Overview of Java 8 CompletableFutures (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 the basic completable futures features

Class CompletableFuture<T>

java.lang.Object
    java.util.concurrent.CompletableFuture<T>

All Implemented Interfaces:

CompletionStage<T>, Future<T>

```
public class CompletableFuture<T>
extends Object
implements Future<T>, CompletionStage<T>
```

A Future that may be explicitly completed (setting its value and status), and may be used as a CompletionStage, supporting dependent functions and actions that trigger upon its completion.

When two or more threads attempt to complete, completeExceptionally, or cancel a CompletableFuture, only one of them succeeds.

In addition to these and related methods for directly manipulating status and results, CompletableFuture implements interface CompletionStage with the following policies:

Overview of Completable Futures

Overview of Completable Futures

- The completable future framework overcomes Java future limitations

Class CompletableFuture<T>

java.lang.Object
    java.util.concurrent.CompletableFuture<T>

All Implemented Interfaces:

CompletionStage<T>, Future<T>

```
public class CompletableFuture<T>
extends Object
implements Future<T>, CompletionStage<T>
```

A Future that may be explicitly completed (setting its value and status), and may be used as a CompletionStage, supporting dependent functions and actions that trigger upon its completion.

When two or more threads attempt to complete, completeExceptionally, or cancel a CompletableFuture, only one of them succeeds.

In addition to these and related methods for directly manipulating status and results, CompletableFuture implements interface CompletionStage with the following policies:

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

Overview of Completable Futures

- The completable future framework overcomes Java future limitations
  - Some features are basic

Class CompletableFuture<T>

java.lang.Object
    java.util.concurrent.CompletableFuture<T>

All Implemented Interfaces:

CompletionStage<T>, Future<T>

```
public class CompletableFuture<T>
extends Object
implements Future<T>, CompletionStage<T>
```

A Future that may be explicitly completed (setting its value and status), and may be used as a CompletionStage, supporting dependent functions and actions that trigger upon its completion.

When two or more threads attempt to complete, completeExceptionally, or cancel a CompletableFuture, only one of them succeeds.

In addition to these and related methods for directly manipulating status and results, CompletableFuture implements interface CompletionStage with the following policies:

Overview of Completable Futures

- The completable future framework overcomes Java future limitations
  - Some features are basic
  - Some features are advanced

Class CompletableFuture<T>

java.lang.Object
    java.util.concurrent.CompletableFuture<T>

All Implemented Interfaces:

CompletionStage<T>, Future<T>

```
public class CompletableFuture<T>
extends Object
implements Future<T>, CompletionStage<T>
```

A Future that may be explicitly completed (setting its value and status), and may be used as a CompletionStage, supporting dependent functions and actions that trigger upon its completion.

When two or more threads attempt to complete, completeExceptionally, or cancel a CompletableFuture, only one of them succeeds.

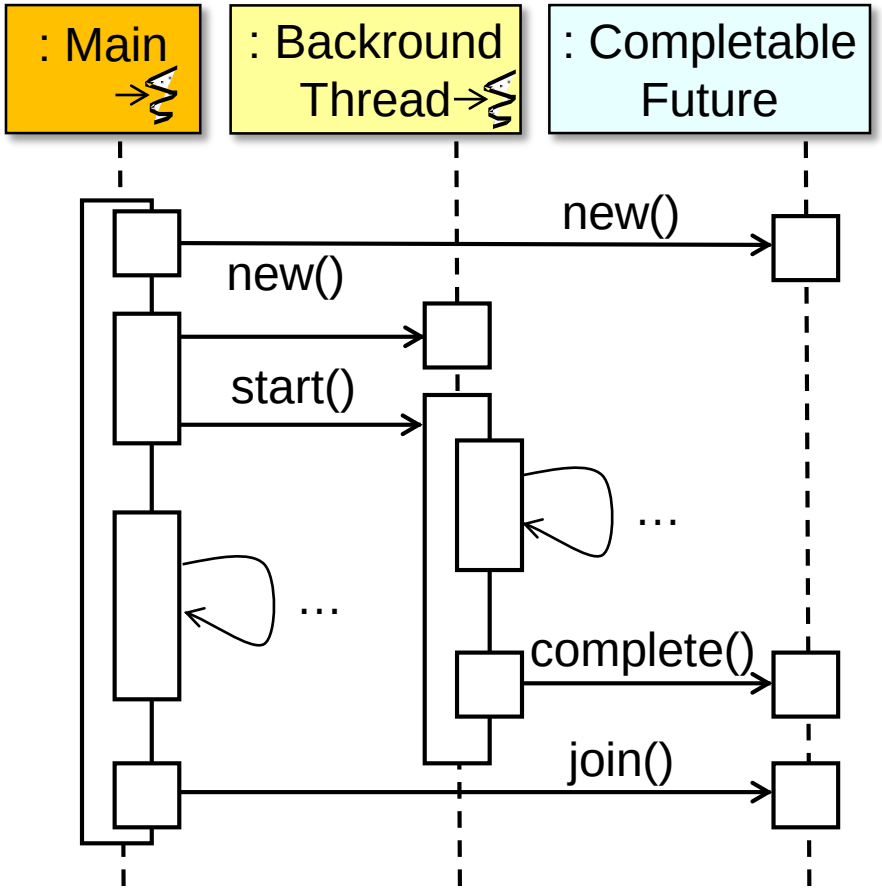
In addition to these and related methods for directly manipulating status and results, CompletableFuture implements interface CompletionStage with the following policies:

