Hochschule Bonn-Rhein-Sieg
University of Applied Sciences

Fachbereich Informatik
Prof. Dr. Peter Becker

Klausur — Sommersemester 2012
Datenstrukturen und Algorithmen
24. September 2012

Bevor Sie mit der Bearbeitung dieser Klausur beginnen, lesen Sie bitte folgende Hinweise. Diese Hinweise sind bei der Bearbeitung zu beachten.

1. Prüfen Sie die Vollständigkeit Ihres Exemplars. Es sollte

   - dieses Hinweisblatt und
   - acht Aufgaben auf acht Blättern

   umfassen.

2. Tragen Sie auf jedem Lösungsblatt oben an den vorgesehenen Stellen Ihren Namen und Ihre Matrikelnummer ein. Blätter ohne diese Angaben werden nicht bewertet.

   Hinter den Aufgaben ist jeweils hinreichend Platz für die Lösungen freigelassen. Reicht der Platz nicht aus, benutzen Sie die Rückseiten, wobei die Zuordnungen von Lösungen zu Aufgaben deutlich erkennbar sein müssen.

3. Geben Sie dieses Deckblatt zusammen mit den Aufgabenstellungen und den Lösungen sowie alles weitere beschriebene Papier ab.

4. Zugelassene Hilfsmittel: keine

5. Mit $\geq$ 50 Punkten haben Sie die Klausur bestanden.

6. Ergebnis (bitte nichts eintragen):

1 (10)	2 (15)	3 (10)	4 (20)	5 (12)	6 (10)	7 (15)	8 (12)	$\sum_{Punkte}$ (104)

Viel Erfolg!

Name: **Matrikel:**

Aufgabe 1 (10 Punkte)

Wir wollen in Java Personen als Objekte mit den folgenden Eigenschaften modellieren:

- Jede Person hat Namen und Alter, die bei der Instanziierung unveränderlich festgelegt werden.
- Jede Person ist entweder Frau oder Mann.
- Eine Frau lacht “hihihihi”, ein Mann “hohohoho”. Für Personen soll es eine Methode `lachen()` geben.
- Wenn eine Frau schläft, dann macht sie “zsss zsss”, ein Mann “schnarch schnarch”. Für Personen soll es eine Methode `schlafen()` geben.
- Wenn man eine Frau nach ihrem Alter fragt, macht sie sich stets fünf Jahre jünger als sie eigentlich ist. Ein Mann gibt dagegen immer sein richtiges Alter an. Für Personen soll es eine Methode `gibAlter()` geben.
- Für jede Person soll es eine Methode `dasBinIch()` geben, die Name, angebliches Alter (nach Aussage der Person), Lach- und Schlafgeräusche ausgibt.

Erstellen Sie in Java eine Klassenhierarchie für Personen mit den oben genannten Eigenschaften. Verwenden Sie in angemessener Weise Vererbung und abstrakte Klassen.

Lösung:

```
public abstract class Person {
    private String name;
    private int    alter;

    public Person(String name, int alter) {
        this.name = name;
        this.alter = alter;
    }

    public abstract String lachen();

    public abstract String schlafen();

    public int gibAlter() { return this.alter; }

    public void dasBinIch() {
        System.out.println("Ich bin " + this.name);
        System.out.println("Mein Alter ist " + this.gibAlter());
        System.out.println("Wenn ich lache, mache ich " + this.lachen());
        System.out.println("Wenn ich schlafe, mache ich " + this.schlafen()
    }
}
```

Name: **Matrikel:**

```
public class Mann extends Person {
    public Mann(String name, int alter) {
        super(name,alter);
    }

    public String lachen() { return "hohohoho"; }

    public String schlafen() { return "schnarch schnarch"; }
}

public class Frau extends Person {
    public Frau(String name, int alter) {
        super(name,alter);
    }

    public String lachen() { return "hihihihi"; }

    public String schlafen() { return "zsss zsss"; }

    public int gibAlter() { return super.gibAlter() - 5; }
}
```

Name: **Matrikel:**

Aufgabe 2 (5+5+5=15 Punkte)

Eine Chiffrierer kann textuelle Nachrichten mit Hilfe eines Schlüssels ver- und entschlüsseln.

