Enhancing Java Completable Futures: Framework Extensibility (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

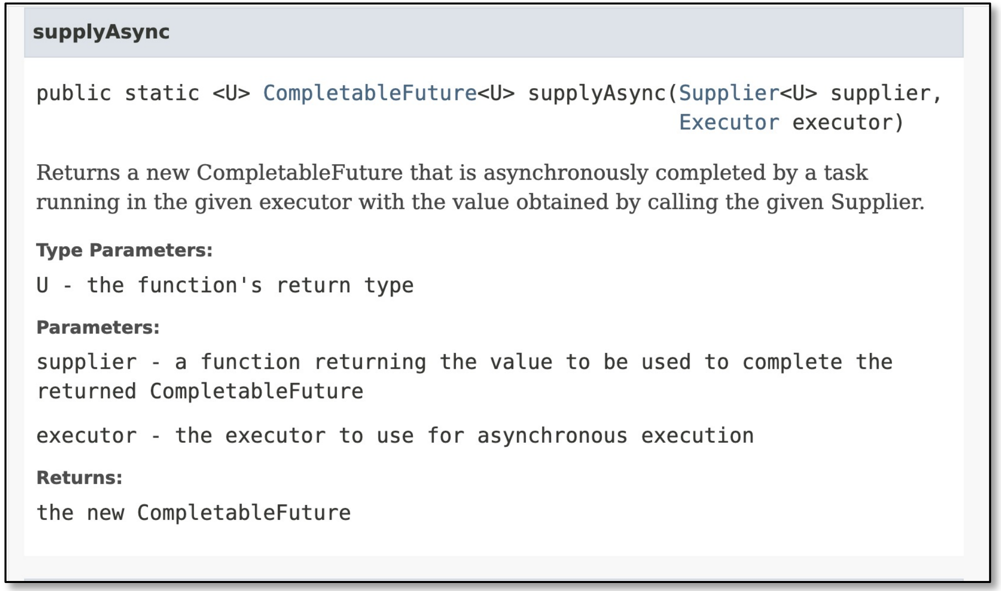
Learning Objectives in this Part of the Lesson

- Evaluate the pros of using the Java completable futures framework
- Evaluate the cons of using the Java completable futures framework
- Understand enhancements to the Java completable futures framework
  - Enhanced timeout handling
  - Enhancing extensibility

Enhancing Java Completable Future Extensibility

Enhancing Java CompletableFuture Extensibility

- Java 8 had overloaded methods for specifying an executor, but not for every stage of the computation



supplyAsync

```
public static <U> CompletableFuture<U> supplyAsync(Supplier<U> supplier,
                                                   Executor executor)
```

Returns a new CompletableFuture that is asynchronously completed by a task running in the given executor with the value obtained by calling the given Supplier.

Type Parameters:

`U` - the function's return type

Parameters:

`supplier` - a function returning the value to be used to complete the returned CompletableFuture

`executor` - the executor to use for asynchronous execution

Returns:

