

Übung 10: Dynamische Datenstrukturen und Rekursion

Abgabetermin: TT.MM.JJJJ

Name: _____

Matrikelnummer: _____

Gruppe: G1 (Prähofer) G2 (Wolfinger) G3 (Wolfinger)

Aufgabe	Punkte	gelöst	abzugeben schriftlich	abzugeben elektronisch	Korr.	Pkte
Aufgabe 09.X	12	<input type="checkbox"/>	Prosabeschreibung Java-Programm Testergebnisse	Java-Programm	<input type="checkbox"/>	

Aufgabe 10.X: Elementare Operationen auf linearen Listen

Gegeben seien folgende zwei Klassen *List* und *Node* zur Repräsentation von einfach-verketteten linearen Listen:

```
class List {
    Node head; // first node in list
    public List() { ... }
} // List
```

```
class Node {
    int val; // value in node
    Node next; // next node in list
    public Node(int v, Node n) {...}
} // Node
```

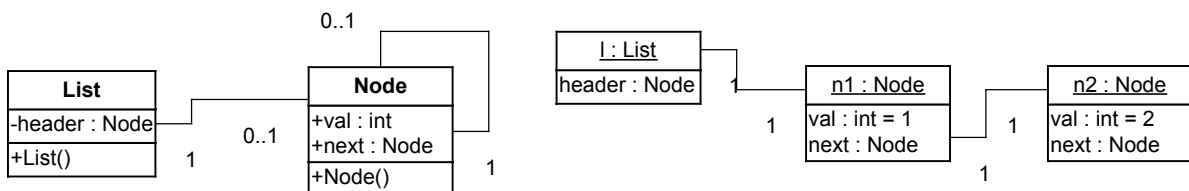
Vervollständigen Sie die beiden Konstruktoren und implementieren Sie in der Klasse *List* folgende Methoden:

- *void prepend(Node n)*: fügt den Wert val vorne in die Liste ein,
- *void append(Node n)*: fügt den Wert val hinten in die Liste ein,
- *boolean isSorted()*: stellt fest, ob die Werte der Knoten der Liste aufsteigend sortiert sind,
- *void insert(Node n)*: fügt den Wert val in eine sortierte Liste an der richtigen Stelle ein, sodass die neue Liste wieder sortiert ist und
- *String toString()*: liefert eine Zeichenkette mit den Werten der Knoten, durch Beistriche voneinander getrennt.

Testen Sie Ihre Methoden ausführlich, auch für Fehlerfälle.

Lösungsidee

Unten eine grafische Darstellung der Klassen *List* und *Node*, sowie ein Objektdiagramm, das zeigt, wie die Objektstruktur einer Liste mit zwei Knoten aussieht, beides in UML:



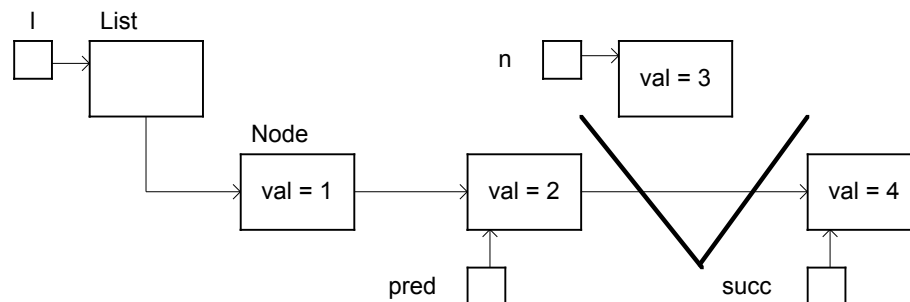
Prosabeschreibung

Klasse Node

- Int-Variable val um den Wert zu speichern
- Referenzvariable next vom Typ Node
- Konstruktor mit Wert val und nächsten Knoten n

Klasse List

- Referenzvariable head zeigt vom Typ Node auf den ersten Knoten; ist null wenn Liste leer
- Konstruktor List ohne Parameter legt eine leere Liste an
- Methode prepend:
 - neuer Knoten wird angelegt mit neuem Wert und aktueller head als next;
 - dieser neue Knoten wird head
- Methode append:
 - man läuft die Kette der Knoten durch bis man am Ende ist, d.h. next vom aktuellen Knoten null ist
 - dann erzeugt man einen neuen Knoten mit Wert und next = null; dieser wird als letzter Knoten angehängt
 - ist die Liste leer, erzeugt man einen neuen Knoten und dieser wird head
- Methode sorted:
 - Man läuft die Kette der Knoten durch bis der next-Knoten des aktuellen Knotens null ist
 - Man vergleicht jeweils den aktuellen und den nächsten mit \leq ; ist dies nicht der Fall gibt man false zurück
 - Ist man die Kette ganz durchgelaufen und hat man keine Verletzung der \leq -Beziehung gefunden gibt man true zurück
 - Ist die Liste leer, gibt man sofort true zurück
- Methode insert (siehe Grafik)
 - Man überprüft auf sorted; wenn nicht sorted dann Fehler und return
 - Man verwendet dann zwei Variablen succ und pred, die die Einfügestelle darstellen (pred = Vorgänger und succ = Nachfolger, d.h. Einfügen nach pred und vor succ)
 - Suchen der Einfügestelle:
 - Am Anfang wird succ gleich head und pred = null (repräsentiert Einfügestelle vor head)
 - Man läuft nun durch die Kette der Knoten indem man pred auf succ und succ auf succ.next stellt bis entweder Ende erreicht (succ == null) oder die Einfügestelle gefunden
 - Einfügen: Man unterscheidet folgende Fälle
 - pred == null: Einfügen des neuen Knotens an erste Stelle als neuer head
 - sonst Einfügen nach pred mit pred.next = neuer Knoten n und n.next = succ
- Methode toString
 - Durchlaufen der Kette der Knoten und Wert jedes Knotens in einen StringBuffer geben

