RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHR- UND FORSCHUNGSGEBIET INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY
http://www-i2.informatik.rwth-aachen.de/lufgi2

Prof. Dr. Jürgen Giesl
Darius Dlugosz, Thomas v. d. Maßen, Antje Nowack

Übung *Informatik I - Programmierung* - Blatt 8

(Lösungsvorschlag)

Aufgabe 1

```
/**
 * Die Klasse ZeichenElement repraesentiert eine Glied (Zeichen)
 * einer Zeichenkette.
 *
 * @see Zeichenkette
 *
 * @author Darius Dlugosz
 * Umgebung: JDK 1.3, Windows 2000
 * Erstellt: 11.12.2001
 * Letzte Aenderung: 14.12.2001
 */
public class ZeichenElement {

  /** Das Zeichen */
  private char zeichen;

  /** Das naechste Zeichen(element) der Kette */
  private ZeichenElement next;

  /**
   * Erzeugt ein neues Zeichen(element).
   *
   * @param das Zeichen das in der Kette aufgenommen wird.
   */
  public ZeichenElement(char zeichen) {
    this.zeichen = zeichen;
  }

  /**
   * Setzt das Zeichen neu.
   *
   * @param zeichen das neue Zeichen
```

```
 */
public void setZeichen(char zeichen) {
  this.zeichen = zeichen;
}

/**
 * Setzt den Nachfolger des Zeichen(elementes).
 *
 * @param next der neue Nachfolger
 */
public void setNext(ZeichenElement next) {
  this.next = next;
}

/**
 * Gibt das Zeichen des Elementes.
 *
 * @return das aktuelle Zeichen
 */
public char getZeichen() {
  return zeichen;
}

/**
 * Gibt das naechste Zeichen(element) zurueck.
 *
 * @return das naechste Zeichenelement.
 */
public ZeichenElement getNext() {
  return next;
}

/**
 * Vergleicht das Zeichen mit dem Zeichen von ele.
 *
 * @param ele ein Zeichen(element)
 *
 * @return true, falls das Zeichen gleich dem von ele ist.
 */
public boolean equals(ZeichenElement ele) {
  if (ele == null) return false;
  else return zeichen == ele.getZeichen();
}
```

```
///

  public String toString() {
    return (new Character(zeichen)).toString();
  }

  public String gibString() {
    if (next == null) return toString();
    else return toString() + next.gibString();
  }

}

/**
 * Die Klasse Zeichenkette repraestiert ein Folge
 * von Zeichen.
 *
 * @see ZeichenElement
 *
 * @author Darius Dlugosz
 * Umgebung: JDK 1.3, Windows 2000
 * Erstellt: 12.12.2001
 * Letzte Aenderung: 14.12.2001
 */
public class Zeichenkette {

  /** Das erste (Zeichen-)Element der Zeichenkette */
  private ZeichenElement kopf;

  /** Das letzte (Zeichen-)Element der Zeichenkette (Liste) */
  private ZeichenElement fuss;


  /**
   * Erzeugt eine leere Zeichenkette.
   */
  public Zeichenkette() {
  }

  /**
   * Liefert das erste Zeichen der Zeichenkette.
   *
   * @return das erste Zeichen der Zeichenkette
   */
  public ZeichenElement getKopf() {
```

```
  return kopf;
}

/**
 * Fuegt am Ende der Zeichenkette das Zeichen zeichen ein.
 *
 * @param zeichen das Zeichen das am Ende hinzugefuegt wird
 */
public void add(char zeichen) {
  if (fuss == null)  kopf = fuss = new ZeichenElement(zeichen);
  else {
    ZeichenElement z = new ZeichenElement(zeichen);
    fuss.setNext(z);
    fuss = z;
  }
}

/**
 * Bestimmt die Laenge der Zeichenkette.
 *
 * @return Laenge der Zeichenkette
 */
public int laenge() {
  return laenge(kopf);
}

/**
 * Liefert die Laenge der Zeichenkette beginnend mit rest.
 *
 * @param rest das erste Element der retslichen Zeichenkette
 *
 * @return Laenge der Zechein(elementen) beginnend mit rest
 */
private static int laenge(ZeichenElement rest) {
  if (rest == null) return 0;
  else return 1 + laenge(rest.getNext());
}

/**
 * Ueberprueft, ob prefix der Anfang der Zeichenkette ist.
