

Musterlösung 7

Aufgabe 1 (Generische Datentypen)

Die Lösung ist auch zu finden unter:

<http://www.inf.fu-berlin.de/lehre/WS05/ALP3/loesungen/U7.A1.jar>

```
1 package U7.A1;
2
3 /**
4  * Abbildung von K auf V-Werte. "null" ist weder im Definitions- noch im
5  * Wertebereich erlaubt/enthalten.
6  */
7 public class MapImpl<K extends Comparable<K> & Clonable<K>,
8         V extends Comparable<V> & Clonable<V>>
9     implements Map<K, V>, Clonable<MapImpl<K, V>> {
10 private class Entry {
11     K key;
12
13     V val;
14
15     Entry next;
16
17     Entry(K k, V v) {
18         key = k;
19         val = v;
20     }
21 }
22
23 private Entry start; // erste Element der verketteten Liste
24
25 /**
26  * "val" unter dem Schlüssel "key" eintragen. Falls bereits ein Eintrag für
27  * "key" verzeichnet ist, ersetzt "val" den alten Wert.
28  *
29  * @throws IllegalArgumentException falls key oder val == null
30  *
31  */
32 public void enter(K key, V val) {
33     if (key == null || val == null)
34         throw new IllegalArgumentException("null-Werte nicht erlaubt");
35     Entry next = start, prev = null;
36     while (next != null && next.key.compareTo(key) < 0) {
37         prev = next;
38         next = next.next;
39     }
40     if (next != null && next.key.compareTo(key) == 0) {
41         next.val = val; // Update
42     } else {
43         Entry e = new Entry(key, val);
44         e.next = next;
45         if (prev == null)
46             start = e;
47         else
48             prev.next = e;
49     }
50 }
```

```

51
52  /**
53   * @return Bild-Wert von "key"
54   * @throws NotFound, falls kein Eintrag zu "key"
55   */
56  public V lookup(K key) throws NotFound {
57      Entry e = start;
58      while (e != null && e.key.compareTo(key) < 0)
59          e = e.next;
60      if (e != null && e.key.compareTo(key) == 0)
61          return e.val;
62      throw new NotFound();
63  }
64
65  /** erstellt tiefe Kopie */
66  public MapImpl<K, V> copy() {
67      MapImpl<K, V> res = new MapImpl<K, V>();
68      if (start == null)
69          return res;
70      Entry my = start, other;
71      other = res.start = new Entry(my.key.copy(), my.val.copy());
72      while (my.next != null) {
73          my = my.next;
74          other.next = new Entry(my.key.copy(), my.val.copy());
75          other = other.next;
76      }
77      return res;
78  }
79
80  /** prüft auf tiefe Gleichheit */
81  public boolean equals(Object o) {
82      if (!(o instanceof MapImpl))
83          return false;
84      if (o == this)
85          return true; // Abkürzung
86      Entry my = start, other;
87      try {
88          other = ((MapImpl<K, V>) o).start;
89          while (my != null
90              && other != null
91              && my.key.compareTo(other.key) == 0
92              && my.val.compareTo(other.val) == 0) {
93              my = my.next;
94              other = other.next;
95          }
96          return my == null && other == null;
97      } catch (ClassCastException e) {
98          return false;
99      }
100  }
101 }

```

Aufgabe 2 (Abbildung in Haskell)

<http://www.inf.fu-berlin.de/lehre/WS05/ALP3/loesungen/U7.A2.zip>

```
module Maps (Map(..)) where -- exportiere Map und alles was dazu gehört

class Map map where
  enter  :: (Ord k) => map k v -> k -> v -> map k v
  lookup2 :: (Ord k) => map k v -> k -> v -- es gibt schon ein "lookup"

module ListMaps(ListMap, newListMap) where

import Maps

data ListMap k v = M [(k,v)] deriving Eq

newListMap = M [] -- leere Abbildung

instance Map ListMap where
  enter (M l ) k v = M (doenter l k v) where
    -- findet rekursiv die richtige Stellung und fügt ein
    -- neues Tupel ein bzw. ersetzt den alten Wert mit v
    doenter ((k,v):l) k2 v2
      | k < k2 = (k,v):(doenter l k2 v2)
      | k == k2 = (k,v2):l
      | k > k2 = (k2,v2):(k,v):l
    doenter [] k v = [(k,v)]
  lookup2 (M ((k,v):l)) k2
    | k < k2 = lookup2 (M l) k2
    | k == k2 = v
    | otherwise = error "not found"
  lookup2 (M []) k = error "not found"

instance (Show k, Show v) => Show (ListMap k v) where
  show (M []) = ""
  show (M ((k,v):l)) = show k ++ " -> " ++ show v ++ "\n" ++ show (M l)
```

Aufgabe 3 (Parametrische Polymorphie)

```

6 |
7 | import java.util.Vector;
8 | class Person<Id> {
9 |     Id id;
10 |     String name;
11 |     Person(Id id, String name) {
12 |         this.id = id;
13 |         this.name = name;
14 |     }
15 |     <Id> void setId(Id i) {
16 |         id = i;
17 |     }
18 | }
19 | class Student<Key> extends Person<Key> {
20 |     int matr;
21 |     Student(int id, String name, int matr) {
22 |         super(id, name);
23 |         this.matr = matr;
24 |     }
25 |     public static void main(String[] args) {
26 |         Vector<Person<Integer>> persVerz = new Vector<Person<Integer>>();
27 |         Vector<Student<Integer>> studVerz = new Vector<Student<Integer>>();
28 |         persVerz.add(new Person<Integer>(23, "Hans"));
29 |         Person<Integer> otto = new Student<Integer>(24, "Otto", 12345);
30 |         studVerz.add(otto);
31 |         Student<? extends Integer> x =
32 |             new Student<Character>('a', "Klaus", 23456);
33 |         x = new Student<Integer>(26, "Karl", 34567);
34 |         studVerz.add(x);
35 |         personenAusgeben(studVerz);
36 |         studentenAusgeben(studVerz);
37 |         persVerz = studVerz;
38 |         personenAusgeben(persVerz);
39 |     }
40 |     static void personenAusgeben(Vector<Person<Object>> v) {
41 |         for (int i = 0; i < v.size(); i++)
42 |             System.out.println(i + ": " + v.get(i).name);
43 |     }
44 |     static void studentenAusgeben(Vector<?> v) {
45 |         for (int i = 0; i < v.size(); i++)
46 |             System.out.println(i
47 |                 + ": "
48 |                 + v.get(i).name
49 |                 + " ("
50 |                 + v.get(i).matr
51 |                 + ")");
52 |     }
53 | }