**Java-Programm**

```

class Node {

    int val;
    Node next;

    public Node(int val) {
        this(val, null);
    }
    public Node(int val, Node next) {
        this.val = val;
        this.next = next;
    } // Node

} // Node

class List {

    Node head;

```

```

public List() {
    head = null;
} // List

public void prepend(int val) {
    Node n = new Node(val);
    n.next = head;
    head = n;
} // prepend

public void append(int val) {
    Node n = new Node(val);
    if (head == null)
        head = n;
    else {
        Node last = head;
        while (last.next != null) {
            last = last.next;
        }
        ; // while
        last.next = n;
    } // else
} // prepend

public boolean sorted() {
    if (head == null)
        return true;
    else {
        Node cur = head;
        while (cur.next != null) {
            if (cur.val > cur.next.val)
                return false;
            cur = cur.next;
        } // while
        return true;
    } // else
} // sorted

public void insert(int val) {
    Node n = new Node(val);
    if (!sorted()) {
        Out.println("ERROR: list not sorted prior to insert");
        return;
    } // if
    Node pred = null;
    Node succ = head;
    while (succ != null && n.val > succ.val) {
        pred = succ;
        succ = succ.next;
    } // while
    if (pred == null) // insert in front of head
        head = n;
    else
        pred.next = n;
    n.next = succ;
} // insert

public String toString() {
    StringBuffer sb = new StringBuffer("[");
    Node cur = head;
    while (cur != null) {
        sb.append(cur.val);
        cur = cur.next;
        if (cur != null)
            sb.append(", ");
    }
    sb.append("]");
    ; // while
    return sb.toString();
} // toString

} // List

```

Testspezifikation:

- Methode prepend:
 - Mit leerer Liste
 - Mit nicht-leerer Liste
- Methode append:

- Mit leerer Liste
- Mit Liste mit einem Element
- Mit Liste mit vielen Elementen
- Methode sorted:
 - Mit leerer Liste => true
 - Mit Liste mit einem Element => true
 - Mit sortierter Liste mit zwei Elementen => true
 - Mit unsortierte Liste mit zwei Elementen => false
 - Mit sortierter Liste mit mehreren Elementen => true
 - Mit unsortierte Liste mit mehreren Elementen => false
- Methode insert:
 - Einfügen in leere Liste
 - Einfügen in Liste mit 1 Element am Anfang
 - Einfügen in Liste mit 1 Element am Ende
 - Einfügen in Liste mit mehreren Elementen am Anfang
 - Einfügen in Liste mit mehreren Elementen am Ende
 - Einfügen in Liste mit mehreren Elementen in der Mitte
- Methode toString
 - Mit leerer Liste
 - Mit Liste mit einem Element
 - Mit Liste mit vielen Elementen