 *
 * @param prefix eine Zeichenkette
 *
 * @return true, falls die Zeichenkette mit prefix beginnt;
```

```
 *         false, sonst.
 */
public boolean beginntMit(Zeichenkette prefix) {
  if (prefix == null) return false;
  else return beginntMit(kopf, prefix.getKopf());
}

/**
 * Ueberpruft ob der Anfang der Zeichenkette beginnend mit ele1 gleich
 * der Zeichenkette, beginnend mit ele2 ist.
 *
 * @return true, falls der Anfang der Zeichenkette, die mit ele1 beginnt mit
 *         der Zeichenkette, die mit ele2 beginnt, gleich ist;
 *         false, sonst.
 */
public static boolean beginntMit(ZeichenElement ele1, ZeichenElement ele2) {
  if (ele1 == null && ele2 == null) return true;
  else if (ele1 != null && ele2 == null) return true;
  else if (ele1 == null && ele2 != null) return false;
  else if (ele1.equals(ele2)) return beginntMit(ele1.getNext(), ele2.getNext());
  else return false;
}

/**
 * Ueberprueft, ob suffix das Ende der Zeichenkette ist.
 *
 * @param prefix eine Zeichenkette
 *
 * @return true, falls die Zeichenkette mit suffix endet;
 *         false, sonst.
 */
public boolean endetMit(Zeichenkette suffix) {
  if (suffix == null) return false;
  else return endetMit(kopf, suffix.getKopf());
}


/**
 * Ueberpruft ob das Ende der Zeichenkette beginnend mit ele1 gleich
 * der Zeichenkette, beginnend mit ele2 ist.
 *
 * @return true, falls das Ende der Zeichenkette, die mit ele1 beginnt mit
 *         der Zeichenkette, die mit ele2 beginnt, gleich ist;
 *         false, sonst.
```

```
 */
private static boolean endetMit(ZeichenElement ele1, ZeichenElement ele2) {
  ZeichenElement beginn = gibBeginn(ele1, ele2);
  if (beginn == null) return false;
  else if (endetMit2(beginn, ele2)) return true;
  else return endetMit(ele1.getNext(), ele2);
}

/**
 * Ueberpruft ob das Ende der Zeichenkette beginnend mit ele1 gleich
 * der Zeichenkette, beginnend mit ele2 ist.
 *
 * @return true, falls das Ende der Zeichenkette, die mit ele1 beginnt mit
 *         der Zeichenkette, die mit ele2 beginnt, gleich ist;
 *         false, sonst.
 */
private static boolean endetMit2(ZeichenElement ele1, ZeichenElement ele2) {
  if (ele1 == null && ele2 == null) return true;
  else if (ele1 != null ^ ele2 != null) return false;
  else if (ele1.equals(ele2)) return endetMit2(ele1.getNext(), ele2.getNext());
  else return false;
}

/**
 * Ueberprueft, ob die Zeichenkette zeichenfolge in der Zeichenkette
 * enthalten ist.
 *
 * @param zeichenfolge eine Zeichenkette
 *
 * @return true, falls die zeichenfolge in der Zeichenkette enthalten ist.
 */
public boolean enthaelt(Zeichenkette zeichenfolge) {
  if (zeichenfolge == null) return false;
  else return enthaelt(kopf, zeichenfolge.getKopf());
}

/**
 * Ueberpruft ob das die Zeichenkette beginnend mit ele2 in der Zeichenkette,
 * beginnend mit ele1, enthalten ist.
 *
 * @param zeichenfolge eine Zeichenkette
 *
 * @return true, falls das die Zeichenkette, die mit ele2 beginnt in
 *         der Zeichenkette, die mit ele2 beginnt, enthalten ist;
```

```
   *         false, sonst.
   */
  private static boolean enthaelt(ZeichenElement ele1, ZeichenElement ele2) {
    ZeichenElement beginn = gibBeginn(ele1, ele2);
    if (beginn == null) return false;
    else if (beginntMit(beginn, ele2)) return true;
    else return enthaelt(ele1.getNext(), ele2);
  }

  /**
   * Liefert das erste Zeichenelement in start, das mit dem Zeichen(element)
   * ele beginnt.
