Open in app ↗

Search Medium

Get unlimited access to the best of Medium for less than $1/week. Become a member

The elusive and beautiful Java Method Reference

Donald Raab · Follow
Published in Javarevisited
6 min read · Feb 19, 2022

Listen Share More

I love lambdas in Java 8, but method references are elusive and amazing

Photo by Jonatan Pie on Unsplash

Lambdas are flexible anonymous bits of code

We can solve a lot of interesting problems in Java using lambdas. We can use expression lambdas for simple things, and statement lambdas for more complex things. Lambdas can call out to other methods on the current object (`this`) or objects that are in scope, like the current element of an iteration, or a final local variable outside of the lambda. We can always simplify a lambda by putting code in another method.

Writing good lambdas requires discipline. For instance, it is important to use intention revealing names for parameters. Here's a simple example of a using a lambda to filter a `List` of `Strings` .

```
@Test
public void filterStringsLambda()
{
    var list = Lists.mutable.with(
            "Atlanta",
            "Atlantic City",
            "Boston",
            "Boca Raton");

    var actual = list.stream()
            .filter(string -> string.startsWith("At"))
            .collect(Collectors.toList());

    var expected = List.of("Atlanta", "Atlantic City");

    Assertions.assertEquals(expected, actual);
}
```

In this code, the lambda is the parameter passed to the `filter` method in the form of a `Predicate` . In this example, the `Predicate` takes a parameter of type `String` which I give the name `string` . The expression after the separator (`->`) will be evaluated for each element of the list and will only include those elements which evaluate to `true` .

There are several methods in the `Stream` API that will take a `Predicate` as a parameter. The methods include `filter` , `anyMatch` , `allMatch` , `noneMatch` .

There is no easy way for me to use a method reference here because I need to

pass the parameter `"At"` to the method `startsWith`. Parameters are kryptonite for method reference usage. We can simulate a method reference here by using a lambda and extracting it into a separate method as follows.

```
@Test
public void filterStringsLambdaInMethod()
{
    var list = Lists.mutable.with(
            "Atlanta",
            "Atlantic City",
            "Boston",
            "Boca Raton");

    var actual = list.stream()
            .filter(this.stringStartsWith("At"))
            .collect(Collectors.toList());

    var expected = List.of("Atlanta", "Atlantic City");

    Assertions.assertEquals(expected, actual);
}

private Predicate<String> stringStartsWith(String prefix)
{
    return string -> string.startsWith(prefix);
}
```

Having to create a method on a class to generate lambdas that can leverage local variables in scope is less than ideal. I would love to be able to just use the `startsWith` method as a method reference.

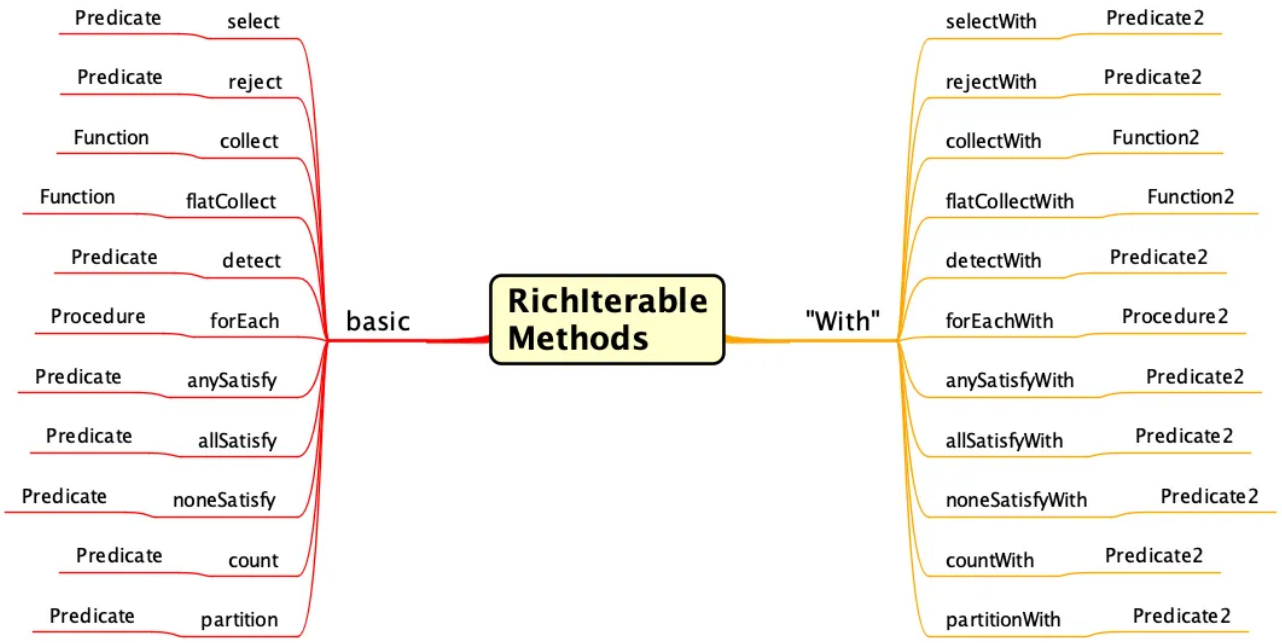
How to satisfy a Method Reference Preference?

