Understand Method Groupings in the Java Completable Futures API

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

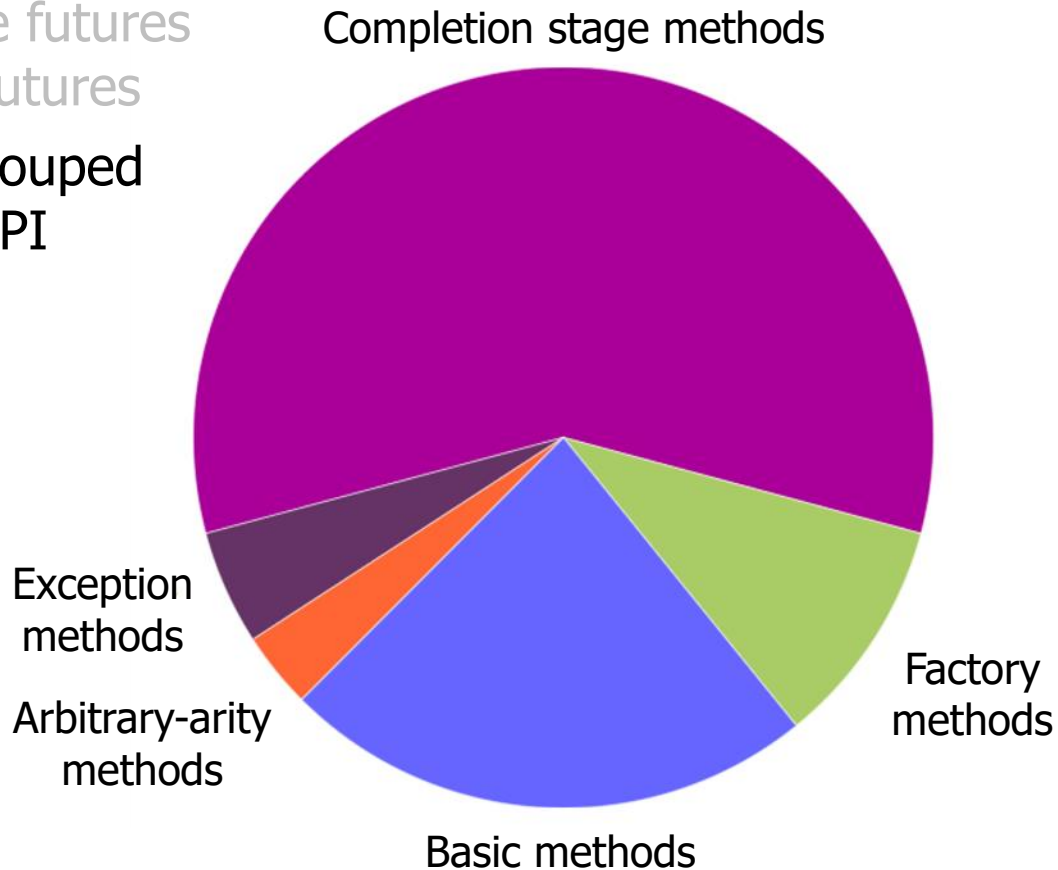**Professor of Computer Science**

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

Factory methods

Basic methods

