

Autoren: Marius Birk
Pieter Vogt
Tutor: Florian Brandt

Abgabe: 09.06.2020, 12:00 Uhr

Smileys:

| A1 | A2 | A3 | Σ |
|----|----|----|----------|
| | | | |

Objektorientierte Modellierung und Programmierung

Abgabe Übungsblatt Nr.07

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

Aufgabe 1

```
1  public class LambdaTest {
2
3  public static void main(String[] args) {
4      // @SuppressWarnings("unchecked")
5      // Function<Double, Double> chain = makeChain(new
6          //     Function[]{inverse, id, timesTen, divideByPi});
7      // AUFGABE 1 B
8
9      // i) map x to itself
10     Function<Double> id = (x) -> x;
11     System.out.println(id.calculate(5.0));
12
13     // ii) map x to inverse self:
14     Function<Double> inverse = (x) -> x * -1;
15     System.out.println(inverse.calculate(5.0));
16
17     // iii) map x to x*10
18     Function<Double> timesTen = (x) -> x * 10;
19     System.out.println(timesTen.calculate(5.0));
20
21     // iv) map x to x/pi
22     Function<Double> divideByPi = (x) -> x / Math.PI;
23     System.out.println(divideByPi.calculate(5.0));
24
25     // AUFGABE 1 C
26     Function round = (x) -> Math.round(x.doubleValue());
27     System.out.println(round.calculate(5.5421235223));
28     System.out.println(round.calculate(5.5421235223).
29         getClass().toString());
30
31     // AUFGABE 1 D
32     Function testChain = (x) -> x.doubleValue();
33     Number a = inverse.calculate(id.calculate(timesTen.
34         calculate(divideByPi.calculate(5.0))));
```

```
33     }
34 }
35 /*
36     public Function makeChain(final Function[] funs, Number n)
37     {
38         Function c;
39         c =(x) -> funs [];
40
41         return c;
42     }
43 */
44 }
```

Aufgabe 2

```
1     import java.util.stream.Stream;
2     public class StreamTest {
3         public static void main(final String[] args){
4             final Stream<Integer> naturals = Stream.iterate(1, x
5                 ->x+1);
6             final Stream<Integer> integers = Stream.iterate(0, (
7                 Integer x) ->{
8                     if(x<=0){
9                         x=x*-1;
10                        x=x+1;
11                    }else{
12                        x=x*-1;
13                    }
14                    return x;
15                });
16
17             System.out.println("Naturals:" + filterAndSum(
18                 naturals));
19             System.out.print("Integers:" + filterAndSum(integers)
20                 );
21         }
22
23         public static int filterAndSum(Stream<Integer> stream) {
24             Stream<Integer> result = stream.filter(x -> x % 2 ==
25                 0).limit(10);
26             int result1 = result.mapToInt(Integer::intValue).sum
27                 ();
28             if (result != null) {
29                 return result1;
30             } else {
31                 return 0;
32             }
33         }
34     }
```

28 }

Aufgabe 3

```
1 import java.io.*;
2 import java.util.ArrayList;
3 import java.util.List;
4
5 class Person implements Serializable {
6     private String firstname;
7     private String lastname;
8     private String sortname;
9     public Person() { }
10    public Person(String firstname, String lastname) {
11        this.firstname = firstname;
12        this.lastname = lastname;
13        updateSortname();
14    }
15    public String getFirstname() {
16        return firstname;
17    }
18    public void setFirstname(String firstname) {
19        this.firstname = firstname;
20        updateSortname();
21    }
22    public String getLastname() {
23        return lastname;
24    }
25    public void setLastname(String lastname) {
26        this.lastname = lastname;
27        updateSortname();
28    }
29    public String getSortname() {
30        return sortname;
31    }
32    public void updateSortname() {
33        sortname = lastname + firstname;
34    }
35    @Override
36    public String toString() {
37        return firstname + " " + lastname + " (" + sortname + ")";
38    }
39    public static List<Person> load(String filename) throws
        IOException {
40        List<Person> persons = new ArrayList<>();
41        DataInputStream f = new DataInputStream(new
            BufferedInputStream(new FileInputStream(filename)));
42        try{
```

```
43     while(f != null){
44         try{
45             persons.add(load(f));
46         }catch (EOFException e){
47             break;
48         }
49     }
50 } catch( FileNotFoundException e){
51     System.out.print("File not found!");
52 }catch (IOException e){
53     System.out.print("End of File");
54 } catch (ClassNotFoundException e) {
55     e.printStackTrace();
56 }finally {
57     f.close();
58 }
59 return persons;
60 }
61 public static Person load(DataInputStream in) throws
62     IOException, ClassNotFoundException {
63     Person person = null;
64     ObjectInputStream o = new ObjectInputStream(in);
65     if(in != null){
66         person=(Person) o.readObject();
67     }
68     return person;
69 }
70 public static void save(String filename, List<Person> list)
71     throws IOException {
72     DataOutputStream f = new DataOutputStream(new
73         BufferedOutputStream(new FileOutputStream(filename)));
74     try{
75         for(int i = 0; i<list.size();i++){
76             save(f, list.get(i));
77         }
78     }catch(IOException e){
79         System.out.print(e);
80     }finally {
81         f.close();
82     }
83 }
84 public static void save(DataOutputStream out, Person person)
85     throws IOException {
86     ObjectOutputStream o = new ObjectOutputStream(out);
87     o.writeObject(person);
88 }
89 public static List<Person> unserialize(String filename)
90     throws IOException, ClassNotFoundException {
91     List<Person> persons = new ArrayList<>();
```

```
87     DataInputStream f = new DataInputStream(new
      BufferedInputStream(new FileInputStream(filename)));
88     ObjectInputStream o = new ObjectInputStream(f);
89     try{
90         while(f != null){
91             try{
92                 persons.add((Person)o.readObject());
93             }catch (EOFException e){
94                 break;
95             }
96         }
97     } catch( FileNotFoundException e){
98         System.out.print("File not found!");
99     }catch (IOException e){
100         System.out.print(e);
101     } catch (ClassNotFoundException e) {
102         e.printStackTrace();
103     }finally {
104         f.close();
105     }
106     return persons;
107 }
108 public static void serialize(String filename, List<Person>
      persons) throws IOException {
109     DataOutputStream f = new DataOutputStream(new
      BufferedOutputStream(new FileOutputStream(filename)));
110     ObjectOutputStream o = new ObjectOutputStream(f);
111     try{
112         for(int i = 0; i<persons.size();i++){
113             o.writeObject(persons.get(i));
114         }
115     }catch(IOException e){
116         System.out.print("Error initialize Output123");
117     }finally {
118         f.close();
119         o.close();
120     }
121 }
122 }
123 public class PersonTest {
124     public static void main(String[] args) throws IOException,
      ClassNotFoundException {
125         List<Person> persons = new ArrayList<>();
126         persons.add(new Person("Willy", "Wonka"));
127         persons.add(new Person("Charlie", "Bucket"));
128         persons.add(new Person("Grandpa", "Joe"));
129         System.out.println(persons);
130
131         Person.save("persons.sav", persons);
```

```
132     persons = Person.load("persons.sav");
133     System.out.println(persons);
134     Person.serialize("persons.ser", persons);
135     persons = Person.unserialize("persons.ser");
136     System.out.println(persons);
137 }
138
139 }
```