Institut für Informatik
der Universität München
Prof. Dr. M. Hofmann
Dr. J. Johannsen, Dr. R. Matthes

SS 2004

14.07.2004

# Übungen zur Vorlesung
# **Informatik II**

Probeklausur

Diese Aufgaben sind Beispiele für mögliche Klausuraufgaben. Sie haben den Zweck, Sie mit den Typen von Aufgaben, die in der Klausur gestellt werden könnten, vertraut zu machen. Dieser Satz von Aufgaben entspricht nicht dem Umfang der 120-minütigen Klausur, sondern nur ca. 75 Minuten.

**Aufgabe 1** **4 Punkte**

In dieser Aufgabe finden Sie mehrere Java-Dateien, von denen genau eine nicht kompiliert. Begründen Sie genau, warum sie nicht kompiliert und geben Sie an, wie man durch Ändern, Hinzufügen oder Weglassen einer einzigen Zeile im angegebenen Quelltext erreichen würde, dass er doch kompiliert.

*Hinweis:* Hier geht es nicht um Tippfehler oder vergessene “;” oder Klammern.

```
public interface Autotuer{

    boolean tuerOeffnen();
    boolean fensterHeben(double gewuenschterHub);
}

public class MeineAutotuer implements Autotuer{

    public boolean tuerOeffnen(){
        return false;
        // vorerst immer Ausgabe, dass es nicht funktioniert hat
    }

    protected static void tuerSchliessen(){
        return;
        // soll immer funktionieren
    }

    boolean fensterHeben(double dieHubhoehe){
        return false;
        // vorerst immer Ausgabe, dass es nicht funktioniert hat
    }
}
```

**Aufgabe 2** **6 Punkte**

Lösen Sie nochmals die Hausaufgaben S-12 und/oder S-13 zum Hoare-Kalkül.

– bitte wenden –

**Aufgabe 3** **4 Punkte**

Hier finden Sie ein Java-Programm, das einwandfrei kompiliert, aber bei Ausführung einen Laufzeitfehler erzeugt. Geben Sie an, welcher Art dieser Laufzeitfehler ist (z.B. `ArithmeticException`), und begründen Sie, warum dieser Fehler auftritt.

```
import java.util.Random;

public class Beispiel {

    private final static int KONSTANTE = -20;
    double[] werte;

    public Beispiel(int groesse) {
        werte = new double[groesse];
        Random generator = new Random();
        for (int i=0; i<werte.length; i++)
            werte[i] = generator.nextDouble()*KONSTANTE;
    }

    public static void main(String[] args) {
        Integer i = new Integer(-1);
        multipliziere(i);
        Beispiel meines = new Beispiel(i.intValue());
        System.out.println(meines);
    }

    private static void multipliziere(Integer i) {
        int summe = 0;
        for (int l = 0; l < KONSTANTE; l++)
            summe += i.intValue();
        i = new Integer(summe);
    }
}
```

**Aufgabe 4** **6+6 Punkte**

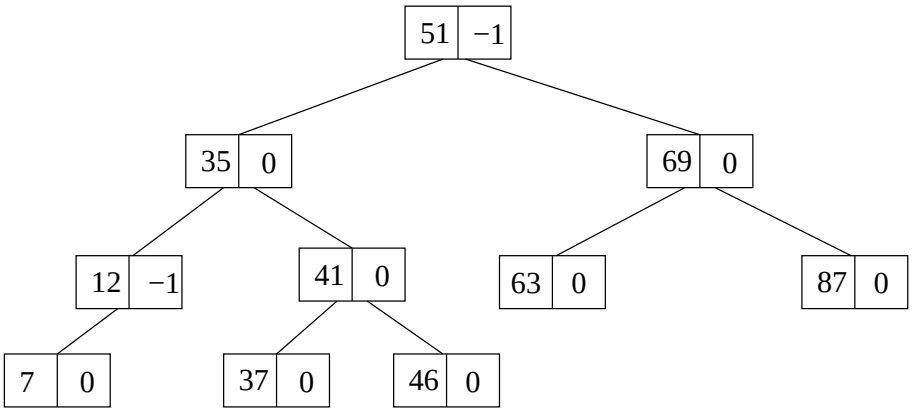
Auf der Vorlesungshomepage finden Sie die Datei `Pizza.java`, die 7 Klassen enthält. Bearbeiten Sie die folgenden Aufgaben ohne Verwendung eines Java-Systems:

a) Zeichnen Sie ein UML-Klassendiagramm für alle Klassen außer `Pizza` mit allen Beziehungen dieser Klassen. Dabei sollen Variablen und Methoden nicht dargestellt werden.

b) Zeichnen Sie ein UML-Objektdiagramm für den Inhalt der Variablen `meinePizza` direkt nach Ausführung der `main`-Methode von `Pizza`.

**Aufgabe 5** **6 Punkte**

Betrachten Sie den folgenden AVL-Baum:




Jedes Rechteck symbolisiert dabei einen inneren Knoten `Build(...)` des Baumes, der Eintrag im linken Feld ist der Datensatz, und der im rechten der Balancefaktor des Knotens. Die leeren Blätter `Empty()` sind um der Übersichtlichkeit willen nicht dargestellt.

Zeichnen Sie den AVL-Baum, der entsteht, wenn Sie in diesen Baum mit der AVL-Einfügeoperation den Wert 43 einfügen.

**Aufgabe 6** **12 Punkte**

Schreiben Sie eine Unterklasse `MeineListe` von `java.util.LinkedList`, die eine neue Methode enthält, mit der eine gegebene Funktion auf alle Elemente einer Liste angewendet wird, ähnlich der Funktion `List.map` in OCaml. Die anzuwendende Funktion wird dabei in Form eines Objekts vom Typ `Mapper` übergeben, der wie folgt definiert ist:

```
public interface Mapper {
   Object map( Object o ) ;
}
```

Die zu implementierende Methode ist demnach:

```
void mapListe( Mapper m )
```

*Hinweis:* Aus der Klasse `java.util.LinkedList` sollten Sie nur die Methode

```
ListIterator listIterator()
```

verwenden, die einen Iterator für die Liste liefert. Das Interface `java.util.ListIterator` spezifiziert bekanntlich unter anderem die folgenden Methoden:

```
void add(Object o)
boolean hasNext()
boolean hasPrevious()
Object next()
Object previous()
void remove()
void set(Object o)
```