Fakultät IV
Elektrotechnik/Informatik

Klausur Einführung in die Informatik II für Elektrotechniker

2. August 2006

Musterlösung
Stand: 27. Juli 2006

Aufgabe	Punkte	Erreichte Punkte
1	5	
2	7	
3	6	
4	8	
5	9	
6	9	
Summe	44	

Aufgabe 1 (5 Punkte) Theorie.

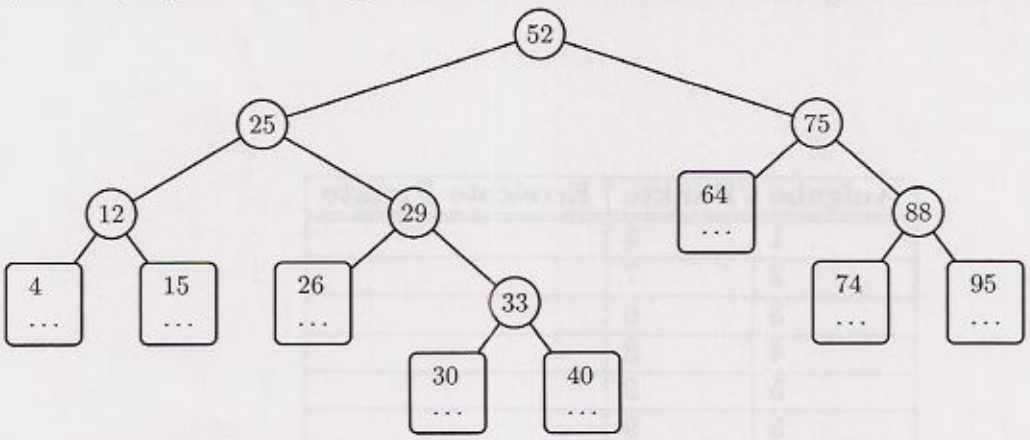
1. (2 Punkte) Wozu dienen in Java *generische Klassen*, wie deklariert und benutzt man sie? Nennen sie einen Vorteil dieser Klassen.

   Lösung:
   Generische Klassen besitzen Typparameter, die innerhalb der Klasse wie Typnamen verwendet werden können. Bei der Erzeugung von Objekten muss man dann in spitzen Klammern nach dem Klassennamen einen konkreten Typ angeben, der für diesen Typparameter benutzt werden soll. Diese Klassen werden z.B. für abstrakte Datentypen benutzt, wie etwa die Collection-Klassen in Java. Vorteile: beim Herausnehmen von Objekten aus diesen Collection-Klassen muss man nicht casten; der Compiler kann mehr Typfehler erkennen.
   (1 Punkt für die Beschreibung, 1 Punkt für richtigen Vorteil.)

2. (1 Punkt) Nennen Sie einen Vorteil und einen Nachteil des Sortieralgorithmus *Quicksort*.

   Lösung:
   Vorteil: im Mittel schnellster bekannter Sortieralgorithmus; Nachteil: im schlechtestem Fall quadratischer Aufwand. **(0.5 Punkte pro richtigem Vor-/Nachteil.)**

3. (2 Punkte) Eignet sich der folgende Baum zum effizienten Suchen? Begründen Sie Ihre Antwort.



   Lösung:
   Nein, denn er ist (a) nicht geordnet (Knoten 74 ist außer der Reihe) und (b) ist er nicht balanciert (es gibt einen Pfad der Länge 2 und Pfade der Länge 4). **(1 Punkt für „nein", 1 Punkt für die Begründung.)**

Aufgabe 2 (7 Punkte) Java.

