Theory: Method references

19 minutes 0 / 5 problems solved

As you know, lambda expressions allow you to use code as data and pass it as a method's arguments. Another way to do it is to use method references. They are often even more readable than corresponding lambda expressions. Besides, method references force developers to decompose a program into a set of short methods with clear areas of responsibility.

§1. Make code clearer with method references

By method reference, we mean a function that refers to a particular method via its name and can be invoked any time we need it. The base syntax of a method reference looks like this:

```
objectOrClass :: methodName
```

where `objectOrClass` can be a **class name** or a **particular instance** of a class.

Here is an example, we create a reference to the standard static method `max` of the `Integer` class.

```
BiFunction<Integer, Integer, Integer> max = Integer::max;
```

Here, `Integer::max` is a method reference to a static method.

This code works because the definition of the method `int max(int a, int b)` fits the type `BiFunction<Integer, Integer, Integer>`: they both mean taking two integer arguments and returning an integer value.

> You may already know that `BiFunction<Integer, Integer, Integer>` is not the only way to create a suitable object in this case. It is possible to use `IntBinaryOperator` and some other classes. We will consider them in other topics.

How we have the `max` object which can be used as a function by invoking the `apply` method. Let's invoke it!

```
System.out.println(max.apply(50, 70)); // 70
```

So, once assigned to an object, a method reference works in the same way as a lambda expression.

Here is an alternative way to create the same object using a lambda expression:

```
BiFunction<Integer, Integer, Integer> max = (x, y) -> Integer.max(x, y);
```

It is recommended to use method references rather than lambda expressions whenever possible. Your code will be shorter, more readable, and easier to test.

> Note that we can refer to both standard and our custom methods using method references.

§2. Kinds of method references

It's possible to write method references to both static and instance (non-static) methods.

In general, there are four kinds of method references:

Current topic:

Method references

Topic depends on:

✕ Lambda expressions

Topic is required for:

Functional interfaces

Table of contents:

- reference to a static method;
- reference to an instance method of an existing object;
- reference to an instance method of an object of a particular type;
- reference to a constructor.

1) Reference to a static method

The general form is the following:

```
ClassName :: staticMethodName
```

Let's take a look at the reference to the static method `sqrt` of the class `Math`:

```
Function<Double, Double> sqrt = Math::sqrt;
```

Now we can invoke the `sqrt` method for double values:

```
sqrt.apply(100.0d); // the result is 10.0d
```

The `sqrt` method can be also written using the following lambda expression:

```
Function<Double, Double> sqrt = x -> Math.sqrt(x);
```

2) Reference to an instance method of an object

The general form looks like this:

```
objectName :: instanceMethodName
```

Let's check out the example of a reference to the `indexOf` method of a particular string.

```
String whatsGoingOnText = "What's going on here?";


Function<String, Integer> indexWithinWhatsGoingOnText = whatsGoingOnText::indexOf;
```

Here is the result of applying it to different arguments:

```
System.out.println(indexWithinWhatsGoingOnText.apply("going")); // 7
System.out.println(indexWithinWhatsGoingOnText.apply("Hi"));    // -1
```

As you can see, actually we always work with the `whatsGoingOnText` object captured from the context.

The following example of a lambda expression is a full equivalent of the reference above and can make your understanding of the situation better:

```
Function<String, Integer> indexWithinWhatsGoingOnText = string -
> whatsGoingOnText.indexOf(string);
```

3) Reference to an instance method of an object of a particular type

Here is a general form of a reference:

```
ClassName :: instanceMethodName
```

In that case, you need to pass an instance of the class as the function argument.

Let's focus on the following reference to an instance the method `doubleValue` of the class `Long`:

```
Function<Long, Double> converter = Long::doubleValue;
```

Now we can invoke the `converter` for long values:

```
converter.apply(100L); // the result is 100.0d
converter.apply(200L); // the result is 200.0d
```

Also, we can write the same converter using the following lambda expression:

```
Function<Long, Double> converter = val -> val.doubleValue();
```

4) Reference to a constructor

This reference has the following declaration:

```
ClassName :: new
```

For example, let's consider our custom class `Person` with a single field `name`.

```
class Person {
    String name;

    public Person(String name) {
        this.name = name;
    }
}
```

Here is a reference to the constructor of this class:

```
Function<String, Person> personGenerator = Person::new;
```

This function produces new `Person` objects based on their names.

```
Person johnFoster = personGenerator.apply("John Foster"); // we have a John Foster
 object
```

Here is the corresponding lambda expression that does the same.

```
Function<String, Person> personGenerator = name -> new Person(name);
```

Further, we will use lambda expressions and method references together.

> More practical examples will be explored in the following topics. For now, it is enough to grasp the general idea and the syntax of method references.

§3. Conclusion

You've learned a new way to create function objects by using method references. It has much in common with lambda expressions but allows writing more readable and decomposed code. At the end of this topic, you should memorize all four types of method references and be ready to use them in your programs when you need to convey a piece of code into some method.

Report a typo

137 users liked this theory. **7** didn't like it. **What about you?**

Start practicing

This content was created over 3 years ago and updated 6 days ago. Share your feedback below in comments to help us improve it!

Comments (0) Hints (0) Useful links (1) Show discussion