Objektorientierte Programmierung und Modellierung

WS 2008/2009

UNIVERSITÄT KOBLENZ · LANDAU

Institut für Informatik

Prof. Dr. Ralf Lämmel
Dr. Manfred Jackel

1. Teilklausur

Modul "OOPM Vorlesung/Übung"

16.12.2008

Gruppe A

Name	
Vorname	
E-Mail-Adresse	@uni-koblenz.de
Matrikelnummer	

Studiengang:
☐ Computervisualistik (Diplom)
☐ Informatik (Diplom)
☐ Computervisualistik (BSc)
☐ Informatik (BSc)
☐ Anglistik & Medienmanagement (BSc)
☐ B Ed _____
☐ Informationsmanagement (BSc)

☐ Diese Prüfungsleistung melde ich verbindlich als Freiversuch im Sinne der Prüfungsordnung an (nur alte PO)

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

Auswertung:

	1	2	3	4	5	GESAMT
Punkte						

Aufgabe 01 (12 Punkte)

Die folgende algebraische Spezifikation von Listen wurde in der Vorlesung gegeben:

```
specification List<T>
 exports
  sorts List<T>
  variables
   t, t1, t2 : T
   l, l1, l2 : List<T>
  constructor symbols
   Nil : -> List<T>
   Cons : T * List<T> -> List<T>
  function symbols
   head : List<T> -> T
   tail : List<T> -> List<T>
   append : List<T> * List<T> -> List<T>
  equations
   (1)  head(Cons(t,l)) = t
   (2)  tail(Cons(t,l)) = l
   (3)  append(Nil,l) = l
   (4)  append(Cons(t,l1),l2) = Cons(t,append(l1,l2))
  end
```

Beweisen Sie:

```
append(Cons(t1, Nil), Cons(t2, Nil)) = Cons(t1, Cons(t2, Nil))
```

append(Cons(t1, Nil), Cons(t2, Nil))

= Cons(t1,append(Nil,Cons(t2,Nil))) (4)

= Cons(t1,Cons(t2,Nil)) (3)

Aufgabe 02 (12 Punkte)

Sortieren Sie mit SelectionSort (Sortieren durch Auswählen)!

Geben Sie das Ergebnis jedes Durchlaufs in der nachfolgenden Tabelle an. Notieren Sie alle Durchläufe, auch wenn nicht getauscht wird!

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

Eine Reservetabelle, falls Sie diese benötigen.

4	7	3	8	2	6	5	1

Aufgabe 03 (12 Punkte)

Geben Sie den exakten Aufwand T(n) und den asymptotischen Aufwand O(T(n)) des nachfolgenden Programmstücks an. Den Aufwand jeder Zeile=jeder Basisoperation haben wir schon hingeschrieben.

```
i=0;                    (1)      1
sum=0                   (1)      1
while (i<2*n) {         (2)      2*(2n+1)
     if (odd(i))        (1)      2n
          sum +=i;      (1)      n
     i++;               (1)      2n
}
```

Hinweis: odd(i) ist genau dann true, wenn i ungerade ist.

Die while-Schleife wird 2n-mal Durchläufen, n mal mit ungeradem i. Das ergibt den Aufwand

T(n)=9n+4

O(T(n)) =O(n)

Aufgabe 04 (12 Punkte)

Diese Aufgabe umfasst 3 Multiple-Choice Cluster mit je 4 Ankreuzfragen. Für jedes Cluster gilt: wenn alle 4 Kreuze an der richtigen Stelle stehen, gibt es 4 Punkte für das Cluster. Ein falsches Kreuz gibt **einen Punkt Abzug**. Wer 2 richtige und 2 falsche Kreuzchen in einem Cluster macht, erhält 1+1-1-1=0 Punkte. Zum Trost: es gibt keine negativen Gesamtpunktzahlen, jedes Cluster bringt 0 bis 4 Punkte. Wenn Sie eine Frage nicht beantworten, **gibt dies 0 Punkte**!

		Ja	Nein	Frage
a)				**Design by Contract**
	1.	☐	☒	Eine Zusicherung, die am Anfang einer Methode gilt, gilt auch am Ende.
	2.	☒	☐	Eine Klasseninvariante eines korrekten Programmes ist unmittelbar nach Ausführen des Konstruktors erfüllt.
	3.	☒	☐	Es gibt partiell korrekte Programme, die terminieren.
	4.	☒	☐	Die schwächste Vorbedingung ist die beste.
b)				**Datenstrukturen**
	1.	☐	☒	Der Keller funktioniert nach dem Prinzip „first in“, „first out“.
	2.	☒	☐	Der Parametertyp generischer Typen muss ein Referenztyp sein.
	3.	☒	☐	int[] ist ein Referenztyp.
	4.	☒	☐	Jede rekursive Java-Funktion kann in eine nicht-rekursive Java-Funktion umgeschrieben werden.
c)				**Verschiedenes**
	1.	☐	☒	Getter-Methoden sollen public und Setter-Methoden müssen private sein.
	2.	☒	☐	$O(n*\log(n)) \subseteq O(n^2)$
	3.	☐	☒	Sortieren von n Elementen hat mindestens den Aufwand $O(n^2)$.
	4.	☐	☒	Binäres Suchen hat den Aufwand $O(n/2)$.

Aufgabe 05 (12 Punkte)

Der folgende Datentyp stellt ein Darlehenskonto dar. Das maximale Darlehen darf durch Auszahlungen nicht überschritten werden. Die Höhe des Darlehens wird durch eine negative Zahl dargestellt.

```
public class Darlehen {
  double maxLoan;        //maxLoan soll <= 0 sein, maximales Darlehen
  double actualLoan;     // das aktuell gezogene Darlehen

  /**
   * Erzeugt ein neues Darlehenskonto mit Kreditrahmen
   * in der angegebenen Höhe.
   */
  public Darlehen(double hoehe) { // Konstruktor
   maxLoan = hoehe;
   actualLoan = 0.0;
  }

  /**
   * Belastet das aktuelle Darlehen um den angegebenen Betrag. Das
   * Darlehen darf nicht überzogen werden.
   */
  public abheben(double betrag) {

    // Vorbedingung {P}

    actualLoan -= betrag;   // Das Darlehen darf nicht überzogen werden!

    // Nachbedingung {Q}

  }
}
```

Geben Sie die schwächste Vorbedingung P und die stärkste Nachbedingunge Q an! Es muss sichergestellt werden, dass das Darlehen nicht überzogen wird und die Methode `abheben()` nicht zum Rückzahlen angewandt werden kann.

Geben Sie die Zusicherungen als mathematische Formel an.

P = (betrag>=0) & actualLoan-betrag >=maxLoan

Q = ActualLoan >= maxLoan