Autoren: Marius Birk
Pieter Vogt
Tutor: Florian Brandt

Abgabe: 30.06.2020, 12:00 Uhr

Smileys:

| A1 | A2 | A3 | $\sum$ |
|---|---|---|---|
| | | | |

# Objektorientierte Modellierung und Programmierung

# Abgabe Uebungsblatt Nr.10

(Alle allgemeinen Definitionen aus der Vorlesung haben in diesem Dokument bestand, es sei den sie erhalten eine explizit andere Definition.)

## Aufgabe 1

```
1   import java.util.ArrayList;
2   import java.util.Scanner;
3
4   public class SearchThread extends Thread{
5       private ArrayList<Integer> search;
6       private int searched;
7       private boolean found;
8       public SearchThread(ArrayList<Integer> second, int
            searched) {
9           search = second;
10          this.searched = searched;
11      }
12
13      public void run(){
14          if(search.contains(searched)){
15              found = true;
16          }else {
17              found = false;
18          }
19      }
20      public static void main(String[] args){
21          int width= (int) (Math.random()*10);
22          int[] array = new int[width];
23          ArrayList<Integer> first = new ArrayList<>();
24          ArrayList<Integer> second = new ArrayList<>();
25          for(int i =0; i<array.length;i++){
26              double rand=Math.round(Math.random()*10);
27              array[i]=(int) rand;
28          }
29          for(int i=0; i<(array.length/2);i++){
30              first.add(array[i]);
31          }
32          for(int i = array.length/2;i<array.length;i++){
33              second.add(array[i]);
34          }
```

```
35              System.out.print("Array: ");
36              for(int i = 0; i<array.length;i++){
37                  System.out.print(array[i]+", ");
38              }
39              System.out.println();
40              Scanner in = new Scanner(System.in);
41              System.out.println("Eingabe: ");
42              int searched = in.nextInt();
43
44              SearchThread search1 = new SearchThread(first,
                    searched);
45              SearchThread search2 = new SearchThread(second,
                    searched);
46              search1.start();
47              search2.start();
48
49              try{
50                  search1.join();
51                  search2.join();
52              }catch (InterruptedException e){}
53              System.out.println(search1.getName()+": "+search1
                    .found);
54              System.out.println(search2.getName()+": "+search2
                    .found);
55
56              if(search1.found==true || search2.found==true){
57                  System.out.println("Found: true");
58              }else{
59                  System.out.print("Found: false");
60              }
61          }
62      }
```

# Aufgabe 2

## QuickSortThreaded

```
1  public class QuickSortThreaded extends QuickSort implements
       Runnable{
2  private int[] numbers;
3  private int leftIndex;
4  private int rightIndex;
5  public QuickSortThreaded(int[] numbers, int leftIndex, int
       rightIndex) {
6   this.numbers = numbers;
7   this.leftIndex = leftIndex;
8   this.rightIndex = rightIndex;
9  }
```

```
10
11  /**
12   * sortiert das uebergebene Array in aufsteigender
        Reihenfolge
13   * gemaess dem QuickSort-Algorithmus (parallel!)
14   */
15  public static void sort(int[] numbers) {
16   QuickSortThreaded left =new QuickSortThreaded(numbers, 0,
        numbers.length/2);
17   QuickSortThreaded right = new QuickSortThreaded(numbers,
        numbers.length/2, numbers.length);
18   Thread t1 = new Thread(left);
19   Thread t2 = new Thread(right);
20
21   t1.start();
22   t2.start();
23   try{
24    t1.join();
25    t2.join();
26   }catch (InterruptedException e){}
27   QuickSort.sort(numbers);
28  }
29
30  /**
31   * der Quicksort-Algorithmus wird auf dem Array zwischen den
32   * angegebenen Indizes ausgefuehrt
33   */
34  protected void quickSort(int[] numbers, int leftIndex, int
       rightIndex) {
35   super.quickSort(numbers, leftIndex, rightIndex);
36  }
37
38  @Override
39  public void run() {
40   this.quickSort(numbers, leftIndex, rightIndex);
41  }
42 }
```