The entire completable futures framework resides in one large class!!!

Basic Completable Futures Features

Basic Completable Futures Features

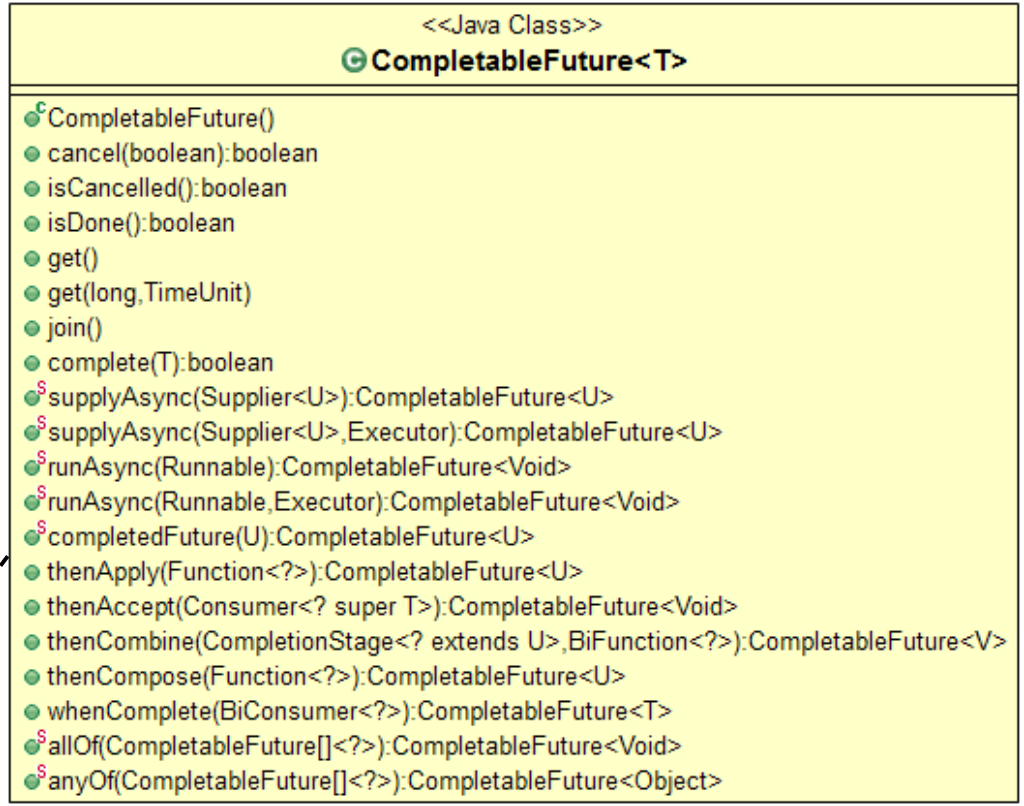
- Basic completable future features

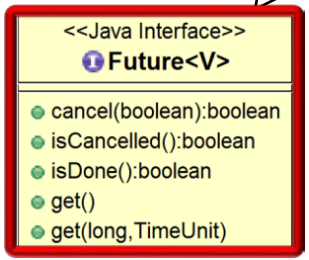


See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8

Basic Completable Futures Features

- Basic completable future features
