(/)

Method References in Java

Last modified: February 10, 2019

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

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

Core Java (https://www.baeldung.com/tag/core-java/)

1. Overview

One of the most welcome changes in Java 8 was the introduction of lambda expressions (https://www.baeldung.com/java-8-lambda-expressions-tips), as these allow us to forego anonymous classes, greatly reducing boilerplate code and improving readability.

Method references are a special type of lambda expressions. They're often used to create simple lambda expressions by referencing existing methods.

There are four kinds of method references:

- Static methods
- Instance methods of particular objects
- Instance methods of an arbitrary object of a particular type
- Constructor

In this tutorial, we'll explore method references in Java.

2. Reference to a Static Method

We'll begin with a very simple example, capitalizing and printing a list of *Strings*:

```
List<String> messages = Arrays.asList("hello", "baeldung", "readers!");
```

We can achieve this by leveraging a simple lambda expression calling the *StringUtils.capitalize()* (https://commons.apache.org/proper/commons-lang/apidocs/org/apache/commons/lang3/StringUtils.html#capitalize-java.lang.String-) method directly:

```
messages.forEach(word -> StringUtils.capitalize(word));
```

Or, we can use a method reference to simply refer to the *capitalize* static method:

```
messages.forEach(StringUtils::capitalize);
```

Notice that method references always utilize the *::* operator.

3. Reference to an Instance Method of a Particular Object

To demonstrate this type of method reference, let's consider two classes:

```
public class Bicycle {

    private String brand;
    private Integer frameSize;
    // standard constructor, getters and setters
}

public class BicycleComparator implements Comparator {

    @Override
    public int compare(Bicycle a, Bicycle b) {
        return a.getFrameSize().compareTo(b.getFrameSize());
    }

}
```

And, let's create a *BicycleComparator* object to compare bicycle frame sizes:

```
BicycleComparator bikeFrameSizeComparator = new BicycleComparator();
```

We could use a lambda expression to sort bicycles by frame size, but we'd need to specify two bikes for comparison:

```
createBicyclesList().stream()
  .sorted((a, b) -> bikeFrameSizeComparator.compare(a, b));
```

Instead, we can use a method reference to have the compiler handle parameter passing for us:

```
createBicyclesList().stream()
  .sorted(bikeFrameSizeComparator::compare);
```

The method reference is much cleaner and more readable, as our intention is clearly shown by the code.

4. Reference to an Instance Method of an Arbitrary Object of a Particular Type

This type of method reference is similar to the previous example, but without having to create a custom object to perform the comparison.

Let's create an *Integer* list that we want to sort:

```
List<Integer> numbers = Arrays.asList(5, 3, 50, 24, 40, 2, 9, 18);
```

If we use a classic lambda expression, both parameters need to be explicitly passed, while using a method reference is much more straightforward:

```
numbers.stream()
  .sorted((a, b) -> Integer.compare(a, b));
numbers.stream()
  .sorted(Integer::compare);
```

Even though it's still a one-liner, the method reference is much easier to read and understand.

5. Reference to a Constructor

We can reference a constructor in the same way that we referenced a static method in our first example. The only difference is that we'll use the *new* keyword.

Let's create a *Bicycle* array out of a *String* list with different brands:

```
List<String> bikeBrands = Arrays.asList("Giant", "Scott", "Trek", "GT");
```

First, we'll add a new constructor to our *Bicycle* class:

```
public Bicycle(String brand) {
    this.brand = brand;
    this.frameSize = 0;
}
```

Next, we'll use our new constructor from a method reference and make a *Bicycle* array from the original *String* list:

```
bikeBrands.stream()
  .map(Bicycle::new)
  .toArray(Bicycle[]::new);
```

Notice how we called both *Bicycle* and *Array* constructors using a method reference, giving our code a much more concise and clear appearance.

6. Additional Examples and Limitations

As we've seen so far, method references are a great way to make our code and intentions very clear and readable. However, we can't use them to replace all kinds of lambda expressions since they have some limitations.

Their main limitation is a result of what's also their biggest strength: **the output from the previous expression needs to match the input parameters of the referenced method signature**.