Use the `With` methods in Eclipse Collections.

> *This is the way*

For many of the methods available in the Eclipse Collections API, there is a corresponding method with the suffix of `With`. Each `With` method takes a different named functional interface that takes two parameters (e.g. `Predicate2`, `Function2`, etc.). The following mind map shows some of the basic methods in the Eclipse Collections API along with their corresponding `With` equivalents and

functional interface types they take as parameters.



RichIterable basic and “With” Methods

How do these extra methods help you with using method references with parameters? Let’s walk through some examples.

Basic Using Lambda

Let’s see the example of filtering a List of Strings using one of the basic Eclipse Collections methods with a lambda.

```
@Test
public void selectStringsLambda()
{
    var list = Lists.mutable.with(
            "Atlanta",
            "Atlantic City",
            "Boston",
            "Boca Raton");

    var actual = list.select(string -> string.startsWith("At"));

    var expected = List.of("Atlanta", "Atlantic City");

    Assertions.assertEquals(expected, actual);
}
```

`With` Method Reference

Now let's see how we can satisfy our method reference preference using the "`With`" equivalent of `select`.

```
@Test
public void selectStringsWithMethodReference()
{
    var list = Lists.mutable.with(
            "Atlanta",
            "Atlantic City",
            "Boston",
            "Boca Raton");

    var actual = list.selectWith(String::startsWith, "At");

    var expected = List.of("Atlanta", "Atlantic City");

    Assertions.assertEquals(expected, actual);
}
```

> *I have spoken*

If you didn't just have an "aha!" moment, don't be alarmed. We still can't pass parameters to method references directly. There is no syntax in Java currently to support it. There is a trick that is happening here.

Let me try and explain how this works. The method `selectWith` takes two parameters. The first parameter is a `Predicate2`, which as it turns out will match the signature of `String::startsWith`. To be more specific, the `Predicate2<String, String>` matches the signature of `String::startsWith`. The second parameter `selectWith` takes is any type of parameter, which in this case happens to be a `String`.

Here's the exact signature of `selectWith` on `RichIterable`.

```
<P> RichIterable<T> selectWith(
        Predicate2<? super T, ? super P> predicate,
        P parameter);
```

I added the next section after I initially published the blog. A friend suggested a minor improvement would be to include a "how to" example so developers could see there is no magic and can begin leveraging for their own code. Thanks for the suggestion Rustam!

A `selectWith` pattern implementation example

There is a class in Eclipse Collections named `IteratorIterate`. It includes many of the basic eager iteration patterns in Eclipse Collections that allow the patterns to be used with any `Iterable` type in Java. I'm sharing this example because `Iterator` is a basic enough concept that most Java developers should be able to read and understand the code. The following shows the implementation of `selectWith` in `IteratorIterate` that is method reference friendly for method references with a single parameter.

```
public static <T, P, R extends Collection<T>> R selectWith(
        Iterator<T> iterator,
        Predicate2<? super T, ? super P> predicate,
        P injectedValue,
        R targetCollection)
{
    while (iterator.hasNext())
    {
        T item = iterator.next();
        if (predicate.accept(item, injectedValue))
        {
            targetCollection.add(item);
        }
    }
    return targetCollection;
}
```

This pattern can be used with any type that can create an `Iterator`.