Arbitrary-arity methods

Exception methods

Grouping the Java Completable Future API

Grouping the Java Completable Future API

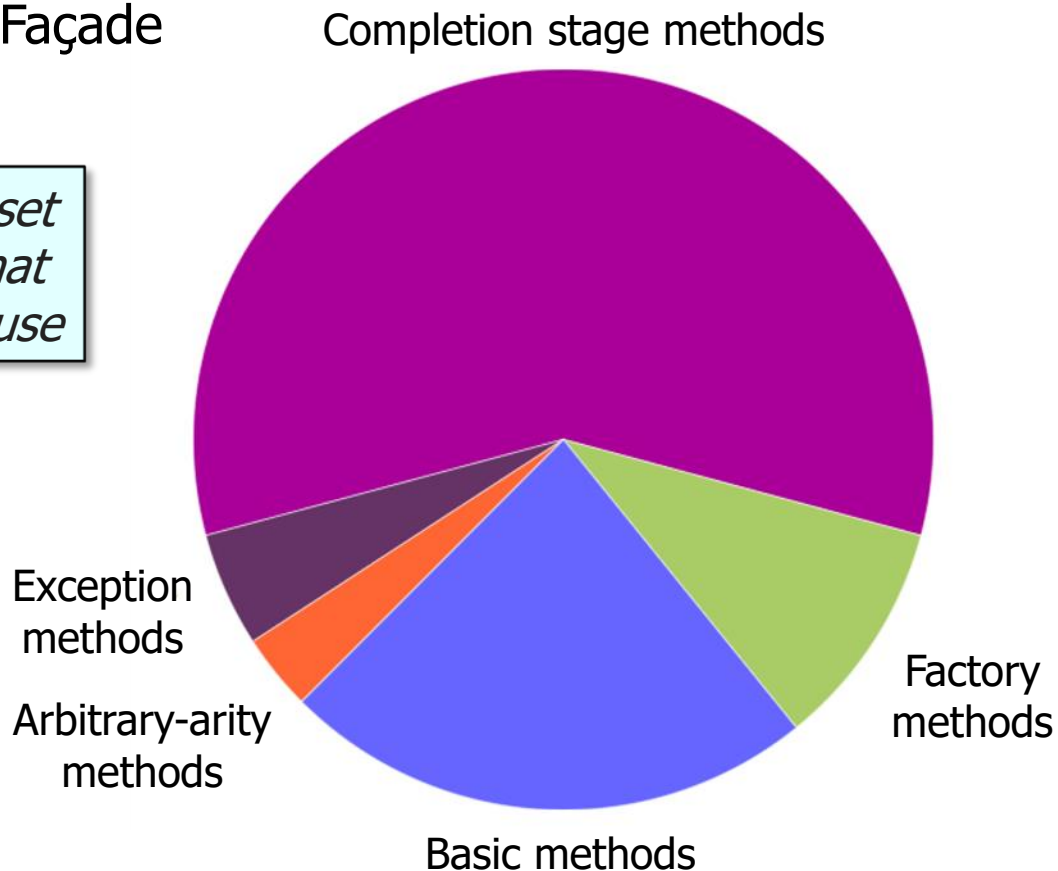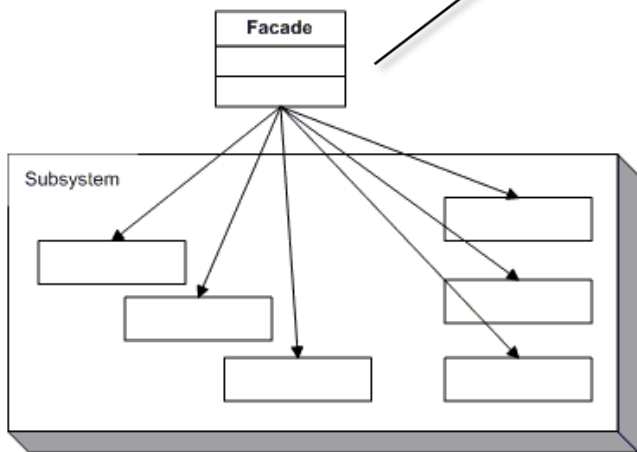- The entire completable future framework resides in 1 public class with 60+ methods!!!

