Java Lambda Expressions Tutorial with examples

Lambda expression is a new feature which is introduced in Java 8. A lambda expression is an anonymous function. A function that doesn't have a name and doesn't belong to any class. The concept of lambda expression was first introduced in LISP programming language.

Java Lambda Expression Syntax

To create a lambda expression, we specify input parameters (if there are any) on the left side of the lambda operator `->`, and place the expression or block of statements on the right side of lambda operator. For example, the lambda expression `(x, y) -> x + y` specifies that lambda expression takes two arguments x and y and returns the sum of these.

```
//Syntax of lambda expression
(parameter_list) -> {function_body}
```

Lambda expression vs method in Java

A method (or function) in Java has these main parts:
1. Name
2. Parameter list
3. Body
4. return type.

A lambda expression in Java has these main parts:
Lambda expression **only has body and parameter list**.
1. **No** name – function is anonymous so we don't care about the name
2. Parameter list
3. Body – This is the main part of the function.
4. **No** return type – The java 8 compiler is able to infer the return type by checking the code. you need not to mention it explicitly.

Where to use the Lambdas in Java

To use lambda expression, you need to either create your own functional interface or use the pre defined functional interface provided by Java. An

interface with **only single abstract method** is called functional interface(or Single Abstract method interface), for example: Runnable, callable, ActionListener etc.

To use function interface:
Pre Java 8: We create anonymous inner classes.
Post Java 8: You can use lambda expression instead of anonymous inner classes.

Java Lambda expression Example

Without using Lambda expression: Prior to java 8 we used the anonymous inner classe to implement the only abstract method of functional interface.

```
import java.awt.*;
import java.awt.event.*;
public class ButtonListenerOldWay {
    public static void main(String[] args) {
        Frame frame=new Frame("ActionListener Before Java8");

        Button b=new Button("Click Here");
        b.setBounds(50,100,80,50);

        b.addActionListener(new ActionListener(){
           public void actionPerformed(ActionEvent e){
             System.out.println("Hello World!");
           }
        });
        frame.add(b);

        frame.setSize(200,200);
        frame.setLayout(null);
        frame.setVisible(true);
    }
}
```

By using Lambda expression: Instead of creating anonymous inner class, we can create a lambda expression like this:

```
import java.awt.*;
public class ButtonListenerNewWay {
   public static void main(String[] args) {
      Frame frame=new Frame("ActionListener java8");

      Button b=new Button("Click Here");
      b.setBounds(50,100,80,50);
```

```
        b.addActionListener(e -> System.out.println("Hello World!"));
        frame.add(b);

        frame.setSize(200,200);
        frame.setLayout(null);
        frame.setVisible(true);
    }
}
```

Note:
1. As you can see that we used less code with lambda expression.
2. Backward compatibility: You can use the lambda expression with your old code. Lambdas are backward compatible so you can use them in existing API when you migrate your project to java 8.

Lets see few more examples of Lambda expressions.

Example 1: Java Lambda Expression with no parameter

```
@FunctionalInterface
interface MyFunctionalInterface {

        //A method with no parameter
    public String sayHello();
}
public class Example {

   public static void main(String args[]) {
        // lambda expression
       MyFunctionalInterface msg = () -> {
              return "Hello";
       };
        System.out.println(msg.sayHello());
    }
}
```

Output:

```
Hello
```

Example 2: Java Lambda Expression with single parameter

```
@FunctionalInterface
interface MyFunctionalInterface {
```

```
       //A method with single parameter
    public int incrementByFive(int a);
}
public class Example {

   public static void main(String args[]) {
        // lambda expression with single parameter num
       MyFunctionalInterface f = (num) -> num+5;
        System.out.println(f.incrementByFive(22));
    }
}
```

Output:

```
27
```

Example 3: Java Lambda Expression with Multiple Parameters

```
interface StringConcat {

    public String sconcat(String a, String b);
}
public class Example {

   public static void main(String args[]) {
        // lambda expression with multiple arguments
       StringConcat s = (str1, str2) -> str1 + str2;
        System.out.println("Result: "+s.sconcat("Hello ",
"World"));
    }
}
```

Output:

```
Result: Hello World
```

Example 4: Iterating collections using foreach loop

```
import java.util.*;
public class Example{
    public static void main(String[] args) {
       List<String> list=new ArrayList<String>();
       list.add("Rick");
       list.add("Negan");
       list.add("Daryl");
       list.add("Glenn");
```

```
        list.add("Carl");
        list.forEach(
            // lambda expression
            (names)->System.out.println(names)
        );
    }
}
```