Learning Objectives in this Part of the Lesson

• Understand the basic completable futures features

```java
import java.lang.Object;
import java.util.concurrent.CompletableFuture;

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

All Implemented Interfaces:
CompletableFuture<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:
```
Basic CompletableFuture Features
Basic CompletableFuture Features

• Basic CompletableFuture features

See github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex8
Basic CompletableFuture Features

- Basic CompletableFuture features
- Support the Future API

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

- Basic CompletableFuture features
- Support the Future API
- Can (time- ) block & poll

String f1 = "62675744/15668936";
String f2 = "609136/913704";

ForkJoinTask<BigFraction> f =
commonPool().submit(() -> {
    BigFraction bf1 =
        new BigFraction(f1);
    BigFraction bf2 =
        new BigFraction(f2);
    return bf1.multiply(bf2);
});
...

BigFraction result = f.get();
// f.get(10, MILLISECONDS);
// f.get(0, 0);

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

- Basic CompletableFuture features
- Support the Future API
  - Can (time-) block & poll
  - Can be cancelled & tested if cancelled/done

```java
String f1 = "62675744/15668936";
String f2 = "609136/913704";

ForkJoinTask<BigFraction> f =
    commonPool().submit(() -> {
        BigFraction bf1 =
            new BigFraction(f1);
        BigFraction bf2 =
            new BigFraction(f2);
        return bf1.multiply(bf2);
    });

...  
if (!f.isDone()
        || !f.isCancelled())
    f.cancel();
```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html)
• Basic CompletableFuture features
• Support the Future API
  • Can (time-) block & poll
  • Can be cancelled & tested if cancelled/done
  • cancel() doesn’t interrupt the computation by default.

String f1 = "62675744/15668936";
String f2 = "609136/913704";

ForkJoinTask<BigFraction> f =
  commonPool().submit(() -> {
    BigFraction bf1 =
      new BigFraction(f1);
    BigFraction bf2 =
      new BigFraction(f2);
    return bf1.multiply(bf2);
  });
...
if (!f.isDone() || !f.isCancelled())
  f.cancel();

See www.nurkiewicz.com/2015/03/completablefuture-cant-be-interrupted.html
Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
  - Define a join() method

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#join](docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#join)
Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
  - Define a join() method
    - Behaves like get() without using checked exceptions

```java
futures.stream()
        .map(CompletableFuture::join)
        .collect(toList())
```

CompletableFuture::join can be used as a method reference in a Java stream.
Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
  - Define a join() method
    - Behaves like get() \textit{without} using checked exceptions

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

Mixing checked exceptions & Java streams is ugly..
Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
  - Define a `join()` method
    - Behaves like `get()` *without* using checked exceptions
  - There is no timed version of `join()`
Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
  - Define a join() method
  - Can be completed explicitly

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#complete
Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
  - Define a join() method
  - Can be completed explicitly
    - i.e., sets result returned by get()/join() to a given value

```java
CompletableFuture<...> future = new CompletableFuture<>();
new Thread (() -> {
    ...
    future.complete(...);
}).start();
...
System.out.println(future.join());
```
Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
  - Define a join() method
- Can be completed explicitly
  - i.e., sets result returned by get()/join() to a given value

```java
CompletableFuture<...> future = new CompletableFuture<>();
new Thread () -> {
  ...
  future.complete(...);
}).start();
...
System.out.println(future.join());
```

Create an incomplete future

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#CompletableFuture](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#CompletableFuture)
Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
  - Define a join() method
  - Can be completed explicitly
    - i.e., sets result returned by get()/join() to a given value

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

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

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

Create/start a new thread that runs concurrently with the main thread

See [docs.oracle.com/javase/8/docs/api/java/lang/Thread.html](docs.oracle.com/javase/8/docs/api/java/lang/Thread.html)
Basic CompletableFuture Features

- Basic CompletableFuture features
  - Support the Future API
  - Define a join() method
- Can be completed explicitly
  - i.e., sets result returned by get()/join() to a given value

```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 docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#complete
Basic CompletableFuture Features

- Basic CompletableFuture features
- Support the Future API
- Define a join() method
- Can be completed explicitly
  - i.e., sets result returned by get()/join() to a given value

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

final CompletableFuture<Long> zero 
   = CompletableFuture 
     .completedFuture(0L);

new Thread () -> {
   ... 
   future.complete(zero.join());
}

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

A completable future can be initialized to a value/constant

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#completedFuture](docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#completedFuture)
End of Basic Java
CompletableFuture Features