JAC444 - Lecture 8

Functional Programming in Java
Segment 2 - Lambda Expressions

Objectives

Upon completion of this lecture, you should be able to:

- Utilize Lambda Expression in Java
- Create Functional Interfaces
- Discover Principles of Functional Programming in Java

Lambda Expressions

In this lesson you will be learning about:

- Syntax of Lambda Expression
- Functional Interfaces
- Variable capture
- Method references

Lambda Expressions

A lambda expression has the form:

```
(params) -> body
```

params - is a list of zero or more types
body - zero or more statements

Example:

```
(int x, int y) -> x + y;
() -> System.out.println(“It is nice to learn”);
```

Example of Lambda Expressions

1. The type of params can be inferred from context

```
k -> k % 2 == 0;
```

If there is one param and can be inferred, the parentheses are not required.
If the body has only one statement the curly brackets are not required

2. If the body has more statements, the curly brackets are mandatory

```
operands -> {
    Integer total = 0;
    for (Integer x : operands)
        total += x;
    return total;
};
```

Implementation of Lambdas

A lambda expression is converted into a function

Example:

```
s -> System.out.println(s)
```

could be converted into a static function

```
public static void generatedName(String s) {
    System.out.println(s);
}
```

Questions:

1. What class should it go in?
2. How could one call it?

Functional Interface

A *functional interface* is an interface with a single abstract method.

Example:

```
@FunctionalInterface
public interface CircleMeasurement {
   /**
    * @param radius circle radius
    * @return the value of circle measure
    */
   double value(double radius);
}
```

Design Decision

A lambda expression is an object whose type is the type of the functional interface.

1. The method generated has the same signature of the method from functional interface

2. The lambda expression body becomes the body of the method from the functional interface

Variable Capture

Lambda expression can use variables outside the body of expression (this is called variable capture).

Outside variable must be final or effective final.

Example:

```
String s = "Effective final variable";
new Thread(() -> System.out.println(s)).start();
```

Method Reference

A *method reference* is the syntax sugar for a lambda expression that has only one method. The *method reference* uses a special operator **::**

Use **objectTYpe::instanceMethod**

Example: **Consumer<String>** is from **java.util.function**

```
Consumer<String> c = s -> System.out.println(s);
```

Lambda can be replaced by a method reference expression

```
Consumer<String> c = System.out::println;
```

More on from: https://www.codementor.io/eh3rrera/using-java-8-method-reference-du10866vx

Predefined Functional Interfaces

Java 8 has a package called **`java.util.function`**

One can find most of the necessary functional interfaces in this package

For example:

Predicate functional interface defines a predicate method which is boolean-valued function of one argument

```
@FunctionalInterface
public interface Predicate<T> {
        //Evaluates this predicate on the given argument.
        boolean test(T t)
}
```