Institut für Informatik SS 2004
der Universität München
Prof. Dr. M. Hofmann
Dr. J. Johannsen, Dr. R. Matthes 17.6.04

Übungen zur Vorlesung
Informatik II

Blatt 8

Abgabe der Hausaufgaben bis spätestens am 28.6.04, 14:00 Uhr über
`http://lehre.tcs.ifi.lmu.de/info2/Abgabe/abgabe.php`,
Bearbeitung in Gruppen zu max. 3 Personen ist zulässig.

Bitte halten Sie sich an folgende Grundsätze zur Bearbeitung der Aufgaben. Die Bewertung Ihrer Abgaben kann andernfalls deutlich herabgesetzt werden.

Verwenden Sie genau die Signatur der Aufgabenstellung, insbesondere die Namen der Klassen und Methoden in der vorgegebenen Klein- und Großschreibung.

Verwenden Sie keine „magischen" Größen in Ihren Programmen, d. h., weisen Sie Konstanten (außer den Zahlen 0 und 1 und dem leeren String) immer einer mit `final` deklarierten und in Großbuchstaben geschriebenen „Variablen" zu und verwenden Sie anschließend nur diese. Wer gute Gründe hat für eine Abweichung von dieser Regel, der kommentiere seinen Code entsprechend.

Jede Klasse und `public`-Methode muß angemessen mit `javadoc`-Kommentaren dokumentiert werden.

Programmieraufgabe P-18 (`MergeSort.java`): **6 Punkte**
Programmieren Sie nach dem Vorbild der Quicksort-Implementierung aus der Vorlesung den Algorithmus Mergesort. Bei diesem Algorithmus wird das übergebene Array `a` in der Mitte geteilt, also bei `m = a.length/2`. Die Teilarrays `a[0..m]` und `a[m+1..a.length-1]` werden durch rekursive Aufrufe sortiert. Anschliessend werden die beiden sortierten Teilarrays zusammengeführt, d.h. so elementweise in ein neues Array `b` kopiert, dass dieses anschliessend sortiert ist. Dabei wird ausgenutzt, dass die Teilarrays bereits sortiert sind. Das kleinste noch nicht in `b` kopierte Element ist dann immer entweder das kleinste noch nicht verwendete Element des ersten Teilarrays `a[0..m]` oder das kleinste noch nicht verwendete Element des zweiten Teilarrays `a[m+1..a.length-1]`. (Dies mag Sie an ein unfaires "Reißverschlußsystem" im Straßenverkehr erinnern.) Schließlich muß `b` in `a` zurückkopiert werden.

Schreiben Sie dazu eine Klasse `MergeSort`, in der es eine statische Methode

```
public static void sortieren( Object[] , Comparator )
```

gibt, die das übergebene Array sortiert. Diese Sortierung soll auf Vergleichen mithilfe des übergebenen `Comparator`-Objektes basieren.

Bemerkung: Die obige Darstellung der Methoden-Signatur ohne Namen für die Parameter ist die übliche Praxis, die auch in der API verwendet wird.

Programmieraufgabe P-19 (`Median.java`): **8 Punkte**

Erstellen Sie eine Klasse `Median` mit einer statischen Methode

```
public static Object median( Object[] a, Comparator c )
```

Diese Methode soll einen *Median* des übergebenen Arrays `a` bestimmen, wobei das `Comparator`-Objekt `c` die Ordnung auf den Objekten vorgibt. Grob gesprochen, ist ein Median ein Objekt, das sowohl mindestens so groß ist wie die Hälfte der Objekte des Arrays als auch höchstens so groß wie die Hälfte der Objekte des Arrays. Genauer legen wir fest: Hat das Array `a` die Länge n, dann ist ein Median das Objekt `a[`$n/2$`]`, wenn `a` zuvor aufsteigend sortiert wurde. *Bemerkung*: Sind die Elemente in `a` Zahlen, so ist ein Median (bzgl. der natürlichen Ordnung) ein Element x in `a`, das den Ausdruck $\sum_{i<n} |x - \mathtt{a}[i]|$ minimiert.

Diese Definition ist bereits eine triviale Implementierung (da Sie schon Sortiermethoden kennen), die Sie zum Testen verwenden können. Die verlangte Methode `median` darf den Array aus Effizienzgründen *nicht sortieren*. Stattdessen sollen Sie die Methode

```
public static int partition( Object[], int, int, Comparator )
```

aus der Klasse `Sortieren` von der Vorlesungshomepage zuhilfe nehmen. Da bei der Aufteilung des Arrays durch `partition` meistens nicht gerade die Mitte ausgeben wird, ist es nötig, ein allgemeineres Problem zu lösen (Stichwort: Einbettung, siehe Informatik I), nämlich ein i.-kleinstes Element auszuwählen, d. h., ein Element, das nach Sortierung des Arrays an Stelle i stehen würde. Dafür aber liefert `partition` eine gute Hilfe, da man danach nur noch rekursiv eine kleinere Instanz dieses allgemeineren Problems lösen muß. *Beispiel*: Man sucht man in einem Array `a` der Länge 50 das 17.-kleinste Element. Liefert `partition` auf `a[0..49]` den Wert 27, so sind alle Werte in `a[28..49]` größer oder gleich denen in `a[0..27]`, also muss das 17.-kleinste in `a[0..27]` liegen. Liefert dann `partition` auf `a[0..27]` den Wert 12, so muss analog das 17.-kleinste in `a[13..27]` liegen. Da die 13 Elemente in `a[0..12]` alle kleiner oder gleich diesen sind, ist das gesuchte Element also das 4.-kleinste in `a[13..27]`.

Hinweis: Der eigentliche Algorithmus würde in ca. 10 Zeilen Platz finden. (Dazu kommen natürlich Kommentare etc.) Sie brauchen über die Methode `partition` nur zu wissen, was in dem zugehörigen Kommentar (in Javadoc) steht.

Programmieraufgabe P-20 (`Zahlenleser.java`): **6 Punkte**

Erstellen Sie eine Klasse `Zahlenleser`, die Zahlen von der Konsole einzulesen gestattet. Wird für ein `Zahlenleser`-Objekt die Methode `liesInt()` aufgerufen, so gibt es einen Prompt auf der Konsole aus, das den Benutzer auffordert, einen Integer-Wert einzugeben. Der vom Benutzer eingegebene Wert soll als Ergebnis des Methodenaufrufs zurückgegeben werden. Gibt der Benutzer einen String ein, der keinen Integer-Wert repräsentiert, so soll er auf diesen Fehler aufmerksam gemacht und zu einer erneuten Eingabe aufgefordert werden. Will der Benutzer keinen Wert eingeben, so kann er dies durch eine leere Eingabe (mit `EOF` – unter Unix mit Ctrl-D einzugeben – oder Return) deutlich machen. Dem aufrufenden Programm wird dies mit der Methode `boolean zahlGelesen()` mitgeteilt, die dann `false` liefert (und sonst `true`).

Analog soll es eine Methode `liesDouble()` geben, die einen Double-Wert einzugeben gestattet.

Hinweis: die Methoden `Integer.parseInt(String)` und `Double.parseDouble(String)` lösen Ausnahmen (z.B. eine `NumberFormatException`) aus, wenn der eingegebene String keine Zahl des entsprechenden Typs repräsentiert.