CS 240: How Java Lambdas Work Transcript

Start slide description.

Jerod Wilkerson presents a recorded lecture with the aid of a slide deck. All visual content is described in the audio.

End slide description.

[00:00:01] **JEROD WILKERSON:** In this video, I will show you how Java Lambdas work kind of behind the scenes so you'll understand how Lambda expressions work. They're implemented in Java in a specific way because of the way the programing language works.

[00:00:14] It's not necessarily the way they work in all languages, but it's the way they work in Java. So, we have a few different steps that the JVM has to go through. So, first it infers at runtime a data type for the Lambda expression.

[00:00:29] So, when you declare a lambda expression, you're not specifying a data type for it, but the the JVM means to know a data type for it, so it can infer that based on the code that you do right.

[00:00:40] I'll have a little bit more to say about that later. And it also can infer a return type or it can determine if the return type is void, and so it does that and the parameter types. You could explicitly specify the parameter types that's usually not done, and so generally the JVM needs to infer those.

[00:01:02] And it can infer all of that from the method that's being called. So, in the previous example, I showed an example of calling Arrays that sort. So, the JVM knows I'm calling arrays that sort, so it can look at the definition of the sort method and infer all of this information.

[00:01:17] So, then the JVM constructs an in-memory instance of a class expected by the called method. So, the called method in this case is arrays that sort, and that is taking as its second parameter a comparator.

[00:01:32] So, when it sees my lambda expression, it knows that, that is a lambda expression that is representing a comparator.

[00:01:39] And so, it will create an in-memory instance of a class that implements the comparator interface and the part before the arrow names the parameters in the method, and that acts as a declaration, so I can use those parameters throughout my code. And then that constructed instance is passed in the method call.

[00:02:05] Now all of that you could do yourself. And so, that's why we've always been able to pass code around in Java, but it was just a little bit tedious.

[00:02:11] Now that we have the JVM doing some of the work for us, it is a lot simpler from the programmers point of view, but in reality the exact same things are happening, the JVM is just doing some of it for you. Okay, So let's look at that specific example in detail, the example that we just went through.

[00:02:28] So, this was the code example, the Lambda expression example that I gave you before.

[00:02:32] And so, if we look at those steps that I just went through in step one, the JVM infers that the method being called is arrays that sort of can see us, arrays that sort, but arrays that sort is a method that's using a generic type.

[00:02:46] So, it seems that type is a T array, an array of some type. And when we call it, we pass it, we pass it some array, so it can tell what T is based on, what type of array we pass it.

[00:03:01] Then the type of the second parameter that's the second parameter type is using wildcard syntax. So, it's a comparator that is its data type is super T. And so, if you looked in arrays that sort, you'd see that's the way that method is declared.

[00:03:18] The JVM when it's called can tell what T is and so it knows that that's a string. So, that allows the JVM to construct an instance of the comparator that looks like this.

[00:03:31] So, this instance is a comparator of type string, and that means its compare method takes two strings and it's going to add the return, that's why you don't have to write that in Lambda expression, because the JVM will write that and then it will take the code that you put for the lambda and just put that after the return.

[00:03:50] And then the JVM will invoke this sort method with the comparator instance past as the second parameter.

[00:03:58] So, that kind of brings up a question. What data type can be used for the lambda expression? So, one of the things that you can do is you could actually create a variable to hold the lambda expression, but if you do that, you'd have to know a data type. And so, we need to understand some things about what is the data type of a lambda expression? Data type can be anything that fits the definition of a functional interface.

[00:04:25] So, we need to know what a functional interface is. A functional interface is any interface with exactly one abstract method. If it has two abstract methods, it's not a functional interface and it can't be used with lambdas.

[00:04:38] And so, if it has exactly one and there are other kinds of methods you can have it in or in an interface there didn't used to be, but now there are in modern versions of Java or in the more recent versions.

[00:04:49] So, an interface can have any number of static or default methods and even declared object methods. So, methods that exist on the object class, you can have those and still be a functional interface, but then it has to have exactly one abstract method.

[00:05:08] So, here are some examples of that. Comparable, I mentioned that in a previous video and when you create a thread, you pass on a chunk of code and the code that you pass it is something that implements the comparable interface.

[00:05:20] And comparable has exactly one abstract method, so it fits the definition of a functional interface. I'm sorry, I just gave you the definition of Runnable.

[00:05:30] Runnable is the code that you pass to a thread. And comparable is what I've been using as examples for sort. And sort functions in Java will take a comparable to determine how something should sort and the comparable interface has exactly one abstract method, so it's a functional interface. Some event handlers have one abstract method, others have more than one.

[00:05:54] But event handlers that have exactly one are functional interfaces and can be the data type for lambda.

[00:06:01] And when I say the data type, what I'm really talking about is usually when you're using a lambda, you're going to have that lambda expression as a parameter that's being passed to some other method.

[00:06:12] That other method has to have a data type for that parameter, and that data type is going to be a functional interface, so these are all examples of that.

[00:06:20] When lambdas were added to the Java language, though, we needed more than just the few interfaces that existed with 1 abstract method, and so they created a bunch of other interfaces. And they created them in the Java that "util" that function package that created 43 interfaces, all that are functional interfaces, so 43 different interfaces that have exactly one method.

[00:06:44] So, those are all functional interfaces and then you can create your own. So, usually one of the 43 or one of the previously existing ones like comparable or Runnable will suit your needs and you should use those rather than create your own.

[00:06:57] But you could run into a situation where there really isn't an interface that has the method you want to call or something like that. And in that case you can create your own interface, just like I did in some previous examples.

[00:07:09] And as long as that interface has exactly one abstract method, it can be used as the data type for a lambda expression.