<<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>

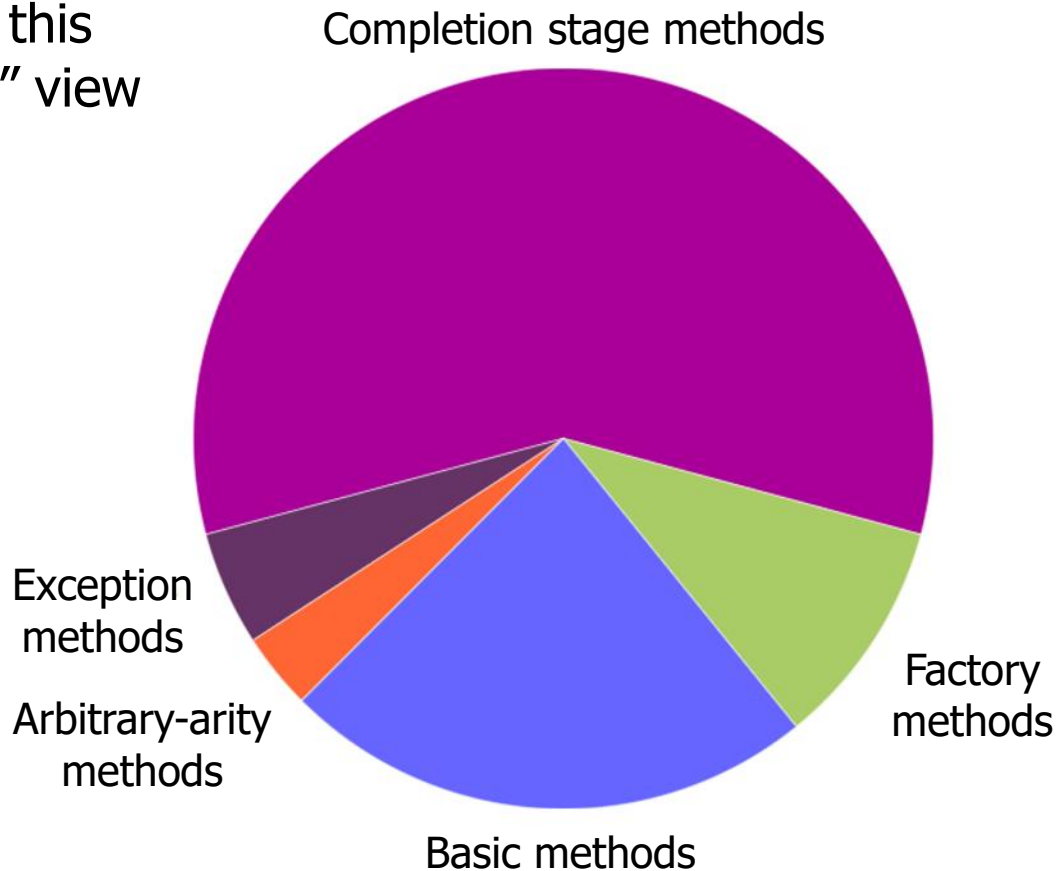See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html