(a) Definieren Sie eine allgemeine Java-Schnittstelle für Chiffrierer. Gehen Sie dabei davon aus, dass eine Nachricht (sowohl im Klartext als auch verschlüsselt) als String repräsentiert wird und ein Schlüssel als Integer-Wert.

(b) Implementieren Sie die Schnittstelle aus (a) am Beispiel eines *Dummy-Chiffrierers*, bei dem (unabbhängig vom Schlüssel) Klartext und verschlüsselte Nachricht stets identisch sind.

(c) Implementieren Sie eine Klasse, die zwei beliebige Chiffrierer durch Konkatenation (Hintereinanderausführung) zu einem neuen Chiffrierer kombiniert.

Lösung:

(a)
```
public interface Chiffrierer {
    String verschluesseln(String nachricht, int schluessel);
    String entschluesseln(String nachricht, int schluessel);
}
```

(b)
```
public class DummyChiffrierer implements Chiffrierer {
    public String verschluesseln(String nachricht, int schluessel) {
        return nachricht;
    }

    public String entschluesseln(String nachricht, int schluessel) {
        return nachricht;
    }
}
```

(c)
```
public class KonkatChiffrierer implements Chiffrierer {
    private Chiffrierer chiff1;
    private Chiffrierer chiff2;

    public KonkatChiffrierer(Chiffrierer c1, Chiffrierer c2) {
        this.chiff1 = c1;
        this.chiff2 = c2;
    }

    public String verschluesseln(String nachricht, int schluessel) {
        return chiff2.verschluesseln(chiff1.verschluesseln(nachricht,sc
    }

    public String entschluesseln(String nachricht, int schluessel) {
        return chiff1.entschluesseln(chiff2.entschluesseln(nachricht,sc
    }
}
```

Name: **Matrikel:**

Aufgabe 3 (2+2+2+2+2=10 Punkte)

Es sei der folgende Quelltext gegeben:

```
import java.io.*;
public class Currywurst extends Fastfood {
  public static void main(String[] args) throws IOException {
    new Currywurst().eat();
  }
  // insert code here
}
class Fastfood {
  void eat() throws IOException { }
}
```

Für welche der folgenden Methodendefinitionen, jeweils eingefügt für die Zeile mit dem Kommentar "`insert code here`", lässt sich der Quelltext kompilieren?

(a) `void eat() { }`

(b) `void eat() throws Exception { }`

(c) `void eat(int x) throws Exception { }`

(d) `void eat() throws FileNotFoundException { }`

(e) `void eat() throws RuntimeException { }`

Erläutern Sie jeweils in einem Satz, warum eine Kompilierung möglich bzw. nicht möglich ist.

Lösung:

(a) Compilierung möglich, da eine überschreibende Methode auf das Auslösen einer in der Oberklasse deklarierten Exception verzichten kann.

(b) Compilierung nicht möglich. In der Unterklasse darf keine allgemeinere geprüfte Exception ausgelöst werden.

(c) Compilierung möglich. Weil die Methode in der Unterklasse eine andere Signatur hat, liegt kein Überschreiben vor und die Methode darf beliebige Exceptions auslösen.

(d) Compilierung möglich. In der Unterklasse ist eine Spezialisierung der deklarierten Exception erlaubt.

(e) Compilierung möglich. Ungeprüfte Exceptions unterliegen keinen Beschränkungen.

Name: **Matrikel:**

Aufgabe 4 (15+5=20 Punkte)

Wir wollen eine generische doppelt verkettete Liste implementieren, die die folgenden Operationen unterstützt:

- `append`: Hängt ein neues Element an das Ende der Liste an.
- `search`: Prüft für ein Objekt o, ob sich o in der Liste befindet oder nicht.
- `deleteAt`: Löscht das i-te Element der Liste.

(a) Geben Sie die Klasse komplett an (Klassenrahmen, innere Klassen, Instanzvariablen, Methodenköpfe und Implementierungen).

