Objektorientierte Programmierung & Modellierung

Universität Koblenz-Landau
Institut für Informatik

Prof. Dr. Ralf Lämmel
Dr. Manfred Jackel

WS 2007/8

2. Teilklausur

18.01.2008

Gruppe A/B

Bitte in Druckschrift leserlich ausfüllen!

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).

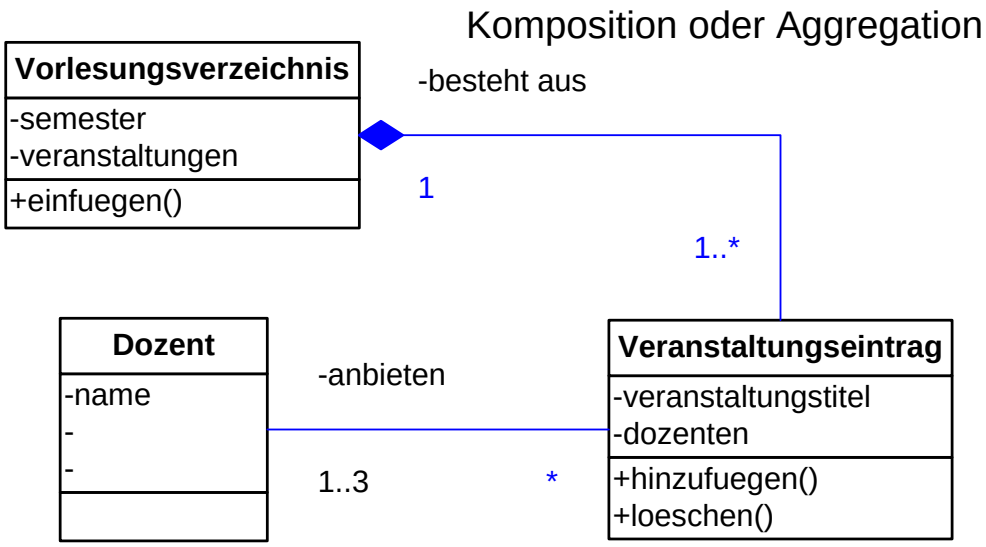
Auswertung:

	1	2	3	4	GESAMT
Punkte					

Aufgabe 01 (12 Punkte)

Das folgende Klassendiagramm modelliert ein Vorlesungsverzeichnis mit Veranstaltungen, die von Dozenten angeboten und von Studenten belegt werden.

a) Stellen Sie die Multiplizitäten der angedeuteten Assoziationen „besteht aus“ und „anbieten“ fest. Welche sind Aggregationen oder Kompositionen?
b) Ergänzen Sie die Java-Klasse Vorlesungsverzeichnis, so dass die Assoziation „besteht aus“ über das Attribut „veranstaltungen“ und die Methode „einfuegen“ implementiert wird. Wählen Sie einen geeigneten Datentyp für veranstaltungen aus.
c) Ergänzen Sie die Java-Klasse Veranstaltungseintrag, so dass die Assoziation „anbieten“ über das Attribut „dozenten“ und die Methode „hinzufuegen“ und „loeschen“ implementiert wird. Wählen Sie einen geeigneten Datentyp für dozenten aus, der bis zu 3 Dozenten pro Veranstaltung zulässt.



```
class Vorlesungsverzeichnis {

     pivate String semester;

     private Vector<Veranstaltungseintrag> veranstaltungen;


     public void einfuegen(Veranstaltungseintrag e) {

          veranstataltungen.add(e);









     }
}

class Veranstaltungseintrag {

     private String veranstaltungstitel;
     private  Dozenten[] dozenten = new Dozenten[3];
     private dozZahl=0;


     public void hinzufuegen(Dozent d) {

          for (int i=0; i<dozZahl; i++) if (dozenten[i]==d) return;
          if (dozZahl==3) return;
          dozenten{dozZahl]=d;
          dozZahl++;


}

     public void loeschen(Dozent d) {
          for (int i=0; i<dozZahl; i++) {
               if (dozenten{i]==d) {
                    for (int j=i; j<dozZahl,j++)
                         dozenten[j]=dozenten[j+1];
                         dozzahll--;
                         return;
                    }
          }



     }

}
```