  - Supports the Future API

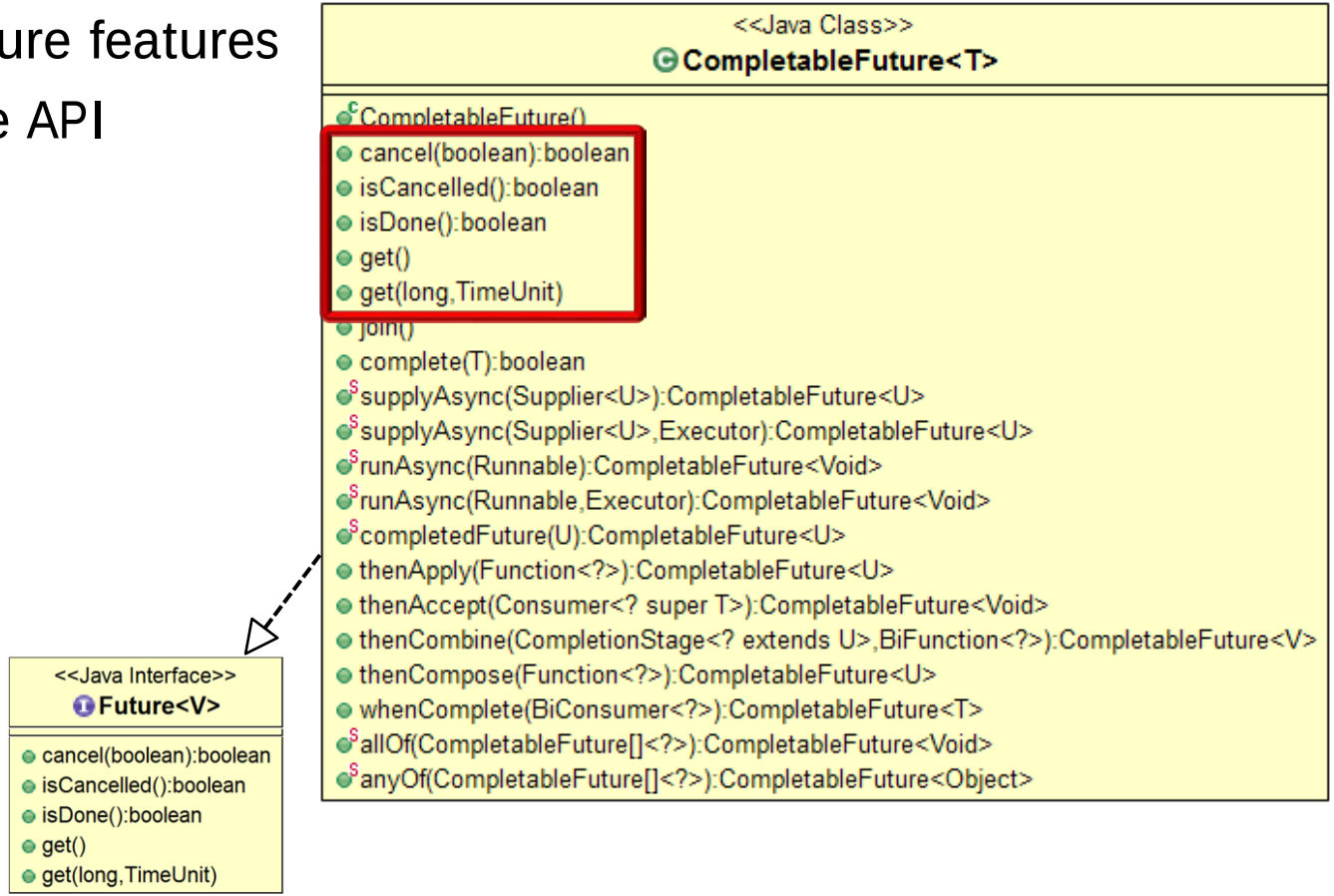




See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html

Basic Completable Futures Features

- Basic completable future features
  - Supports the Future API



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

Basic Completable Futures Features

- Basic completable future features
  - Supports the Future API
  - Defines a join() method