(b) Geben Sie den Aufwand für die von Ihnen implementierten Listenoperationen an.

Lösung:

(a)
```
public class Liste<T extends Comparable<T>> {

    private class Item {
        Item prev;
        Item next;
        T    value;
    }

    private Item first;
    private Item last;

    public Liste() { }

    public void append(T val) {
        Item item = new Item();

        item.value = val;
        item.next = null;
        item.prev = last;
        if (last==null) {
            first = last = item;
        }
        else {
            last.next = item;
            last = item;
        }
    }

    public boolean search(T val) {
```

Name: **Matrikel:**

```
            Item item = first;

            while (item != null && item.value.compareTo(val)!=0) {
                item = item.next;
            }

            return item != null;
        }

        public void deleteAt(int i) {
            Item item = first;
            Item iprev;
            Item inext;
            int  count = 0;

            while (item != null && count < i) {
                item = item.next;
                count++;
            }

            if (item == null)
                return;

            iprev = item.prev;
            inext = item.next;

            if (iprev != null) {
                iprev.next = inext;
            }
            else {
                first = inext;
            }

            if (inext!=null) {
                inext.prev = iprev;
            }
            else {
                last = iprev;
            }
        }
    }
```

(b) Es sei n die Anzahl der Elemente in der Liste. Dann liegt der folgende Aufwand vor:

`append`: $O(1)$

`search`: $O(n)$

`deleteAt`: $O(n)$

Name: **Matrikel:**

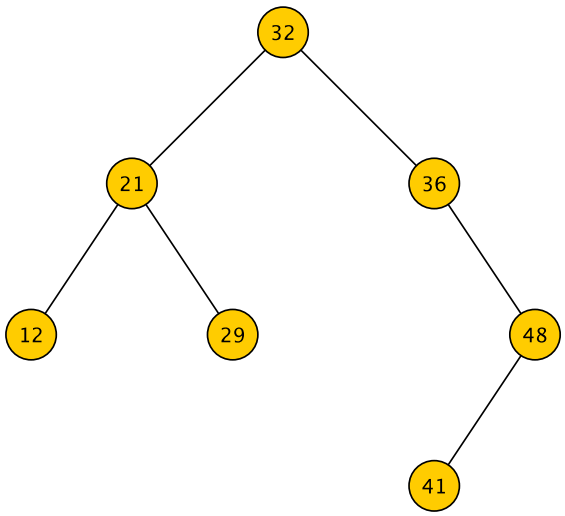
Aufgabe 5 (3+4+2+3=12 Punkte)

(a) Es sei T ein Baum der Höhe n. Wie viele Schlüssel enthält T mindestens, wie viele höchstens?

(b) In einen anfangs leeren Suchbaum werden nacheinander die folgenden Schlüssel eingefügt: 45, 19, 27, 66, 78, 52, 41, 55

Wie sieht der Baum nach dem Einfügen aller Schlüssel aus?

(c) Gegeben sei der folgende Suchbaum:



Geben Sie eine mögliche Reihenfolge der Schlüssel an, durch die dieser Baum bei sequentiellen Einfügen erzeugt würde.

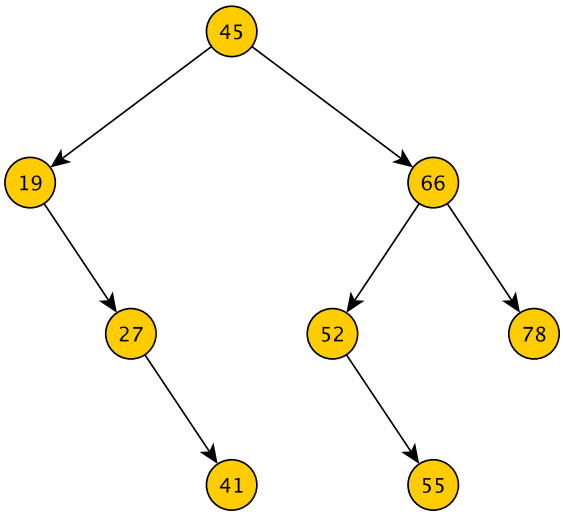
(d) Wir löschen die 48 aus dem Baum von (c). Wie sieht der Baum jetzt aus? Was passiert, wenn wir als nächstes die 32 löschen?