1. (2 Punkte) Gegeben seien die folgenden drei Klassen:

```
class A {
    B b;
}
```

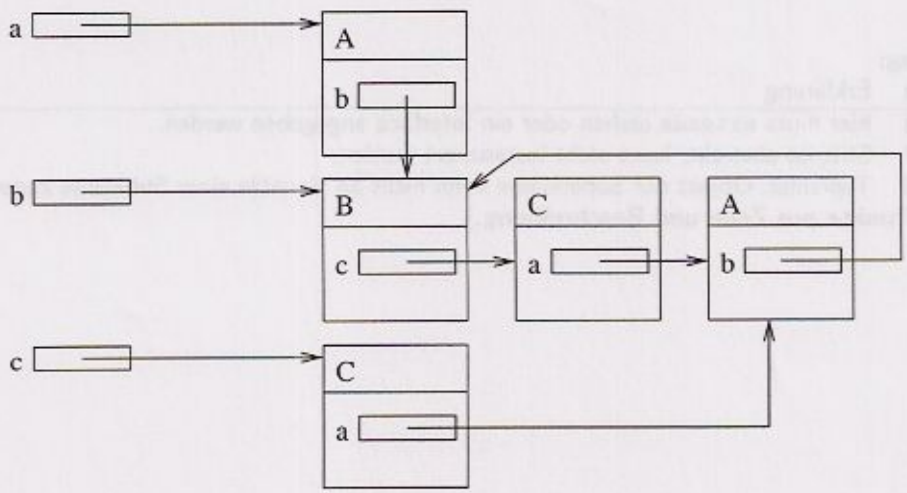
```
class B {
    C c;
}
```

```
class C {
    A a;
}
```

Stellen sie grafisch die Objektstruktur dar, die durch folgenden Java-Code erzeugt wird. Zeichnen Sie pro Objekt ein Kasten und Pfeile für die Referenzen zwischen den Objekten. Lassen Sie auch erkennen, auf welche Objekte die deklarierten Variablen verweisen.

```
A a = new A();
B b = new B();
C c = new C();
a.b = b;
b.c = new C();
b.c.a = new A();
b.c.a.b = b;
c.a = b.c.a;
```

Lösung:



Hinweis für die Korrektur: die Kästen müssen nicht genau so gemalt werden wie in der Musterlösung. Wichtig ist die korrekte Anzahl von Objekten und richtige Referenzen. Die Namen der lokalen Variablen können auch direkt an den Objekten stehen, und die Attribute in den Objekten müssen nicht unbedingt benannt werden. **(0.5 Punkte Abzug pro Fehler.)**

2. (1 Punkt) Wozu dient in Java das Schlüsselwort `implements`?

Lösung:
`Implements` gibt an, dass eine Klasse ein bestimmtes Interface implementiert, also alle Methoden besitzt, die in dem Interface angegeben sind.
(0.5 Punkte für Begriffe Klasse und Interface, 0.5 Punkte für das Implementieren von Methoden.)

3. (1 Punkt) Wozu dient in Java das Schlüsselwort `public`? An welchen Stellen im Programm kann man es benutzen?

Lösung:
Durch den Modifizierer `public` werden Methoden, Attribute und Klassen markiert, die von allen anderen Klassen des Systems aus verwendet werden können, also auch über Package-Grenzen hinweg. **(0.5 Punkte für Zugriff von allen anderen Klassen aus, 0.5 Punkte für Verwendung mit Attributen, Klassen und Methoden.)**

4. (3 Punkte) Welche Fehler enthalten die folgenden Java-Klassen? Geben Sie jeweils die Zeilennummer an und beschreiben Sie den Fehler. Folgefehler (also Fehler, die aus anderen Fehlern resultieren) sollen ignoriert werden.

```
1   abstract class Stift {
2       private int farbe;
3       Stift(int farbe) {
4           this.farbe = farbe;
5       }
6   }
7
8   class Bleistift implements Stift {
9       Bleistift(int farbe) {
10          super(farbe);
11      }
12  }
13
14  class Hauptklasse {
15      public static void main(String[] args) {
16          Bleistift b = new Bleistift(0);
17          Stift s = new Stift(1);
18          b = s;
19      }
20  }
```

Lösung:

Zeile	Erklärung
8	hier muss extends stehen oder ein Interface angegeben werden.
17	Stift ist abstrakt, kann nicht instanziiert werden.
18	Typfehler. Objekt der Superklasse kann nicht an Variable einer Subklasse zugewiesen werden

(0.5 Punkte pro Zeile und Beschreibung.)