<<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#join

Basic Completable Futures Features

- Basic completable future features
  - Supports the Future API
  - Defines a join() method
    - Behaves like get() without using checked 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>
```

Basic Completable Futures Features

- Basic completable future features
  - Supports the Future API
  - Defines a join() method
    - Behaves like get() without using checked exceptions

```
futures
   .stream()
   .map(future
         -> future.join())
   .collect(toList()
```

<<Java Class>>

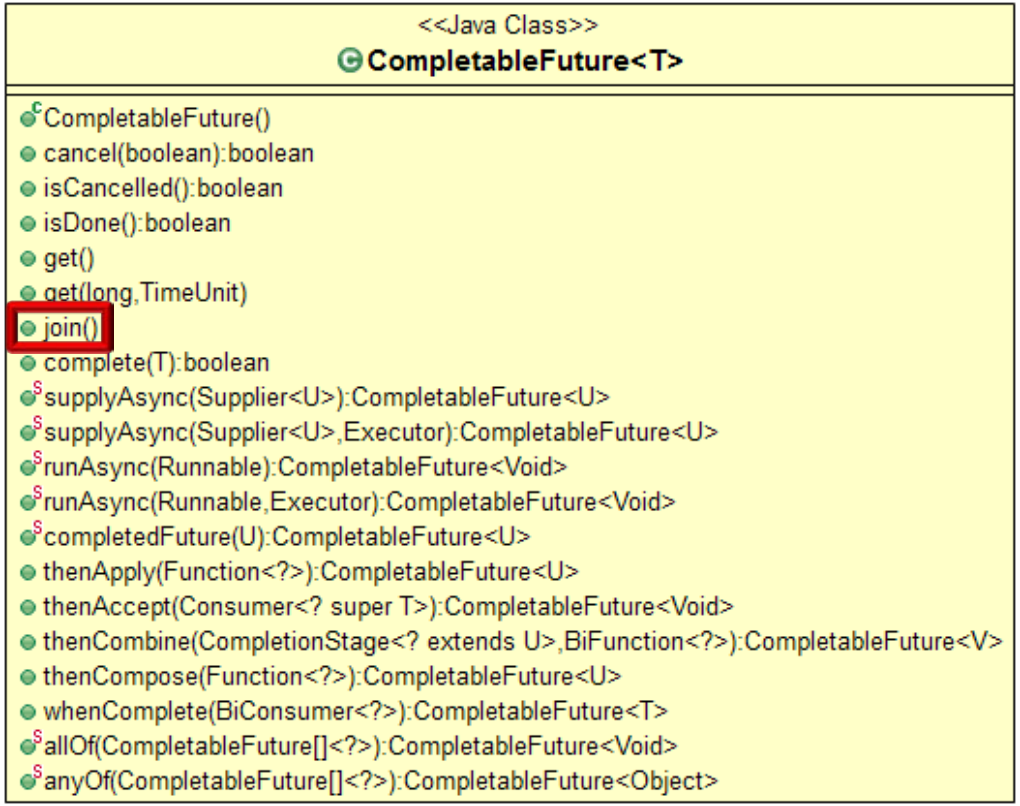
CompletableFuture<T>

- 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>

Basic Completable Futures Features

- Basic completable future features
  - Supports the Future API
  - Defines a join() method
    - Behaves like get() without using checked exceptions

