Java

Lambda Expressions

Hannes Ueck, Jakob Krude

28. Januar 2021

Java-Kurs

What is a lambda expression?

It is a short block of code that takes in parameters and returns a value. They are similar to methods, but they don't have a name.

It basically implements an interface or abstract class with only one method, without needing to define a class and method.

Example

Here you can see, that in line 9 the interface 'Addable' is implemented and then used to add two numbers.

```
1  interface Addable{
2      int add(int a,int b);
3  }
4
5  public class LambdaExpressionExample{
6      public static void main(String[] args) {
7
8          // Multiple parameters with data type in lambda
       expression
9          Addable ad2=(int a,int b)->(a+b);
10         // Prints: 300
11         System.out.println(ad2.add(100,200));
12     }
13 }
```

Syntax

Basic Structure: (*argument-list*) –>{*body*}

- **argument-list**: It can be empty or have multiple comma separated arguments.
- **arrow-token**: It is used to link arguments-list and body of expression.
- **body**: It contains expressions and statements for lambda expression.

Another example

```
1
2  public class LambdaHello {
3      // A interface is declared that has the method print()
4      interface Printer { void print(); }
5
6      public static void main(String[] args) {
7          // This is the implementation of the
8          // interface Printer
9          Printer p = () -> System.out.println("Hello World!")
       ;
10         // Now the lambda defined in the
11         //previous line is executed
12         p.print();     // "Hello World!"
13     }
14 }
15
```

Stream

Now we combine collections with lambda expressions.

For most applications of lambda expressions you want to consume the elements one by one.
For example, you want to add 1 to every number in a List.
For that we need following functions which every collection implements by default:

- **.stream()** one by one return every member of a collection
- **.map(lambda expression)** the lambda expression is applied for every member of the collection
- **.collect(Collectors.toList())** every member is put back into a list

Stream with Collection

The parameter of map() is a lambda expression that is executed with every number in the list.
The stream() returns every number exactly once and collect() collects all the numbers into a list again.

```
1  public void addOne(List<Integer> numbers) {
2      // Add one to every member of numbers
3      // numbers: [1,2,3,4,5]
4      result = numbers
5                  .stream()
6                  .map((number) -> {number++})
7                  .collect(Collectors.toList());
8      return result;          //result: [2,3,4,5,6]
9  }
10
```

Filter

If you want to filter out elements of a collection you can use .filter().

```
1  public List<Integer> addOne(List<Integer> numbers) {
2      // Add one to every member of numbers
3      // numbers: [1,2,3,4,5]
4      result = numbers
5                  .stream()
6                  .filter((number) -> {number > 3})
7                  .collect(Collectors.toList());
8      return result;              // result: [1,2,3]
9  }
10
```

Sort

If you want to sort a list you can define an implementation for the Interface Comparator<T> by using a lambda expression.
Then you can pass the lambda expression to the .sort() function as a parameter.

```
1 public List<Integer> sort(List<Integer> numbers){
2     // Use lambda expression to implement the interface
      Comparator
3     // numbers: [2,1,5,4]
4     Comparator<Integer> comp = (n1, n2) -> n1.compareTo(n2);
5     numbers.sort(comp);
6     return numbers;          // numbers: [1,2,4,5]
7 }
8
```