## QuickSort

```
1     public class QuickSort {
2
3  /**
4   * sortiert das uebergebene Array in aufsteigender
        Reihenfolge
5   * gemaess dem QuickSort-Algorithmus
6   */
7  public static void sort(int[] numbers) {
```

```
8    new QuickSort().quickSort(numbers, 0, numbers.length - 1);
9   }
10
11  /**
12   * der Quicksort-Algorithmus wird auf dem Array zwischen den
13   * angegebenen Indizes ausgefuehrt
14   */
15  protected void quickSort(int[] numbers, int leftIndex, int
       rightIndex) {
16   if (leftIndex < rightIndex) {
17    int pivotIndex = divide(numbers, leftIndex, rightIndex);
18    quickSort(numbers, leftIndex, pivotIndex - 1);
19    quickSort(numbers, pivotIndex + 1, rightIndex);
20   }
21  }
22
23  /**
24   * liefert den Index des Pivot-Elementes und ordnet das
        Array innerhalb
25   * der angegebenen Indizes so um, dass alle Zahlen links vom
         Index
26   * kleiner oder gleich und alle Zahlen rechts vom Index
        groesser
27   * oder gleich dem Pivot-Element sind
28   */
29  protected int divide(int[] numbers, int leftIndex, int
       rightIndex) {
30   int pivotIndex = choosePivotIndex(numbers, leftIndex,
        rightIndex);
31   int pivotValue = numbers[pivotIndex];
32   // das Pivot-Element kommt nach ganz rechts im Array
33   swap(numbers, pivotIndex, rightIndex);
34   int left = leftIndex - 1;
35   int right = rightIndex;
36   // ordne das Array so um, dass jeweils alle Elemente links
        vom
37   // Zeiger left kleiner und alle Elemente rechts vom Zeiger
        right
38   // groesser als das Pivot-Element sind
39   do {
40    left++;
41    while (left <= rightIndex && numbers[left] <= pivotValue)
42     left++;
43    right--;
44    while (right >= leftIndex && numbers[right] >= pivotValue)
45     right--;
46    if (left < right) {
47     swap(numbers, left, right);
48    }
```

```
49    } while (left < right);
50    // platziere das Pivot-Element an seine korrekte Position
51    if (left < rightIndex) {
52     swap(numbers, left, rightIndex);
53     return left;
54    } else {
55     return rightIndex;
56    }
57   }
58
59   /**
60    * waehlt einen beliebigen Index zwischen den angegebenen
         Indizes
61    */
62   protected int choosePivotIndex(int[] numbers, int leftIndex,
         int rightIndex) {
63    // in diesem Fall einfach der mittleren Index
64    return (leftIndex + rightIndex) / 2;
65   }
66
67   /**
68    * tauscht die Elemente des Arrays an den angegebenen
         Indizes
69    */
70   protected void swap(int[] numbers, int index1, int index2) {
71    if (index1 != index2) {
72     int tmp = numbers[index1];
73     numbers[index1] = numbers[index2];
74     numbers[index2] = tmp;
75    }
76   }
77
78  }
```

## QuickSortTest

```
1      public class QuickSortTest {
2
3   public static void main(String[] args) {
4    int[] numbers = {2, 3, 9, 33, -2, 4, 55, 66, -234};
5    print(numbers);
6    QuickSort.sort(numbers);
7    print(numbers);
8
9    int[] numbers2 = {2, 3, 9, 33, -2, 4, 55, 66, -234};
10   print(numbers2);
11   QuickSortThreaded.sort(numbers2);
12   print(numbers2);
```

```
 }

 private static void print(int[] numbers) {
  for (int number : numbers) {
   System.out.print(number + " ");
  }
  System.out.println();
 }

}
```

# Aufgabe 3

```
    import java.awt.Color;

    public class ImageFilterThreaded extends ImageFilter {

        @Override
        protected Color[][] filterMatrix(float[][] filter) {
            //TODO implement this
            //MAtrix teilen um größe zu erhalten
            return super.filterMatrix(filter);
        }

    }
```

# 1 Aufgabe 4

```
import javax.swing.*;
import java.io.ObjectOutputStream;

public class NameOutput extends Thread {
    private String name;
    private int len;

    public static void main(String[] args) throws
       InterruptedException {
        int random = 3;
        Thread[] arr = new Thread[random];
        String[] names = new String[random];
        int len = arr.length;

        for(int i =0;i<arr.length;i++){
            arr[i] = new Thread(new NameOutput());
        }
        for(int i =0; i<arr.length;i++){
```

```
18              names[i] = arr[i].getName();
19          }
20
21          while (true) {
22              synchronized (NameOutput.class) {
23
24
25              }
26          }
27      }
28  //wait und notify sollen besser sein
29
30      public void run() {
31          while(true){
32              System.out.println(this.getName());
33              try {
34                  sleep(1000);
35              } catch (InterruptedException e) {
36                  e.printStackTrace();
37              }
38          }
39      }
40  }
```