- 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

Facade

Subsystem

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

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

Completion stage methods

Factory methods

Basic methods

Arbitrary-arity methods

Exception methods

See en.wikipedia.org/wiki/Earthrise

Grouping the Java Completable Future API

- Some completable future features are basic

Completion stage methods

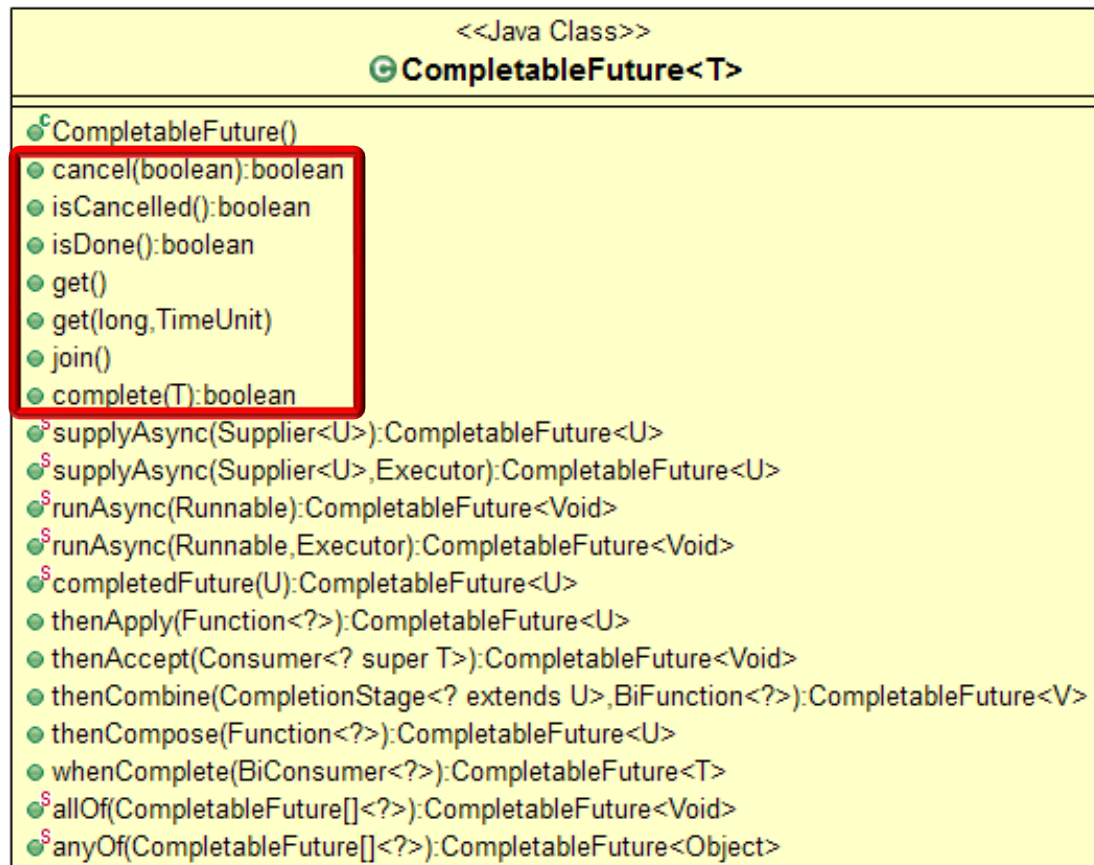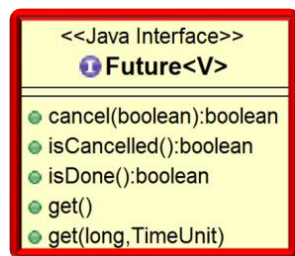Factory methods
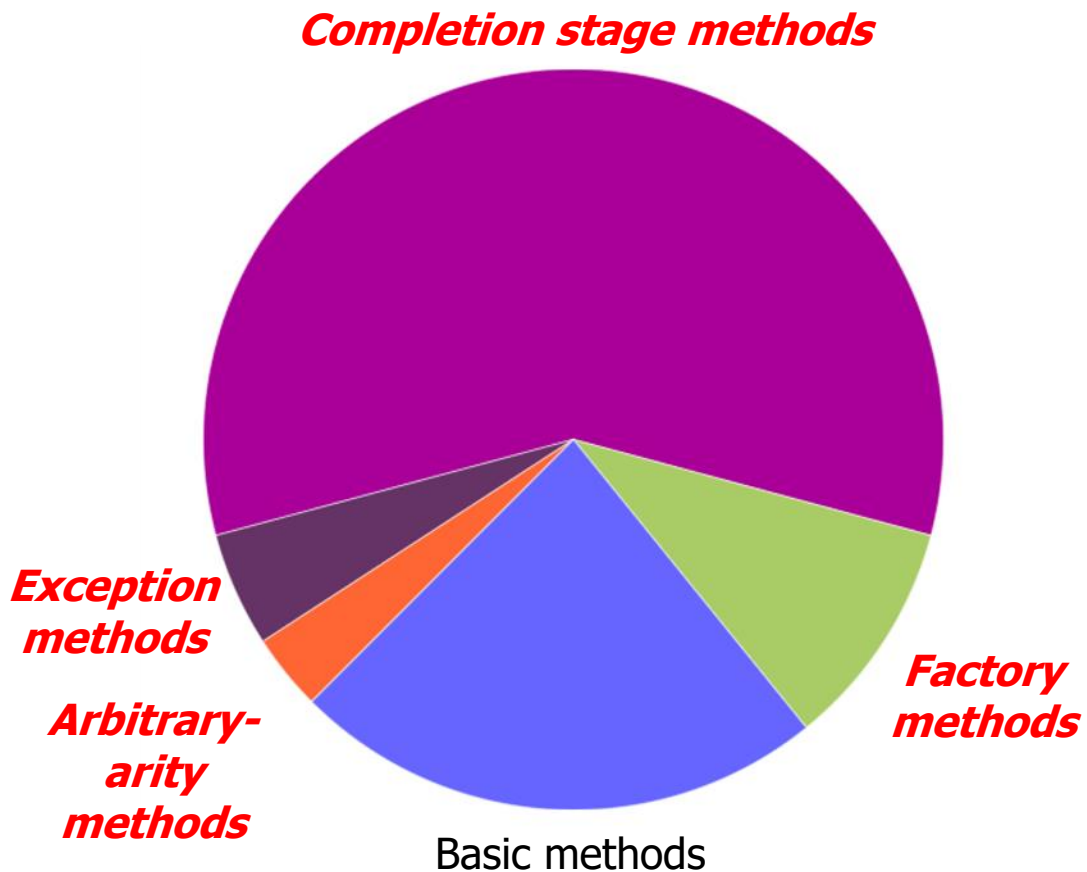
Exception methods

