Autoren: Marius Birk
Pieter Vogt
Tutor: Florian Brandt

Abgabe: 07.07.2020, 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  class Output extends Thread {
2   public void run() {
3    synchronized (InOut.getLock()) {
4     if (!InOut.isEntered()) {
5      try {
6       InOut.getLock().wait();
7      } catch (InterruptedException e) {
8      }
9     }
10    System.out.println(InOut.getValue() * InOut.getValue());
11   }
12  }
13 }
14
15 class Input extends Thread {
16
17  public void run() {
18   synchronized (InOut.getLock()) {
19    InOut.setValue(IO.readInt("Value: "));
20    InOut.setEntered(true);
21    InOut.getLock().notify();
22   }
23  }
24 }
25
26 public class InOut {
27
28  private static Object lock = new Object();
29  private static boolean entered = false;
30  private static int value = 0;
31
32  public static Object getLock() {
33   return lock;
34  }
35
```

```
36  public static boolean isEntered() {
37   return entered;
38  }
39
40  public static void setEntered(boolean entered) {
41   InOut.entered = entered;
42  }
43
44  public static int getValue() {
45   return value;
46  }
47
48  public static void setValue(int value) {
49   InOut.value = value;
50  }
51
52  public static void main(String[] args) {
53   new Output().start();
54   new Input().start();
55  }
56
57 }
58
59 import java.util.concurrent.Semaphore;
60 class OutputThreaded extends Thread {
61     static Semaphore semaphore = new Semaphore(1);
62     public void run() {
63         try {
64             semaphore.acquire();
65             System.out.println(InOut.getValue() * InOut.
                   getValue());
66         } catch (InterruptedException e) {
67             e.printStackTrace();
68         }finally {
69             semaphore.release();
70         }
71     }
72 }
73
74 class InputThreaded extends Thread {
75     static Semaphore semaphore = new Semaphore(1);
76     public void run() {
77         try {
78             semaphore.acquire();
79             InOut.setValue(IO.readInt("Value: "));
80             InOut.setEntered(true);
81         } catch (InterruptedException e) {
82             e.printStackTrace();
83         }finally {
```

```
84              semaphore.release();
85          }
86      }
87  }public class InOutThreaded {
88      private static Object lock = new Object();
89      private static boolean entered = false;
90      private static int value = 0;
91      public static boolean isEntered() {
92              return entered;
93          }
94      public static void setEntered(boolean entered) {
95              InOutThreaded.entered = entered;
96          }
97      public static int getValue() {
98              return value;
99          }
100     public static void setValue(int value) {
101             InOutThreaded.value = value;
102         }
103     public static void main(String[] args) {
104
105         new Output().start();
106         new Input().start();
107     }
108 }
```

Aufgabe 2

```
1  import java.util.concurrent.BrokenBarrierException;
2  import java.util.concurrent.CyclicBarrier;
3
4  public class Barriers {
5
6   private final static int NUMBER = 3;
7   public static CyclicBarrier barrier = new CyclicBarrier(
       NUMBER);
8
9   public static void main(String[] args) {
10   NumberRunner[] runner = new NumberRunner[NUMBER];
11   for (int i = 0; i < NUMBER; i++) {
12    runner[i] = new NumberRunner(i);
13   }
14   for (int i = 0; i < NUMBER; i++) {
15    runner[i].start();
16   }
17  }
18
19 }
```

```
20  class NumberRunner extends Thread {
21
22   private int number;
23
24   public NumberRunner(int n) {
25    number = n;
26   }
27   @Override
28   public void run() {
29    for (int i = 0; i < 100; i++) {
30     for(int j=0; j<10;j++){
31      System.out.println("Thread " + number + ": " + i+j);
32     }
33     if(i%10==0){
34      try {
35       Barriers.barrier.await();
36      } catch (InterruptedException e) {
37       e.printStackTrace();
38      } catch (BrokenBarrierException e) {
39       e.printStackTrace();
40      }
41     }
42
43     try {
44      if(Barriers.barrier.await()==3) {
45       Barriers.barrier.reset();
46      }
47     } catch (InterruptedException e) {
48      e.printStackTrace();
49     } catch (BrokenBarrierException e) {
50      e.printStackTrace();
51     }
52    }
53    //in der aufgabe steht jeweils 10 ausgaben. Dürfen es auch
          weniger als 10 sein solange es nicht mehr als 10 sind?
54   }
55  }
```

Aufgabe 3

```
1      import java.util.Collection;
2  import java.util.Collections;
3  import java.util.Comparator;
4
5  public class KnapsackRecursive extends Knapsack {
6
7   public KnapsackRecursive(int capacity, Collection<Item>
       candidates) {
```

```
8     super(capacity, candidates);
9    }
10
11   @Override
12   public Selection pack() {
13    return new Selection();
14   }
15
16  }
17
18  import java.util.Collection;
19  import java.util.HashMap;
20  import java.util.Map;
21
22  public class KnapsackDynamic extends Knapsack {
23
24   public KnapsackDynamic(int capacity, Collection<Item>
        candidates) {
25    super(capacity, candidates);
26   }
27
28   @Override
29   public Selection pack() {
30    //TODO: implement this
31    return new Selection();
32   }
33
34  }
35
36  import java.util.ArrayList;
37  import java.util.Collection;
38  import java.util.Collections;
39  import java.util.Comparator;
40
41  public class KnapsackGreedy extends Knapsack {
42
43   public KnapsackGreedy(int capacity, Collection<Item>
        candidates) {
44    super(capacity, candidates);
45   }
46
47   @Override
48   public Selection pack() {
49    Selection select = new Selection();
50    Selection test = new Selection();
51    Collections.sort(candidates, new Comparator<Item>() {
52     @Override
53     public int compare(Item o1, Item o2) {
54      return o1.getValue()-o2.getValue();
```

```
55      }
56    });
57    Collections.reverse(candidates);
58    while(capacity!=0){
59     if(capacity>=30){
60      select = new Selection(test, candidates.get(1));
61      capacity=capacity-30;
62     }else if(capacity<30 && capacity>= 5){
63      select = new Selection(select, candidates.get(2));
64      capacity=capacity-5;
65     }else if(capacity<5 && capacity>=1){
66      select = new Selection(select, candidates.get(0));
67      capacity=capacity-1;
68     }
69    }
70    return new Selection(select);
71   }
72  }
```