```
futures
  .stream()
  .map(future
       -> try { future.get();
       } catch (Exception e){
       })
  .collect(toList())
```



Mixing checked exceptions & Java 8 streams is ugly..

Basic Completable Futures Features

- Basic completable future features
  - Supports the Future API
  - Defines a join() method
  - Can be explicitly completed

<<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#complete

Basic Completable Futures Features

- Basic completable future features
  - Supports the Future API
  - Defines a join() method
  - Can be explicitly completed
    - i.e., sets result returned by get() or join() to a given value

<<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#complete

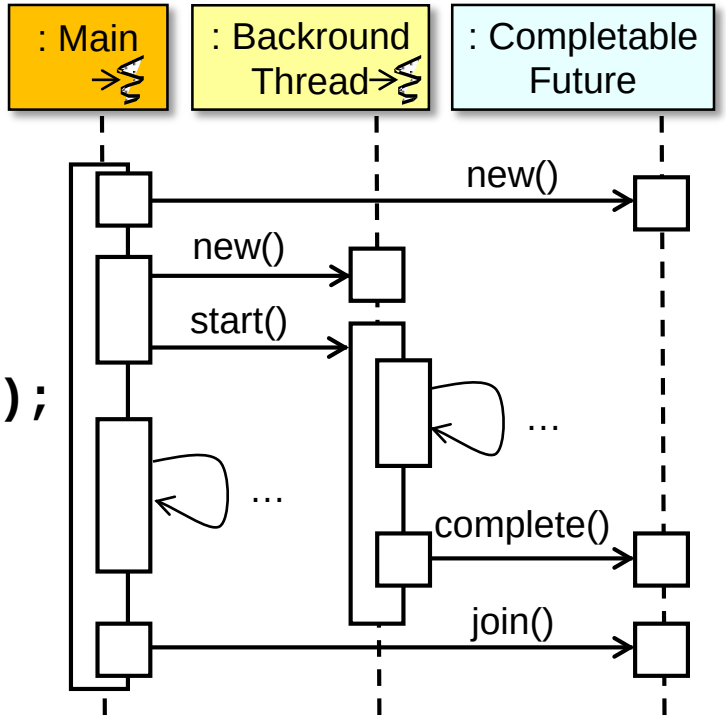
Basic Completable Futures Features

- Example of basic completable future features

```
CompletableFuture<BigInteger>
  future = new CompletableFuture<>();

new Thread (() -> {
  BigInteger bi1 =
    new BigInteger("18802723413348219 6");
  BigInteger bi2 =
    new BigInteger("2434101");
  future.complete(bi1.gcd(bi2));
}).start();


...
System.out.println("GCD = " + future.join());
```



See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8

Basic Completable Futures Features

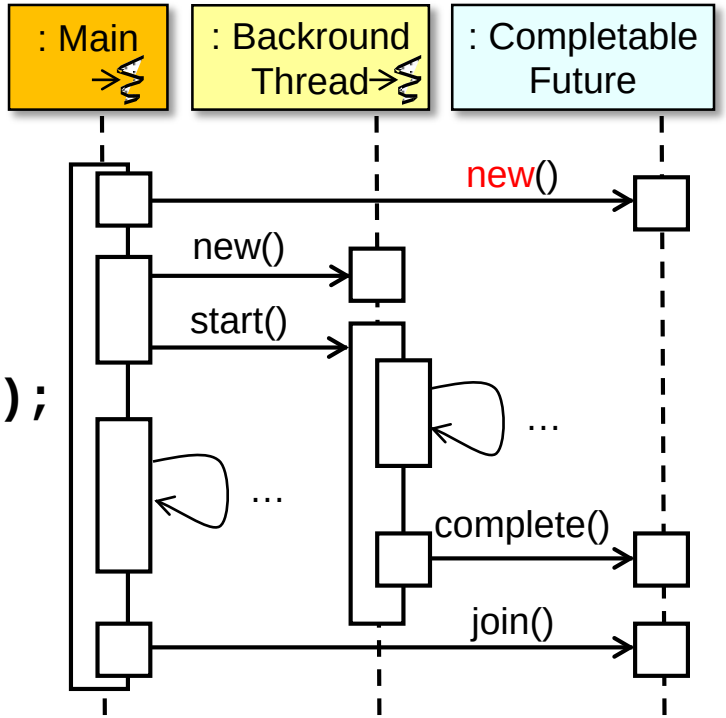
- Example of basic completable future features

```
CompletableFuture<BigInteger>
  future = new CompletableFuture<>();

new Thread (() -> {
  BigInteger bi1 =
    new BigInteger("18802723413348219 6");
  BigInteger bi2 =
    new BigInteger("2434101");
  future.complete(bi1.gcd(bi2));
}).start();