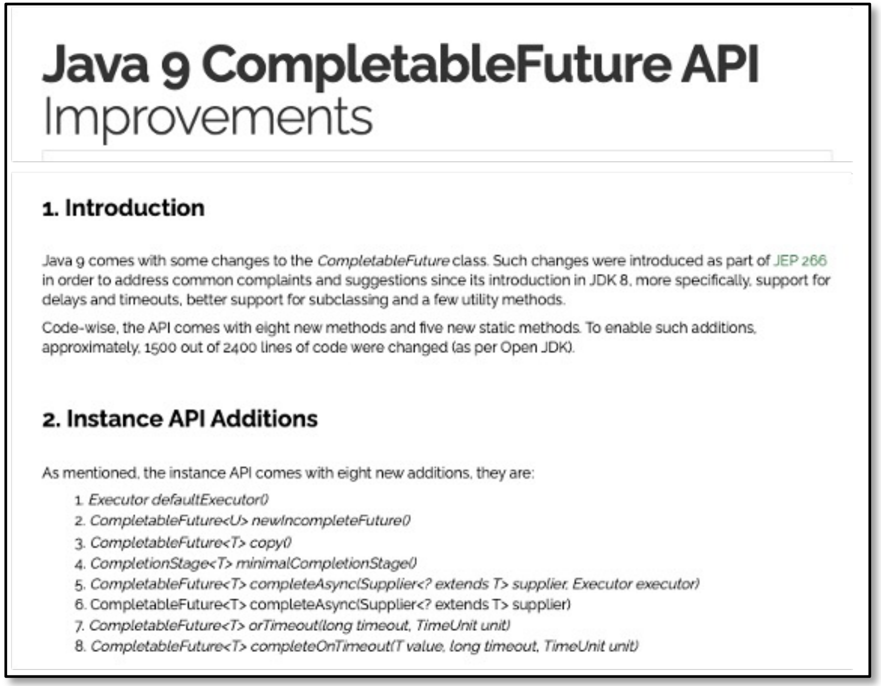
the new CompletableFuture

Enhancing Java CompletableFuture Extensibility

- Java 8 had overloaded methods for specifying an executor, but not for every stage of the computation
  - This design choice meant that while developers could specify an executor for some operations, others would default to the common ForkJoinPool

Enhancing Java CompletableFuture Extensibility

- Java 9 enhances the Java 8 completable future framework to better support custom Executor implementations



Java 9 CompletableFuture API Improvements

1. Introduction

Java 9 comes with some changes to the *CompletableFuture* class. Such changes were introduced as part of JEP 266 in order to address common complaints and suggestions since its introduction in JDK 8, more specifically, support for delays and timeouts, better support for subclassing and a few utility methods.

Code-wise, the API comes with eight new methods and five new static methods. To enable such additions, approximately, 1500 out of 2400 lines of code were changed (as per Open JDK).

2. Instance API Additions

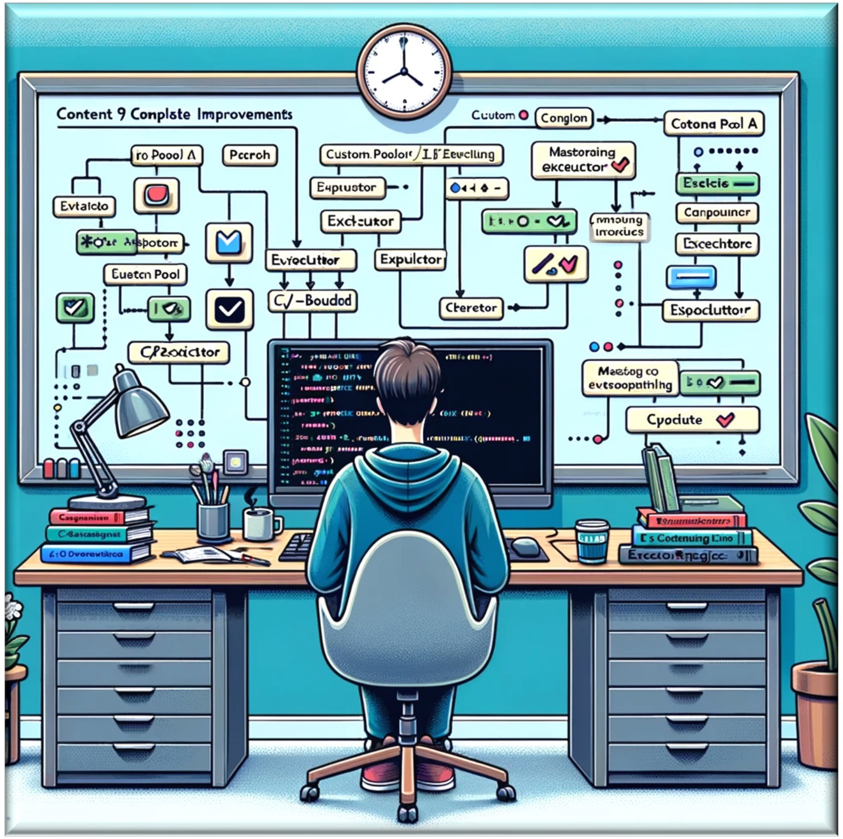
As mentioned, the instance API comes with eight new additions, they are:

