(/)

Last modified: September 27, 2019

by baeldung (https://www.baeldung.com/author/baeldung/)

Java (https://www.baeldung.com/category/java/) +

Java Collections (https://www.baeldung.com/category/java/java-collections/)

Java Streams (https://www.baeldung.com/tag/java-streams/)

1. Overview

In this tutorial, we'll see how to create null-safe streams from Java collections.

To start with, some familiarity with Java 8's Method References, Lambda Expressions, *Optional* and Stream API

If you are unfamiliar with any of these topics, kindly take a look at our previous articles first: New Features in Java 8 (https://www.baeldung.com/java-8-new-features), Guide To Java 8 Optional (https://www.baeldung.com/java-optional) and Introduction to Java 8 Streams (https://www.baeldung.com/java-8-streams-introduction).

2. Maven Dependency

Before we begin, there's one Maven dependency that we're going to need for certain scenarios:

```
</dependency>
```

The *commons-collections4 (https://search.maven.org/classic/#search%7Cgav%7C1%7Cg%3A%22org.apache.commons%22%20AND%20a%3A%22commons-collections4%22)* library can be downloaded from Maven Central.

3. Creating Streams from Collections

The basic approach to creating a *Stream* (https://www.baeldung.com/java-8-streams-introduction) from any type of *Collection* is to call the *stream()* or *parallelStream()* methods on the collection depending on the type of stream that is required:

```
Collection<String> collection = Arrays.asList("a", "b", "c");
Stream<String> streamOfCollection = collection.stream();
```

Our collection will most likely have an external source at some point, we'll probably end up with a method similar to the one below when creating streams from collections:

```
public Stream<String> collectionAsStream(Collection<String> collection) {
    return collection.stream();
}
```

This is can cause some problems. When the provided collection points to a *null* reference, the code will throw

4. Making Created Collection Streams Null-Safe

4.1. Add Checks to Prevent *Null* Dereferences

To prevent unintended *null* pointer exceptions, **we can opt to add checks to prevent *null* references** when creating streams from collections:

```
Stream<String> collectionAsStream(Collection<String> collection) {
    return collection == null
      ? Stream.empty()
      : collection.stream();
}
```

This method, however, has a couple of issues.

First, the *null* check gets in the way of the business logic decreasing the overall readability of the program.

Second, the use of *null* to represent the absence of a value is considered a wrong approach post-Java SE 8: There is a better way to model the absence and presence of a value.

It's important to keep on mind that an empty *Collection* isn't the same as a *null Collection*. While the first one is indicating that our query doesn't have results or elements to show, the second one is suggesting that a kind of error just happened during the process.

4.2. Use *emptyIfNull* Method from *CollectionUtils* Library

We can opt to use Apache Commons' *CollectionUtils* (https://commons.apache.org/proper/commons-collections/apidocs/org/apache/commons/collections4/CollectionUtils.html) library to make sure our stream is *null* safe. This library provides an *emptyIfNull* method which returns an immutable empty collection given a *null* collection as an argument, or the collection itself otherwise:

```
1 public Stream<String> collectionAsStream(Collection<String> collection) {
2     return emptyIfNull(collection).stream();
3 }
```

This is a very simple strategy to adopt. However, it depends on an external library. If a software development policy restricts the use of such a library, then this solution is rendered *null* and void.

4.3. Use Java 8's *Optional*

Using *Optional* can be arguably considered as the best overall strategy to create a null-safe collection from a stream.

Let's see how we can use it followed by a quick discussion below:

```
public Stream<String> collectionToStream(Collection<String> collection) {
    return Optional.ofNullable(collection)
      .map(Collection::stream)
      .orElseGet(Stream::empty);
}
```

- ***Optional.ofNullable(collection)*** creates an *Optional* object from the passed-in collection. An empty *Optional* object is created if the collection is *null.*
- ***map(Collection::stream)*** extracts the value contained in the *Optional* object as an argument to the *map* method (*Collection.stream()*)
- ***orElseGet(Stream::empty)*** returns the fallback value in the event that the *Optional* object is empty, i.e the passed-in collection is *null*.

As a result, we proactively protect our code against unintended *null* pointer exceptions.

4.4. Use Java 9's *Stream OfNullable*

As usual, the full source code that accompanies the article is available over on GitHub (https://github.com/eugenp/tutorials/tree/master/core-java-modules/core-java-collections-2).

Comments are closed on this article!

Examining our previous ternary example in section 4.1. and considering the possibility of some elements could be *null* instead of the *Collection*, we have at our disposal the *ofNullable* method in the *Stream* class.

We can transform the above sample to:

```
1  Stream<String> collectionAsStream(Collection<String> collection) {
```

5. Conclusion

In this article, we briefly revisited how to create a stream from a given collection. We then proceeded to explore the three key strategies for making sure the created stream is null-safe when created from a collection.

Finally, we pointed out the weakness of using each strategy where relevant.

CATEGORIES

SPRING (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
REST (HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SECURITY (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/)
HTTP CLIENT-SIDE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIES

JAVA “BACK TO BASICS” TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)

ABOUT

ABOUT BAELDUNG (/ABOUT)