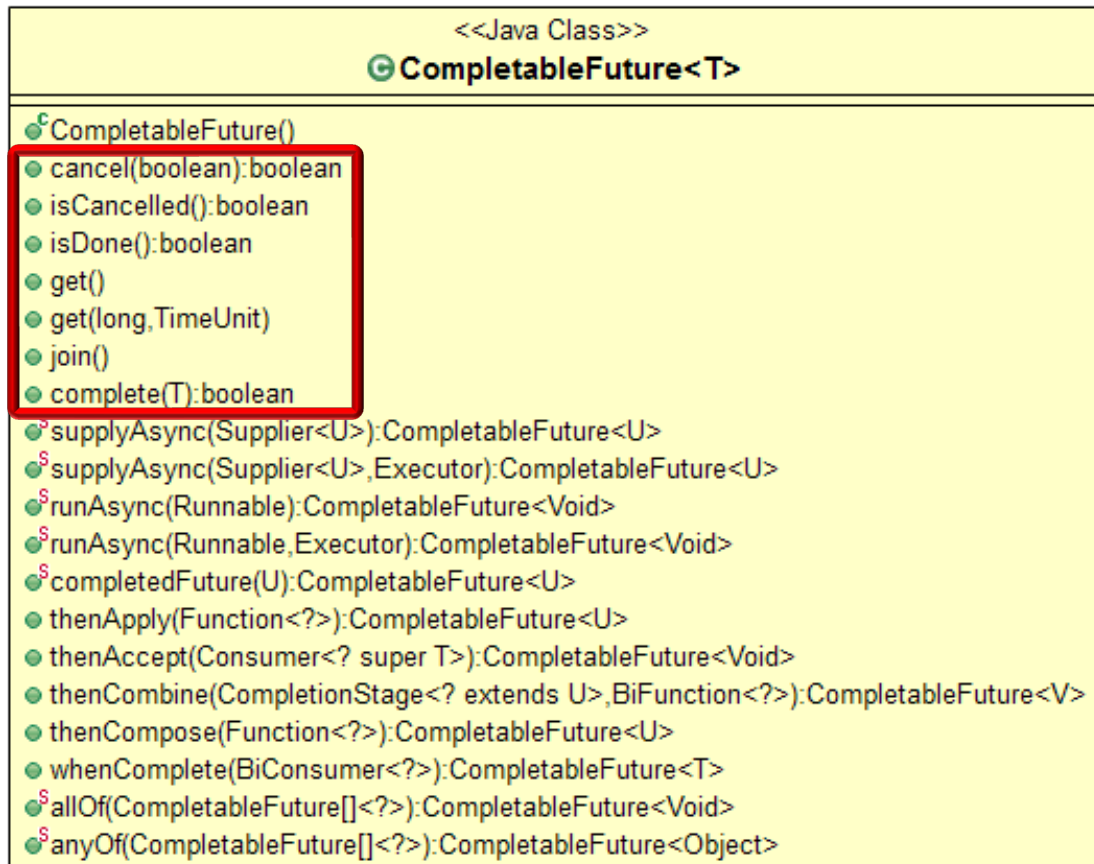
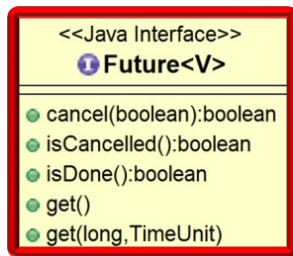
Grouping the Java Completable Future API

- Some completable future features are basic



Grouping the Java Completable Future API

- Some completable future features are basic
- e.g., the Java Future API + some simple enhancements



