JAVA 8 - LAMBDA EXPRESSIONS

http://www.tutorialspoint.com/java8/java8_lambda_expressions.htm Copyright © tutorialspoint.com

Lambda expressions are introduced in Java 8 and are touted to be the biggest feature of Java 8. Lambda expression facilitates functional programming, and simplifies the development a lot.

Syntax

A lambda expression is characterized by the following syntax −

```
parameter -> expression body
```

Following are the important characteristics of a lambda expression −

- **Optional type declaration** − No need to declare the type of a parameter. The compiler can inference the same from the value of the parameter.
- **Optional parenthesis around parameter** − No need to declare a single parameter in parenthesis. For multiple parameters, parentheses are required.
- **Optional curly braces** − No need to use curly braces in expression body if the body contains a single statement.
- **Optional return keyword** − The compiler automatically returns the value if the body has a single expression to return the value. Curly braces are required to indicate that expression returns a value.

Lambda Expressions Example

Create the following Java program using editor and save in some folder like C:\>JAVA.

Java8Tester.java

```
public class Java8Tester {
   public static void main(String args[]){
      Java8Tester tester = new Java8Tester();

      //with type declaration
      MathOperation addition = (int a, int b) -> a + b;

      //with out type declaration
      MathOperation subtraction = (a, b) -> a - b;

      //with return statement along with curly braces
      MathOperation multiplication = (int a, int b) -> { return a * b; };

      //without return statement and without curly braces
      MathOperation division = (int a, int b) -> a / b;

      System.out.println("10 + 5 = " + tester.operate(10, 5, addition));
      System.out.println("10 - 5 = " + tester.operate(10, 5, subtraction));
      System.out.println("10 x 5 = " + tester.operate(10, 5, multiplication));
      System.out.println("10 / 5 = " + tester.operate(10, 5, division));

      //with parenthesis
      GreetingService greetService1 = message ->
      System.out.println("Hello " + message);

      //without parenthesis
      GreetingService greetService2 = (message) ->
      System.out.println("Hello " + message);

      greetService1.sayMessage("Mahesh");
      greetService2.sayMessage("Suresh");
   }
   
   interface MathOperation {
      int operation(int a, int b);
   }
   
   interface GreetingService {
      void sayMessage(String message);
   }
   
   private int operate(int a, int b, MathOperation mathOperation){
      return mathOperation.operation(a, b);
   }
}
```

Verify the Result

Compile the class using **javac** compiler as follows −

```
$javac Java8Tester.java
```

Now run the Java8Tester as follows −

```
$java Java8Tester
```

It should produce the following output −

```
10 + 5 = 15
10 - 5 = 5
10 x 5 = 50
10 / 5 = 2
Hello Mahesh
Hello Suresh
```

Following are the important points to be considered in the above example.

- Lambda expressions are used primarily to define inline implementation of a functional interface, i.e., an interface with a single method only. In the above example, we've used various types of lambda expressions to define the operation method of MathOperation interface. Then we have defined the implementation of sayMessage of GreetingService.
- Lambda expression eliminates the need of anonymous class and gives a very simple yet powerful functional programming capability to Java.

Scope

Using lambda expression, you can refer to final variable or effectively final variable *whichisassignedonlyonce*. Lambda expression throws a compilation error, if a variable is assigned a value the second time.

Scope Example

Create the following Java program using editor and save in some folder like C:\>JAVA.

Java8Tester.java

```
public class Java8Tester {

   final static String salutation = "Hello! ";
   
   public static void main(String args[]){
      GreetingService greetService1 = message ->
      System.out.println(salutation + message);
      greetService1.sayMessage("Mahesh");
   }
   interface GreetingService {
      void sayMessage(String message);
   }
}
```

Verify the Result

Compile the class using **javac** compiler as follows −

```
$javac Java8Tester.java
```

Now run the Java8Tester as follows −

```
$java Java8Tester
```

It should produce the following output −

```
Hello! Mahesh
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js