...
System.out.println("GCD = " + future.join());
```

Make object



Basic Completable Futures Features

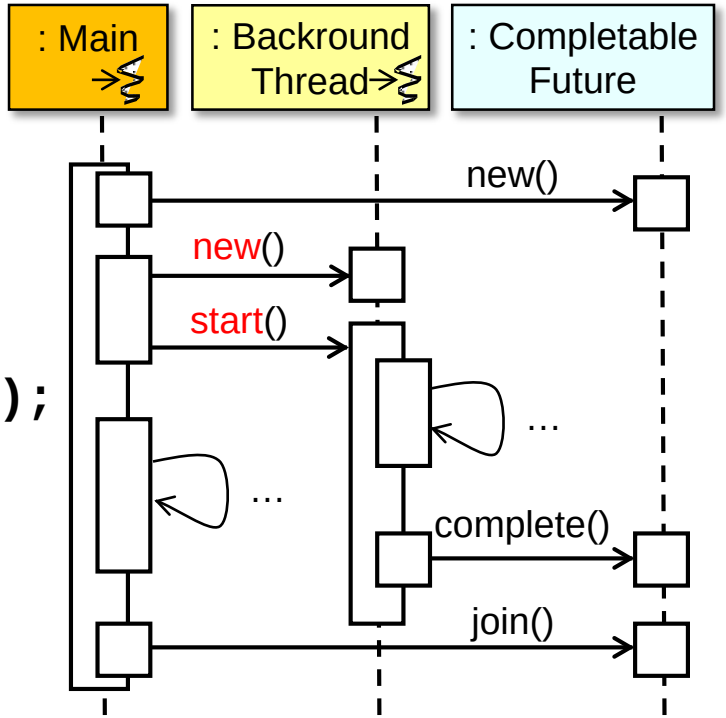
- Example of basic completable future features

```
CompletableFuture<BigInteger>
  future = new CompletableFuture<>();

new Thread (() -> {
  BigInteger bi1 =
    new BigInteger("18802723413348219 6");
  BigInteger bi2 =
    new BigInteger("2434101");
  future.complete(bi1.gcd(bi2));
}).start();

...
System.out.println("GCD = " + future.join());
```

Start computation in a background thread



Basic Completable Futures Features

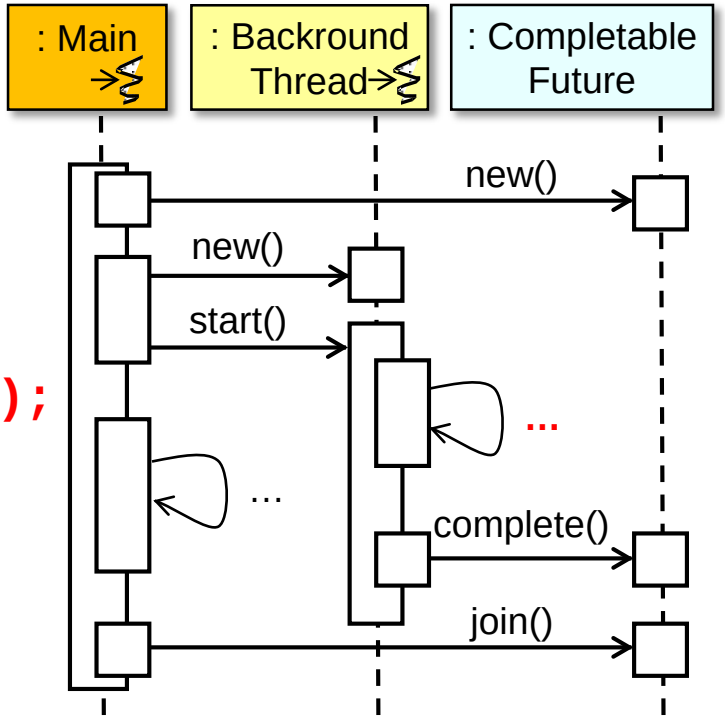
- Example of basic completable future features

```
CompletableFuture<BigInteger>
  future = new CompletableFuture<>();

new Thread (() -> {
  BigInteger bi1 =
    new BigInteger("188027234133482196");
  BigInteger bi2 =
    new BigInteger("2434101");
  future.complete(bi1.gcd(bi2));
}).start();

