03 Iteratoren - Arbeitsblatt

1. Verschieden Möglichkeiten zum Iterieren

a) Sei `int sumOf(Collection<Integer> c)` eine Methode, die alle in der Collection `c` enthalten Elemente aufsummiert. Programmieren Sie die Methode auf mindestens zwei verschiedene Arten. Welches sind die Vor- und Nachteile?

b) Programmieren Sie eine Methode `void removeSecondElem(Collection<Integer> c, int v)`, die genau eines der Elemente `v` entfernt, falls es mehr als eines gibt: Wenn beim Aufruf `c` das Element `v` ein- oder keinmal enthält, soll `c` nicht verändert werden. Ansonsten wird die Anzahl enthaltener `v`-Elemente um eins reduziert. Nehmen Sie an, dass `c` insgesamt sehr viele Elemente enthält. Welche der gezeigten Zugriffs-Möglichkeiten benutzen Sie? Warum?

© Dr. Barbara Geissmann, Michael Henninger, Prof. Dr. Wolfgang Weck

c) Ordnen Sie die folgenden Eigenschaften einer oder mehreren der Möglichkeiten zu, mit denen auf alle Elemente einer Collection zugegriffen werden kann:

Eigenschaft	toArray	stream	forEach	Iterator
Es werden immer alle Elemente der Collection bearbeitet. Vorzeitiger Abbruch, wie z.B. bei der sequenziellen Suche, ist nicht möglich.				
Unterstützt funktionales Programmieren mit immutable Collections.				
Es entsteht eine vollständige Kopie der Datenstruktur. Diese kann verändert werden, ohne dass die originale Datenstruktur davon beeinflusst wird.				
Einzelne bearbeitete Elemente können bei Bedarf direkt aus der Collection entfernt werden.				
Der Aufwand sowohl für Speicher als auch für Laufzeit entspricht O(n), wobei n die Anzahl der Elemente in der Collection ist.				
Erlaubt unter gewissen Randbedingungen sehr leicht Multi-Core Prozessoren durch parallele Verarbeitung auszunutzen.				

2. Einfacher ListIterator

Untenstehend finden Sie eine zu einfache Art, einen `Iterator` auf einer verlinkten Liste zu implementieren. Warum ist diese Implementierung nicht gut?

```
class MyIterator<E> implements Iterator<E> {
    private List<E> list;
    private int next = 0;

    MyIterator(List<E> list) { this.list = list; }

    public boolean hasNext() { return next < list.size(); }

    public E next() { return list.get(next++); }

    public void remove() { … }
}
```