Lösung:

(a) mindestens n, höchstens $2^n - 1$

(b)

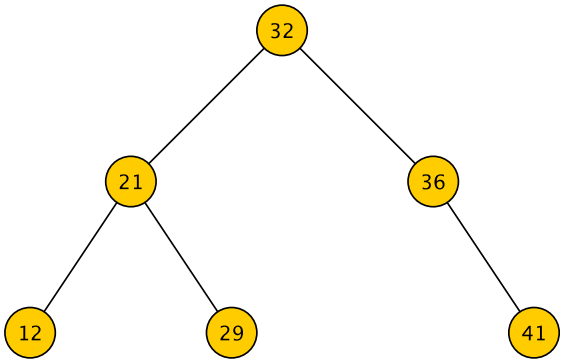
Name: **Matrikel:**



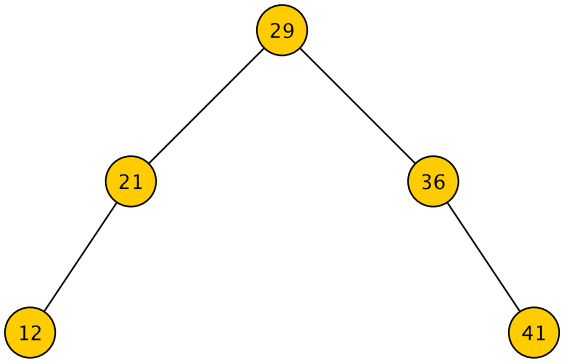
(c) Eine mögliche Reihenfolge ist: 32, 21, 36, 12, 29, 48, 41 (Ebenen des Baums von oben nach unten und in jeder Ebene von links nach rechts, entspricht einer Breitensuche).

Eine andere mögliche Reihenfolge liefert z.B. eine Preoder-Traversierung: 32, 21, 12, 29, 36, 48, 41

(d) Die 48 hat nur einen Unterbaum. Dieser Unterbaum wird hochgezogen:



Für das Löschen der 32 nehmen wir das größte Element des linken Unterbaums in die Wurzel (alternativ das kleinste aus dem rechten Unterbaum):



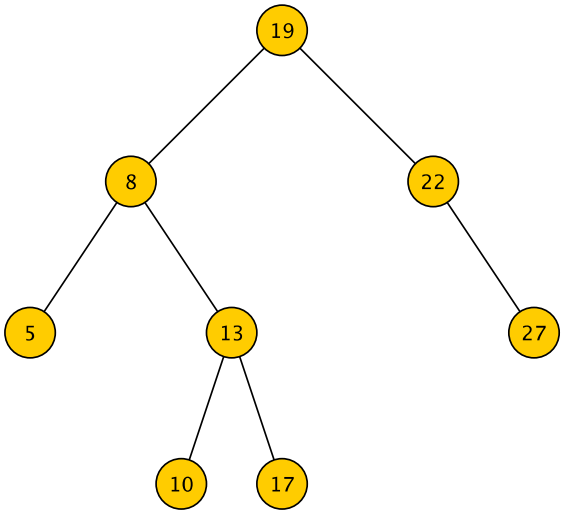
Name: **Matrikel:**

Aufgabe 6 (7+3=10 Punkte)

(a) In einen leeren AVL-Baum werden nacheinander die folgenden Schlüssel eingefügt: 46, 61, 78, 65, 70, 84, 93

Geben Sie für jeden Schlüssel an, wie der AVL-Baum nach dem Einfügen des Schlüssels (inklusive eventuell notwendiger Ausgleichsoperationen) aussieht.