Arbitrary-arity methods

Basic methods
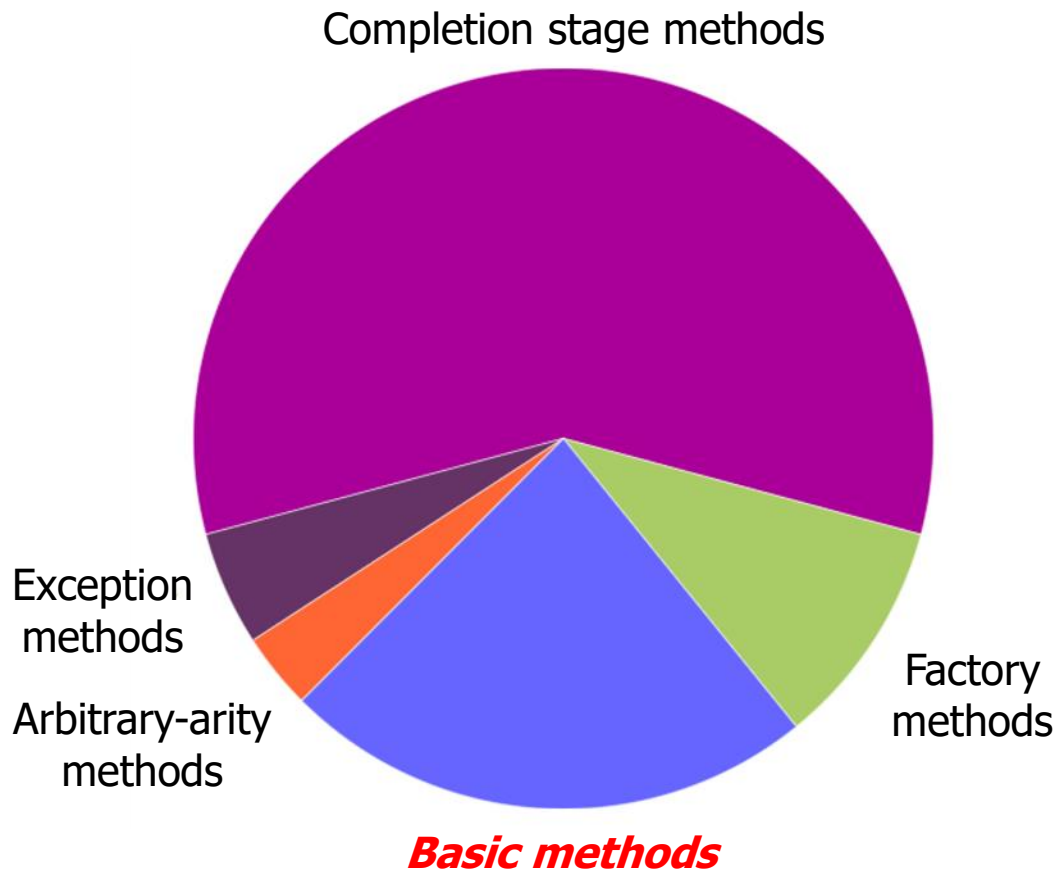
Grouping the Java Completable Future API

- Some completable future features are basic

  - e.g., the Java Future API + some simple enhancements

<<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>>
① Future<V>

- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long,TimeUnit)

