

Autoren: Marius Birk  
Pieter Vogt  
Tutor: Florian Brandt

Abgabe: 01.06.2020, 12:00 Uhr

Smileys:

A1	A2	A3	$\Sigma$

## Objektorientierte Modellierung und Programmierung

### Abgabe Uebungsblatt Nr.06

(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.*;
2 import java.util.function.Consumer;
3
4 public class HashMapMultiSet implements MultiSet{
5     public static void main(String[] args){
6         MultiSet<String, Integer> Test = (MultiSet<String,
7             Integer>) new HashMap<String, Integer>();
8     }
9
10    @Override
11    public Iterator iterator() {
12        return null;
13    }
14
15    @Override
16    public void forEach(Consumer action) {
17        List<Object> all = new ArrayList<>();
18
19        for (int i = 0; i < values.size(); i++)
20            for (int j = 0; j < frequency.get(i); j++)
21                all.add(values.get(i));
22
23        all.forEach(action);
24    }
25
26    @Override
27    public Spliterator spliterator() {
28        return null;
29    }
30
31    @Override
32    public void add(Object element) {
33        int index= values.indexOf(element);
34        int count = frequency.get(index);
```

```
35         int prevCount = 0;
36         if (index != -1) {
37             prevCount = frequency.get(index);
38             frequency.set(index, prevCount + count);
39         }
40         else if (count != 0) {
41             values.add(element);
42             frequency.add(count);
43         }
44
45     }
46
47     @Override
48     public int count(Object element) {
49         int index = values.indexOf(element);
50         int number = frequency.get(index);
51         return number;
52     }
53 }
54 interface MultiSet<T, I extends Number> extends Iterable {
55     public void add(T element);
56     public default int count(T element){
57         return 0;
58     }
59     public List<Object> values = null;
60     public List<Integer> frequency = null;
61
62 }
```

## Aufgabe 2

### Beispiel1

Aufgrund der Typeerasure ist es nicht möglich eine generische Exception zu werfen. Angenommen es würde durch einen Fehler im Compiler tatsächlich kompiliert, was würde passieren? Sobald eine Exception geworfen werden müsste, müsste sich die JRE für eine Exception entscheiden. Zum Zeitpunkt der Implementierung waren noch alle Exceptions voneinander unterscheidbar. Nun ist durch die Typeerasure des Compilers alles innerhalb der `{}`  entfernt und durch `Number` ersetzt da beide Klassen `Number` subclasses. Die JRE weiss nicht welche Exception geworfen werden muss und es kommt zum Laufzeitfehler.

### Beispiel2

## Aufgabe 3

```
1  import java.util.*;
2
3  public class StarkEnterprises implements Company{
4      private HashMap<Integer, String> employee = new HashMap
5          <>();
6      private HashMap<Integer, String> project = new HashMap
7          <>();
8      private HashMap<Integer, Integer> relation = new HashMap
9          <>();
10     public StarkEnterprises(){
11     }
12     @Override
13     public void addEmployee(int id, String name) throws
14         DuplicateIdException {
15         if(employee.containsKey(id)) {
16             throw new DuplicateIdException();
17         } else {
18             if (employee.containsValue(name)) {
19                 throw new DuplicateIdException();
20             }
21         }else{
22             employee.put(id, name);
23         }
24     }
25
26     @Override
27     public String getEmployeeName(int id) {
28         return employee.get(id);
29     }
30
31     @Override
32     public void addProject(int id, String name) throws
33         DuplicateIdException {
34         if(project.containsKey(id)){
35             throw new DuplicateIdException();
36         }else{
37             if(project.containsValue(name)){
38                 throw new DuplicateIdException();
39             }
40         }else{
41             project.put(id, name);
42         }
43     }
44
45     @Override
46     public String getProjectName(int id) {
47         return project.get(id);
48     }
49 }
```

```
45     @Override
46     public void assignEmployeeToProject(int employeeId, int
47         projectId) throws UnknownIdException {
48         try{
49             relation.put(employeeId, projectId);
50         }catch(Exception e){
51             throw new UnknownIdException();
52         }
53     }
54
55     @Override
56     public void removeEmployeeFromProject(int employeeId, int
57         projectId) throws UnknownIdException {
58         try{
59             relation.remove(employeeId, projectId);
60         }catch(Exception e){
61             throw new UnknownIdException();
62         }
63     }
64
65     @Override
66     public Collection<Integer> getEmployees() {
67         List<String> employeeByValue = new ArrayList(employee
68             .values());
69         Collections.sort(employeeByValue);
70         List<String> employeeName = new ArrayList<>(employee.
71             values());
72         List<Integer> employeeId = new ArrayList(employee.
73             keySet());
74         List<Integer> sorted = new ArrayList<>();
75         for(int i = 0; i<employeeByValue.size();i++){
76             for(int j =0; j<employeeName;j++){
77                 if(employeeByValue.get(i).equals(employeeName
78                     .get(j))){
79                     employeeId.add(j);
80                 }
81             }
82         }
83         return employeeId;
84     }
85
86     @Override
87     public Collection<Integer> getProjectsForEmployee(int
88         employeeId) throws UnknownIdException {
89         List<Integer> ID= new ArrayList<>();
90         List<Integer> project = new ArrayList(relation.values
91             ());
92         for(int i =0; i<relation.size();i++){
93             if(employeeId==relation.get(i)){
```

```
86         ID.add(project.get(i));
87     }
88 }
89 List<String> name = new ArrayList<>();
90 for(int i = 0; i<ID.size();i++){
91     for(int j = 0; j<project.size();j++){
92         if(ID.get(i) == project.get(j)){
93             name.add(this.getProjectName(j));
94         }
95     }
96 }
97 return ID;
98 }
99 }
100 class DuplicateIdException extends Throwable {
101
102 }
103
104 class UnknownIdException extends Throwable {
105
106 }

1 public class StarkTest {
2
3     public static void main(String[] args) {
4         Company stark = new StarkEnterprises();
5         try {
6             stark.addEmployee(0, "Tony");
7             stark.addEmployee(1, "Pepper");
8             stark.addEmployee(2, "Jarvis");
9             stark.addProject(0, "Suit");
10            stark.addProject(1, "Jarvis");
11            stark.addProject(2, "Jarvis");
12            stark.addProject(3, "Finances");
13            stark.assignEmployeeToProject(0, 0);
14            stark.assignEmployeeToProject(0, 1);
15            stark.assignEmployeeToProject(1, 3);
16            stark.assignEmployeeToProject(2, 0);
17            stark.assignEmployeeToProject(2, 2);
18            System.out.println(stark);
19        } catch (InvalidIdException e) {
20            System.out.println("Invalid ID: " + e.getId());
21        }
22    }
23
24 }
```