Informatik für Informationsmanager II

Universität
Koblenz-Landau
Institut für Informatik

Prof. Dr. Steffen Staab
Dr. Manfred Jackel

SS 2006

Abschlussklausur

04.08.2006

Lösung

Bitte in Druckschrift leserlich ausfüllen!

Name _____

Vorname _____

E-Mail-Adresse _____@uni-koblenz.de

Matrikelnummer _____

Studiengang:

☐ Computervisualistik (Diplom)
☐ Informatik (Diplom)
☐ Informationsmanagement (BSc)
☐ Informationsmanagement (MSc)
☐ Anglistik & Medienmanagement (MSc)
☐

☐ Diese Prüfungsleistung melde ich verbindlich als Freiversuch im Sinne der Prüfungsordnung an.

☐ Diese Prüfungsleistung ist mein 2. Versuch (Nachklausur). Erstversuch im _____ (Semester).

Auswertung:

	1	2	3	4	5	6	GESAMT
Punkte							

Aufgabe 1 (12 Punkte)

Diese Aufgabe umfasst 3 Multiple-Choice-Unteraufgaben. Innerhalb jeder Unteraufgabe gilt: wenn alle 4 Kreuze an der richtigen Stelle stehen, gibt es 4 Punkte für die Unteraufgabe. Ein falsches Kreuz gibt einen Punkt Abzug. Wer 2 richtige und 2 falsche Kreuzchen in einer Unteraufgabe macht, erhält 1+1-1-1=0 Punkte. Es gibt keine negativen Gesamtpunktzahlen pro Unteraufgabe, jede Unteraufgabe bringt 0 bis 4 Punkte.

Aufgabe 1a (4 Punkte):

		Ja	Nein	Frage
a)				**Interface**
	1.	☐	☒	`class A extends B,C implements D {…}` ist korrekter Java-Code
	2.	☐	☒	Eine Klasse, die nur abstrakte Methoden besitzt, ist ein Interface.
	3.	☒	☐	Die Klasse String implementiert das Interface Comparable.
	4.	☒	☐	Abstrakte Methoden darf man überladen.

Aufgabe 1b (4 Punkte):

				Wir setzen voraus: `class A extends B implements C {…}`
	1.	☐	☒	`A a = new B;` ist (im Gültigkeitsbereich von A und B) korrekt.
	2.	☐	☒	`B b = new C;` ist (im Gültigkeitsbereich von A und B) korrekt.
	3.	☐	☒	B muss alle abstrakten Methoden von C implementieren.
	4.	☐	☒	Angenommen. A und B definieren beide die virtuelle Methode int vm(). Weiter gebe es die Deklaration der Objektreferenz `B x;` Dann ruft x.vm() immer die virtuelle Methode von B auf.

Aufgabe 1c (4 Punkte):

c)				**Vererbung**
	1.	☐	☒	Ein Konstruktor einer abgeleiteten Klasse muss alle Konstruktoren der Oberklasse aufrufen.
	2.	☒	☐	Eine abgeleitete Klasse kann protected-Membervariablen der Oberklasse verwenden.
	3.	☐	☒	Die Oberklasse kann private-Membervariablen der abgeleiteten Klasse verwenden.
	4.	☒	☐	Alle Klassen in Java sind von der Klasse `Object` abgeleitet (außer Object selbst natürlich).

Aufgabe 2 (12 Punkte)

Die folgende Klasse implementiert generische Listen. Ergänzen Sie die Klasse um eine Methode `public int getLength()`, die die Länge der Liste rekursiv berechnet.

```
public class List {
     private ObjectNode head;

     public List() {
          head = null;
     }

     public List(ObjectNode head) {
          this.head = head;
     }

     public void insertFirst(Object data) {
          head = new ObjectNode(data, head);
     }

     public Object getHead() {
          return head.data;
     }

     public boolean empty() {
          return head == null;
     }

     public List getTail() {
          if (head == null)
               return new List(null);
          else if (head.getNext() == null)
               return new List(null);
          else
               return new List(head.getNext());
     }

/**
*
* Bitte fügen Sie den Code in den Methodenrumpf ein:
**/

     public int getLength() {

          if (head==null) return 0;
          return 1 +getTail().getLength();


     }
}
```

Aufgabe 3 (12 Punkte)

Gegeben sei folgendes Code-Fragment:

```
     int i,n; int[] a;
     …
(1)   while (n>0) {
(2)       i = i+1;
(3)       n = n/2;
(4)       a[i] = n;
     }
```

a) Geben Sie für jede nummerierte Zeile die dort verwendeten Elementaroperationen (Zuweisung, Variablenzugriff, Indizierung, Addition, Division, Vergleich) und ihre jeweilige Anzahl an.

(1) Lesen(n)+Vergleichen = 2

(2) Lesen(i)+Addition+Zuweisung=3

(3) Lesen(n)+Division+Zuweisung=3

(4) Lesen(n)+Lesen(i)+Indizierung+Zuweisung=4

Die Teilaufgaben b) und c) müssen Studierende „Anglistik und Medienmanagement“ nicht lösen

b) Berechnen Sie den exakten Aufwand A(n) der while-Schleife in Abhängigkeit von n unter der Maßgabe, dass jede Elementaroperation eine Aufwandseinheit erfordert.