Aufgabe 3 (6 Punkte) Numerik.

Die Kosinusfunktion kann folgendermaßen definiert werden:

$$\cos(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!}$$

1. (4 Punkte) Schreiben Sie eine Methode

   `double cos(double x, int n)`

   welche den Kosinus von x nach der oben genannten Formel berechnet. Der Approximationsprozess soll abgebrochen werde, sobald eine Genauigkeit von n Nachkommastellen erreicht ist.

   Hinweis: Sie dürfen die Methode `double pow(double x, double n)` *aus der Klasse* `Math` *benutzen um die Potenz x^n zu berechnen. Die Fakultätsfunktion*

   $$\begin{aligned} 0! &= 1 \\ (n+1)! &= (n+1) \cdot n! \end{aligned}$$

   müssen Sie aber selbst definieren.

   Lösung:

```
1   double cos(double x, int n) {
2       double eps = Math.pow (10, -n);
3       double sum = 0.0;
4       double oldSum;
5       int k = 0;
6       do {
7           oldSum = sum;
8           sum = sum + (Math.pow(-1, k) * (Math.pow(x, 2 * k) / fac(2 * k)));
9           k++;
10      } while (notClose(sum, oldSum, eps));
11      return sum;
12  }
13
14  int fac(int n) {
15      if (n == 0) {
16          return 1;
17      } else {
18          return fac(n-1) * n;
19      }
20  }
21
22  boolean notClose(double x, double y, double eps) {
23      return Math.abs(x - y) > eps;
24  }
```

   (1 Punkt für Approximation, 0.5 Punkte für Test auf ausreichende Genauigkeit, 0.5 Punkte für Summenbildung, 1 Punkt für Fakultät, 1 Punkt für Korrektheit insgesamt.)

2. (2 Punkte) In Java gibt es die Funktion `double Math.cos(double x)`, welche ebenfalls die Kosinusfunktion berechnet.

   Schreiben Sie eine Methode

   ```
   boolean testCos(double x, int n, double eps)
   ```

   welche prüft, ob die Differenz der Ergebnisse der Funktionen `Math.cos(x)` und `cos(x, n)` (aus der vorigen Unteraufgabe) weniger als der Betrag `eps` ist.

   Lösung:

   ```
   1       boolean testCos(double x, int n, double eps) {
   2           return (Math.abs(Math.cos(x) - cos(x, n)) < eps);
   3       }
   ```

   (1 Punkt für korrekten Aufruf der Methoden, 1 Punkte für Genauigkeit).

Aufgabe 4 (8 Punkte) Abstrakte Datentypen.

In dieser Aufgabe soll ein abstrakter Datentyp `AuftragsListe` zum Speichern von Bestellungen erstellt werden, welcher Objekte der folgenden Klasse verwalten kann:

```
class Bestellung {
    boolean express;
    String ware;
    int anzahl;
    Bestellung(boolean express, String ware, int anzahl) {
        this.express = express;
        this.ware = ware;
        this.anzahl = anzahl;
    }
}
```

Eine Bestellung ist entweder normal (Attribut `express = false`) oder eine Expressbestellung (Attribut `express = true`). Dies muss in den zu programmierenden Methoden beachtet werden.

Eine `AuftragsListe` besitzt drei Methoden, die weiter unten zu programmieren sind:

`boolean istLeer()` Prüfen, ob die Liste leer ist.

`void rein(Bestellung b)` Eine Bestellung einfügen.

`Bestellung raus()` Eine Bestellung aus der Liste holen.

