Learning Objectives in this Part of the Lesson

• Recognize key basic & advanced features of the Java 8 completable future framework

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 Java 8 completable future framework provides an async concurrent programming model.

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html
Overview of Completable Futures

- The Java 8 completable future framework provides an async concurrent programming model
- Supports dependent actions that trigger upon completion of async operations

Task 1: Get start page asynchronously

Task 2: Count images on the page asynchronously

Task 3: Count images on all hyperlinked pages asynchronously

Task 4: Combine results to create the total asynchronously

These dependencies can be modeled via a data flow diagram
The Java 8 completable future framework provides an async concurrent programming model.

- Supports dependent actions that trigger upon completion of async operations.

Async operations can be forked, chained, & joined.

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletionStage.html
Overview of Completable Futures

- The Java 8 completable future framework provides an async concurrent programming model.
- Supports dependent actions that trigger upon completion of async operations.
- Async operations can run concurrently in thread pools.

See www.nurkiewicz.com/2013/05/java-8-definitive-guide-to.html
Overview of Completable Futures

- The Java 8 completable future framework provides an async concurrent programming model.
- Supports dependent actions that trigger upon completion of async operations.
- Async operations can run concurrently in thread pools.
  - Either the common fork-join pool or a user-designed pool.
Overview of Completetable Futures

• The completetable future framework overcomes Java future limitations
The completable future framework overcomes Java future limitations

- *Can* be completed explicitly

```java
CompletableFuture<...> future = new CompletableFuture<>();

new Thread () -> {
    ...
    future.complete(...);
}.start();
```

```
... System.out.println(future.join());
```

After complete() is done calls to join() will unblock

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8](http://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8)
Overview of Completetable Futures

- The completetable future framework overcomes Java future limitations
  - *Can* be completed explicitly
  - *Can* be chained together fluently to handle async results efficiently

```java
CompletableFuture.supplyAsync(reduceFraction)
    .thenApply(BigFraction::toMixedString)
    .thenAccept(System.out::println);
```

The action of each “completion stage” is triggered when the future from the previous stage completes asynchronously.

See [en.wikipedia.org/wiki/Fluent_interface](en.wikipedia.org/wiki/Fluent_interface)
Overview of Completable Futures

- The completable future framework overcomes Java future limitations
  - *Can* be completed explicitly
  - *Can* be chained together fluently to handle async results efficiently
  - *Can* be triggered reactively/efficiently as a *collection* of futures w/out undue overhead

```java
CompletableFuture<List<BigFraction>> futureToList = Stream
  .generate(generator)
  .limit(sMAX_FRACTIONS)
  .map(reduceFractions)
  .collect(FuturesCollector.toFutures());

futureToList.thenAccept(printList);

Print out the results after all async fraction reductions have completed
```
Overview of Completable Futures

- The completable future framework overcomes Java future limitations
  - *Can* be completed explicitly
  - *Can* be chained together fluently to handle async results efficiently
  - *Can* be triggered reactively/efficiently as a *collection* of futures w/out undue overhead

```java
CompletableFuture<List<BigFraction>> futureToList = Stream
    .generate(generator)
    .limit(sMAX_FRACTIONS)
    .map(reduceFractions)
    .collect(FuturesCollector.toFutures());

futureToList.thenAccept(printList);
```

Completable futures can also be combined with Java 8 streams
Overview of Completable Futures

- Some completable future features are basic

```java
CompletableFuture

- CompletableFuture()
- cancel(boolean): boolean
- isCancelled(): boolean
- isDone(): boolean
- get()
- get(long, TimeUnit)
- join()
- complete(T): boolean

- supplyAsync(Supplier<U>, Executor): CompletableFuture<U>
- supplyAsync(Supplier<U>, Executor): CompletableFuture<U>
- runAsync(Runnable): CompletableFuture<Void>
- runAsync(Runnable, Executor): CompletableFuture<Void>
- completedFuture(U): CompletableFuture<U>
- thenApply(Function<T>): CompletableFuture<U>
- thenAccept(Consumer<? super T>): CompletableFuture<Void>
- thenCombine(CompletionStage<? extends U>, BiFunction<?>): CompletableFuture<V>
- thenCompose(Function<?>): CompletableFuture<U>
- whenComplete(BiConsumer<?>, Throwable): CompletableFuture<T>
- allOf(CompletableFuture[]): CompletableFuture<Void>
- anyOf(CompletableFuture[]): CompletableFuture<Object>
```
Overview of Completable Futures

• Some completable future features are basic
  • e.g., the Java Future API + a few simple enhancements

Only slightly better than the conventional Future interface
Overview of Completable Futures

- Other completable future features are more advanced.
Overview of Completable Futures

- Other completable future features are more advanced
  - Factory methods
    - Initiate async two-way or one-way functionality
Overview of Completable Futures

- Other completable future features are more advanced
  - Factory methods
  - Chaining methods
    - Serve as completion stage for async result processing & composition

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletionStage.html
Overview of Completable Futures

- Other completable future features are more advanced
  - Factory methods
  - Chaining methods
  - "Arbitrary-arity" methods that process futures in bulk
  - Combine multiple futures into a single future

See en.wikipedia.org/wiki/Arity
End of Overview of Java 8
CompletableFuture Futures
(Part 1)