Overview of Basic Java

CompletableFuture Features

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 features in the Java completable futures framework
- e.g., get(), isDone(), isCanceled(), cancel(), join(), & complete()

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

- Basic CompletableFuture features

We showcase these basic features in case study ex8

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8](https://github.com/douglascraigschmidt/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, e.g.
  - Can block, time-wait, & poll

```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);
    });
...

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](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html)
Basic CompletableFuture Features

- Basic CompletableFuture features
- Support the Future API, e.g.
  - Can block, time-wait, & 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

- Basic CompletableFuture features
- Support the Future API, e.g.
  - Can block, time-wait, & poll
- Can be cancelled & tested if cancelled/done
- cancel() doesn’t interrupt the computation by default..

```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();
```

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
    - Blocks awaiting results

See www.educative.io/answers/what-is-completablefuturejoin-in-java
Basic CompletableFuture Features

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

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


CompletableFuture::join can be used as a method reference in a Java stream
Mixing checked exceptions & Java streams is ugly..

Basic CompletableFuture Features

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

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

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

```java
futures
    .stream()
    .map(f -> rethrowSupplier(f::get).get())
    .collect(toList())
```

Exception laundering is also an option, but may be overkill compared with join()

See [stackoverflow.com/a/27644392/3312330](https://stackoverflow.com/a/27644392/3312330)
Basic CompletableFuture Features

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

See [tedblob.com/completablefuture-join-vs-get](http://tedblob.com/completablefuture-join-vs-get)
Basic CompletableFuture Features

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

you complete me

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 future result to a given value, which is then returned by get()/join()

```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 future result to a given value, which is then returned by get()/join()

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

... Create an incomplete future

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

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

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#CompletableFuture](http://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 future result to a given value, which is then returned by get()/join()

```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 calling thread

See 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 future result to a given value, which is then returned by get()/join()

```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 future result to a given value, which is then returned by get()/join()

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

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

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

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

A completable future can be initialized to a value/constant
End of Overview of Basic Java CompletableFuture Features