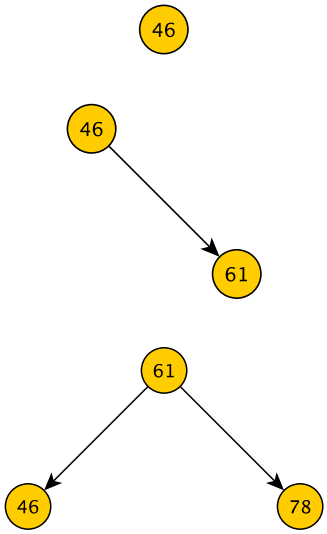
(b) Gegeben sei der folgende AVL-Baum:



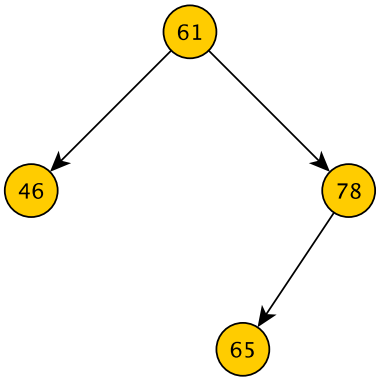
Wie sieht der Baum nach dem Löschen der 5 aus?

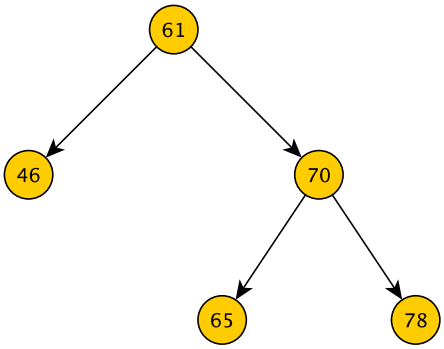
Lösung:

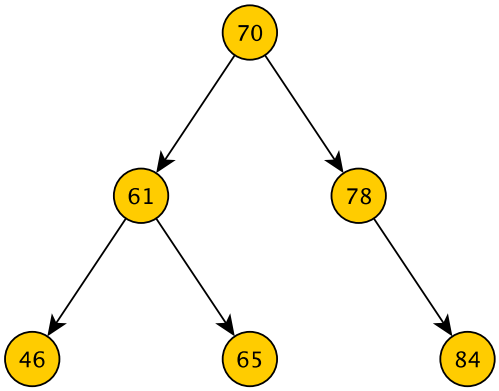
(a)

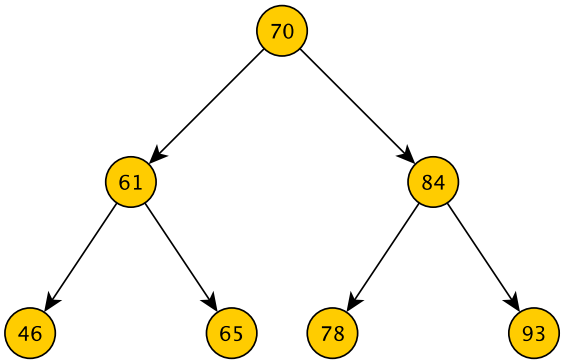


Name: **Matrikel:**



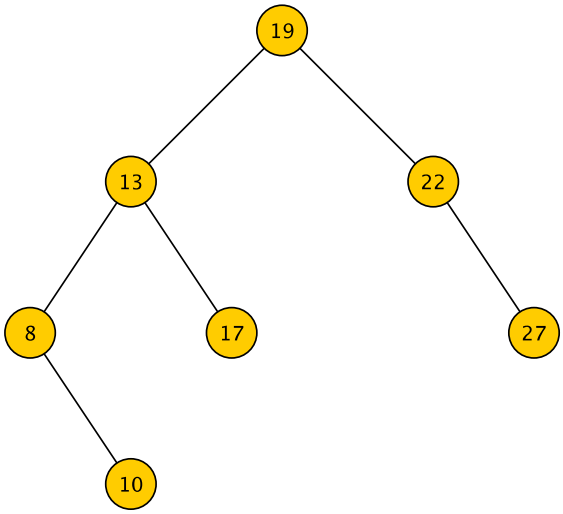






Name: **Matrikel:**

(b) Nach dem Löschen der 5 ist am Knoten 8 das AVL-Kriterium verletzt. Es liegt der zu Fall 3 (c) symmetrische Fall vor, siehe Skript Folie 277f. Es entsteht der folgende Baum:



Name: **Matrikel:**

Aufgabe 7 (6+6+3=15 Punkte)

Gegeben ist die folgende Zahlenfolge:

$$45, 17, 29, 36, 18, 34, 52, 67, 39, 8, 12, 88, 55, 32, 61$$

(a) Erläutern Sie, wie diese Zahlenfolge mit Hilfe des Mergesort-Algorithmus aufsteigend sortiert wird.

