RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

Lehrstuhl für Hochleistungsrechnen
RWTH Aachen · D-52074 Aachen · GERMANY
http://programmierung.informatik.rwth-aachen.de

Prof. Christian Bischof, Ph.D.
Andre Vehreschild, Jakob T. Valvoda

Übung *Programmierung WS 06/07* – Blatt 8

Lösungen müssen bis zum **18. Dezember 2006, 17:00 Uhr** in den Kasten Ihrer Übungsgruppe eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55). Bitte vergessen Sie nicht, die **Nummer Ihrer Übungsgruppe**, sowie die **Namen** und **Matrikelnummern** der Mitglieder Ihrer 2er-Gruppe auf jedes Lösungsblatt Ihrer Abgabe zu schreiben. Bitte heften Sie ihre Blätter zusammen.

Alle erstellten Java-Programme sind sowohl in gedruckter Form abzugeben, als auch per E-Mail an den jeweiligen Tutor zu senden. Vergessen Sie auch bei der elektronischen Abgabe per E-Mail nicht die Angabe der **Nummer Ihrer Übungsgruppe**, sowie die jeweiligen **Namen** und **Matrikelnummern**.

Die **Präsenz-Übung** findet statt am **Freitag, den 15. Dezember 2006**. Weitere Informationen sind auf der **Webseite der Vorlesung** veröffentlicht.

Aufgabe 1 (2 + 4 = 6 Punkte)

Eine Hierarchie aus geometrischen Basisobjekten `Primitive`, sowie die einzelnen Objekte Kugel `Sphere` und Kegel `Cone` und die schwer vorstellbare Hyper-Sphäre `HyperSphere` seien wie folgt implementiert:

```
public abstract class Primitive {
  public String getName() {
    return "primitive";
  }
}
public class Sphere extends Primitive {
  public String getName() {
    return "sphere";
  }
  public float getRadius() {
    return 1.0f;
  }
}
public class Cone extends Primitive {
  public String getName() {
    return "cone";
  }
  public float getRadius() {
    return 0.5f;
  }
```

```
  public float getHeight() {
    return 1.0f;
  }
}
public class HyperSphere extends Sphere {
  public String getName() {
    return "h-sphere";
  }
  public int getDimension() {
    return 5;
  }
}
```

Analysieren Sie das folgende Hauptprogramm

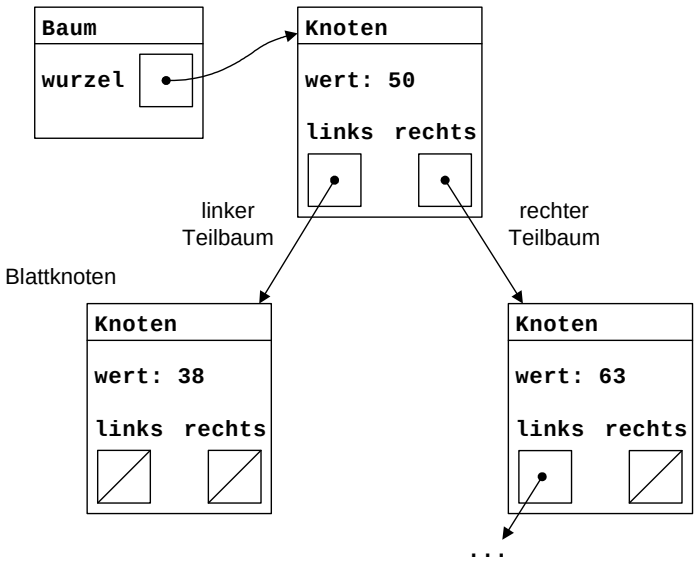
```
public class GraphicsJuggler {
  public static void main( String [] args ) {
    Primitive primitive;
    Sphere sphere;
    Cone cone;
    HyperSphere hyper;

    // Anweisung 1
    hyper = new HyperSphere();
    System.out.println( hyper.getDimension() );
    // Anweisung 2
    primitive = new Cone();
    System.out.println( primitive.getName() );
    // Anweisung 3
    primitive = new Sphere();
    System.out.println( primitive.getRadius() );
    // Anweisung 4
    primitive = new HyperSphere();
    System.out.println( primititve.getName() );
    // Anweisung 5
    hyper = new Sphere();
    System.out.println( hyper.getRadius() );
    // Anweisung 6
    primitive = (Primitive)(Sphere)new HyperSphere();
    System.out.println( primitive.getName() );
    // Anweisung 7
    cone = (Cone)(Primitive)new Sphere();
    System.out.println( cone.getRadius() );
    // Anweisung 8
    sphere = (Sphere)(Primitive)new HyperSphere();
    System.out.println( sphere.getName() );
  }
}
```

a) Geben Sie die Klassenhierarchie als UML-Diagramm für die Klassen `Primitive`, `Sphere`, `HyperSphere` und `Cone` an. Geben Sie die Attribute und die Methoden an. Verwenden Sie hierfür die In Übung 7 Aufgabe 3 eingeführte Notation.

b) Analysieren Sie die Stellen im Hauptptrogramm von `GraphicsJuggler`, welche mit den Kommentaren `//Anweisung 1` bis `//Anweisung 8` gekennzeichnet sind. Geben Sie jeweils an, ob die Anweisungen (d.h. die nachfolgende Zuweisung und der Methodenaufruf) beim Kompilieren oder bei der Ausführung zu Fehlern führen. Schreiben Sie Ihre Antwort und Begründung in eigenen Worten. Falls die Anweisungen korrekt sind, dann geben Sie jeweils die Ausgabe an.

