Entwurfsmuster

Jan Fässler

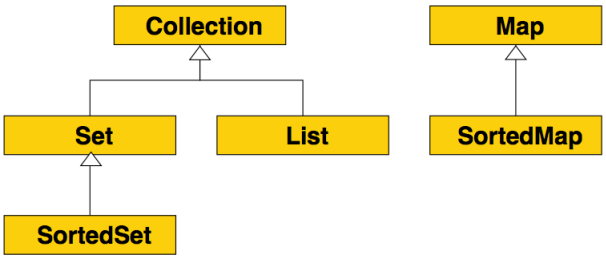
3. Semester (HS 2012)

If you use this documentation for a exam, you should offer a beer to the authors!

Inhaltsverzeichnis

1 Java Collection Framework

1.1 Die Interfaces



1.2 Der Iterator

Ein Iterator ist immer zwischen zwei Elemente. Es gibt also in einer Collection mit n Elementen, n + 1 mögliche Positionen an denen der Iterator stehen kann.

Listing 1: Iterator

```
1 interface Iterator<E> {
2   boolean hasNext();      // there is an element which can be jumped over
3   E next();               // returns the jumped over element
4   void remove();          // removes the last element returned by next
5 }
```

1.3 Die Implementierung

Interface	Implementation				Historical
Set	HashSet		TreeSet		
List		ArrayList		LinkedList	Vector Stack
Map	HashMap		TreeMap		Hashtable Properties

ArrayList
: Eine Implementierung welche ein Array darstellt bei dem man die grösse verändern kann.

LinkedList
: Eine doppelt verkettete Liste.

HashSet
: Die Elemente werden in einer Hash-Tabelle gespeichert.

TreeSet
: Die Elemente werden in einer Baumstruktur gespeichert.

Maps
: Eine Map ist wie ein Wörterbuch aufgebaut. Jeder Eintrag besteht aus einem Schlüssel (key) und dem zugehörigen Wert (value). Jeder Schlüssel darf in einer Map nur genau einmal vorhanden sein.

Wenn eine Klasse ein Interface implementiert, müssen immer alle Funktionen des Interface implementiert werden. In einer abstrakten Collection-Klasse können alle Funktionen bis auf zwei realisiert werden. Für das Hinzufügen und für den Iterator benötigt es Kentnisse über die Datenstruktur. Hier das ein Beispiel einer Abstrakten Klasse:

Listing 2: Abstract Collection

```
1  abstract class AbstractCollection<E> implements Collection<E> {
2    public abstract Iterator<E> iterator();
3    public abstract boolean add(E x);
4    public boolean isEmpty() { return size() == 0; }
5    public int size() {
6      int n = 0; Iterator<E> it = iterator();
7      while (it.hasNext()) {
8        it.next(); n++;
9      }
10     return n;
11   }
12   public boolean contains(Object o) {
13     Iterator<E> e = iterator();
14     while (e.hasNext()) {
15       Object x = e.next();
16       if(x == o || (o != null) && o.equals(x)) return true;
17     }
18     return false;
19   }
20   public void clear() {
21     Iterator<E> it = iterator();
22     while (it.hasNext()) {
23       it.next();
24       it.remove();
25     }
26   }
27   public boolean remove(Object o) {
28     Iterator<E> it = iterator();
29     while (it.hasNext()) {
30       Object x = it.next();
31       if(x == o || (o != null && o.equals(x))) {
32         it.remove();
33         return true;
34       }
35     }
36     return false;
37   }
38   public boolean containsAll(Collection<?> c) {
39     Iterator<?> it = c.iterator();
40     while (it.hasNext()) {
41       if(!contains(it.next())) return false;
42     }
43     return true;
44   }
45   public boolean addAll(Collection<? extends E> c) {
46     boolean modified = false;
47     Iterator<? extends E> it = c.iterator();
48     while (it.hasNext()) {
49       if(add(it.next())) modified = true;
50     }
51     return modified;
52   }
```