...
System.out.println("GCD = " + future.join());
```

Start computation in a background thread



See docs.oracle.com/javase/8/docs/api/java/math/BigInteger.html

Basic Completable Futures Features

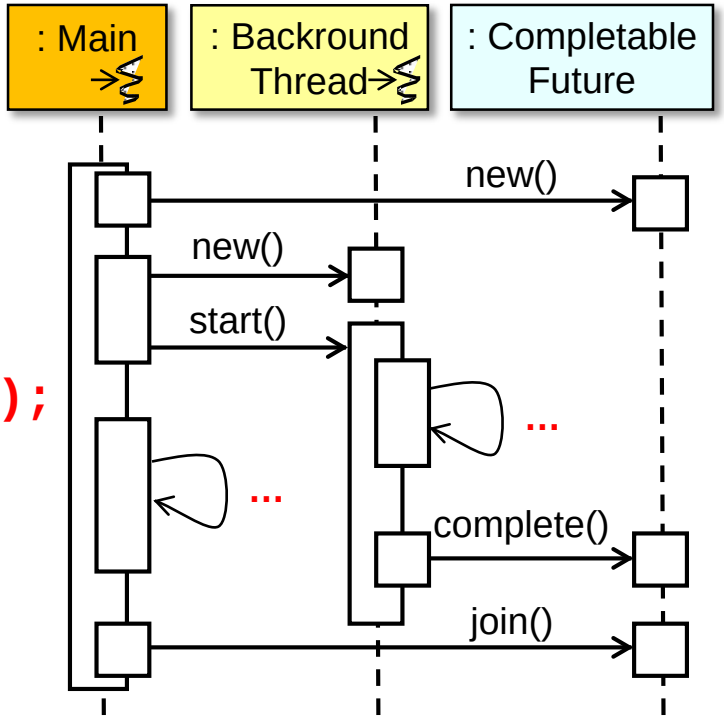
- Example of basic completable future features

```
CompletableFuture<BigInteger>
  future = new CompletableFuture<>();

new Thread (() -> {
  BigInteger bi1 =
    new BigInteger("18802723413348219 6");
  BigInteger bi2 =
    new BigInteger("2434101");
  future.complete(bi1.gcd(bi2));
}).start();

...
System.out.println("GCD = " + future.join());
```

These computations run concurrently



Basic Completable Futures Features

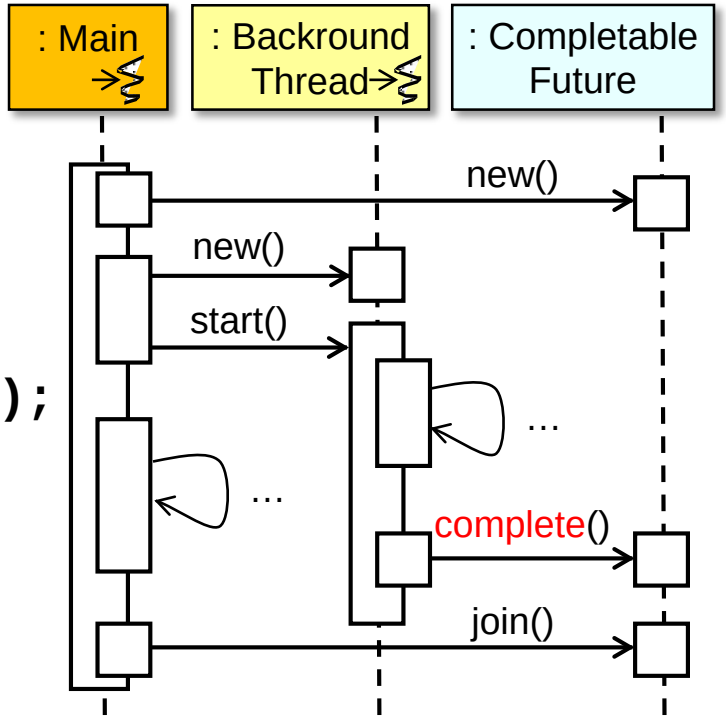
- Example of basic completable future features

```
CompletableFuture<BigInteger>
  future = new CompletableFuture<>();

new Thread (() -> {
  BigInteger bi1 =
    new BigInteger("18802723413348219 6");
  BigInteger bi2 =
    new BigInteger("2434101");
  future.complete(bi1.gcd(bi2));
}).start();
```

CompletableFuture can be completed explicitly

