

Übungen zu Algorithmen

Wintersemester 2003/2004

Blatt 11

Aufgabe 11.1 (20 Punkte)

Betrachten Sie die Klasse `BackListe` in `/home/ainf/Uebung/Blatt11/BackListe.java`,
und vervollständigen Sie den Code der Methode

```
/** Macht das Element vor dem aktuellen zum aktuellen.
 *
 * @return true, falls Rueckschritt geklappt hat
 */
public boolean back()
```

so daß beim Testen der Klasse `BackListe` mit der unveränderten Klasse aus
`/home/ainf/Uebung/Blatt11/BackListeTest.java` **keine** Fehlermeldungen ent-
stehen. Ändern Sie **keine** anderen Klassen.

Musterlösung vom 14.01.2004:

```
/* BackListe_loesTest.java */
import AlgoTools.IO;

/**
 * @version      22.12.2003
 * @author      Ralf Kunze Olaf Mueller
 *
 * Programmbeschreibung: Testet die Loesung der Klasse
 *                      BackListe mit Integers
 */

public class BackListe_loesTest {

    public static void main (String argv[]) {

        BackListe_loes l = new BackListe_loes(); // instanziiere BackListe
        int i; // deklariere int

        IO.println("Bitte Integers eingeben (Abbruch mit 0):");
```

```

i = IO.readInt();

while(i != 0) {
    l.insert(new Integer(i));           // fuege in Liste ein
    l.advance();                       // vorruecken
    i = IO.readInt();
}

IO.println("Ausgabe der gesamten Liste:");
l.reset();                             // An den Anfang
while(!l.endpos()) {                  // Bis zum Ende
    IO.println(((Integer)l.elem()).intValue()); // alle Elemente ausgeben
    l.advance();                       // vorruecken
}

IO.println("Ausgabe in umgekehrter Reihenfolge:");
while(l.back()) {                    // Alle Elemente von hier
    IO.println(((Integer)l.elem()).intValue()); // bis zurueck zum Anfang
}                                     // ausgeben

IO.print("Zusaetzlicher Rueckschritt zum Testen ergibt das");
IO.print(" Element: ");
l.back();                             // noch ein Rueckschritt
                                        // hier darf kein
                                        // Fehler auftauchen

if (!l.empty())
    IO.print(((Integer)l.elem()).intValue()); // Zahl ausgeben
                                        // hier auch nicht

IO.println();
while (!l.empty()) l.delete();        // Liste loeschen
                                        // Nochmal
                                        // Rueckschritt versuchen
if(l.back()) {                        // Falls Rueckschritt moegl.
    IO.println(((Integer)l.elem()).intValue()); // Zahl ausgeben
}                                       // Auch hier darf kein
                                        // Fehler auftauchen
}
}

/***** BackListe_loes.java *****/

/**
 * @version      22.12.2003
 * @author      Ralf Kunze Olaf Mueller
 *
 * Programmbeschreibung: erweitert ADT Liste um back
 */

public class BackListe_loes extends VerweisListe {

```

```

/** Macht das Element vor dem aktuellen zum aktuellen.
 * Liefert true, wenn der Rueckschritt moeglich war; sonst false
 *
 * @return true, falls Rueckschritt geklappt hat
 */

public boolean back() {

    if (empty()) // leere Liste
        return false; // -> kein Rueckschritt moeglich

    else {
        Object token = new Object(); // ein Markerobjekt erzeugen
        int counter = 0;

        insert(token); // fuege Objekt als Markierung ein
        reset(); // Ruecksetzen der Liste
        while (! (token == elem())) { // zaehle Listenelemente, die
            advance(); // vor der Markierung existieren
            counter++;
        }
        delete(); // loesche Markerobjekt

        if (counter == 0) // war schon am Anfang ?
            return false; // -> kein Rueckschritt moeglich

        else { // ruecke von Anfang counter-1
            reset(); // Schritte vor
            for (int i=1; i<counter; i++)
                advance();
            return true; // neue Position erreicht
        }
    }
}
}
}

```

Aufgabe 11.2 (30 Punkte)

Implementieren Sie den Quicksort-Algorithmus iterativ — also **nicht** rekursiv — in der Klasse `QSortIter`. Anstatt des Rekursionsaufrufs werden in einer Schleife die beiden Grenzen der noch zu sortierenden Teilfolgen bis zur Weiterverarbeitung in einem *Keller* abgelegt. Sie werden bei Bedarf wieder heruntergeholt. Auf diese Art und Weise wird der Laufzeitkeller der Rekursion nachgebildet; erklären Sie ihrem Tutor was damit gemeint ist.