1. *Executor defaultExecutor()*
2. *CompletableFuture<U> newIncompleteFuture()*
3. *CompletableFuture<T> copy()*
4. *CompletionStage<T> minimalCompletionStage()*
5. *CompletableFuture<T> completeAsync(Supplier<? extends T> supplier, Executor executor)*
6. CompletableFuture<T> completeAsync(Supplier<? extends T> supplier)
7. *CompletableFuture<T> orTimeout(long timeout, TimeUnit unit)*
8. *CompletableFuture<T> completeOnTimeout(T value, long timeout, TimeUnit unit)*

See www.baeldung.com/java-9-completablefuture

Enhancing Java CompletableFuture Extensibility

- Java 9 enhances the Java 8 completable future framework to better support custom Executor implementations
  - These enhancements include methods that allow for greater flexibility & control over async computation steps



Enhancing Java CompletableFuture Extensibility

- Here are some new methods that enable virtual thread management for custom Executor implementations integrated with CompletableFuture

Methods	Params		
default Executor	()	Executor	Returns default *Executor* used for methods that don't specify an *Executor*
new Incomplete Future	()	Completable Future<T>	Returns a new *CompletableFuture* that will be returned by a *CompletionStage* method
complete Async	Supplier<T>	Completable Future<T>	Complete the *CompletableFuture* asynchronously using the value given by the *Supplier*

See www.baeldung.com/java-9-completablefuture

Enhancing Java CompletableFuture Extensibility

- Here are some new methods that enable virtual thread management for custom Executor implementations integrated with CompletableFuture

Methods	Params		
default Executor	()	Executor	Returns default *Executor* used for methods that don't specify an *Executor*
new Incomplete Future	()	Completable Future<T>	Returns a new *CompletableFuture* that will be returned by a *CompletionStage* method
complete Async	Supplier<T>	Completable Future<T>	Complete the *CompletableFuture* asynchronously using the value given by the *Supplier*

See docs.oracle.com/javase/9/docs/api/java/util/concurrent/CompletableFuture.html#defaultExecutor

Enhancing Java CompletableFuture Extensibility

- Here are some new methods that enable virtual thread management for custom Executor implementations integrated with CompletableFuture

Methods	Params		
default Executor	()	Executor	Returns default *Executor* used for methods that don't specify an *Executor*
new Incomplete Future	()	Completable Future<T>	Returns a new *CompletableFuture* that will be returned by a *CompletionStage* method
complete Async	Supplier<T>	Completable Future<T>	Complete the *CompletableFuture* asynchronously using the value given by the *Supplier*

See docs.oracle.com/javase/9/docs/api/java/util/concurrent/CompletableFuture.html#newIncompleteFuture

Enhancing Java CompletableFuture Extensibility

- Here are some new methods that enable virtual thread management for custom Executor implementations integrated with CompletableFuture

Methods	Params		
default Executor	()	Executor	Returns default *Executor* used for methods that don't specify an *Executor*
new Incomplete Future	()	Completable Future<T>	Returns a new *CompletableFuture* that will be returned by a *CompletionStage* method
complete Async	Supplier<T>	Completable Future<T>	Complete the *CompletableFuture* asynchronously using the value given by the *Supplier*

See docs.oracle.com/javase/9/docs/api/java/util/concurrent/CompletableFuture.html#completeAsync

Extending the Java Completable Future Framework

Extending the Java CompletableFuture Framework

- Customize the completable futures framework to use virtual threads by default

```
class CompletableFutureEx<T> extends CompletableFuture<T> {
  private static Executor sEXEC = Executors
    .newVirtualThreadPerTaskExecutor();

  public Executor defaultExecutor() { return sEXEC; }

  public <T> CompletableFuture<T> newIncompleteFuture()
  { return new CompletableFutureEx<>(); }

  public static <T> CompletableFuture<T>
    supplyAsync(Supplier<T> supplier) {
    return new CompletableFutureEx<T>().completeAsync(supplier);
  } ...
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Loom/ex6

Extending the Java CompletableFuture Framework

- Customize the completable futures framework to use virtual threads by default

