Bellevue University

**Module 9**

Donnell Perkins

Intermediate Java Programming – CSD 405

Instructor: Darrell Payne

Due: 07/09/2023

History

The Lambda expression feature was introduced when Java updated to version 8 back in March of 2014. Lambda expressions can help to simply the code, especially when working with event handling. Before Lambda expression developers used anonymous inner classes to handle events and that usually took, at least, 5 lines of code to get one thing accomplished. The code files were longer and more overwhelming and Lambda expressions was introduced to alleviate that problem.

|  |  |
| --- | --- |
| A screen shot of a computer code  Description automatically generated | A screen shot of a computer  Description automatically generated |

In the example above, the Anonymous Inner class must call the interface EventHandler<ActionEvent> and the method *handle* to process any events. And if you notice that we used the letter ‘e’ to let you know when we are handling an event. There already seems like a lot going on and we haven’t processed the action that we plan to handle. But all of this is needed before anything can happen.

With the second example all the introduction that happened for the anonymous inner class is done when you call *setOnAction*. And instead of having to tell the code what we are doing with ‘e’, the complier already understands that you are using a lambda expression to process an event. So, there is no need to call an interface EventHandler and the *handle* method. And that was all done in 3 lines of code.

Uses/Syntax/Limitations

The complier does quite a bit when it comes to working with Lambda expressions. As well as working with the syntax for lambda expression. It’s kind of like working with If statements that we have been using. If you only have one statement, you don’t need the curly braces. The complier will automatically assume you have one Lambda event that you are processing. For the compiler to recognize multiple lambda expressions, we use an interface called the SAM (Single Abstract Method) interface. Which turns out to be a functional interface, which an instance of it is called a function object. This is what makes the Lambda function/expression.

Reading around online I found a site that described the functionality of lambda expressions as being able to use them as a “… method argument, or code as data.”. And to me that helps me understand what all happens when using lambda expressions. Like I mentioned earlier you can create lambda expressions without curly braces that only carries out one line of code/expression. But you can also use parentheses around the parameter of the lambda if you need to pass in more than one parameter. It can be broken down something like:

|  |  |  |
| --- | --- | --- |
| No Parentheses & No Curly Braces | Parentheses & No Curly Braces | Parentheses & Curly Braces |
| parameter -> expression | (param1, param2) -> expression | (param1, param2) -> {statements of code} |

Lambda expressions are limited in how they can be used. “They have to immediately return a value, and they cannot contain variables, assignments or statements such as **if** or **for**. To do more complex operations, a code block can be used with curly braces. If the lambda expression needs to return a value, then the code block should have a return statement.”

Code Example

For this paper I decided to create a simple lambda expression and use the for-each loop as well as an ArrayList class to iterate through a list of companies and their profits.

Line 16 I create the string ArrayList and give it the name companies. Then I added four random wealthy companies of the past few years, and that’s lines 18 to 21. Then we get to where I used the lambda function on line 25. We create the parameter n and the compiler will know that n will be declared to whatever is being passed through. In this case it is Strings for the string of companies and their wealth. So I call forEach(n) and then send my expression using the (->). For my expression I use the System.out.println to print each item in the ArrayList.

Just for comparison’s sake I created a for loop that does the exact same thing but more code to execute, and that is on lines 29 and 30. I am use to creating and recognizing what all comes in the standard for loop. You create the integer that will loop and initialize, then the check statement for the loop, and then how the variable will increment. Line 30 is where the statement of code will go that we are looping.

The lambda expression is explaining what is happening using the keywords for and each and combining them, that way you know that each item of the collection will be looped through.

A screen shot of a computer code

Description automatically generated

# Works Cited

GeekforGeeks. (2023, July 09). *Lambda Expressions Java 8*. Retrieved from Geek For Geeks: https://www.geeksforgeeks.org/lambda-expressions-java-8/

Liang, Y. D. (2019). *Introduction to Java Programming and Data Structures.* Hoboken, NJ: Pearson.

W3Schools. (2023, July 09). *Java Lambda*. Retrieved from W3Schools: https://www.w3schools.com/java/java\_lambda.asp