Algorithmen und Programmierung II 14.03.07 Prof. Dr. E, Ehses

---

Name: Matr.Nr.:

Studiengang: Medieninformatik (BA2)

Vorbemerkungen:

- Die Bearbeitungszeit beträgt 60 Minuten. Sie können maximal 50 Punkte erreichen. Hilfsmittel sind nicht zugelassen!
- Die Klausur umfasst insgesamt 5 Seiten. Überprüfen Sie bitte *sofort* die Vollständigkeit!
- Geben Sie die Lösung auf dem jeweiligen Aufgabenblatt an. Bei Platzmangel können Sie auch die dazugehörige Rückseite verwenden.
- Lesen Sie die Aufgabenstellung jeweils gut und in Ruhe durch. An sich richtige Lösungen, die nicht der Aufgabenstellung entsprechen, werden nicht gewertet.
- Viel Erfolg!

Punktzahl:

Aufgabe 1:	/22=	/4+	/4+	/ 5 +	/5 +	/4
Aufgabe 2:	/28=	/5+	/6+	/6/ +	/ 6 +	/5

Summe: /50

Note:

Aufgabe 1 Objektorientierung

a) Was genau bewirkt eine Importanweisung?. Und was ist der Unterschied zwischen

```
import a.b.C und import a.b.*.
```

Für die nächsten Fragen dienen der folgende Algorithmus und die beiden Schnittstellen als Vorbild:

```
public static double mDouble(double[] a) {
  double m = a[0];
  for (int i = 0; i < a.length; i++)
      if (m < a[i]) m = a[i];
  return m;
}

public interface Comparable {
  /**
   * @return 0, wenn this == a, positive Zahl, wenn this > a,
        *            sonst negative Zahl
        */
       public int compareTo(Object a);
}

public interface Comparator {
     /**
      * @return 0, wenn a == b, positive Zahl, wenn a > b,
      *            sonst negative Zahl
      */
     public int compare(Object a, Object b);
}
```

b) Was tut der Algorithmus und wie lautet x nach
`double[] a = {-1.2, -10.4}; double x = mDouble(a)`?

c) Schreiben Sie eine Variante des Algorithmus, die für Arrays von Objekten zu gebrauchen ist (`Typ Object[]`). Es wird erwartet, dass jedes Arrayelement zu einer Klasse gehört, die Schnittstelle `Comparable` implementiert.

d) Schreiben Sie eine Klasse `StringLengthComparator`, die die Schnittstelle `Comparator` implementiert. Und für sich festlegt, dass ein längerer String als größer als ein kürzerer String betrachtet wird.

e) Welche Ausnahme erzeugen im Allgemeinen die Methoden `compare` und `compareTo`, wenn der Vergleich nicht möglich ist? Bitte begründen Sie die Antwort.

Aufgabe 2 Algorithmen

Wir haben für einen Listenknoten die folgende Deklaration (Listenende bei null-Referenz):

```
class DoubleNode {
    double value;
    DoubleNode nextNode;
}
```

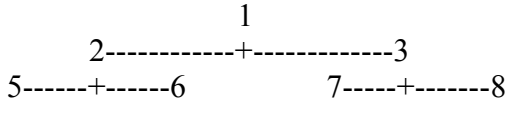
a) Schreiben Sie eine Methode, die die Summe aller Zahlenwerte in einer verketteten Liste zurückgibt..

```
public static double summe(DoubleNode anfang) {
```

b) Ergänzen Sie die folgende Methode passend für eine Klasse `LinkedList`. Die Klasse hat die Instanzvariable `DoubleNode first`, die auf das erste Listenelement zeigt.

```
public boolean equals(Object other) {
    if (! (other instanceof LinkedList) ) return false;
```

c) Geben Sie für den folgenden Baum an, in welcher Reihenfolge die Knoten bei Preorder-, Inorder- und bei Postorder-Traversierung bearbeitet werden.



Preorder:

Inorder:

Postorder:

d) Für einen Binärbaum gelten die folgenden Deklarationen:

```
class Node {
    Comparable key;
    Object value;
    Node left;
    Node right;
}

private Node root;
```

Der Baum ist so organisiert, dass alle key-Werte des linken Unterbaums kleiner sind als der key des aktuellen Knotens und alle key-Werte des rechten Unterbaums größer sind. Größer und kleiner meint in Bezug auf das Ergebnis der Methode compareTo. Die Suche starte immer mit dem durch root referierten Knoten (bei leerem Baum ist root gleich null). Die Suche soll den Inhalt des value-Attributs des gefundenen Knoten zurückgeben oder null, wenn der gesuchte Knoten nicht zu finden ist.

Vervollständigen Sie die Methode get:

```
Object get(Object key) {
```

e) Welche mittlere Komplexität (Rechenzeitaufwand) haben die folgenden Algorithmen (die Problemgröße n ist jeweils gleich der Anzahl der vorhandenen Datenelemente):

lineare Suche in einem unsortierten Feld:

binäre Suche in einem sortierten Feld:

Suche in einem ausgeglichenen Suchbaum (s. Aufgabe d):

Einfaches Suchverfahren (z.B. Bubblesort):

Effizientes Sortierverfahren (z.B. Quicksort):