Aufgabe 2 (2 + 2 + 2 + 3 = 9 Punkte)

Der in der Vorlesung angesprochene binäre Baum zeichnet sich dadurch aus, dass jeder Knoten n_i des Baums auf genau zwei Teilbäume $L(n_i)$ und $R(n_i)$ verweist. Die Teilbäume eines Knotens können auch leer sein. Ein Knoten n_j ohne weitere Verweise auf Teilbäume (d.h. $L(n_j) = R(n_j) = \emptyset$ bzw. beide Attribute sind `null`) wird Blattknoten genannt.



Eine Erweiterung des binären Baumes sind die binären Suchbäume. Binäre Suchbäume haben die (Zusatz-)Eigenschaft, dass für jeweils zwei Knoten n_i und n_j im Baum gilt

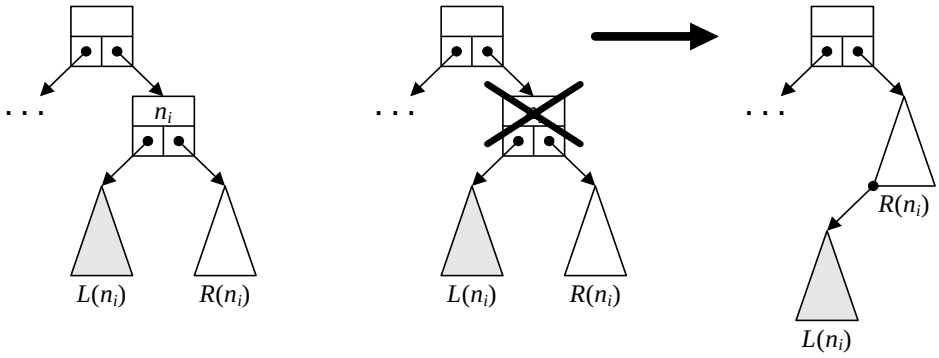
$$\begin{aligned} \text{Wert}(n_j) &< \text{Wert}(n_i) \quad \text{falls } n_j \in L(n_i) \\ \text{Wert}(n_i) &\leq \text{Wert}(n_j) \quad \text{falls } n_j \in R(n_i) \end{aligned}$$

In binären Suchbäumen kann effizient gesucht werden. Es kann ebenfalls effektiv nach dem kleinsten und größten Element gesucht werden. Das kleinste Element wird ermittelt, indem ausgehend von einer Wurzel n_i jeweils der linke Teilbaum $L(n_i)$ durchsucht wird. Ist der Teilbaum leer, dann ist n_i das kleinste Element. Ansonsten wird die Wurzel des Teilbaums $L(n_i)$

und ihr linker Teilbaum betrachtet, bis das Element gefunden ist. Das größte Element sucht man analog im rechten Teilbaum.

Soll ein neues Element in den Suchbaum eingetragen werden, dann muss auf das Einhalten der obigen Größenrelation geachtet werden. Hierzu geht man wie folgt vor: Ist der Baum leer, so wird das neue Element die Wurzel. Befinden sich bereits Knoten im Baum, so wird der Baum durchlaufen, wobei anhand des Werts des neuen Elements jeweils entschieden wird, ob der linke oder der rechte Teilbaum weiter durchlaufen werden soll. Ist der jeweilige linke oder rechte Teilbaum leer, dann wird das neue Element dort eingefügt.

Das Löschen eines Knotens in einem binären Suchbaum muss ebenfalls die Größenrelation erhalten. Soll ein Knoten n_i gelöscht werden, dann ersetzt man ihn durch die Wurzel des Teilbaumes $R(n_i)$. Den Teilbaum $L(n_i)$ fügt man an den (per Definition leeren) linken Teilbaum des kleinsten Knotens von $R(n_i)$ an. Falls jedoch der rechte Teilbaum $R(n_i)$ leer ist, dann wird der Knoten n_i mit der Wurzel des Teilbaums $L(n_i)$ ersetzt. Das Löschen ist in der nachfolgenden Abbildung für den allgemeinen Fall schematisch dargestellt:



Hinweis: Der kleinste Knoten $n_{\min}$ im Teilbaum $R(n_i)$ muss nicht notwendigerweise ein Blattknoten sein, sondern kann einen Knoten $R(n_{\min}) \neq \emptyset$ haben.

a) Implementieren Sie die Klassen `Baum` und `Knoten`, welche einen binären Suchbaum für `int`-Werte repräsentieren. Implementieren Sie für die Klasse `Baum` den Konstruktor `Baum()` der einen leeren binären Suchbaum initialisiert. Implementieren Sie für die Klasse `Knoten` den Konstruktor `Knoten( int )` welcher einen Knoten mit dem übergebenen Wert erzeugt und beide Teilbäume initial auf `null` setzt.

b) Implementieren Sie in der Klasse `Baum` die Methode `einfuegen( int )`, welche für den angegebenen Wert einen neuen Blattknoten erzeugt und diesen nach dem obigen (rekursiven) Verfahren in den binären Suchbaum einfügt.

c) Implementieren Sie in der Klasse `Baum` die Methode `boolean suche( int )`, welche den angegebenen Wert im binären Suchbaum sucht und den Wert `true` zurückliefert, falls der Wert im Baum vorhanden ist.

d) Implementieren Sie in der Klasse `Baum` die Methode `loesche( int )`, welche den ersten Knoten löscht, der den angegebenen Wert hat. Verwenden Sie hierzu die oben beschriebene und in der Abbildung skizzierte Methode.