   *
   * @param start Beginn einer Zeichenkette
   * @param ele  ein Zeichenelement
   *
   * @return das erste Zeichenelement in start, das mit dem Zeichen(element)
   *         ele beginnt.
   */
  private static ZeichenElement gibBeginn(ZeichenElement start, ZeichenElement ele) {
    if (ele == null) return null;
    else if (start == null) return null;
    else if (ele.equals(start)) return start;
    else return gibBeginn(start.getNext(), ele);
  }

}
```

Aufgabe 2

```
/**
 * Datentyp fuer Elemente einer Liste.
 * Jedes Element enthaelt einen Baumknoten Node
 *
 * @see List, Node
 *
 * @author Darius Dlugosz
 * Umgebung: JDK 1.3, Windows 2000
 * Erstellt: 11.12.2001
 * Letzte Aenderung: 13.12.2001
 */
public class Element {
```

```
/** Baumknoten (der Inhalt des Elementes) */
private Node node;

/** Nachfolger des Elementes */
private Element next;

/**
 * Erzeugt ein neues Listenelement ohne Nachfolger.
 * Das Element enthaelt ein Baumknotenelement von Typ Node, das einen Baumknoten
 * repraestieret. Der Baumknoten wird mit name beschriftet.
 *
 * @param name Element mit einem Baumknoten, dessen Beschriftung name ist
 */
public Element(String name) {
  this.node = new Node(name);
}

/**
 * Liefert den Baumknoten des Elementes zurueck.
 *
 * @return name Baumknoten des Elementes
 */
public Node getNode() {
  return node;
}

/**
 * Setzt den Nachfolger des Elementes auf next.
 *
 * @param next der neue Nachfolger
 */
public void setNext(Element next) {
  this.next = next;
}

/**
 * Liefert den Nachfolger des Elmenetes zurueck.
 *
 * @return Der Nachfolger des Elementes
 */
public Element getNext() {
  return next;
}
```

```
  /**
   * @return Name des Nachfolgers
   */
  public String toString() {
    return node.getName();
  }

}
```

```
/**
 * Liste der Nachfolger eines Knotens
 *
 * @see
 *
 * @author Darius Dlugosz
 * Umgebung: JDK 1.3, Windows 2000
 * Erstellt: 11.12.2001
 * Letzte Aenderung: 13.12.2001
 */
public class List {

  /** Das Erste Element der Nachfolger(liste). */
  private Element first;

  /** Das letzte Element der Nachfolger(liste). */
  private Element last;

  /**
   * Erzeugt eine leere Nachfolgerliste
   */
  public List() {
```

```
}

/**
 * Liefert das erste Nachfolgerelement zurueck.
 *
 * @return das erste Element der Nachfolgerliste.
 */
public Element getFirst() {
  return first;
}

/**
 * Liefert das erste Element der Liste mit dem Namen name zurueck.
 *
 * @param name der Name nach dem in der Liste gesucht wird
 *
 * @return das erste Element mit dem Namen name
 */
public Element getElement(String name) {
  return getElement(name, first);
}

/**
 * Liefert das erste Element der Liste, beginnend mit first,  mit dem Namen
 * name zurueck.
 *
 * @param name der Name nach dem in der Liste gesucht wird
 * @param first der Anfangselement mit der die Suche startet
 *
 * @return das erste Element mit dem Namen name
 */
public Element getElement(String name, Element first) {
  if (first == null) return null;
  else if (first.getNode().getName().equals(name)) return first;
  else return getElement(name, first.getNext());
}

/**
 * Liefert den ersten Nachfolgerknoten der Liste mit dem Namen name zurueck.
 *
 * @param name der Name nach dem in der Liste gesucht wird
 *
 * @return der erste Nachfolgerknoten der Liste mit dem Namen name
 */
```

```
  public Node getNode(String name) {
    Element element = getElement(name);
    if (element != null) return element.getNode();
    else return null;
  }

  /**
   * Fuegt der Nachfolgerliste an letzter Stelle einen Knoten mit der
   * Beschriftung name.
   *
   * @param name die Beschriftung des Knotens, der hinzugefuegt wird
   */
  public void add(String name) {
    if (first == null) first = last = new Element(name);
    else {
      last.setNext(new Element(name));
      last = last.getNext();
    }
  }

  /**
   *
   */
  public String toString() {
    return "( " + toString(first) + ")";
  }

  /**
   *
   */
  private static String toString(Element first) {
    if (first != null) return first.toString() + " " + toString(first.getNext());
    else return "";
  }

}
```

```
/**
 * Datentyp fuer Elemente eines Baumes.