Verwenden Sie `/home/ainf/Vorlesung/VerweisKeller.java`

Testen Sie Ihre Klasse, indem Sie in der `main`-Methode der Klasse `Aufg11_2` eine Folge von `ints` einlesen und diese sortiert wieder ausgeben.

Hinweis: Der Keller kann nicht mit `ints` arbeiten, sondern verlangt Objekte. Daher sollen auf dem Keller Objekte der Klasse `Integer` abgelegt werden. Wie ein `Integer` Objekt erzeugt und davon

wieder ein int erhalten werden kann, ist der Dokumentation der Klasse Integer zu entnehmen.

Musterlösung vom 14.01.2004:

```

/***** QsortIter.java *****/
import AlgoTools.IO;

/**
 * @version      22.12.2003
 * @author      Ralf Kunze  Olaf Mueller
 *
 * Programmbeschreibung: Implementiert Quicksort iterativ
 * unter Verwendung eines Kellers, in dem
 * die Grenzen der noch zu
 * sortierenden Teilfolgen abgelegt werden
 * und von dem diese zur
 * Weiterverarbeitung wieder heruntergeholt werden.
 */

public class QSortIter {

    /**
     * Sortiert das Array a innerhalb der Grenzen unten und oben mit dem
     * iterative Quicksort
     *
     * @param a Zu sortierendes Array
     * @param unten Untergrenze
     * @param oben Obergrenze
     */
    public static void sort(int[] a, int unten, int oben){
        int i,j,x,hilf;

        Keller k=new VerweisKeller();

        k.push(new Integer(oben));           // die Grenzen werden auf den
        k.push(new Integer(unten));         // Keller gelegt

        while (!k.empty()){                 // solange noch was zu sortieren ist
                                           // Grenzen vom Keller holen und aus
                                           // Integer-Objekt den Wert rausholen

            i=unten =((Integer)k.top()).intValue();
            k.pop();                         // und vom Keller räumen
            j=oben  =((Integer)k.top()).intValue();
            k.pop();

            x=a[(i+j)/2];                    // Vergleichselement

            do{
                while(a[i]<x) i++;
                while(a[j]>x) j--;           // x fungiert als Bremse
                if (i<=j){
                    hilf=a[i];             // tauschen

```



```

        IO.println();
    }
}

```

Aufgabe 11.3 (30 Punkte)

Implementieren Sie das Interface des abstrakten Datentyps *Liste* aus `/home/ainf/Vorlesung/Liste.java` mit Hilfe des ADT *Keller* in einer Klasse *KellerListe*, in der die Listenoperationen unter Verwendung der Keller-Operationen umgesetzt werden. Speichern Sie dazu alle Listenelemente, die **vor** dem aktuellen Element liegen, in einem VerweisKeller *k1* und alle restlichen Listenelemente in einem VerweisKeller *k2*. Die *KellerListe* soll Instanzen der Klasse *Object* verwalten können.

Achten Sie darauf, daß nur Methoden oder Variablen außerhalb der Klasse sichtbar sind, die auch wirklich von außen zugänglich sein müssen (Stichwort: *Information Hiding*). Warum ist dies sinnvoll?

Testen Sie Ihre *KellerListe* mit einer von Ihnen angepaßten Version der Klasse `/home/ainf/Vorlesung/VerweisListeTest.java`.

Zusätzlich dokumentieren Sie die Klasse *KellerListe* javadoc-konform. Dazu gehört die Angabe der Version, des Autors und die Programmbeschreibung. Ausserdem sollen die Rückgabewerte, die formalen Parameter und die *RuntimeExceptions* javadoc-konform dokumentiert werden. Suchen Sie sich dazu die entsprechenden *Tags* (wie z.B. `@version`) aus der javadoc-Dokumentation. Erzeugen Sie dann mit dem Kommando `javadoc` eine HTML-Seite mit der Dokumentation.

Hinweis:

Sichern Sie die Methoden gegen Fehlbedienung analog zur Vorgehensweise im Skript Abschnitte 9.1 und 9.2. Falls ein Fehler auftritt, werfen Sie eine *RuntimeException*. Die Fehlermeldungen dürfen nicht erkennen lassen, dass der Listenimplementation die Klasse *VerweisKeller* zugrunde liegt.

