Technische Universität Braunschweig

Institut für Programmierung und Reaktive Systeme

Programmieren II

Dr. Werner Struckmann

3. September 2007

Name:

Vorname:

Matrikelnummer: **Kennnummer:**

Anrede: ☐ Frau ☐ Herr

angestrebter Abschluss: ☐ Bachelor ☐ Master ☐ Diplom ☐ Magister

Fachrichtung: ☐ Informatik ☐ Wirtschaftsinformatik ☐ IST ☐ Mathematik

☐ Mobilität u. Verkehr ☐ Physik ☐ Maschinenbau ☐ Sonstiges: _____

Die Bearbeitungszeit beträgt 120 Minuten. Die Klausur besteht aus 5 Aufgaben. Sie haben die Klausur bestanden, wenn Sie mindestens 40 von 80 möglichen Punkten erreicht haben.

Aufgabe	1	2	3	4	5	Σ
max. Punkte	20	12	12	20	16	80
Punkte						

Note:

Bitte prägen Sie sich Ihre Kennnummer gut ein. Aus Datenschutzgründen wird das Klausurergebnis nur unter dieser Kennnummer bekannt gegeben. Aus den gleichen Gründen können Ergebnisse weder telefonisch noch per E-Mail mitgeteilt werden.

Die Ergebnisse der Klausur erfahren Sie ab dem 14. September 2007 auf der WWW-Seite zu dieser Veranstaltung. Ihre Klausur können Sie am

Dienstag, den 18. September 2007,

von 9:00–11:00 Uhr und von 13:00–15:00 Uhr im Raum 251 des Informatikzentrums einsehen.

Aufgabe 1: Unter dem *Modalwert* einer Folge versteht man denjenigen Wert, der in der Folge am häufigsten auftritt. Beispielsweise ist 5 der Modalwert der Folge

$$3, 12, 12, 5, 5, 0, -3, 0, 5, -5, 12.$$

Der Modalwert einer Folge ist i. Allg. nicht eindeutig bestimmt.

Schreiben Sie eine Methode `static int modal(int[] a)`, die ein Feld ganzer Zahlen als Parameter erhält und den Modalwert des Felds als Rückgabewert liefert. Falls der Modalwert der Folge nicht eindeutig bestimmt ist, kann ein beliebiger Modalwert zurückgegeben werden. Erläutern Sie kurz die Idee Ihres Algorithmus.

20 Punkte

Lösung: Zunächst wird das Feld sortiert. Hierzu kann ein beliebiges Sortierverfahren verwendet werden. Dann wird gezählt, wie häufig das Element `a[0]` vorkommt. Das Element `a[0]` ist der vorläufige Modalwert. In einem Durchlauf durch das Feld werden die Position `mi` und die Häufigkeit `ml` des vorläufigen Modalwerts gespeichert und ggf. aktualisiert.

Es ist sinnvoll, eine Methode `static int anz(int[] a, int i)` zu schreiben, die zählt, wie oft das Element an Position i im Feld a auftritt.

```
static int anz(int[] a, int i) {
  return (i == a.length-1 || a[i] != a[i+1]) ? 1 : 1+anz(a,i+1);
}

static int modal(int[] a) {
  assert a != null && a.length > 0;
  ... Sortieren von a ...;
  int mi = 0,
      ml = anz(a, 0),
      i  = ml;
  while (i < a.length) {
    int k = anz(a,i);
    if (k > ml) {
      mi = i;
      ml = k;
    }
    i += k;
  }
  return a[mi];
}
```

Da der zweite Teil des Verfahrens in Linearzeit abläuft, wird die Komplexität durch die Laufzeit des Sortieralgorithmus bestimmt. Wird z. B. Quicksort gewählt, dann liegt die Laufzeit des Algorithmus im Mittel in $O(n \log(n))$, wobei n die Länge des Felds a ist.

Alternative Lösung: Wenn das Feld vorher nicht sortiert wird, dann muss zur Bestimmung der Häufigkeit eines Elements das Feld vollständig durchlaufen werden. Es ergibt sich der folgende Algorithmus. Seine Laufzeit liegt in $\Theta(n^2)$, wobei n wiederum die Länge des Felds a ist.

```
static int anz(int[] a, int i) {
  int x = 0;
  for (int e : a)
    if (e == a[i])
      x++;
  return x;
}

static int modal(int[] a) {
  assert a != null && a.length > 0;
  int mi = 0,
      ma = anz(a, 0);
  for (int j = 1; j < a.length; j++) {
    int k = anz(a, j);
    if (k > ma) {
      mi = j;
      ma = k;
    }
  }
  return a[mi];
}
```

Aufgabe 2: Bitte kreuzen Sie an. Für jede richtige Antwort erhalten Sie einen Punkt, für jede falsche Antwort wird ein Punkt abgezogen. Kein Kreuz bzw. zwei Kreuze bedeuten 0 Punkte. Die minimale Gesamtpunktzahl für diese Aufgabe beträgt 0 Punkte. Alle Fragen dieser Aufgabe beziehen sich auf Java.