```
class CompletableFutureEx<T> extends CompletableFuture<T> {
  private static Executor sEXEC = Executors
    .newVirtualThreadPerTaskExecutor();

  public Executor defaultExecutor() { return sEXEC; }

  public <T> CompletableFuture<T> newIncompleteFuture()
  { return new CompletableFutureEx<>(); }

  public static <T> CompletableFuture<T>
    supplyAsync(Supplier<T> supplier) {
    return new CompletableFutureEx<T>().completeAsync(supplier);
  } ...
```

Customization requires the use of inheritance

Extending the Java CompletableFuture Framework

- Customize the completable futures framework to use virtual threads by default

```
class CompletableFutureEx<T> extends CompletableFuture<T> {
  private static Executor sEXEC = Executors
    .newVirtualThreadPerTaskExecutor();

  public Executor defaultExecutor() { return sEXEC; }

  public <T> CompletableFuture<T> newIncompleteFuture()
  { return new CompletableFutureEx<>(); }

  public static <T> CompletableFuture<T>
    supplyAsync(Supplier<T> supplier) {
    return new CompletableFutureEx<T>().completeAsync(supplier);
  } ...
```

Creates a virtual thread for each async task

There are other ways to do this that we'll explore in the next part of the lesson

Extending the Java CompletableFuture Framework

- Customize the completable futures framework to use virtual threads by default

```
class CompletableFutureEx<T> extends CompletableFuture<T> {
  private static Executor sEXEC = Executors
    .newVirtualThreadPerTaskExecutor();

  public Executor defaultExecutor() { return sEXEC; }

  public <T> CompletableFuture<T> newIncompleteFuture()
  { return new CompletableFutureEx<>(); }

  public static <T> CompletableFuture<T>
    supplyAsync(Supplier<T> supplier) {
    return new CompletableFutureEx<T>().completeAsync(supplier);
  } ...
```

Return the default Executor

Extending the Java CompletableFuture Framework

- Customize the completable futures framework to use virtual threads by default

```
class CompletableFutureEx<T> extends CompletableFuture<T> {
  private static Executor sEXEC = Executors
    .newVirtualThreadPerTaskExecutor();

  public Executor defaultExecutor() { return sEXEC; }

  public <T> CompletableFuture<T> newIncompleteFuture()
  { return new CompletableFutureEx<>(); }

  public static <T> CompletableFuture<T>
    supplyAsync(Supplier<T> supplier) {
    return new CompletableFutureEx<T>().completeAsync(supplier);
  } ...
```

Factory method creates this subclass instance

Extending the Java CompletableFuture Framework

- Customize the completable futures framework to use virtual threads by default

```
class CompletableFutureEx<T> extends CompletableFuture<T> {
  private static Executor sEXEC = Executors
    .newVirtualThreadPerTaskExecutor();

  public Executor defaultExecutor() { return sEXEC; }

  public <T> CompletableFuture<T> newIncompleteFuture()
  { return new CompletableFutureEx<>(); }

  public static <T> CompletableFuture<T>
    supplyAsync(Supplier<T> supplier) {
    return new CompletableFutureEx<T>().completeAsync(supplier);
  } ...
```

Submit supplier to run asynchronously

Extending the Java CompletableFuture Framework

- Customize the completable futures framework to use virtual threads by default

```
class CompletableFutureEx<T> extends CompletableFuture<T> {
  private static Executor sEXEC = Executors
    .newVirtualThreadPerTaskExecutor();

  public Executor defaultExecutor() { return sEXEC; }

  public <T> CompletableFuture<T> newIncompleteFuture()
  { return new CompletableFutureEx<>(); }

  public static <T> CompletableFuture<T>
    supplyAsync(Supplier<T> supplier) {
    return new CompletableFutureEx<T>().completeAsync(supplier);
  } ...
```

Arrange to run supplier in the "default" Executor

End of Enhancing Java Completable Futures: Framework Extensibility (Part 1)