Musterlösung vom 14.01.2004:

```

/***** KellerListe.java *****/
import AlgoTools.IO;

/**
 * Implementiert den ADT Liste mit
 * Hilfe des ADT Keller
 *
 * Die Doku wurde mit dem Kommando
 * <tt>javadoc -version -author -d doku KellerListe</tt> erzeugt.
 *
 * @version      22.12.2003
 * @author       Ralf Kunze  Olaf Mueller
 */

public class KellerListe implements Liste {

```

```

private Keller k1;
private Keller k2;

/**
 * Erzeugt eine leere Kellerliste
 */
public KellerListe() {
    k1 = new VerweisKeller();
    k2 = new VerweisKeller();
}

/**
 * Die Methode testet, ob eine Kellerliste leer ist.
 *
 * @return true, falls die Kellerliste leer ist
 */
public boolean empty() {
    return k1.empty() && k2.empty();
}

/**
 * Die Methode testet, ob man sich am Ende einer Kellerliste befindet.
 *
 * @return true, falls am Ende der Kellerliste
 */
public boolean endpos() {
    return k2.empty();
}

/**
 * Fuegt ein Objekt in die Kellerliste ein.
 *
 * @param x Das einzufuegende Objekt
 */
public void insert(Object x) {
    k2.push(x);
}

/**
 * Geht in der Liste einen Eintrag weiter.
 *
 * @throws RuntimeException falls bereits am Ende der Liste.
 */
public void advance() {
    if(endpos())
        throw new RuntimeException("Liste.advance: am Ende");
    else {
        k1.push(k2.top());
        k2.pop();
    }
}

/**

```

```

    * Nach der Methode wieder am Anfang der Liste
    */
public void reset() {
    while(!k1.empty()) {
        k2.push(k1.top());
        k1.pop();
    }
}

/**
 * Liefert das Objekt an der entsprechenden Stelle in der Kellerliste.
 *
 * @throws RuntimeException falls die Liste leer ist.
 * @return Object, welches sich an der aktuellen Stelle befindet.
 */
public Object elem() {
    if(k2.empty())
        throw new RuntimeException("Liste.elem: kein aktueller Listeneintrag");
    return k2.top();
}

/**
 * Loescht das Element an der aktuellen Stelle.
 *
 * @throws RuntimeException falls die Liste leer ist.
 */
public void delete() {
    if(k2.empty())
        throw new RuntimeException("Liste.delete: kein aktueller Listeneintrag");
    else
        k2.pop();
}
}

/***** KellerListeTest.java *****/
import AlgoTools.IO;

/**
 * @version      22.12.2003
 * @author       Ralf Kunze  Olaf Mueller
 *
 * Programmbeschreibung: Testet die Klasse KellerListe mit Studenten
 */

public class KellerListeTest {

    public static void main (String argv[]) {

        KellerListe l = new KellerListe();           // KellerListe erz.
        Student s;                                   // deklariere Student

        s = new Student("Erika", "Muster", 22, 8, 1979, "INF", 2000); // erzeuge Student
    }
}

```

```

l.insert(s); // fuege in Liste ein
l.advance(); // eins weiter in l

s = new Student("Willi", "Wichtig", 18, 2, 1960, "MAT", 2002); // erzeuge Student
l.insert(s); // fuege in Liste ein
l.advance(); // eins weiter in l

s = new Student("Hein", "Bloed", 18, 5, 1981, "ASW", 2001); // erzeuge Student
l.insert(s); // fuege in Liste ein
l.advance(); // eins weiter in l

s = new Student("Susi", "Sorglos", 10, 7, 1978, "BMI", 2000); // erzeuge Student
l.insert(s); // fuege in Liste ein

l.reset(); // an den Anfang

while (!l.endpos()) { // 1., 3., 5.. loeschen
    l.delete();
    if (!l.endpos())
        l.advance();
}

// l.advance(); // Fatal Error: in advance am Ende

l.reset(); // an den Anfang

while (!l.endpos()) { // Liste ausgeben
    IO.println(((Student)l.elem()).vorname+" "+((Student)l.elem()).nachname);
    l.advance(); // ein weiter in l
}
l.reset(); // an den Anfang

while (!l.empty()) l.delete(); // Liste loeschen
//l.delete(); // Fatal Error: in delete: Kein aktueller Listeneintrag!

// l.elem(); // Fatal Error: in elem: Kein aktueller Listeneintrag!
}
}

```

Aufgabe 11.4 (20 Punkte)

a) Implementieren Sie in einer Klasse `ArrayKeller` den ADT Keller mit Hilfe eines Arrays nach der Skizze aus Abschnitt 9.2 des Skriptes. Nutzen Sie hierzu das Interface `Keller` aus `/home/ainf/Vorlesung/Keller.java`.

Die Größe des Kellers soll bei der Instanziierung eines `ArrayKeller`-Objektes angegeben werden können.

Sorgen Sie für eine geeignete Fehlerbehandlung für den Fall, dass zuviele Elemente in den `ArrayKeller` eingefügt werden.

