CS 240: Method References Transcript

[00:00:00] **INSTRUCTOR:** The last thing I want to talk about with lambda expressions is method references.

[00:00:05] So, lambda expressions as hopefully you've learned are a shortened syntax or a simplified way to do something we could have done before.

[00:00:14] Method references are an even shorter syntax that we can use for certain lambda expressions.

[00:00:20] So, if we look at this example.

Start slide description. Following content is shown:

Method References

Simplified syntax for lambda expressions that simply call an existing method.

Consider a lambda expression that uses the Java forEach(Consumer<? super T>) method (defined in Iterable) to print the contents of a list:

```
List<Integer> intList = Arrays.asList(23, 5, 10, 71);

intList.forEach(x -> System.out.println(x));
```

The lambda expression simply calls an existing method, passing its parameter list to the method, so it can be replaced with a method reference:

```
List<Integer> intList = Arrays.asList(23, 5, 10, 71);

intList.forEach(System.out::println);
```

End slide description.

[00:00:23] So, one of the methods that is now built into Java into the list class is a for each method.

[00:00:29] It's pretty common to want to do something to iterate through the elements of some list and do something to all of the elements.

[00:00:37] So, in this example, we want to print them all out.

[00:00:40] So, we can create our list, and we can call four each and use a lambda expression to print everything out.

[00:00:47] So, that's simple, but we can make it even more simple because this is a special lambda expression that's just calling one method.

[00:00:54] And it's calling that method on the parameter that's passed in.

[00:00:58] And so, if that's the case, then that's an example where you can use a method reference to shorten the syntax.

[00:01:04] So, here is an example with the method reference.

[00:01:07] So, instead of declaring the variable with the array syntax, I can just say System dot out.

[00:01:14] And then instead of dot print line, it's colon colon print line.

[00:01:18] That indicates that I'm not trying to call this method right here.

[00:01:22] I'm trying to specify a method reference or, in other words, a lambda function.

[00:01:27] So that gets translated; the JVM can recognize that and translate that into a lambda that looks like this.

[00:01:36] OK.

[00:01:37] So, I already said this, but the double colon indicates a method reference instead of a method call.

Start slide description. Following content is shown:

Method References (cont.)

- The double colon indicates a method reference, instead of a method call.
- A parameter list is not needed or allowed for a method reference and there is no -> operator.
- Method references can be used for static method, instance method, and constructor invocations:
    - Static method: ClassName::methodName
    - Instance method: objectReference::methodName (equivalent to x -> objectReference.methodName(x))
    - Instance Method (with instance defined as the first parameter in the expression):
    - ClassName::methodName (equivalent to (x, y) -> x.methodName(y))
    - Constructor invocation: ClassName::new

End slide description.

[00:01:43] And if you're using a method reference, you don't specify a parameter list.

[00:01:48] In fact, you can't specify a parameter list.

[00:01:50] By definition, method references don't have a parameter list.

[00:01:53] So, we don't have that, and there are a few different ways we can use these or a few different circumstances where we can use them.

[00:02:00] If we’re just calling a static method, then we can do that by calling class name colon colon method name and we can shorten the syntax in that way.

[00:02:10] If we’re calling some instance method where we are doing something to some object that would get passed in a parameter list, that can be shortened.

[00:02:22] That’s the example that I showed you before, where we just specify object reference, colon colon method name.

[00:02:28] That’s equivalent to having some variable and passing that in and passing it as the parameter to the method name.

[00:02:35] So, if I just shorten it this way, then it’s the same thing.

[00:02:41] It’s also possible sometimes we have two parameters, and what we’re doing in our lambda is we’re calling some method on the first parameter and passing the second parameter as a parameter.

[00:02:55] And so that’s what this looks like.

[00:02:56] If I have my lambda expression is X,Y and the code is X and then call some method, so call some method on X and pass Y, I can shorten that to a method reference.

[00:03:08] So, I can just say class name colon colon method again.

[00:03:11] And the JVM will infer or determine that this is what I’m trying to do.

[00:03:18] And finally, I can shorten constructor calls.

[00:03:23] So, if my lambda expression is just creating some object, I can just do class name colon colon a new, and that is a shortened syntax again for a lambda expression that would create a new instance.

[00:03:35] So, these are a little bit trickier to recognize when you can use them.

[00:03:38] I don’t always recognize them.

[00:03:40] Sometimes I’ll write out the regular lambda and then my code editor will tell me, hey, that could be shortened to a method reference.

[00:03:47] So, when you’re kind of new to lambda functions or lambda expressions, you probably write these out and then when you see that it can be shortened to a method reference, just tell your editor to do that for you.

[00:03:59] And that’ll help you get used to writing method references.

[00:04:01] You don’t actually have to write them as method references, but the whole point of lambdas is short and simple syntax.

[00:04:08] So, we might as well make it as short and simple as we can.

[00:04:11] So, it’s a good idea to use method references when you can.