Concurrent programming

Kristoffer Klokker

2022

Contents

1 Introduction

Concurrency is the act of having multiple execution done simultaneously which interact with each other.
This is done to utilise multiple CPU cores rather than rely on CPU speed.
Not only this but instead of having single powerfull computers, bigger networks of computers can be used.
The benefits comes at a cost of complexity, due to the all possible outcomes of different timed execution.

2 Anonymous,- and lambdafunctions

For at simple class which is given in an argument, instead of creating a class and then parsing it, the class can be created in the argument field.
For instance a class which implements comparable, it can be programmed as such:

```
1  public interface StringExecute {
2      public void run(String content);
3  }
4
5  public static void doAndMeasure( StringExecutable
       runnable ) {
6      long t1 = System.currentTimeMillis();
7      runnable.run();
8      System.out.println( "Elapsed time: " + (System.
           currentTimeMillis() - t1) + "ms" );
9  }
10
11 public static void anonFunc() {
12     doAndMeasure(new StringExecute() {
13         public static void run(String content) {
14             System.out.println(content + " Wow!");
15         }
16     });
17 }
18
19 public static void lambdaFunc() {
20     doAndMeasure( (content) -> System.out.println(
           content + " Wow!"););
21 }
22
```

```
23 public static void lambdaFuncOpt() {
24     doAndMeasure( (content) -> System.out::println;); //
          Only prints content and not + " Wow!"
25 }
```

Here lambda function is only possible due to the compiler knowing what type of object is created due to restrains from the function and the runnable interface only contains a single run function. Another use of lambda expression is when working with maps.

```
26 public static void main() {
27     String text = "Hello world hope your having a good
          day!";
28     Map<Charachter, Interger> occurrences = new HashMap
          <>();
29     for(int i = 0; i < text.length(); i++){
30         final char c = text.charAt(i);
31         if(occurrences.containsKey(c))
32             occurrences.put(c,occurrences.get(c)+1);
33         else
34             occurrences.put(c,1);
35     }
36     for(int i = 0; i < text.length(); i++){
37         final char c = text.charAt(i);
38         occurrences.merge(c,1,(currValue,value) ->
              currValue+value);
39     }
```

Both for loops do the same, the second uses merge which takes a position (c) and a default value (1) and a bifunction which is a function with two inputs. The arguments will automaticly be assigned such currValue is the current hash value and value is the same as the default value 1.