Dokumentieren Sie die Klasse `ArrayKeller` javadoc-konform analog zur Klasse `KellerListe` aus Aufgabe Aufgabe 11.3 und erstellen Sie mit dem Kommando `javadoc`

eine HTML-Seite mit der Dokumentation.

b) Testen Sie Ihren ArrayKeller mit einer von Ihnen angepaßten Version der Klasse Reverse aus /home/ainf/Vorlesung/Reverse.java (s. Abschnitt 9.2 des Skriptes). Wählen Sie hierzu eine ArrayKeller-Größe von zehn Elementen.

Musterlösung vom 14.01.2004:

```
/* ***** ArrayKeller.java * ***** */
/**
 * Implementation des Interfaces vom
 * ADT Keller mit einem nicht dynamischen Array.
 *
 * Die Doku wurde mit dem Kommando
 * <tt>javadoc -version -author -d doku ArrayKeller</tt> erzeugt.
 *
 * @version      22.12.2003
 * @author       Ralf Kunze   Olaf Mueller
 */

public class ArrayKeller implements Keller {

    private static final int N=10; // Standard Kellergroesse

    private Object[] stack;        // Array fuer die Elemente
    private int topindex;          // Kellerposition und -maechtigkeit

    /**
     * Erzeugt einen Leeren Keller in Standardgroesse.
     */
    public ArrayKeller() {
        this(N);
    }

    /**
     * Erzeugt einen Keller variabler Groesse. Der Keller ist jedoch nicht
     * dynamisch erweiterbar.
     *
     * @param n Groesse des Kellers
     */
    public ArrayKeller(int n) {
        stack = new Object[n]; // Platz besorgen
        topindex = 0;          // ganz unten anfangen
    }

    /**
     * Testet, ob der Keller leer ist
     *
     * @return true falls Keller leer
     */
    public boolean empty() {
```

```

    return (topindex == 0);
}

/**
 * Liefert das oberste Element im Keller zurueck
 *
 * @return Das oberste Objekt
 * @throws RuntimeException falls der Keller leer ist
 */
public Object top() {
    if (empty()) { // Falls leer: Fehlermeldung
        throw new RuntimeException("Keller.top: Keller leer");
    }

    return(stack[topindex - 1]); // topindex zeigt 1 ueber Top-Element
}

/**
 * Entfernt das oberste Element vom Keller.
 *
 * @throws RuntimeException falls der Keller leer ist.
 */
public void pop() {
    if (empty()) {
        throw new RuntimeException("Keller.pop: Keller leer");
    }

    topindex--; // Vorletztes Element wird Top-Element
    stack[topindex] = null; // Verweis auf altes Top-Element loeschen
}

/**
 * Legt ein Objekt auf den Keller.
 *
 * @param o Objekt, welches auf den Keller gelegt werden soll.
 * @throws RuntimeException falls der Keller bereits voll ist.
 */
public void push(Object o) { // Legt ein neues Element auf den Keller
    if (topindex>stack.length-1){ // Falls kein Platz mehr: Fehlermeldung
        throw new RuntimeException("Keller.push: Keller voll");
    }

    stack[topindex] = o; // Objekt merken
    topindex++; // neues Element wird zum Top-Element
}
}

/***** ArrayReverse.java *****/
import AlgoTools.IO;

/**
 * @version 22.12.2003

```

```

*   @author          Ralf Kunze   Olaf Mueller
*
*   Programmbeschreibung: Liest eine Folge von ganzen Zahlen ein
*                       und gibt sie in umgekehrter Reihenfolge wieder aus.
*                       Verwendet wird die Wrapper-Klasse Integer,
*                       welche Objekte vom einfachen Typ int enthaelt.
*                       Vor dem Einfuegen in den Keller werden mit
*                       new Integer diese Objekte
*                       erzeugt, nach dem Auslesen aus dem Keller
*                       werden sie nach int gecastet.
*
*                       Es wird die Array-Implementation des Keller genutzt.
*/

public class ArrayReverse {

    public static void main (String argv[]) {

        ArrayKeller k = new ArrayKeller();           // lege leeren Keller an

        int[] a = IO.readInts("Bitte Zahlenfolge:  "); // lies Integer-Folge ein

        for (int i=0; i<a.length; i++)               // pushe jede Zahl als
            k.push(new Integer(a[i]));               // Integer-Objekt

        IO.print("Umgekehrte Reihenfolge:");
        while (!k.empty()) {                          // solange Keller nicht leer
            IO.print(" "+((Integer)k.top()).intValue()); // gib Top-Element aus
            k.pop();                                   // entferne Top-Element
        }
        IO.println();
    }
}

```