Let's see an example of this limitation:

```
createBicyclesList().forEach(b -> System.out.printf(
  "Bike brand is '%s' and frame size is '%d'%n",
  b.getBrand(),
  b.getFrameSize()));
```

This simple case can't be expressed with a method reference, because the *printf* method requires 3 parameters in our case, and using *createBicyclesList().forEach()* would only allow the method reference to infer one parameter (the *Bicycle* object).

Finally, let's explore how to create a no-operation function that can be referenced from a lambda expression.

In this case, we'll want to use a lambda expression without using its parameters.

First, let's create the *doNothingAtAll* method:

```
private static <T> void doNothingAtAll(Object... o) {
}
```

As it is a varargs (https://www.baeldung.com/java-varargs) method, it will work in any lambda expression, no matter the referenced object or number of parameters inferred.

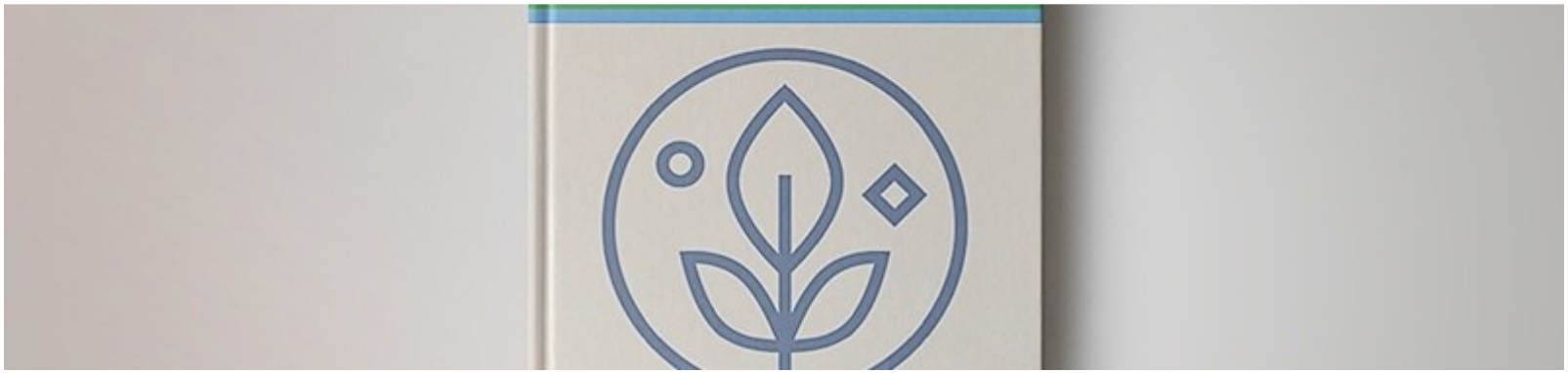
Now, let's see it in action:

```
createBicyclesList()
  .forEach((o) -> MethodReferenceExamples.doNothingAtAll(o));
```

7. Conclusion

In this quick tutorial, we learned what method references are in Java and how to use them to replace lambda expressions, thereby improving readability and clarifying the programmer's intent.

All code presented in this article is available over on GitHub (https://github.com/eugenp/tutorials/tree/master/core-java-8/src/test/java/com/baeldung/java8/lambda/methodreference).



Guest

rswrc

println isn't a static method …. https://docs.oracle.com/javase/8/docs/api/java/io/PrintStream.html#println–(https://docs.oracle.com/javase/8/docs/api/java/io/PrintStream.html#println--)

+ 0 −

2 months ago

(https://www.baeldung.com/author/loredana-crusoveanu/)

Editor

Loredana Crusoveanu (https://www.baeldung.com/author/loredana-crusoveanu/)

Good catch, thanks! I've updated the section with a different example.

+ 0 −

2 months ago

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 (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)
THE COURSES (HTTPS://COURSES.BAELDUNG.COM)
CONSULTING WORK (/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
THE FULL ARCHIVE (/FULL_ARCHIVE)
WRITE FOR BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORS (/EDITORS)