	wahr	falsch
Die Klasse `String` ist final.	√	
Werte einer Enum-Klasse können nicht verändert werden.	√	
Enum-Klassen können nicht instanziiert werden.	√	
Wrapper-Klassen sind unveränderbar (immutable).	√	
Von anonymen Klassen lassen sich keine Objekte erzeugen.		√
Von lokalen Klassen lassen sich keine Objekte erzeugen.		√
Ein Konstruktor darf nicht den Modifikator `private` besitzen.		√
Serialisierung ist eine Möglichkeit, Persistenz von Daten zu erreichen.	√	
Typebounds schränken die möglichen Parameter generischer Klassen ein.	√	
Klassen und Schnittstellen können gleichzeitig als Typebound auftreten.	√	
Ein Semaphor kann den Wert 0 annehmen.	√	
Die Methoden `wait` und `notify` können nur Threads ausführen, die sich in einem Monitor befinden.	√	

12 Punkte

Aufgabe 3: Beschreiben Sie kurz die Aufgaben der folgenden Komponentenklassen.

`JLabel:` Darstellung von Texten und Bildern

`JButton:` Einfache Schaltfläche zum Auslösen von Aktionen

`JCheckBox:` Schalter mit zwei Zuständen, Kennzeichnung durch Häkchen

`JRadioButton:` Schalter(gruppe) mit zwei Zuständen, sich ausschließende Schalter

`JTextArea:` Feld für die Eingabe mehrzeiliger Texte

`JTextField:` Feld für die Eingabe einzeiliger Texte

`JComboBox:` Aufklappbare Auswahlliste

12 Punkte

Aufgabe 4: Gegeben sei die folgende Klasse `LinkedList` zur verketteten Implementierung linearer Listen ganzer Zahlen.

```
public class LinkedList {

  private int item = -1;

  private LinkedList next;

  public void insert(int x) {
    LinkedList l = new LinkedList();
    l.item = x;
    l.next = next;
    this.next = l;
  }

  public String toString() {
    return next == null ? "." : " ! " + next.item + next;
  }

}
```

a) Was gibt das folgende Programmfragment aus?

```
LinkedList l = new LinkedList();
l.insert(-8);
l.insert(3);
l.insert(-5);
l.insert(3);
l.insert(0);
System.out.println(l);
```

b) Ändern Sie die Methode `void insert(int x)` so ab, dass die Elemente in absteigend sortierter Reihenfolge eingefügt werden.

c) Schreiben Sie eine Methode `int summe()`, die die Summe der eingefügten Elemente der aktuellen Liste liefert. Wenn keine Elemente eingefügt wurden, soll das Ergebnis 0 sein. Durch die Berechnung der Summe darf die Liste nicht verändert werden.

20 Punkte

Lösung: Die Lösung zu dieser Aufgabe finden Sie auf Seite 7.

Aufgabe 5: Gegeben sei die folgende Java-Methode:

```
static int f(int n) {
  assert n >= 1;       // Vorbedingung P
  int s = 1,
      i = 0;
  assert ...;          // Schleifeninvariante Q
  while (s < n) {
    s = 2*s;
    i = i + 1;
    assert ...;        // Schleifeninvariante Q
  }
  assert ...;          // Nachbedingung R
  return i;
}
```

a) Welchen Wert berechnet diese Methode? Formulieren Sie eine entsprechende Nachbedingung R.

b) Beweisen Sie die partielle Korrektheit des Programms bezüglich P und R, indem Sie eine geeignete Schleifeninvariante Q angeben.

c) Formulieren Sie Q und R als Java-Ausdrücke, sodass diese in den obigen assert-Anweisungen verwendet werden können.

16 Punkte

Lösung:

a) Offensichtlich berechnet die Methode f den Logarithmus zur Basis 2 von n, wobei ggf. nach oben gerundet wird. Beispielsweise ist $f(8) = 3$ und $f(9) = 4$. Eine geeignete Nachbedingung R ist also $n \leq 2^i < 2n$ oder gleichwertig $i = \lceil \log_2(n) \rceil$.

b) Eine geeignete Schleifeninvariante Q ist $s = 2^i \wedge s < 2n$.

c) Eine einfache Möglichkeit, 2^i in Java auszudrücken, ist `1<<i`. Mögliche Ausdrücke für R und Q sind deshalb `n <= 1<<i && 1<<i < 2*n` sowie `s == 1<<i && s < 2*n`.

Lösung zu Aufgabe 4

a) ␣!␣0␣!␣3␣!␣-5␣!␣3␣!␣-8.

b)
```
public void insert(int x) {
  if (next == null || x > next.item) {
    LinkedList l = new LinkedList();
    l.item = x;
    l.next = next;
    this.next = l;
  } else
    next.insert(x);
}
```

c)
```
public int summe() {
  return next == null ? 0 : next.item + next.summe();
}
```