Aufgabe 02 (12 Punkte)

In der Vorlesung wurde der Datentyp IList als Liste von Integer vorgestellt. Wir definieren, wann eine Liste aufsteigend sortiert ist.

```
isSorted(Nil) = true;
isSorted Cons(i,Nil) = true;
isSorted(Cons(i,Cons(j,list)) = i<=j & isSorted(Cons(j,list))
```

Programmieren Sie die Java-Methode isSorted so aus, die sie diese Definition implementiert.

```
public interface IList
{
  int head() throws ListException;
  IList tail() throws ListException;
  int length();
  boolean isSorted();
}

public class Cons implements IList {

    private int h;
    private IList t;
    public Cons(int h, IList t) { this.h = h; this.t = t; }
    public int head() { return h; }
    public IList tail() { return t; }
    public int length() { return t.length() + 1; }

    public boolean isSorted() {
     // hier ergänzen


     try {
       if ( h<=t.head()) return t.isSorted() else return false;
     } catch (ListException e) {
         return true;
     }
     }
}

public class Nil implements IList {

    public int head() throws ListException { throw new ListException(); }
    public IList tail() throws ListException { throw new ListException(); }
    public int length() { return 0; }
    public boolean isSorted() {
     // hier ergänzen

      return true;

    }
}
```

Aufgabe 03 (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.

		Ja	Nein	Frage
a)				**Sortieren**
	1.	☐	☒	Der exakte Aufwand beim InsertionSort hängt nur von der Anzahl, nicht von der Art der Daten ab.
	2.	☒	☐	Beim Selectionsort ist der Aufwand zum Finden des 1. Elementes der sortierten Folge von der Aufwandsklasse n=Anzahl der Elemente.
	3.	☐	☒	Quicksort geht am schnellsten, wenn man immer das kleinste Element als Pivot-Element nimmt.
	4.	☒	☐	Mergesort gehört zu den effizientesten Sortierverfahren.
b)				**Exceptions**
	1.	☐	☒	Unchecked Exceptions kann man nur durch finally-Blöcke behandeln.
	2.	☐	☒	Auf einen Try-Block muss mindestens ein Catch-Block folgen.
	3.	☐	☒	Eine Runtime-Exception führt immer zum Programmabbruch.
	4.	☒	☐	Wenn der erste Catch-Block die generellste Ausnahme behandelt, wird der speziellere nachfolgende Catch-Block verdeckt.
c)				**Misc**
	1.	☒	☐	Tail recursion ist leicht in eine iterative Lösung zu überführen.
	2.	☐	☒	Doppelte Rekursion kann nicht iterativ berechnet werden.
	3.	☐	☒	Jede iterative Lösung eines Problems ist von geringerer Komplexität als eine rekursive.
	4.	☒	☐	Generische Datentypen ermöglichen die Parametrisierung von Klassendefinitionen durch Referenztypen.

Aufgabe 04 (12 Punkte)

Das folgende Programmstück berechnet die Binärdarstellung einer nicht negativen Zahl als String.

```
(1) if (n==0) string = “0“;                          1
    else {
(2)   string=““;                                     1
(3)   while (n>0) {                                  x+1
(4)       if (n%2 == 0) ch=“0“; else ch=“1“;         x
(5)       string = ch+string;                        x
(6)       n = n/2;                                   x
    }
```

Berechnen Sie T(n). Jede Anweisungszeile wird mit dem Aufwand 1 bewertet.

Hinweis: in der while-Schleife wird n fortlaufend halbiert, was nahe legt, dass der duale Logarithmus in die Rechnung eingehen muss.

Aus obigern zeilenweise Rechnung ergibt sich T(n) = 4x+3.
Wie groß ist x?
Wenn $n=2^m$ ist, wird die Schleife m+1 mal durchlaufen. Daher ist x = |ld(n)|+1 (|ld(n)| sei die größte ganze Zahl <= ld(n)).

T(n) = 4*|ld(n)|+7.