Understand Advanced Java CompletableFuture

Features: Handling Runtime Exceptions (Part 2)

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 advanced features of completable futures, e.g.
  - Factory methods initiate async computations
- Completion stage methods chain together actions to perform async result processing & composition
  - Method grouping
  - Single stage methods
  - Two stage methods (and)
  - Two stage methods (or)
- Apply these methods
- Handle runtime exceptions (Examples)
Examples of Handling Exceptions in Completion Stages
Examples of Handling Exceptions in Completion Stages

- This example shows three ways to handle exceptions with completable futures

```java
CompletableFuture
    .supplyAsync(() ->
        BigFraction.valueOf(100, denominator))
```

... An exception will occur if denominator param is 0!

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8)
Examples of Handling Exceptions in Completion Stages

- This example shows three ways to handle exceptions with completable futures

CompletableFuture
  .supplyAsync(() ->
     BigFraction.valueOf(100, denominator))

...
Examples of Handling Exceptions in Completion Stages

- Using the handle() method to handle exceptional or normal completions

```java
CompletableFuture
  .supplyAsync(() ->
    BigFraction.valueOf(100, denominator))
  .handle((fraction, ex) -> {
    if (fraction == null)
      return BigFraction.ZERO;
    else
      return fraction.multiply(sBigReducedFraction);
  })
  .thenAccept(fraction ->
    System.out.println(fraction.toMixedString()));
```

Handle outcome of the previous stage (always called, regardless of whether an exception is thrown)

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#handle
Examples of Handling Exceptions in Completion Stages

• Using the `handle()` method to handle exceptional or normal completions

```java
CompletableFuture.supplyAsync(() ->
    BigFraction.valueOf(100, denominator))
    .handle(((fraction, ex) -> {
        if (fraction == null)
            return BigFraction.ZERO;
        else
            return fraction.multiply(sBigReducedFraction);
    }))
    .thenAccept(fraction ->
        System.out.println(fraction.toMixedString()));
```

These values are mutually exclusive
Examples of Handling Exceptions in Completion Stages

- Using the handle() method to handle exceptional or normal completions

```java
CompletableFuture.supplyAsync(() ->
    BigFraction.valueOf(100, denominator))
    .handle((fraction, ex) -> {
        if (fraction == null)
            return BigFraction.ZERO;
        else
            return fraction.multiply(sBigReducedFraction);
    })
    .thenAccept(fraction ->
        System.out.println(fraction.toMixedString()));
```

See [en.wikipedia.org/wiki/Error_hiding](en.wikipedia.org/wiki/Error_hiding)
Examples of Handling Exceptions in Completion Stages

• Using the handle() method to handle exceptional or normal completions

CompletableFuture
  .supplyAsync(() ->
      BigFraction.valueOf(100, denominator))

  .handle((fraction, ex) -> {
      if (fraction == null)
          return BigFraction.ZERO;
      else
          return fraction.multiply(sBigReducedFraction);
  })

  .thenAccept(fraction ->
      System.out.println(fraction.toMixedString()));
Examples of Handling Exceptions in Completion Stages

- Using the handle() method to handle exceptional or normal completions

```java
CompletableFuture.supplyAsync(() ->
    BigFraction.valueOf(100, denominator))

    .handle((fraction, ex) -> {
        if (fraction == null)
            return BigFraction.ZERO;
        else
            return fraction.multiply(sBigReducedFraction);
    })

    .thenAccept(fraction ->
        System.out.println(fraction.toMixedString()));
```

*handle() must return a value (& can thus change the return value)*
Examples of Handling Exceptions in Completion Stages

- Using the handle() method to handle exceptional or normal completions

CompletableFuture
  .supplyAsync(() ->
    BigFraction.valueOf(100, denominator))
  .handle((fraction, ex) -> {
    if (fraction == null)
      return BigFraction.ZERO;
    else
      return fraction.multiply(sBigReducedFraction);
  })
  .thenAccept(fraction ->
    System.out.println(fraction.toMixedString()));

Display result as a mixed fraction
Examples of Handling Exceptions in Completion Stages

- Using the exceptionally() method to handle exceptional or normal completions

```java
CompletableFuture
    .supplyAsync(() ->
        BigFraction.valueOf(100, denominator))

    .thenApply(fraction ->
        fraction.multiply(sBigReducedFraction))

    .exceptionally(ex -> BigFraction.ZERO)

    .thenAccept(fraction ->
        System.out.println(fraction.toMixedString()));
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#exceptionally
Examples of Handling Exceptions in Completion Stages

- Using the `exceptionally()` method to handle exceptional or normal completions

```java
CompletableFuture.supplyAsync(() ->
        BigFraction.valueOf(100, denominator))
      .thenApply(fraction ->
        fraction.multiply(sBigReducedFraction))
      .exceptionally(ex -> BigFraction.ZERO)
      .thenAccept(fraction ->
        System.out.println(fraction.toMixedString()));
```

An exception occurs if denominator is 0!
Examples of Handling Exceptions in Completion Stages

- Using the exceptionally() method to handle exceptional or normal completions