Here's an example using `IteratorIterate.selectWith` with a JDK `Set`.

```
@Test
public void selectWithOnIteratorIterate()
{
    Set<String> strings = Set.of(
            "Atlanta",
            "Atlantic City",
            "Boston",
            "Boca Raton");

    HashSet<String> actual = IteratorIterate.selectWith(
            strings.iterator(),
            String::startsWith,
            "At",
            new HashSet<>());
    var expected = Set.of("Atlanta", "Atlantic City");
    Assertions.assertEquals(expected, actual);
}
```

I hope this additional section was helpful.

More Method References Please!

Now that we know how to use a method reference with a `With` method, let me show you some more examples.

```
@Test
public void predicatesWithMethodReference()
{
    var list = Lists.mutable.with(
            "Atlanta",
            "Atlantic City",
            "Boston",
            "Boca Raton");

    var selected1 = list.selectWith(String::startsWith, "At");

    var expected1 = List.of("Atlanta", "Atlantic City");
    Assertions.assertEquals(expected1, selected1);

    var rejected = list.rejectWith(String::startsWith, "At");
```

```
        var expected2 = List.of("Boston", "Boca Raton");
        Assertions.assertEquals(expected2, rejected);

        var selected2 = list.selectWith(String::startsWith, "Bo");

        Assertions.assertEquals(expected2, selected2);

        var detected = list.detectWith(String::endsWith, "y");

        Assertions.assertEquals("Atlantic City", detected);

        var count = list.countWith(String::contains, "c");

        Assertions.assertEquals(2, count);
        Assertions.assertTrue(
            list.anySatisfyWith(String::contains, "a"));
        Assertions.assertTrue(
            list.allSatisfyWith(String::contains, "t"));
        Assertions.assertTrue(
             list.noneSatisfyWith(String::contains, "z"));

        var partitioned = list.partitionWith(String::endsWith, "n");

        Assertions.assertEquals(expected2, partitioned.getSelected());
        Assertions.assertEquals(expected1, partitioned.getRejected());
    }
```

There are a lot of methods that take single parameters that can match `Predicate2`, `Function2`, `Procedure2`, etc. as method references. The With methods in Eclipse Collections increase the total number of places you can use method references instead of lambdas quite a bit.

Enjoy Method References and Lambdas

I hope this blog helped you discover a nifty feature available in Eclipse Collections that can help you find more places to use method references. We had the "`With`" methods in Eclipse Collections years before Method References arrived in Java 8. We initially added them so we could create more opportunities to hoist up anonymous inner classes into static variables to reduce garbage generation. We used to call these "fat free closures" as they didn't require you to keep adding new objects to the heap. The coincidence that this made it easier to leverage these methods with method references was an amazingly pleasant and welcome surprise.

Thank you for reading this blog! I hope you get to enjoy using method references as much as I do now.

I am a Project Lead and Committer for the Eclipse Collections OSS project at the Eclipse Foundation. Eclipse Collections is open for contributions. If you like the library, you can let us know by starring it on GitHub.

Further Learning

10 Best Places to Learn Java Online for Free in 2022

My favorite websites to learn Java online for free, suitable for beginners and people who want to learn to code in Java...

medium.com

Java Eclipse Collections Software Development Programming

Open Source

medium.com

22 Essential Java Libraries and APIs Every Programmer Should Learn in 2022

Most essential Java libraries you can learn to become a better Java developer. It includes Java libraries for logging...

medium.com

Follow

Written by Donald Raab

1.4K Followers · Writer for Javarevisited

Java Champion. Creator of the Eclipse Collections OSS Java library (http://www.eclipse.org/collections/). Inspired by Smalltalk. Opinions are my own.