Only slightly better than the conventional Future interface

Grouping the Java Completable Future API

- Other completable future features are more advanced

Completion stage methods

Exception methods

Arbitrary-arity methods

Factory methods

Basic methods

Grouping the Java Completable Future API

- Other completable future features are more advanced
  - Factory methods

Completion stage methods

Factory methods

Basic methods

Arbitrary-arity methods

Exception methods
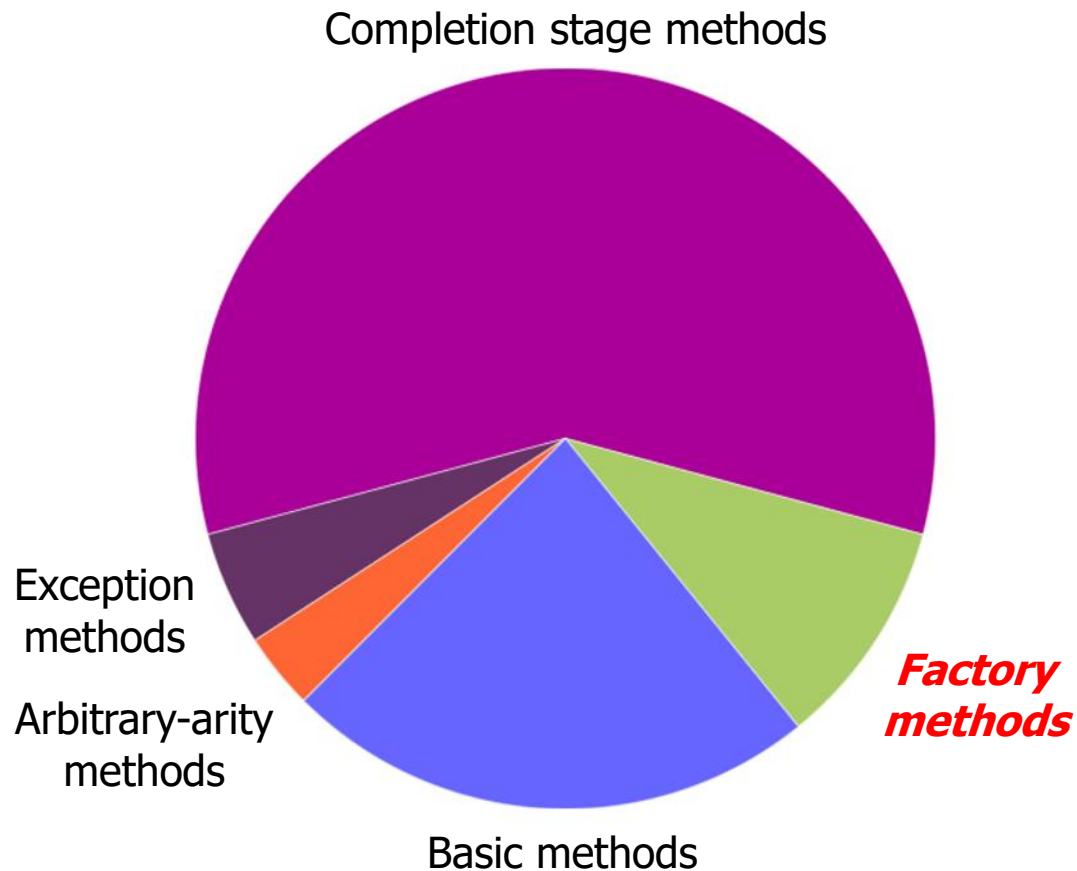
See en.wikipedia.org/wiki/Factory_method_pattern

Grouping the Java Completable Future API

- Other completable future features are more advanced
  - Factory methods
    - Initiate async two-way or one-way computations without using threads explicitly