```java
CompletableFuture.supplyAsync(() -> BigFraction.valueOf(100, denominator))
  .thenApply(fraction -> fraction.multiply(sBigReducedFraction))
  .exceptionally(ex -> BigFraction.ZERO)
  .thenAccept(fraction -> System.out.println(fraction.toMixedString()));
```

Handle case where denominator != 0 (skipped if exception is thrown)
Examples of Handling Exceptions in Completion Stages

- Using the `exceptionally()` method to handle exceptional or normal completions

```java
CompletableFuture
  .supplyAsync(() ->
    BigFraction.valueOf(100, denominator))

  .thenApply(fraction ->
    fraction.multiply(sBigReducedFraction))

  .exceptionally(ex -> BigFraction.ZERO)

  .thenAccept(fraction ->
    System.out.println(fraction.toMixedString()));
```

*Handle case where denominator == 0 & exception is thrown (otherwise skipped)*

`exceptionally()` is akin to `catch()` in a Java try/catch block, i.e., control xfers to it
Examples of Handling Exceptions in Completion Stages

- Using the `exceptionally()` method to handle exceptional or normal completions

```java
CompletableFuture.supplyAsync(() ->
    BigFraction.valueOf(100, denominator))
    .thenApply(fraction ->
        fraction.multiply(sBigReducedFraction))
    .exceptionally(ex -> BigFraction.ZERO)
    .thenAccept(fraction ->
        System.out.println(fraction.toMixedString()));
```

Convert ("swallow") an exception to a 0 result.

See en.wikipedia.org/wiki/Error_hiding
Examples of Handling Exceptions in Completion Stages

- Using the exceptionally() method to handle exceptional or normal completions

```java
CompletableFuture
  .supplyAsync(() ->
    BigFraction.valueOf(100, denominator))

  .thenApply(fraction ->
    fraction.multiply(sBigReducedFraction))

  .exceptionally(ex -> BigFraction.ZERO)

  .thenAccept(fraction ->
    System.out.println(fraction.toMixedString()));
```

Display result as a mixed fraction
Examples of Handling Exceptions in Completion Stages

- Using the `whenComplete()` method to perform a exceptional or normal action

```java
CompletableFuture.supplyAsync(() ->
    BigFraction.valueOf(100, denominator))
    .thenApply(fraction ->
        fraction.multiply(sBigReducedFraction))
    .whenComplete((fraction, ex) -> {
        if (fraction != null)
            System.out.println(fraction.toMixedString());
        else
            System.out.println(ex.getMessage());
    });
```

Called under both normal & exception conditions

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#whenComplete
Examples of Handling Exceptions in Completion Stages

• Using the whenComplete() method to perform a exceptional or normal action

CompletableFuture
  .supplyAsync(() ->
    BigFraction.valueOf(100, denominator))
  .thenApply(fraction ->
    fraction.multiply(sBigReducedFraction))
  .whenComplete((fraction, ex) -> {
    if (fraction != null)
      System.out.println(fraction.toMixedString());
    else
      System.out.println(ex.getMessage());
  });
Examples of Handling Exceptions in Completion Stages

- Using the whenComplete() method to perform a exceptional or normal action

```java
CompletableFuture.supplyAsync(() ->
    BigFraction.valueOf(100, denominator))
    .thenApply(fraction ->
        fraction.multiply(sBigReducedFraction))
    .whenComplete((fraction, ex) -> {
        if (fraction != null)
            System.out.println(fraction.toMixedString());
        else
            System.out.println(ex.getMessage());
    });
```

Handle the normal case
Examples of Handling Exceptions in Completion Stages

- Using the `whenComplete()` method to perform a exceptional or normal action

```java
CompletableFuture
  .supplyAsync(() ->
    BigFraction.valueOf(100, denominator))
  .thenApply(fraction ->
    fraction.multiply(sBigReducedFraction))
  .whenComplete((fraction, ex) -> {
    if (fraction != null)
      System.out.println(fraction.toMixedString());
    else // ex != null
      System.out.println(ex.getMessage());
  });
```

Handle the exceptional case
Examples of Handling Exceptions in Completion Stages

- Using the `whenComplete()` method to perform a exceptional or normal action

```java
CompletableFuture.supplyAsync(() ->
    BigFraction.valueOf(100, denominator))
  .thenApply(fraction ->
    fraction.multiply(sBigReducedFraction))
  .whenComplete((fraction, ex) ->
    if (fraction != null) System.out.println(fraction.toMixedString());
    else // ex != null
      System.out.println(ex.getMessage());
  );
```

`whenComplete()` is like Java Streams.peek(), i.e., it has a side-effect, doesn’t change the return value, & doesn’t swallow the exception

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#peek