 *
 * @see
 *
 * @author Darius Dlugosz
 * Umgebung: JDK 1.3, Windows 2000
 * Erstellt: 11.12.2001
 * Letzte Aenderung: 13.12.2001
 */
public class Node {

  /** Beschriftung des Knotens */
  private String name;

  /** Liste der Nachfolgerknoten; die Elemente der Liste sind vom Typ "Element" */
  private List children;

  /**
   * Erzeugt ein neues Knoten ohne Nachfolger mit der Beschriftung name.
   *
   * @param name Name des Elementes
   */
  public Node(String name) {
    this.name = name;
    //children = new List();
  }

  /**
   * @return name Name des Elementes
   */
  public String getName() {
    return name;
  }

  /**
   * @param name Name, der das Element erhalten soll
   */
```

```
public void setName(String name) {
  this.name = name;
}

/**
 * @return Nachfolger des Elements
 */
public List getChildren() {
  return children;
}

/**
 * @param next Nachfolger, den das Element erhalten soll
 */
public void setChildren(List children) {
  this.children = children;
}

/**
 * @return (nach Breitensuche) Knoten mit der Beschriftung name, falls es einen gibt
 *         null, falls keiner der Knoten den Namen name besitzt.
 */
public Node getNode(String name) {
  Node res = null;
  if (this.name.equals(name)) res = this;
  else {
    if (children != null) {
      res = children.getNode(name);
      if (res == null) {
        boolean found = false;
        // name in Nachfahren in children suchen
        for (Element childElement = children.getFirst(); childElement != null &&
                                  !found; childElement = childElement.getNext()) {
          res = childElement.getNode().getNode(name);
          if (res != null) {
            found = true;
          }
        }
      }
    }
  }
  return res;
}
```

```
  /**
   * @return String des Baumes beginnend mit diesem Knoten.
   */
  public String toString() {
    return "[" + name + " " + children.toString() + "]";
  }

}
```

```
/**
 * Die Klasse Baum repräsentiert einen Baum.
 *
 * @see
 *
 * @author Darius Dlugosz
 * Umgebung: JDK 1.3, Windows 2000
 * Erstellt: 12.12.2001
 * Letzte Aenderung: 14.12.2001
 */
public class Tree {

  /**
   * Die Wurzel des Baumes
   */
  private Node root;

  /**
   * Erzeugt einen Baum mit nur einem Knoten (Wurzel),
   * dessen Bescfriftung name ist.
   *
   * @param name die Beschriftung der Wurzel
   */
  public Tree(String name) {
    root = new Node(name);
```

```
}

/**
 * Dursucht (mit Breitensuche) die Knoten des Baumes nach der Beschriftung name,
 * und liefert anschliessend den Knoten.
 *
 * @param name Die Beschriftung die der gesuchte Knoten enthalten soll.
 *
 * @return der erste mit Breitensuche gefundenen Knoten mit der Beschriftung name
 *         null, falls der Baum keinen Knoten mit name enthält
 */
public Node getNode(String name) {
  if (root == null) return null;
  else return root.getNode(name);
}

/**
 * Fuegt dem Knoten mit der Beschriftung parent, einen Nachfolgerknoten mit
 * der Beschriftung child an letzter Stelle in der Liste der Nachfolger ein.
 * Der Knoten mit der Beschriftung parent wird mit Breitensuche gesucht.
 */
public void add(String parent, String child) {
  Node node = getNode(parent);
  if (node != null) {
    if (node.getChildren() != null) node.getChildren().add(child);
    else  {
      List children = new List();
      children.add(child);
      node.setChildren(children);
    }
  }
}

/**
 * Gibt (mit Tiefensuche) den Baum auf dem Bildschirm aus.
 */
public void print() {
  if (root != null) print(root, "");
}

/**
 *
 */
private void print(Node node, String prefix) {
```

```
    System.out.println(prefix + node.getName());
    if (node.getChildren() != null)
    for (Element child = node.getChildren().getFirst(); child != null;
                                                    child = child.getNext()) {
      print(child.getNode(), prefix + "  ");
    }
  }

}
```