Für die Methode `raus()`, die Bestellungen aus der Liste holt, sollen folgende Regeln gelten.

1. Express-Bestellungen sollen Vorrang vor normalen Bestellungen haben, also immer zuerst herausgegeben werden.
2. Als Einschränkung gilt: nach jeweils drei Express-Bestellungen muss eine normale Bestellung herausgegeben werden, damit eine gewisse Fairness herrscht.
3. Wenn die Liste leer ist, soll `null` abgeliefert werden und angenommen werden, dass noch keine Expresslieferung rausgenommen wurde.

Hinweis: Sie können die Klasse `LinkedList` mit dem Konstruktor `LinkedList()` und den Methoden `boolean isEmpty()`, `void add(Object o)` und `Object removeFirst()` aus der Java-Standardbibliothek benutzen.

Aufgaben:

1. (2 Punkte) Schreiben Sie den Anfang der Klasse `AuftragsListe`. Definieren Sie die benötigten Attribute, um die oben beschriebenen Operationen schreiben zu können. Schreiben Sie einen geeigneten Konstruktor, mit dem man eine leere Auftragsliste erzeugen kann.

   Hinweis: Die Art und Weise, wie die Bestellungsobjekte intern von der Klasse `AuftragsListe` verwaltet werden, ist Ihnen überlassen.

   Lösung:

```
1   class AuftragsListe {
2       private LinkedList normal;
3       private LinkedList express;
4       private int expressPending;
5       private final int expressPendingDefault = 3;
6       AuftragsListe() {
7           normal = new LinkedList();
8           express = new LinkedList();
9           expressPending = expressPendingDefault;
10      } // constructor AuftragsListe
11      // ...
12  }
```

   (1 Punkt für Attribute, 1 Punkt für Konstruktor.)

2. (1 Punkt) Schreiben Sie (in der Klasse `AuftragsListe`) eine Methode

   ```
   boolean istLeer()
   ```

   welche den Wert `true` zurückgibt, wenn keine Bestellungen in der Liste sind und andernfalls den Wert `false`.

   Lösung:

   ```
   1       boolean istLeer() {
   2           return normal.isEmpty() && express.isEmpty();
   3       }
   ```

   (0.5 Punkte Abzug pro Fehler.)

3. (2 Punkte) Schreiben Sie (ebenfalls in der Klasse `AuftragsListe`) eine Methode

   ```
   void rein(Bestellung b)
   ```

   welche der Auftragsliste die gegebene Bestellung hinzufügt.

   Lösung:

   ```
   1       void rein(Bestellung b) {
   2           if (b.express) {
   3               express.add(b);
   4           } else {
   5               normal.add(b);
   6           }
   7       } // rein
   ```

   (0.5 Punkte Abzug für Fehler.)

4. (3 Punkte) Schreiben Sie (ebenfalls in der Klasse `AuftragsListe`) eine Methode

   ```
   Bestellung raus()
   ```

   welche eine Bestellung aus der Liste entfernt und als Ergebnis zurückliefert.

   Beachten Sie hierbei die Einschränkungen, die für die Herausgabe von Bestellungen aus der Liste gelten, so wie sie in der Einleitung angegeben wurden.

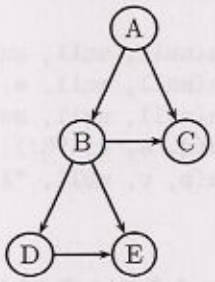
   Lösung:

   ```
   1       Bestellung raus() {
   2           if (!express.isEmpty() && expressPending > 0) {
   3               Bestellung b = (Bestellung) (express.removeFirst());
   4               expressPending--;
   5               return b;
   6           } else if (!normal.isEmpty()) {
   7               Bestellung b = (Bestellung) (normal.removeFirst());
   8               expressPending = expressPendingDefault;
   9               return b;
   10          } else {
   11              return null;
   12          }
   13      }
   ```

   (1 Punkt für richtiges Rausgeben und Löschen, 1 Punkt für Beachtung der Regeln, 1 Punkt für Korrektheit insgesamt).

