Autoren: Marius Birk
Pieter Vogt
Tutor: Florian Brandt

Abgabe: 09.06.2020, 12:00 Uhr

Smileys:

A1	A2	A3	$\sum$

Objektorientierte Modellierung und Programmierung

Abgabe Uebungsblatt Nr.07

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

Aufgabe 1

a) 0/1

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

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

d) 0/2

e) 0/1

Aufgabe 2

1)3/7

```
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
                ->x+1);
5           final Stream<Integer> integers = Stream.iterate(0, (
                Integer x) ->{
6               if(x<=0){
7                   x=x*-1;
8                   x=x+1;
9               }else{
10                  x=x*-1;
11              }
12              return x;
13          });
14
15          System.out.println("Naturals:" + filterAndSum(
                naturals));
16          System.out.print("Integers:" + filterAndSum(integers)
                );
17      }
18
19      public static int filterAndSum(Stream<Integer> stream) {
20          Stream<Integer> result = stream.filter(x -> x % 2 ==
                0).limit(10);
21          int result1 = result.mapToInt(Integer::intValue).sum
                ();
22          if (result != null) {
23              return result1;
24          } else {
25              return 0;
26          }
27      }
```

a) 1/1

b) 2/2

c) 3.5/4

2) 6.5/7

```
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
          IOException, ClassNotFoundException {
62      Person person = null;
63      ObjectInputStream o = new ObjectInputStream(in);
64      if(in != null){
65       person=(Person) o.readObject();
66      }
67      return person;
68     }
69     public static void save(String filename, List<Person> list)
          throws IOException {
70      DataOutputStream f = new DataOutputStream(new
           BufferedOutputStream(new FileOutputStream(filename)));
71      try{
72       for(int i = 0; i<list.size();i++){
73        save(f, list.get(i));
74       }
75      }catch(IOException e){
76       System.out.print(e);
77      }finally {
78       f.close();
79      }
80     }
81     public static void save(DataOutputStream out, Person person)
           throws IOException {
82      ObjectOutputStream o = new ObjectOutputStream(out);
83      o.writeObject(person);
84     }
85     public static List<Person> unserialize(String filename)
          throws IOException, ClassNotFoundException {
86      List<Person> persons = new ArrayList<>();
```

a) 0.5/1

b) 0/1

c) 1/1

d) 0/1

```
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;                                          e) 1/1
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();                                             f) 1/1
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);
```


3) 3.5/6

```
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 }
```