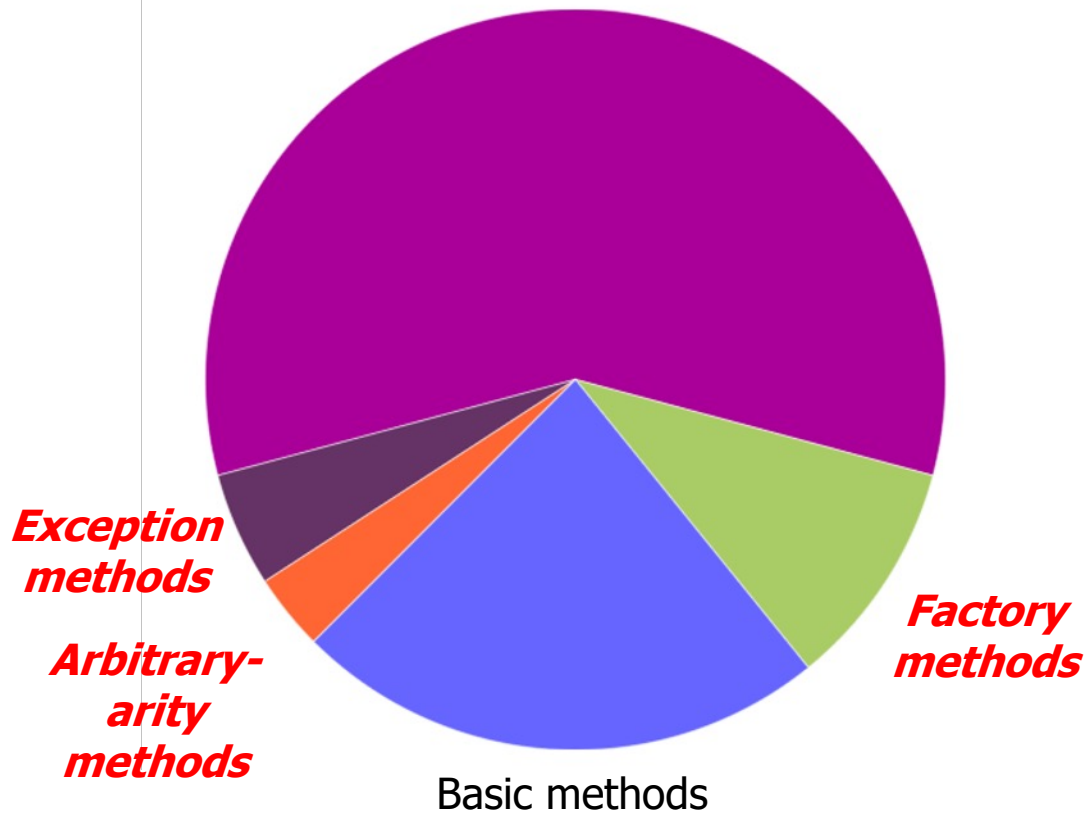
Only slightly better than the conventional Future interface

