```
1 package uebung071;
2
3 public interface Function<R,T extends Number> {
4     R calculate(T wert1);
5 }
6
```

kann R auch eingegrenzt werden?

a) 1/1

```
 1 package uebung071;
 2
 3 public class LambdaTest {
 4     public static void main(String[] args) {
 5         //b
 6         // warum muss man den Typ die in <> steht
   bei den Lambda Ausdrückwn mit angeben
 7         Function<Number, Number> id = (x) -> x;
 8         Function<Number, Double> inverse = (x) ->
   Math.pow(x, -1);
 9         Function<Number, Double> timesTen = (x) ->
   x * 10;
10         Function<Number, Double> divideByPi = (x
   ) -> x / Math.PI;
11                                  b) 2/2
12         //c
13         Function<Long, Double> round = x -> Math.
   round(x);      c) 1/1
14
15         //e
16
17         @SuppressWarnings("unchecked")
18         Function <Double , Double > chain =
   makeChain(new Function [] { inverse , id ,
19                 timesTen , divideByPi });
20
21         double roundedResult = round.calculate(
   chain.calculate(5.5));
22
23         System.out.println(roundedResult);      e) 1/1
24
25
26         }       was heißt static denn? Dass die Methode unabhängig von einer Objektinstanz ist.
27         //d    Wäre die nicht static müsste immer ein LambdaTest Objekt erstellt werden,
28         // warum muss methode static sein?      obwohl dieses dann nie genutzt wird
29         static Function<Double, Double> makeChain(
   final Function<Double, Double>[] funs){
30             return x -> {
31                 double result = x;
32                 for (Function<Double, Double> fun
    : funs) {
```

irgenwo muss der angegeben werden damit der Compiler und die IDE wissen welcher Type gemeint ist, gilt auch für Listen und co.: Etwa List<Integer>

```
33                      result = fun.calculate(result);
34                  }
35                  return result;
36              };
37      }
38  }
39
```

d) 2/2

```java
package uebung072;

import java.util.stream.Stream;

public class StreamTest {
    public static void main(String[] args) {
        //a
        Stream<Integer> naturals = Stream.iterate(1, n -> n + 1);
        //b
        Stream<Integer> integers = Stream.iterate(0, n -> {
            if (n > 0) {
                n = n + 1;
            } else {
                n = (n * -1) - 1;
            }

            return n;
        });

        System.out.println("Ergebnis für naturals: " + filterAndSum(naturals));
        System.out.println("Ergebnis für integers: " + filterAndSum(integers));
    }
        //c
        public static Integer filterAndSum(Stream<Integer> stream) {
            return stream
                    .filter(n -> n % 2 == 0)
                    .limit(10)
                    .reduce(Integer::sum)
                    .orElse(0);
        }
    }


```

a) 1/1

funktioniert so nicht: hier kommt 0, -1, 0, -1... raus b) 1/2

c) 4/4

```java
package uebung073;

import java.io.*;
import java.util.ArrayList;
import java.util.List;

class Person {
    private String firstname;
    private String lastname;
    private String sortname;
    public Person() { }
    public Person(String firstname, String lastname) {
        this.firstname = firstname;
        this.lastname = lastname;
        updateSortname();
    }
    public String getFirstname() {
        return firstname;
    }
    public void setFirstname(String firstname) {
        this.firstname = firstname;
        updateSortname();
    }
    public String getLastname() {
        return lastname;
    }
    public void setLastname(String lastname) {
        this.lastname = lastname;
        updateSortname();
    }
    public String getSortname() {
        return sortname;
    }
    public void updateSortname() {
        sortname = lastname + firstname;
    }
    @Override
    public String toString() {
        return firstname + " " + lastname + " (" + sortname + ")";
```

```
40      }
41      public static List<Person> load(String filename
    ) throws IOException {
42          try {
43              DataInputStream in = new
    DataInputStream(new FileInputStream(filename));
44              List<Person> liste = new ArrayList<>();
45              int size = in.readInt();
46              for(int i = 0; i < size; i++ ){
47                  liste.add(load(in));
48              }
49              return liste;
50          } finally {
51              //was muss hier rein??
52          }
53      }
54      public static Person load(DataInputStream in)
    throws IOException {
55          String firstname = in.readUTF();
56          String lastname = in.readUTF();
57          return new Person(firstname, lastname);
58      }
59      public static void save(String filename, List<
    Person> list) throws IOException {
60
61          try {
62              DataOutputStream out = new
    DataOutputStream(new FileOutputStream(filename));
63              out.writeInt(list.size());
64              for (Person person : list) {
65                  save(out, person);
66              }
67
68          } finally {
69              // out.close(); Warum kann ich out
    nicht schließen???
70          }
71
72
73      }
74      public static void save(DataOutputStream out,
```

0,5/1 Stream nicht geschlossen

try with resources macht das einfacher

der InputStream muss geschlossen werden (auch im Fall einer Exception), das muss im finally Block erfolgen

1/1

0,5/1

weil out nicht in der richtigen Scope definiert wird, sondern erst im try block:
DataOutputStream out = null;
try {out = ...} finally { if (out != null) out.close()} oder besser try with resources

```
74 Person person) throws IOException {
75         out.writeUTF(person.getFirstname());
76         out.writeUTF(person.getLastname());
77     }
78     public static List<Person> unserialize(String
   filename) throws IOException,
   ClassNotFoundException {
79         try (ObjectInputStream in = new
   ObjectInputStream(new FileInputStream(filename
   ))) {
80             return (List<Person>) in.readObject();
81         }
82     }
83     public static void serialize(String filename,
   List<Person> persons) throws IOException {
84         try (ObjectOutputStream out = new
   ObjectOutputStream(new FileOutputStream(filename
   ))) {
85             out.writeObject(persons);
86         }
87     }
88 }
89 public class PersonTest {
90
91     public static void main(String[] args) throws
   IOException, ClassNotFoundException {
92         List<Person> persons = new ArrayList<>();
93         persons.add(new Person("Willy", "Wonka"));
94         persons.add(new Person("Charlie", "Bucket"
   ));
95         persons.add(new Person("Grandpa", "Joe"));
96         System.out.println(persons);
97
98         Person.save("persons.sav", persons);
99         persons = Person.load("persons.sav");
100        System.out.println(persons);
101        Person.serialize("persons.ser", persons);
102        persons = Person.unserialize("persons.ser"
   );
103        System.out.println(persons);
104    }
```

1/1

1/1

1/1

hier wurde korrekt try with resources verwendet, warum nicht vorher?

```
105
106 }
107
108
109
```