<<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

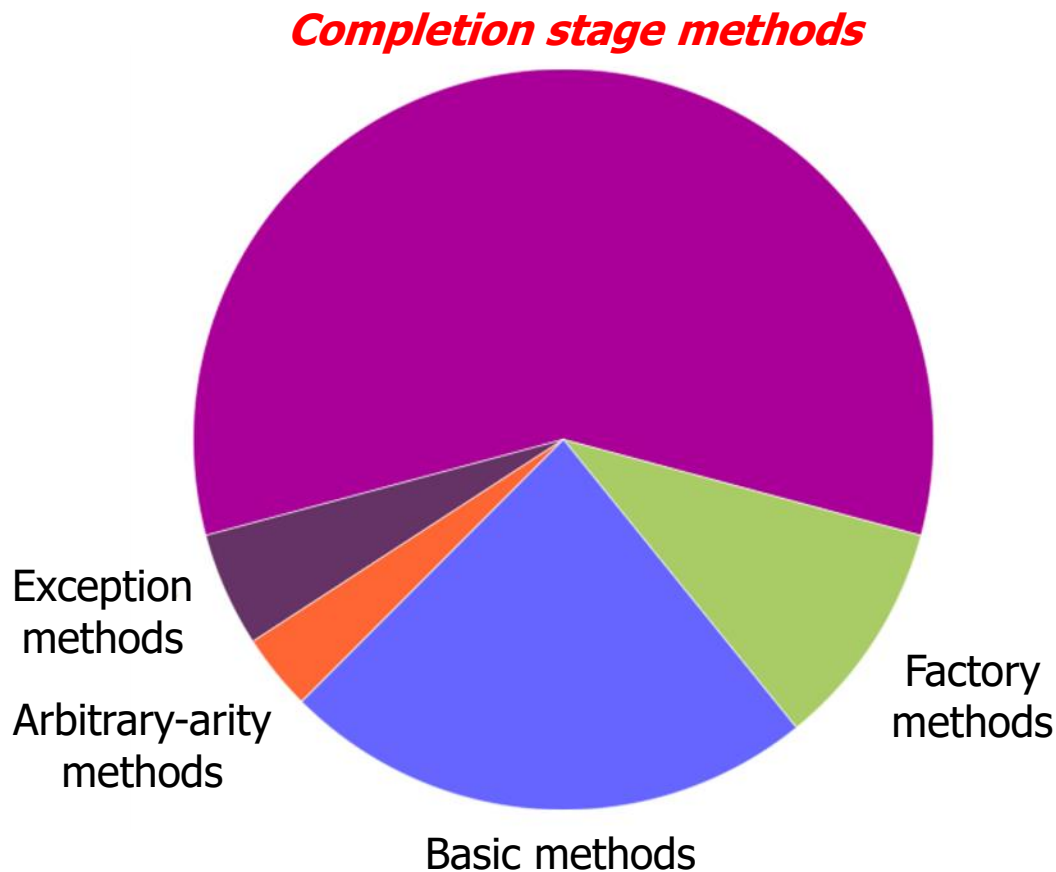   - Initiate async two-way or one-way computations without using threads explicitly


```
                  <<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 *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

Completion stage methods

Exception methods

Arbitrary-arity methods

Basic methods

Factory methods

Grouping the Java Completable Future API

- Other completable future features are more advanced

  - Factory methods

  - Completion stage methods

    - Chain together actions that perform async result processing & composition

<<Java Class>>
C 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>>
I 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 together actions that perform async result processing & composition
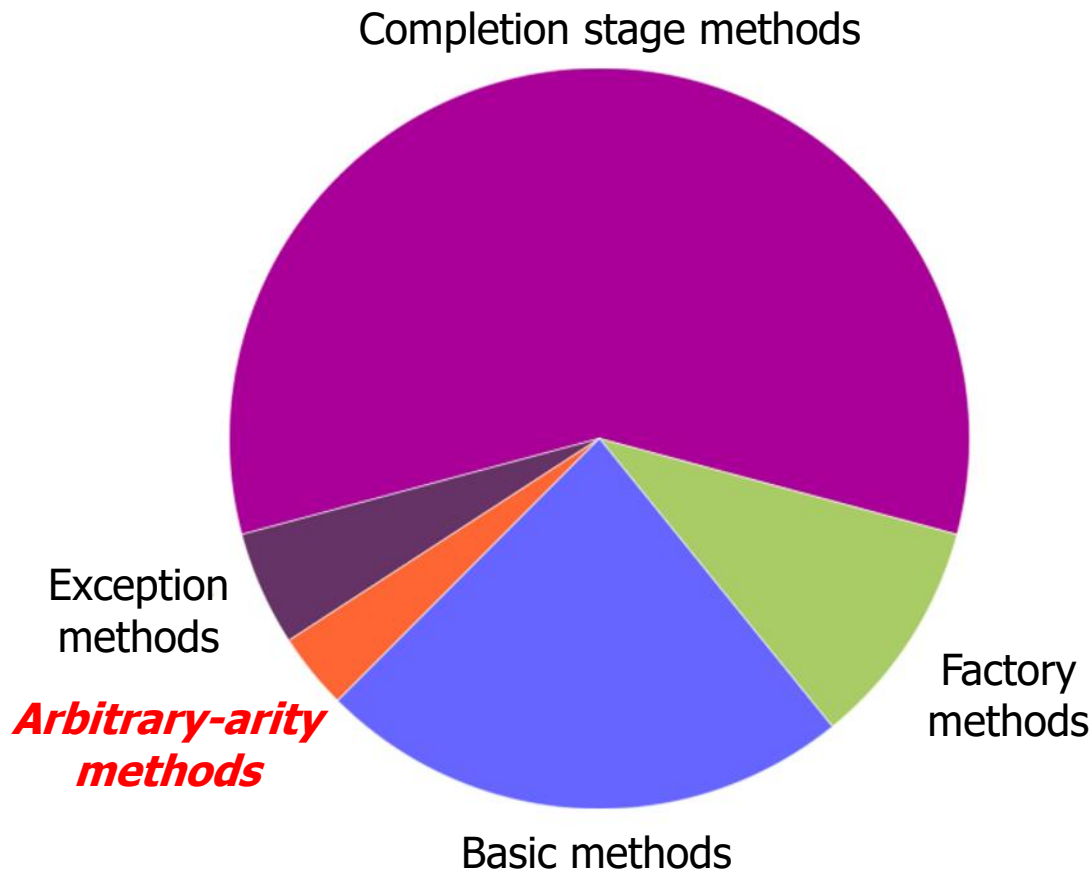
<<Java Class>>
CompletableFuture<T>

- 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>

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