Grouping the Java Completable Future API

- Other completable future features are more advanced

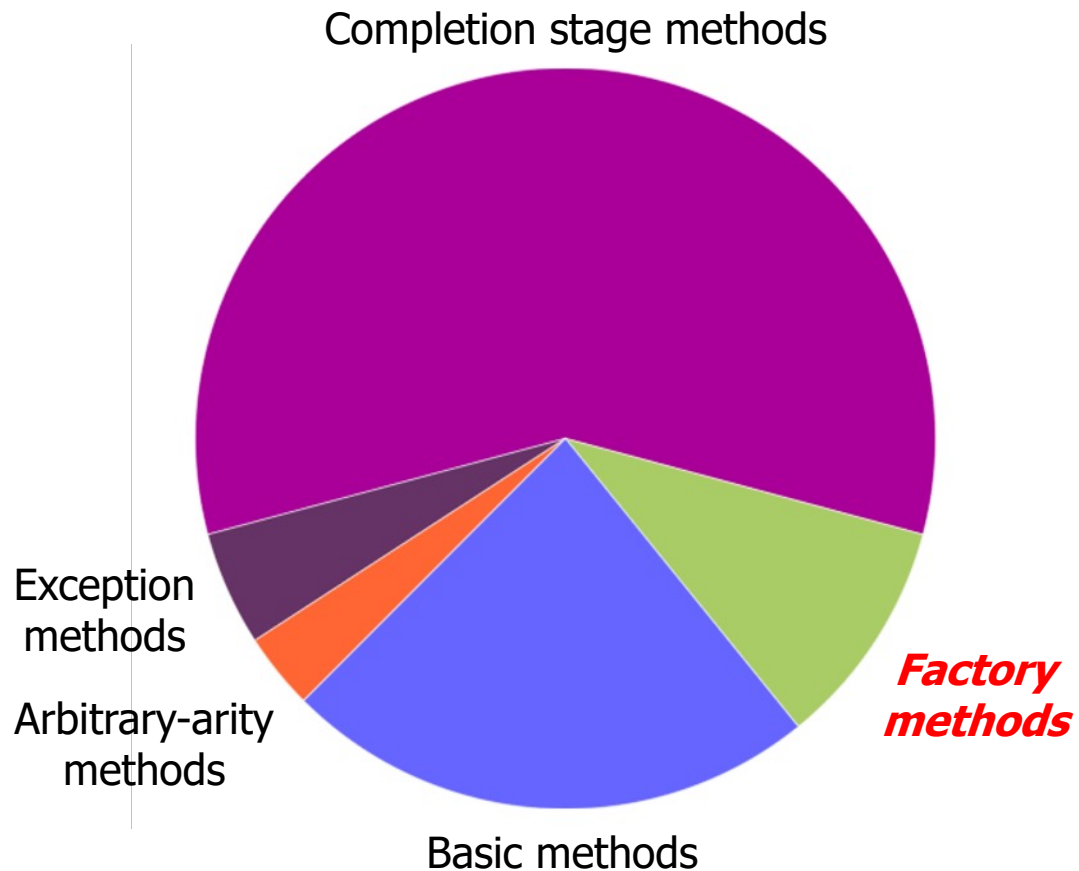


Completion stage methods



Grouping the Java Completable Future API

- Other completable future features are more advanced
 - Factory methods



See en.wikipedia.org/wiki/Factory_method_pattern

Grouping the Java Completable Future API

- Other completable future features are more advanced
 - Factory methods
 - Initiate async computations without explicit thread use

```
<<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
 - Initiate async computations without explicit thread use
 - Both one-way & two-way



```
<<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
 - Initiate async computations without explicit thread use
 - Both one-way & two-way