Aufgabe 5 (9 Punkte) Bäume.

In dieser Aufgabe soll eine erweiterte Datenstruktur für Binärbäume geschrieben werden. Zusätzlich zu den Verweisen auf die linken und rechten Nachfolgerknoten enthält jeder Knoten noch einen Verweis auf seinen rechten Nachbarn auf der gleichen Ebene.
Die folgende Abbildung zeigt, wie solch ein Baum konzeptionell aussieht. Alle Knoten haben Verweise auf bis zu zwei Nachfolger. Außerdem haben alle Knoten einen Verweis auf den rechten Nachbarn, sofern einer existiert.



1. (3 Punkte) Schreiben Sie eine Klasse `Knoten` zur Darstellung von Baumknoten. Jeder Knoten hat sowohl Referenzen auf zwei Kindknoten als auch eine Referenz auf den rechten Nachbarn. Weiterhin soll jeder Knoten eine Referenz auf ein Datenelement der Klasse `Object` enthalten.

   Schreiben Sie einen geeigneten Konstruktor, der die Initialisierung aller Attribute erlaubt.

   Lösung:

```
1   class Knoten {
2       Knoten links;
3       Knoten rechts;
4       Knoten naechster;
5       Object data;
6
7       Knoten(Knoten links, Knoten rechts, Knoten naechster, Object data) {
8           this.links = links;
9           this.rechts = rechts;
10          this.naechster = naechster;
11          this.data = data;
12      }
13  }
```

   (1.5 Punkte für die Attribute, 0.5 Punkt für den Konstruktor, 1 Punkt für die Initialisierung.)

2. (2 Punkte) Schreiben Sie Methode

```
Knoten testBaum()
```

welche den Baum erzeugt, der oben dargestellt ist. Die Datenobjekte sind die Strings `"A"`, `"B"`, `"C"`, usw.

Lösung:

```
1       Knoten testBaum() {
2           Knoten e = new Knoten(null, null, null, "E");
3           Knoten d = new Knoten(null, null, e, "D");
4           Knoten c = new Knoten(null, null, null, "C");
5           Knoten b = new Knoten(d, e, c, "B");
6           Knoten a = new Knoten(b, c, null, "A");
7           return a;
8       }
```

(1 Punkt für Erzeugung aller Knoten, 1 Punkt für richtige Referenzen.)

3. (4 Punkte) Gegeben sei folgendes Interface:

```
interface Besucher {
    void besuche(Knoten k);
}
```

Schreiben Sie eine Methode

```
void ebene(Knoten k, Besucher b)
```

welche alle Knoten, die an dem Knoten `k` hängen, ebenenweise besucht. Für den oben dargestellten Baum soll also die `besuche`-Methode von `b` nacheinander für die Knoten A, B, C, D, E aufgerufen werden.

Hinweis: Beachten Sie, dass ein Knoten auch einen einzigen linken oder rechten Nachfolger haben kann.

Lösung:

```
1       void ebene(Knoten k, Besucher b) {
2           if (k != null) {
3               Knoten n = k;
4               do {
5                   b.besuche(n);
6                   n = n.naechster;
7               } while (n != null);
8               if (k.links != null) {
9                   ebene(k.links, b);
10              } else {
11                  ebene(k.links, b);
12              }
13          }
14      }
```

(1 Punkt für rekursive Traversierung, 1 Punkt für ebenenweise Abarbeitung, 1 Punkt für Aufrufe der Interface-Methode, 1 Punkt für Beachtung von Knoten ohne linken Nachfolger.)

Aufgabe 6 (9 Punkte) Vererbung.

In dieser Aufgabe sollen Sie ein Programm zur Planung Ihres Stundenplans schreiben.