(b) Welche Laufzeit hat Mergesort? Begründen Sie Ihre Anwort.

(c) Was versteht man unter einem *stabilen* Sortierverfahren? Ist Mergesort stabil?

Lösung:

(a) Mergesort basiert auf dem Paradigma Teile-und-Herrsche. Solch ein Algorithmus besteht aus einer (rekursiven) Zerlegung in kleinere Probleme und einer Zusammensetzung der gelösten kleineren Probleme. Die Zerlegung endet, wenn für die Teilprobleme eine explizite Lösung vorliegt.

Konkret bei Mergesort:

- Zerlegung: Eine zu sortierende Folge wird in zwei möglichst gleich große Teile zerlegt.
- Zusammensetzung: Die sortierten Folgen der Teilprobleme werden zu einer sortierten Folge durch Mischen zusammengeführt.
- Explizite Lösung: Teilfolgen der Länge 1 sind bereits sortiert. Hier endet die Zerlegung.

Für Skizzen zum Zerlegen und Zusammensetzen siehe Folie 312 und 314.

(b) Bei n Elementen beträgt die Laufzeit $O(n \log n)$.

Begründung:

- Durch das Zerlegen entsteht ein ausgeglichener Baum aus Teilproblemen mit Höhe $O(\log n)$.
- Zwei Teilfolgen der Länge n_l und n_r können in Zeit $O(n_l + n_r)$ zusammengeführt werden. Damit entsteht auf jeder Ebene des Zerlegungsbaums ein Gesamtaufwand für die Ebene von $O(n)$.
- Da der Baum $O(\log n)$ viele Ebenen hat, ist der Gesamtaufwand für Mergesort $O(n \log n)$.

(c) Objekte, die als gleich angesehen werden, haben in der sortierten Folge die gleiche Reihenfolge, wie in der Ausgangsfolge. Mergesort ist stabil.

Name: **Matrikel:**

Aufgabe 8 (5+7=12 Punkte)

Gegeben Sei eine (zunächst leere) Hash-Tabelle der Größe 7, sowie die Hashfunktion

$$h(x) \quad = \quad x \bmod 7$$

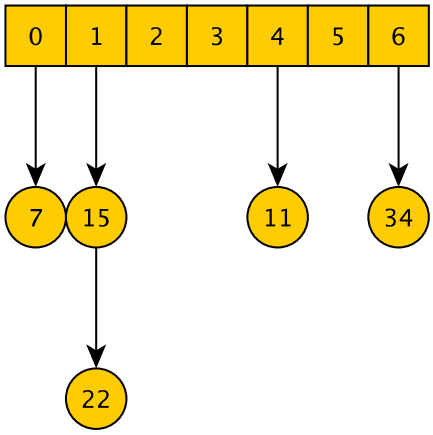
(a) Fügen Sie die folgenden Zahlen in die Hash-Tabelle ein, wobei Kollisionen durch Verkettung behandelt werden:

$$11, 22, 34, 7, 15$$

(b) Implementieren Sie (in Java) eine sinnvolle Hash-Funktion für Strings (ohne die Modulo-Operation).

Lösung:

(a) Nach dem Einfügen der Schlüssel sieht die Hashtabelle wie folgt aus:



(b)
```
public static int hashString(String s) {
      int hash = 0;

      for (int i=0 ; i<s.length() ; i++) {
          hash = (hash<<5) - hash + s.charAt(i);
          // hash = 31*hash + s.charAt(i);

      }

      return hash;
}
```