Wenn das Innere der Schliefe k Durchläufe hat, wird Zeile 1 k+1 mal durchlaufen (k>=0). Dies ergibt 2(k+1)+k(3+3+4) =12k+2. Damit es k Durchläufe gibt, muss n/2 k-mal größer gleich 1 sein, also $2^{k-1} \le n < 2^k$. Damti ergibt sich der Aufwand zu $A(n) = 12\lceil \log_2 n \rceil + 2$, die Klammerung $\lceil\ \rceil$ bedeutet: die nächstgrößere ganze Zahl, falls der Klammerinhalt nicht ganzzahlig ist.

c) Zeigen Sie, dass der asymptotische Aufwand ld(n)=$\log_2(n)$ ist.

$$\begin{aligned} A(n) &= 12\lceil \log_2 n \rceil + 2 \\ &\le 12\log_2 n + 14 \\ &\le 14(\log_2 n + 1) \\ &\le 14(\log_2 n + \log_2 n) = 28\log_2 n \end{aligned}$$

Der exakte Aufwand lässt sich also ab n=2 durch $28\log_2 n$ abschätzen. Damit ist $A(n) \in O(\log_2 n)$

Aufgabe 4 (12 Punkte)

Sortieren Sie die folgenden Zahlen Array-basiert, indem Sie die nachstehenden Tabellen vervollständigen.

a) Sortieren durch Einfügen (insertion sort)

8	**2**	**4**	**1**	**7**	**5**	**3**	**6**
2	8	4	1	7	5	3	6
2	4	8	1	7	5	3	6
1	2	4	8	7	5	3	6
1	2	4	7	8	5	3	6
1	2	4	5	7	8	3	6
1	2	3	4	5	7	8	6
1	2	3	4	5	6	7	8

b) Sortieren durch Austauschen (BubbleSort, ohne Optimierung)

8	**2**	**4**	**1**	**7**	**5**	**3**	**6**
2	4	1	7	5	3	6	8
2	1	4	5	3	6	7	8
1	2	4	3	5	6	7	8
1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8

Aufgabe 5 (12 Punkte)

Leiten Sie aus dem `interface NodeActionInterface` eine Klasse `class EdgeCounter` ab, die die Anzahl der Kanten eines Baumes zählt. Genauer: die Anweisungen

```
EdgeCounter ec  = new EdgeCounter():
t.preorder(ec);
System.out.println(ec);
```

führen zu der Ausgabe „Kantenzahl=..“. Dabei ist t ein Objekt der Klasse Tree des Paketes Kapitel13, das Sie aus der Übung kennen. Wir wiederholen die Deklarationen, die Sie zum Verständnis brauchen:

```
public interface NodeActionInterface {
  /**
   * Methode, deren Implementierung auf einem Knoten arbeitet.
   * @param n Der zu bearbeitende Knoten
   */
  public void action(Node n);
}

public class Tree {
      Node root;
      …
      public void preorder(NodeActionInterface p) {
            // Mantelprozedur für traversePreorder
            traversePreorder(root, p);
      }

      private void traversePreorder(Node node, NodeActionInterface p) {

            if (node == null) return; // leerer Unterbaum
            p.action(node); // Ausführen der Aktion auf Knoten
            traversePreorder(node.left, p); // Durchlaufe linken Unterbaum
            traversePreorder(node.right, p); // Durchlaufe rechten Unterbaum
      }
…
}
```

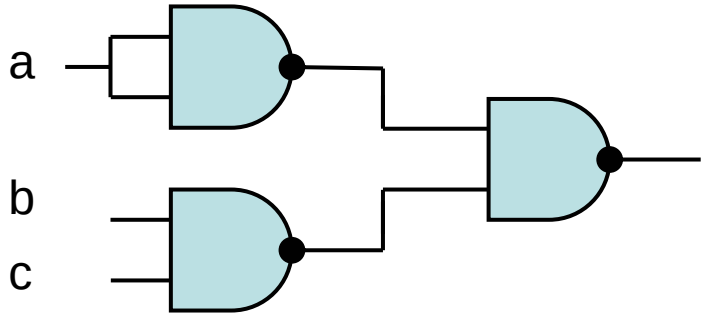
Preorder wendet die action-Methode auf jeden Knoten einmal an. Wir geben die Klasse EdgeCounter teilweise vor, so dass Sie nur die action-Methode vervollständigen müssen.

```
   public class EdgeCounter implements NodeActionInterface {

      private int kantenZahl=0;  // hier zählen wir

      public void action(Node n) { // hier programmieren!

            // if (n==null) return;
            if (n.left!=null)
                  kantenZahl++;
            if (n.right!=null)
                  kantenZahl++;









      }

      public String toString() {
            return„“Kantenzahl=“+kantenZahl;

}
}
```

Aufgabe 6 (12 Punkte)

Geben Sie eine Gatter-Schaltung an, die die Verknüpfung $a \vee b \wedge c$ ausschließlich mit NAND-Gattern realisiert.

$$\begin{aligned} a \vee (b \wedge c) &\equiv \neg\neg(a \vee (b \wedge c)) \\ &\equiv \neg(\neg a \wedge \neg(b \wedge c)) \end{aligned}$$



Alternativ:

$$\begin{aligned} a \vee (b \wedge c) &\equiv \neg\neg(a \vee (b \wedge c)) \\ &\equiv \neg\neg(a \wedge b \vee a \wedge c) \\ &\equiv \neg(\neg(a \wedge b) \wedge \neg(a \wedge c)) \end{aligned}$$