Asynchronous Programming in Java SE 17

Accessing Data Asynchronously on the Web

José Paumard

PHD, Java Champion, JavaOne RockStar

@JosePaumard https://github.com/JosePaumard

Understanding asynchronous programming with the CompletionStage API

Version Check

This is a Java 17 course

Many things are compatible with Java 8

The IDE used is Intellij IDEA 2021.2.1

Ultimate or Community version

Any other Java 17 compatible IDE will work

Asynchronous pipelines to process data

To improve performances for I/O operations

Input / Output may be:

- accessing data from a disk
- over a network
- from a database

This is a Java course:

Basic knowledge of the language

How to write simple lambda expressions

How to create and run a simple application

Elements of concurrent programming:

- threads and executor services

- future objects

Applying Concurrency and Multi-threading
to Common Java Patterns

Agenda of the course

Getting data asynchronously from the Web

How to trigger the processing of this data

How to organize your application to be asynchronous

Controlling which thread is executing your tasks

How to handle errors, reporting them, or recovering from them

Synchronous vs. Asynchronous

Synchronous

You need to wait for a task to complete to continue to work.

```
HTTPClient client = ...;
String response =
          client.get("http://www.mydata.com/data");
```

You need to wait for the server to send you the response to process it

It may be several 100ms

In the meantime your CPU is doing nothing

Asynchronous

The code you write will be executed at some point in the future.

```
List<String> strings = ...;
strings.forEach(s -> System.out.println(s));
```

Printing the elements of the list is done between 0 and N times

At some point in the future

Asynchronous and synchronous are not related to concurrent programming

Asynchronous programming may rely on concurrency, but not always

Blocking vs. Non-Blocking

Synchronous = Blocking

A synchronous code is always blocking. It will slow down your application if it blocks it for a long time.

```
HTTPClient client = ...;
String response =
          client.get("http://www.mydata.com/data");
```

The call to get() is a blocking call

While the response is sent the current thread is ‘blocked’

This is a bad situation: getting a response over the web takes several 100ms

```
List<String> strings = ...;
strings.forEach(s -> System.out.println(s));
```

The call to forEach() is synchronous, your code is blocking there

The call to println() is also synchronous, but does not take a lot of time

This code is fine, it does not block your application

✓

Asynchronous programming may be used to avoid blocking calls

In that case it will make your application faster

Concurrency

Asynchronous + Concurrent

Running a blocking code in another thread is a way to avoid blocking the main thread of your application.

How can you get the result from this other thread?

```
ExecutorService service = ...;
HTTPClient client = ...;
Future<String> future =
    service.submit(() ->
                client.get("http://www.mydata.com/data"));
// do some other stuff
String response = future.get();
```

The call to get() is still a blocking call, but blocks another thread

Your application thread is free to do something else

You can get the response through this future object

By calling future.get(), which is a blocking call

✓

Asynchronous programming may be used to avoid blocking calls

In that case it will make your application faster

Demo

Let us write some code!

And see how fetching data asynchronously can speed up your application

Module Wrap Up

What did you learn?

1) Synchronous vs. asynchronous
   - The difference with concurrent programming
   - Avoid blocking your main thread

2) Performing blocking calls concurrently
   - To avoid blocking your main thread

How can you get your result without blocking your main thread?

Up Next: Triggering a Task on the Outcome of Another Task