1. (1 Punkt) Schreiben Sie eine abstrakte Klasse `Lehrveranstaltung`, welche die abstrakten Methoden `String name()`, `int sws()` und `String typ()` besitzt.

   Lösung:

```
1  abstract class Lehrveranstaltung {
2      abstract String name();
3      abstract int sws();
4      abstract String typ()
5  }
```

   (0.5 Punkte für `abstract`, 0.5 Punkte für abstrakte Methode.)

2. (3 Punkte) Schreiben Sie eine Klasse `Vorlesung`, welche von `Lehrveranstaltung` abgeleitet ist. Den Namen und die Semesterwochenstunden (SWS) soll man beim Aufruf des Konstruktors angeben können. Der Typ einer Vorlesung ist die Zeichenkette `"LV"`. Wenn Sie Attribute verwenden, sollen diese von außen *nicht* sichtbar sein.

   Lösung:

```
1  class Vorlesung extends Lehrveranstaltung {
2      String name;
3      int sws;
4
5      Vorlesung(String name, int sws) {
6          this.name = name;
7          this.sws = sws;
8      }
9      String name() {
10         return name;
11     }
12     int sws() {
13         return sws;
14     }
15     String typ() {
16         return "VL";
17     }
18 }
```

   (0.5 Punkte für `extends`, 0.5 Punkte für Attribute, 0.5 Punkte für Konstruktor und 0.5 Punkte für Methodenimplementierung.)

3. (2 Punkte) Schreiben Sie weiterhin eine Klasse `Uebung`, welche ebenfalls von `Lehrveranstaltung` abgeleitet ist. Eine Übung soll die Vorlesung speichern, zu der sie gehört. Diese Vorlesung und die Semesterwochenstunden (SWS) soll man beim Aufruf des Konstruktors angeben können. Der Typ einer Übung ist die Zeichenkette `"UE"`, der Name entspricht der dazugehörigen Vorlesung. Wenn Sie Attribute verwenden, sollen diese von außen *nicht* sichtbar sein.

Lösung:

```
1   class Uebung extends Lehrveranstaltung {
2       private Vorlesung vl;
3       private int sws;
4
5       Uebung(Vorlesung vl, int sws) {
6           this.vl = vl;
7           this.sws = sws;
8       }
9       String name() {
10          return vl.name();
11      }
12      int sws() {
13          return sws;
14      }
15      String typ() {
16          return "UE";
17      }
18  }
19
```

(0.5 Punkte für `extends`, 0.5 Punkte für Attribute, 0.5 Punkte für Konstruktor und 0.5 Punkte für Methodenimplementierung.)

4. (3 Punkte) Schreiben Sie nun eine Klasse `Stundenplan`, die folgende Methoden zur verfügung stellt:

   - Einen Konstruktor, dem man ein Array von Lehrveranstaltungen übergeben kann. Der Konstruktor soll eine Kopie das übergebenen Arrays anlegen.
   - Eine Methode `int gesamtSWS(String typ)`, welche die Summe der SWS aller Lehrveranstaltungen mit dem angegebenen Typ berechnet.

   Hinweis: Beachten Sie, dass Sie zum Vergleichen von Strings die Methode `boolean equals(String other)` verwenden müssen. Der Operator == funktioniert bei Strings nicht.

Lösung:

```
1   class Stundenplan {
2       private Lehrveranstaltung[] lvs;
3       Stundenplan(Lehrveranstaltung[] lvs) {
4           this.lvs = new Lehrveranstaltung[lvs.length];
5           System.arraycopy(lvs, 0, this.lvs, 0, this.lvs.length);
6       }
7       int gesamtSWS(String typ) {
8           int s = 0;
9           for (int i = 0; i < lvs.length; i++) {
10              if (lvs[i].typ().equals(typ)) {
11                  s = s + lvs[i].sws();
12              }
13          }
14          return s;
15      }
16  }
```

(1 Punkt für Konstruktor, 2 Punkte für Methode `gesamtSWS`.)