Autoren: Marius Birk
Pieter Vogt
Tutor: Florian Brandt

Abgabe: 01.06.2020, 12:00 Uhr

Smileys:

A1	A2	A3	$\sum$

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,
               Integer>) new HashMap<String, Integer>();
7      }
8
9      @Override
10     public Iterator iterator() {
11         return null;
12     }
13
14     @Override
15     public void forEach(Consumer action) {
16         List<Object> all = new ArrayList<>();
17
18         for (int i = 0; i < values.size(); i++)
19         for (int j = 0; j < frequency.get(i); j++)
20         all.add(values.get(i));
21
22         all.forEach(action);
23     }
24
25     @Override
26     public Spliterator spliterator() {
27         return null;
28     }
29
30     @Override
31     public void add(Object element) {
32         int index= values.indexOf(element);
33         int count = frequency.get(index);
34
```

```
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 subclassen. Die JRE weiss nicht welche Exception geworfen werden muss und es kommt zum Laufzeitfehler.

Beispiel2

In diesem Beispiel verhält es sich ähnlich wie im ersten Beispiel. Außer, dass in den Generics Datentypen verwendet wurden, die voneinander erben. Wenn versucht wird eine Exception mit dem Generic NNumberßu werfen und dies fehl schlägt, sodass der catch-Bereich in Kraft treten soll. Kann dies auch nicht geschehen, denn der Generic Integer erbt von Number. Der gesamte Block kann daher nicht funktionieren. Zur Laufzeit hin, verhält es sich genau wie in Beispiel eins. Grund ist wieder Typeerasure. Zur Laufzeit würde versucht werden eine GenericException zu werfen und im catch-Block zu greifen. Daher hätte der try-catch-Block keinen Effekt.

Aufgabe 3

```
1  import java.util.*;
2
3  public class StarkEnterprises implements Company{
4      private HashMap<Integer, String> employee = new HashMap
           <>();
5      private HashMap<Integer, String> project = new HashMap
           <>();
6      private HashMap<Integer, Integer> relation = new HashMap
           <>();
7      public StarkEnterprises(){
8      }
9      @Override
10     public void addEmployee(int id, String name) throws
           DuplicateIdException {
11         if(employee.containsKey(id)) {
12             throw new DuplicateIdException();
13         } else {
14             if (employee.containsValue(name)) {
15                 throw new DuplicateIdException();
16             }
17         }else{
18             employee.put(id, name);
19         }
20     }
21
22     @Override
23     public String getEmployeeName(int id) {
24         return employee.get(id);
25     }
26
27     @Override
28     public void addProject(int id, String name) throws
           DuplicateIdException {
29         if(project.containsKey(id)){
30             throw new DuplicateIdException();
```

```
31          }else{
32              if(project.containsValue(name)){
33                  throw new DuplicateIdException();
34              }
35          }else{
36              project.put(id, name);
37          }
38      }
39
40      @Override
41      public String getProjectName(int id) {
42          return project.get(id);
43      }
44
45      @Override
46      public void assignEmployeeToProject(int employeeId, int
            projectId) throws UnknownIdException {
47          try{
48              relation.put(employeeId, projectId);
49          }catch(Exception e){
50              throw new UnknownIdException();
51          }
52      }
53
54      @Override
55      public void removeEmployeeFromProject(int employeeId, int
            projectId) throws UnknownIdException {
56          try{
57              relation.remove(employeeId, projectId);
58          }catch(Exception e){
59              throw new UnknownIdException();
60          }
61      }
62
63      @Override
64      public Collection<Integer> getEmployees() {
65          List<String> employeeByValue = new ArrayList(employee
                .values());
66          Collections.sort(employeeByValue);
67          List<String> employeeName = new ArrayList<>(employee.
                values());
68          List<Integer> employeeId = new ArrayList(employee.
                keySet());
69          List<Integer> sorted = new ArrayList<>();
70          for(int i = 0; i<employeeByValue.size();i++){
71              for(int j =0; j<employeeName;j++){
72                  if(employeeByValue.get(i).equals(employeeName
                        .get(j))){
73                      employeeId.add(j);
```

```
74                  }
75              }
76          }
77          return employeeId;
78      }
79
80      @Override
81      public Collection<Integer> getProjectsForEmployee(int
           employeeId) throws UnknownIdException {
82          List<Integer> ID= new ArrayList<>();
83          List<Integer> project = new ArrayList(relation.values
               ());
84          for(int i =0; i<relation.size();i++){
85              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  }
```