```

Zeile	Fehler
15	Durch den vorangestellten Typparameter <ID> erhält man eine generische Methode bezüglich eines neuen Typparameters ID, der das ursprüngliche ID verdeckt. Es ist fast so, als hätte man durch eine lokale Variable ein Attribut verdeckt. Der Übersetzer beschwert sich deshalb in Zeile 16 über zwei inkompatible Typen mit selbem Namen.
21-22	Da die Klasse Student<Key> von Person<Key> abgeleitet ist, lautet

	<p>die Signatur des Konstruktors aus Person<ID> folgendermaßen: Person(Key, String). Diesen kann man also nicht mit einem Integer aufrufen, sondern nur mit einem Key. Es gibt zwei Lösungen für das Problem:</p> <p>Man ändere den Konstruktor von Student:</p> <pre>Student(Key id, String name, int matr) { super(id, name); this.matr = matr; }</pre> <p>oder man leitet nicht von der Klasse Person<Key> ab, sondern von der Klasse Person<Integer> :</p> <pre>class Student extends Person<Integer> { int matr; Student(int id, String name, int matr) { super(id, name); this.matr = matr; } }</pre>
30	<p>Analog zu unserem Wissen von Typverträglichkeit: Der statische Typ von otto ist eine Person<Integer>, der dynamische Typ aber Student<Integer>. Da Student<Key> von Person<Key> abgeleitet wurde, ist ein Student mit beliebigem Key verträglich zu einer entsprechenden Person. Wie auch bisher gilt die Umkehrung nicht und man benötigt eine dynamische Typumwandlung.</p> <pre>studVerz.add((Student<Integer>) otto);</pre>
31	<p>Die Zuweisung ist nicht möglich, denn obwohl zwischen den primitiven Typen Verträglichkeit besteht, besteht keine zwischen den Hüllenklassen Character und Integer.</p>
34	<p>Typverträglichkeit besteht nur in eine Richtung: Student<Integer> ist Untertyp von Student<? extends Integer>. Umgekehrt gilt keine Verträglichkeit, da ein Student<? extends Integer> nicht in allen Situationen als Student<Integer> verwendet werden kann (es ist nur lesender Zugriff auf Methoden und Felder des Typparameters erlaubt).</p>
37	<p>Vector<Person<Integer>> ist nicht typverträglich mit Vector<Student<Integer>>. Für eingeschränkte Typverträglichkeit dienen in diesem Fall wieder ? extends (Lesezugriff) und ? super (Schreibzugriff).</p>
40	<p>Andere Typen wie Person<X> sind nicht verträglich mit Person<Object>, daher ist Person<Object> hier nicht als "möglichst allgemeiner" Parameter geeignet. Im Rumpf wird nur auf das name-Attribut von Person zugegriffen, dieses ist in beliebigen Unterklassen von Person mit beliebigem Typparameter verfügbar (sofern nicht verdeckt).</p>
44	<p>Es muss möglich sein auf name und matr zuzugreifen, was mit ? nicht</p>

möglich ist. Erst wenn wir den Typ besser einschränken, mittels `Vector<? extends Student<?>>`, können wir auf die Attribute zugreifen.

Korrigierte Version:

```
1 package U7.A3;
2
3 import java.util.Vector;
4
5 class Person<Id> {
6     Id id;
7     String name;
8     Person(Id id, String name) {
9         this.id = id;
10        this.name = name;
11    }
12    void setId(Id i) {
13        id = i;
14    }
15 }
16
17 public class Student<Id> extends Person<Id> {
18     int matr;
19     Student(Id id, String name, int matr) {
20         super(id, name);
21         this.matr = matr;
22     }
23     public static void main(String[] args) {
24         Vector<Person<Integer>> persVerz = new Vector<Person<Integer>>();
25         Vector<Student<Integer>> studVerz = new Vector<Student<Integer>>();
26         persVerz.add(new Person<Integer>(23, "Hans"));
27         Person<Integer> otto = new Student<Integer>(24, "Otto", 12345);
28         studVerz.add((Student<Integer>) otto);
29         Student<? extends Integer> x;
30         x = new Student<Integer>(26, "Karl", 34567);
31         studVerz.add((Student<Integer>) x);
32         personenAusgeben(studVerz);
33         studentenAusgeben(studVerz);
34         personenAusgeben(persVerz);
35     }
36     static void personenAusgeben(Vector<? extends Person<?>> v) {
37         for (int i = 0; i < v.size(); i++)
38             System.out.println(i + ": " + v.get(i).name);
39     }
40     static void studentenAusgeben(Vector<? extends Student<?>> v) {
41         for (int i = 0; i < v.size(); i++)
42             System.out.println(i + ": " + v.get(i).name + " (" + v.get(i).matr
43                 + ")");
44     }
45 }
```