```
...
System.out.println("GCD = " + future.join());
```

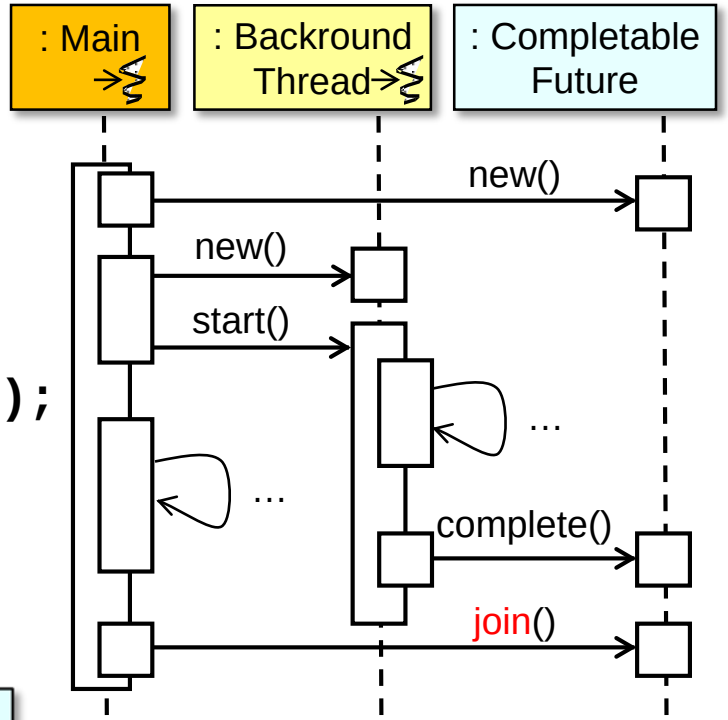


Basic Completable Futures Features

- Example of basic completable future features

```
CompletableFuture<BigInteger>
  future = new CompletableFuture<>();

new Thread (() -> {
  BigInteger bi1 =
    new BigInteger("18802723413348219 6");
  BigInteger bi2 =
    new BigInteger("2434101");
  future.complete(bi1.gcd(bi2));
}).start();
```



join() returns the GCD result

```
...
System.out.println("GCD = " + future.join());
```

Basic Completable Futures Features

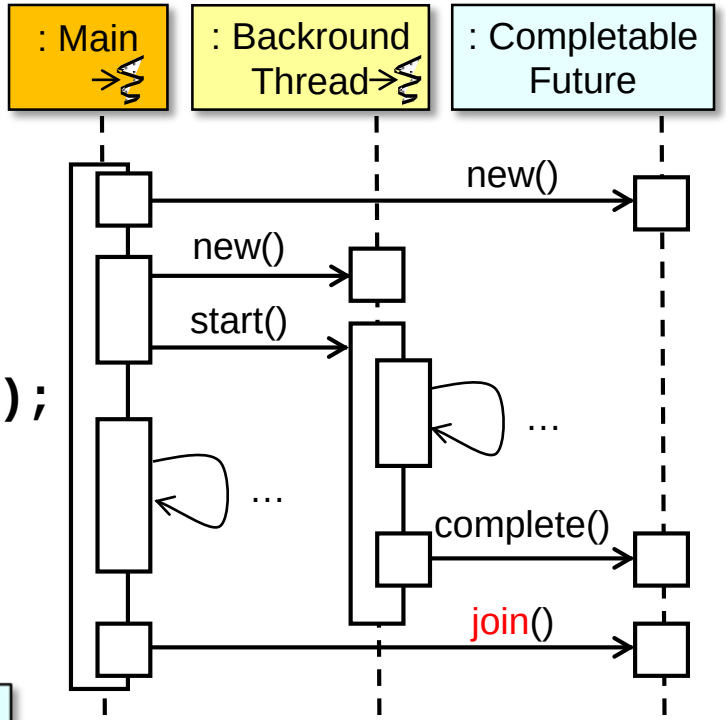
- Example of basic completable future features

```
CompletableFuture<BigInteger>
  future = new CompletableFuture<>();

new Thread (() -> {
  BigInteger bi1 =
    new BigInteger("18802723413348219 6");
  BigInteger bi2 =
    new BigInteger("2434101");
  future.complete(bi1.gcd(bi2));
}).start();
```

join() blocks until this future is completed

```
...
System.out.println("GCD = " + future.join());
```



End of Overview of Java 8 Completable Futures (Part 1)