Motivating the Need for Java 8 Completable Futures (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

- Motivate the need for Java futures by understanding the pros & cons of synchrony & asynchrony
- Know how Java futures provide the foundation for completable futures in Java 8

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

- Java 1.5 (JDK 5) added support for async calls via Java futures

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

- Java 1.5 (JDK 5) added support for async calls via Java futures

- Async calls return a future & continue running the computation in the background
Overview of Java Futures

- `ExecutorService.submit()` is an example of an async call in Java

```
Callable<BigFraction> task = () -> {
    BigFraction bf1 = new BigFraction(f1);
    BigFraction bf2 = new BigFraction(f2);
    return bf1.multiply(bf2);
};

Future<BigFraction> future = ForkJoinPool.common().submit(task);
```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#submit](docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#submit)
Overview of Java Futures

- A future is a proxy that represents the result(s) of an async computation.

Result obtained only after the computation completes

Asynchronous computation

See en.wikipedia.org/wiki/Futures_and_promises
Overview of Java Futures

- A future is a proxy that represents the result(s) of an async computation.

```java
Result get_result ()
begin
  ## Suspend calling thread until result is available.
  if (result == NULL) then
    thread.wait ();
  return result;
end
```

Table tent #’s are a human-known-use of futures!

Overview of Java Futures

- A future is a proxy that represents the result(s) of an async computation.

```java
Result get_result ()
begin
  ## Suspend calling thread until result is available.
  if (result == NULL) then
    thread.wait ();
  return result;
end
```

Table tent #’s are a human-known-use of futures!

E.g., McDonald’s vs Wendy’s model of preparing fast foot
Overview of Java Futures

• When the async computation completes the future is triggered & the result is available

See www.nurkiewicz.com/2013/02/javauutilconcurrentfuture-basics.html
Overview of Java Futures

- When the async computation completes the future is triggered & the result is available
- `get()` can block or (timed-)poll

BigFraction result = future.get();

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html#get
Overview of Java Futures

- When the async computation completes the future is triggered & the result is available
  - `get()` can block or (timed-)poll
  - Results can occur in a different order than the original calls were made

```
searchForWord1
  -------
future1

searchForWord2
  -------
future2

searchForWord3
  -------
future3

future result1
future result2
future result3
```

OUT OF ORDER
Programming with Java Futures
Example of using Java Future via a Callable & the common fork-join pool

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

Callable<BigFraction> call = () -> {
    BigFraction bf1 =
        new BigFraction(f1);
    BigFraction bf2 =
        new BigFraction(f2);
    return bf1.multiply(bf2); }

Future<BigFraction> future =
    commonPool().submit(call);
...
BigFraction result =
    future.get();
```

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8)
Example of using Java Future via a Callable & the common fork-join pool

Callable is a two-way task that returns a result via a single method with no arguments

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

Callable<BigFraction> call = () -> {
    BigFraction bf1 =
        new BigFraction(f1);
    BigFraction bf2 =
        new BigFraction(f2);
    return bf1.multiply(bf2); }

Future<BigFraction> future =
    commonPool().submit(call);
...
BigFraction result =
    future.get();

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Callable.html
Example of using Java Future via a Callable & the common fork-join pool

Java 8 enables the initialization of a Callable via a lambda expression

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

Callable<BigFraction> call = () -> {
    BigFraction bf1 =
        new BigFraction(f1);
    BigFraction bf2 =
        new BigFraction(f2);
    return bf1.multiply(bf2); }

Future<BigFraction> future =
    commonPool().submit(call);
...

BigFraction result =
    future.get();

See "Overview of Java 8 Lambda Expressions & Method References"
• Example of using Java Future via a Callable & the common fork-join pool

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

Callable<BigFraction> call = () -> {
    BigFraction bf1 =
        new BigFraction(f1);
    BigFraction bf2 =
        new BigFraction(f2);
    return bf1.multiply(bf2);
};

Future<BigFraction> future =
    commonPool().submit(call);
...

BigFraction result =
    future.get();
```

Can pass values to a callable via effectively final variables

See javarevisited.blogspot.com/2015/03/what-is-effectively-final-variable-of.html
Programming with Java Futures

- Example of using Java Future via a Callable & the common fork-join pool

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

Callable<BigFraction> call = () -> {
    BigFraction bf1 =
        new BigFraction(f1);
    BigFraction bf2 =
        new BigFraction(f2);
    return bf1.multiply(bf2);
};

Future<BigFraction> future =
    commonPool().submit(call);
...

BigFraction result =
    future.get();
```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html)
• Example of using Java Future via a Callable & the common fork-join pool

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

Callable<BigFraction> call = () -> {
    BigFraction bf1 =
        new BigFraction(f1);
    BigFraction bf2 =
        new BigFraction(f2);
    return bf1.multiply(bf2);
};

Future<BigFraction> future =
    commonPool().submit(call);
...

BigFraction result =
    future.get();

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html)
Example of using Java Future via a Callable & the common fork-join pool

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

Callable<BigFraction> call = () -> {
    BigFraction bf1 =
        new BigFraction(f1);
    BigFraction bf2 =
        new BigFraction(f2);
    return bf1.multiply(bf2); }

Future<BigFraction> future =
    commonPool().submit(call);
...

BigFraction result =
    future.get();
```
Programming with Java Futures

- Example of using Java Future via a Callable & the common fork-join pool

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

Callable<BigFraction> call = () -> {
    BigFraction bf1 = 
        new BigFraction(f1);
    BigFraction bf2 = 
        new BigFraction(f2);
    return bf1.multiply(bf2);
};

Future<BigFraction> future =
    commonPool().submit(call);
...
BigFraction result =
    future.get();
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html#get
Example of using Java Future via a Callable & the common fork-join pool

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

Callable<BigFraction> call = () -> {
    BigFraction bf1 =
        new BigFraction(f1);
    BigFraction bf2 =
        new BigFraction(f2);
    return bf1.multiply(bf2);
};

Future<BigFraction> future =
    commonPool().submit(call);
...
BigFraction result =
    future.get(n, SECONDS);

get() can also perform polling & timed-blocks
End of Motivating the Need for Java 8 Completable Futures (Part 2)