

Übungsblatt 7 - Abgabe vor der Vorlesung 06.12.05

Aufgabe 1 (Generische Datentypen)

Gegeben seien folgende Schnittstellen:

```
/** eine Abbildung von Schlüsseln auf Werte */
interface Map <K,V> {
    void enter(K key, V val);
    V lookup(K key) throws NotFound;
}
interface Cloneable<T> {
    /** stellt eine TIEFE Kopie her */
    T copy();
}
```

Schreiben Sie eine generische Klasse `MapImpl`, die diese Schnittstellen implementiert. Ersetzen Sie in `MapImpl` die aus `Object` geerbte Methode `boolean equals(Object o)` derart, dass auf tiefe Gleichheit überprüft wird. Die Abbildung soll als verkettete Liste von (K,V) -Tupeln repräsentiert werden. Diese Liste soll nach den Schlüsselwerten sortiert sein, so dass nicht immer die gesamte Liste durchsucht werden muss. Schlüssel und Werte sollen über die Schnittstelle `java.lang.Comparable<T>` verglichen werden.

Schreiben Sie geeignete Tests und geben Sie einen Testlauf ab.

Schlagen Sie anschließend im Paket `java.util` nach und finden Sie die entsprechenden Klassen/Schnittstellen aus der Standardbibliothek.

Optional: Führen Sie einen Vergleich der Leistungsfähigkeit ihrer Datenstruktur und derjenigen der Sun API über zufälligen Daten aus.

Aufgabe 2 (Abbildung in Haskell)

Implementieren Sie eine Abbildung (`Map`) als abstrakten Datentyp in Haskell. Definieren Sie eine entsprechende Typklasse `Map` mit den Funktionen `enter` und `lookup` und sehen Sie eine Möglichkeit vor, eine neue leere Abbildung zu erzeugen. `Map` soll polymorph bezüglich der Schlüssel und der Daten sein. Die in Frage kommenden Schlüsselmenge sollen linear geordnet sein, und die Repräsentation sollte von dieser Ordnung Gebrauch machen (vorhandene Schlüssel sortiert halten!). Definieren Sie ein entsprechendes `show`.

Aufgabe 3 (Parametrische Polymorphie)

Finden Sie im folgenden Code alle Fehler und geben Sie einen Verbesserungsvorschlag an, um die Fehler sinnvoll zu beheben.

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

Java Wiederholung

Nach zehn Aufgaben beenden wir die Java-Wiederholung. Wer Interesse an weiteren Aufgaben hat, kann gerne am ACM Programming Contest Training teilnehmen.