Completion stage methods

Factory methods

Basic methods

Exception methods
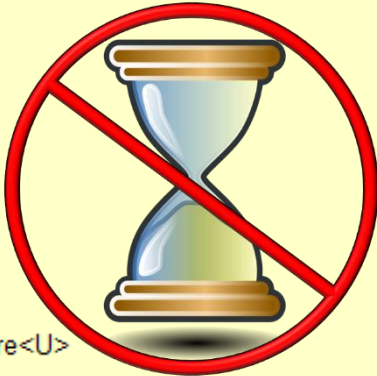
Arbitrary-arity methods

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>

- 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
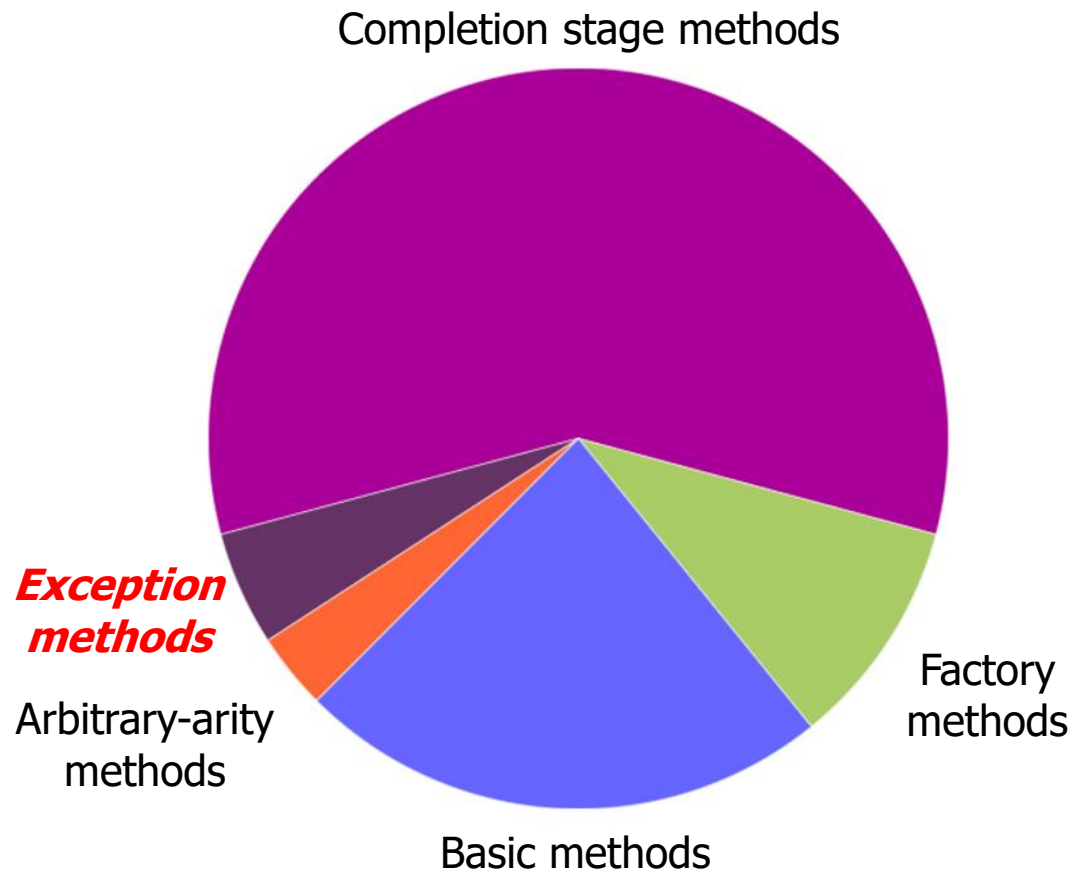
<<Java Class>>
© CompletableFuture<T>

- 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>

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
  - Exception methods

Completion stage methods

Factory methods

Basic 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

<<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 *resilient* by handling erroneous computations gracefully

- All methods are implemented internally via message-passing that's ultimately connected to Java-based thread pools

<<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>

Ensures loose coupling, isolation, & location transparency between components

End of Understand Method Groupings in the Java Completable Futures API