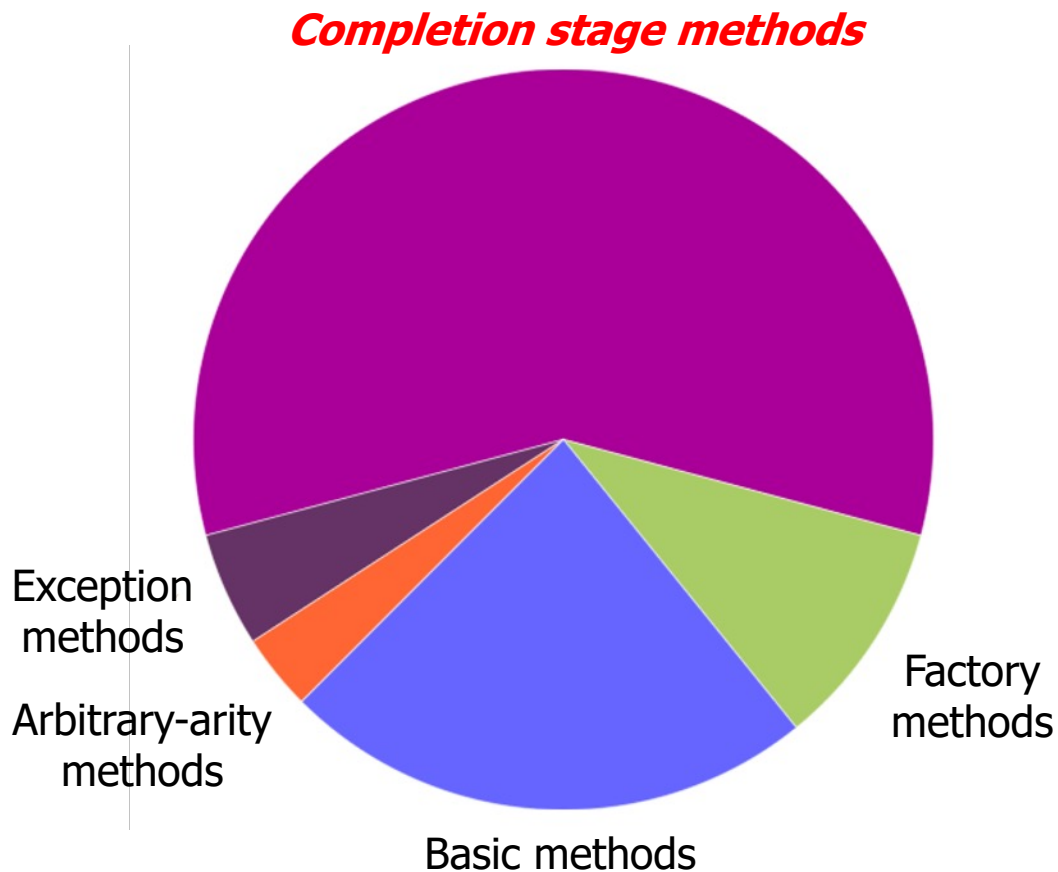


<<Java Class>>	
CompletableFuture<T>	
•	CompletableFuture()
•	cancel(boolean):boolean
•	isCancelled():boolean
•	isDone():boolean
•	get()
•	get(long,TimeUnit)
•	join()
•	complete(T):boolean
• ^S	supplyAsync(Supplier<U>):CompletableFuture<U>
• ^S	supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
•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

Help make programs more *elastic* by leveraging a pool of worker threads

Grouping the Java Completable Future API

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



Grouping the Java Completable Future API

- Other completable future features are more advanced
 - Factory methods
 - Completion stage methods
 - Chain actions that process results of async operations

```
<<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>
```

```
<<Java Interface>>
CompletionStage<T>

thenApply(Function<?>):CompletionStage<U>
thenAccept(Consumer<?>):CompletionStage<Void>
thenCombine(CompletionStage<?>,BiFunction<?>):CompletionStage<V>
thenCompose(Function<?>):CompletionStage<U>
whenComplete(BiConsumer<?>):CompletionStage<T>
```

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

Grouping the Java Completable Future API

- Other completable future features are more advanced
 - Factory methods
 - Completion stage methods
 - Chain actions that process results of async operations
 - Can trigger actions based on 1 or more prior operations

<<Java Interface>>

CompletionStage<T>

- thenApply(Function<?>):CompletionStage<U>
- thenAccept(Consumer<?>):CompletionStage<Void>
- thenCombine(CompletionStage<?>,BiFunction<?>):CompletionStage<V>
- thenCompose(Function<?>):CompletionStage<U>
- whenComplete(BiConsumer<?>):CompletionStage<T>

<<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/CompletionStage.html

Grouping the Java Completable Future API

- Other completable future features are more advanced
 - Factory methods
 - Completion stage methods
 - Chain actions that process results of async operations
 - Can trigger actions based on 1 or more prior operations

<<Java Interface>>

CompletionStage<T>

- thenApply(Function<?>):CompletionStage<U>
- thenAccept(Consumer<?>):CompletionStage<Void>
- thenCombine(CompletionStage<?>,BiFunction<?>):CompletionStage<V>
- thenCompose(Function<?>):CompletionStage<U>
- whenComplete(BiConsumer<?>):CompletionStage<T>

<<Java Class>>

CompletableFuture<T>

- CompletableFuture()
- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long,TimeUnit)
- join()
- complete(T):boolean
- ^SsupplyAsync(Supplier<U>):CompletableFuture<U>
- ^SsupplyAsync(Supplier<U>,Executor):CompletableFuture<U>
- ^SrunAsync(Runnable):CompletableFuture<Void>
- ^SrunAsync(Runnable,Executor):CompletableFuture<Void>
- ^ScompletedFuture(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>
- ^SallOf(CompletableFuture[]<?>):CompletableFuture<Void>
- ^SanyOf(CompletableFuture[]<?>):CompletableFuture<Object>



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

End of Understanding Method Groupings in the Java Completable Futures API (Part 1)