Testprogramm:

```
public class ListTest {

    public static void main(String[] args) {

        List l;

        // Methode toString
        Out.println();
        Out.println("----- Test Methode toString -----");
        // Mit leerer Liste
        l = new List();
        Out.println("Leere Liste []: " + l.toString());
        // Mit Liste mit einem Element
        l.prepend(3);
        Out.println("Leere Liste [3]: " + l.toString());
        // Mit Liste mit mehreren Elementen
        l.prepend(2);
        Out.println("Leere Liste [2, 3]: " + l.toString());
        l.prepend(1);
        Out.println("Leere Liste [2, 3]: " + l.toString());
        l.prepend(0);
        Out.println("Leere Liste [2, 3]: " + l.toString());

        // Methode prepend:
        Out.println();
        Out.println("----- Test Methode prepend -----");
        // Mit leerer Liste
        l.prepend(1);
        Out.println("Liste [1]: " + l.toString());
        // Mit nicht-leerer Liste
        l.prepend(2);
        Out.println("Liste [2, 1]: " + l.toString());
        l.prepend(3);
        Out.println("Liste [3, 2, 1]: " + l.toString());

        // Methode append:
        Out.println();
        Out.println("----- Test Methode append -----");
        l = new List();
        // Mit leerer Liste
        l.append(1);
```

```
Out.println("Liste [1]: " + l.toString());
// Mit nicht-leerer Liste
l.append(2);
Out.println("Liste [1, 2]: " + l.toString());
l.append(3);
Out.println("Liste [1, 2, 3]: " + l.toString());
l.append(4);
l.append(5);
l.append(6);
Out.println("Liste [1, 2, 3, 4, 5, 6]: " + l.toString());

// Methode sorted
Out.println();
Out.println("----- Test Methode sorted -----");
l = new List();
// Mit leerer Liste => true
Out.println("Liste " + l.toString() + " sorted = " + l.sorted());
// Mit Liste mit einem Element => true
l.append(1);
Out.println("Liste " + l.toString() + " sorted = " + l.sorted());
// Mit sortierter Liste mit zwei Elementen => true
l.append(2);
Out.println("Liste " + l.toString() + " sorted = " + l.sorted());
// Mit sortierter Liste mit mehreren Elementen => true
l.append(3);
Out.println("Liste " + l.toString() + " sorted = " + l.sorted());
l.append(3);
Out.println("Liste " + l.toString() + " sorted = " + l.sorted());
l.insert(2);
Out.println("Liste " + l.toString() + " sorted = " + l.sorted());
// Mit unsortierte Liste mit zwei Elementen => false
l = new List();
l.prepend(1);
l.prepend(2);
Out.println("Liste " + l.toString() + " sorted = " + l.sorted());
// Mit unsortierte Liste mit mehreren Elementen => false
l = new List();
l.prepend(1);
l.prepend(2);
l.prepend(3);
Out.println("Liste " + l.toString() + " sorted = " + l.sorted());
l.prepend(3);
Out.println("Liste " + l.toString() + " sorted = " + l.sorted());
l = new List();
l.append(1);
l.prepend(2);
l.prepend(1);
Out.println("Liste " + l.toString() + " sorted = " + l.sorted());

// Methode insert
Out.println();
Out.println("----- Test Methode insert -----");
l = new List();
// Einfügen in leere Liste
l.insert(3);
Out.println("Liste [3]: " + l.toString());
// Einfügen in Liste mit 1 Element am Anfang
l.insert(1);
Out.println("Liste [1, 3]: " + l.toString());
// Einfügen in Liste mit 1 Element am Ende
l.insert(5);
Out.println("Liste [1, 3, 5]: " + l.toString());
// Einfügen in Liste mit mehreren Elementen am Anfang
l.insert(0);
Out.println("Liste [0, 1, 3, 5]: " + l.toString());
// Einfügen in Liste mit mehreren Elementen in der Mitte
l.insert(2);
Out.println("Liste [0, 1, 2, 3, 5]: " + l.toString());
l.insert(4);
Out.println("Liste [0, 1, 2, 3, 4, 5]: " + l.toString());
// Einfügen in Liste mit mehreren Elementen am Ende
l.insert(6);
Out.println("Liste [0, 1, 2, 3, 4, 5, 6]: " + l.toString());
```

```
    } // main  
}
```

Testergebnis:

□

```
----- Test Methode toString -----  
Leere Liste []: []  
Leere Liste [3]: [3]  
Leere Liste [2, 3]: [2, 3]  
Leere Liste [2, 3]: [1, 2, 3]  
Leere Liste [2, 3]: [0, 1, 2, 3]  
  
----- Test Methode prepend -----  
Liste [1]: [1, 0, 1, 2, 3]  
Liste [2, 1]: [2, 1, 0, 1, 2, 3]  
Liste [3, 2, 1]: [3, 2, 1, 0, 1, 2, 3]  
  
----- Test Methode append -----  
Liste [1]: [1]  
Liste [1, 2]: [1, 2]  
Liste [1, 2, 3]: [1, 2, 3]  
Liste [1, 2, 3, 4, 5, 6]: [1, 2, 3, 4, 5, 6]  
  
----- Test Methode sorted -----  
Liste [] sorted = true  
Liste [1] sorted = true  
Liste [1, 2] sorted = true  
Liste [1, 2, 3] sorted = true  
Liste [1, 2, 3, 3] sorted = true  
Liste [1, 2, 2, 3, 3] sorted = true  
Liste [2, 1] sorted = false  
Liste [3, 2, 1] sorted = false  
Liste [3, 3, 2, 1] sorted = false  
Liste [1, 2, 1] sorted = false  
  
----- Test Methode insert -----  
Liste [3]: [3]  
Liste [1, 3]: [1, 3]  
Liste [1, 3, 5]: [1, 3, 5]  
Liste [0, 1, 3, 5]: [0, 1, 3, 5]  
Liste [0, 1, 2, 3, 5]: [0, 1, 2, 3, 5]  
Liste [0, 1, 2, 3, 4, 5]: [0, 1, 2, 3, 4, 5]  
Liste [0, 1, 2, 3, 4, 5, 6]: [0, 1, 2, 3, 4, 5, 6]
```