Übung 6 Judith Koch, Bianca Ploch, Tu Le-Thanh

Aufgabe 1

Folgende kontextfreie Grammatik aus der Vorlesung beschreibt die Ausdrücke Werte,Variablen, Grundrechenarten und Klammerungen:

<expression> → [<expression> +|-]<term>
<term> → [<term> *|/] <factor>
<factor> → <number>|<variable>|(<expression>)
<number> → 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<variable> → a | b | c | x | y |z

Aufgabe 2

Als Basis für die Knoten wurde zunächst eine Klasse TreeNode erstellt, welche die grundlegenden Referenzen besitzt. Diese werden je nach Konstruktor unterschiedlich initiiert.

```
public TreeNode(){
        element = null;
        left = null;
        right = null;
}

public TreeNode(T element){
        this.element = element;
        this.left = null;
        this.right = null;
}

public TreeNode(T element, TreeNode<T> left, TreeNode<T> right){
        this.element = element;
        this.left = left;
        this.right = right;
}

public TreeNode(T element, TerminalNode<T> left, TerminalNode<T> right){
        this.element = element;
        this.left = left;
        this.right = right;
}
```

Darüber hinaus besitzt diese Klasse Getter und Setter für die jeweiligen Elemente und eine Methode zur Ermittlung, ob ein Knoten leer ist.

Die Klasse für die Terminale erbt dann von TreeNode und erweitert diesen mit einer Referenz auf einen Parent. Derzeit wird der Parent noch nicht verwendet. Für diesen Fall gibt es einen Konstruktor, dem nur das Element übergeben wird.

```
public TerminalNode(T element, TreeNode<T> parent){
    this.element = element;
    this.parent = parent;
}
```

Nicht-Terminale werden durch die Klasse TreeNode realisiert.

Aufgabe 3

Für das Einlesen des Strings wird die Klasse MyScanner mit einem Parameter initialisiert, welcher für die weitere Verarbeitung abgespeichert wird. Hierfür wandelt zunächst die scan-Methode den vorgespeicherten String in einen Character-Array um. Dieser wird mit der

Intention, benötigte Klammern hinzuzufügen, in eine ArrayList umgeschrieben.

```
private void makeArrayList(char[] charArray){
    for (int i = 0; i < charArray.length; i++){
        charArrayList.add((Character)charArray[i]);
    }
}
```

Die ArrayList wird aufgrund der bereits implementierten add-Methode bevorzugt benutzt. Sie vereinfacht es, mittels des Indexes an bestimmten Stellen der Liste eine Klammer einzufügen. Die insertBrackets-Funktion iteriert über die ArrayList und prüft währenddessen, ob ein Multiplikationszeichen auftaucht. In solch einem Fall wird vor den ersten Faktor eine öffnende Klammer gesetzt. Da die Indices sich verschieben wird für die geschlossene Klammer nach dem zweiten Faktor eine zusätzliche Addition für die add-Methode benötigt. In Folge dessen wird der Zähler dann manuell um einen Wert inkrementiert.

```
private void insertBrackets(ArrayList<Character> charArrayList){
    for (int i = 0; i < charArrayList.size(); i++){
        if(charArrayList.get(i) == '*'){
            charArrayList.add(i-1, '(');
            charArrayList.add(i+3, ')');
            i++;
        }
    }
}
```

Am Ende wird dann ein geeignet großer Character-Array erstellt und mit der toArray-Funktion neu zugewiesen.

```
public void scan(){
    charArray = stringToScan.toCharArray();
    makeArrayList(charArray);
    if (!checkForBrackets()){
        insertBrackets(charArrayList);
    }

    charArrayToUse = new Character[charArrayList.size()];

    charArrayList.toArray(charArrayToUse);
}
```

Aufgabe 4

Der Parser verarbeitet nun das entstandene Array aus der Scanner-Klasse. Hierzu weist zunächst die parse-Funktion der bereits initialisierten TreeNode “left” den zurückgegebenen Wert aus der rekursiven Funktion term(). Jene Methode setzt den Zeiger “index” solange weiter bis das zu verarbeitende Zeichen keine Klammer mehr ist.

```
if(isBracket(charArray[index])){
    next();
    term();
} else {
```

Entspricht es dem Fall wird term() rekursiv aufgerufen und führt dementsprechend die else-Anweisung aus. In dieser wird zunächst erst mal festgestellt, ob es ein darauf folgendes Symbol gibt und ein Operator ist.

```
if(index + 1 < charArray.length && operation(index+1)){
```

Ist es nicht der Fall wird ein TerminalKnoten mit dem Character an der Stelle an der der Zeiger momentan steht zugeordnet.

```
else {
  terminal = new TerminalNode<Character>(charArray[index]);
```

Andernfalls wird geprüft, ob das übernächste Symbol eine Klammer ist. In dieser Situation wird der Zeiger um 2 erhöht und dementsprechend ein TreeNode rückwärts befüllt, welcher dann dem Terminal zugewiesen wird.

```
else {
  skip(2);
  terminal = new TreeNode<Character>((charArray[index-1]), new TerminalNode<Character>(charArray[index-2]), term());
```

Ist es keine Klammer springt die Methode term() in die letzte If-Anweisung. Wenn nach dem Wert der Operation eine Klammer oder Multiplikation folgt, ist es Teil einer Grundoperation und wird auch so in auffolgener Weise dem Terminal zugewiesen. Der Zeiger wird dann auf die Klammer bzw. Multiplikation gesetzt.

```
new TreeNode<Character>(charArray[index+1], new TerminalNode<Character>(charArray[index]), new TerminalNode<Character>(charArray[index+2])
```

Sollte dem so nicht sein wird ein TreeNode erstellt, der den Operator als Element und den ersten Wert der Rechnung als linken Knoten besitzt.

```
if(index+3 < charArray.length && !multiplication(index+3) && !isBracket(charArray[index+3])){
    terminal = new TreeNode<Character>(charArray[index+1], new TerminalNode<Character>(charArray[index]), ne
    skip(5);
```

Der rechte Knoten besteht aus dem drittnächsten Knoten als Element, dem zuvor als weitergeleiteten linken Knoten und dem danach als weiterführenden rechten.

```
new TreeNode<Character>(charArray[index+3], new TerminalNode<Character>(charArray[index+2]), new TerminalNode<Character>(charArray[index+4]
```

Danach wird der Zeiger um 5 weitere Stellen gesetzt bzw. auf dem Symbol nach der schon verarbeitenden Character-Folge.
Schlussendlich wird der terminal natürlich zurückgegeben.

Die Parse-Funktion überspringt dann anfänglich sämtliche Klammern bis diese dann auf einen Operator trifft. Dieser wird dann zwischengespeichert. Auf dem darauffolgenden Character wird

dann die term-Methode erneut aufgerufen und mit den anderen temporär gespeicherten Elementen “left” und “symbol” in ein ParentNode als “right” dann zurückgegeben. Wenn dem nicht so ist, wird dem ParentNode left zugewiesen.

```
public TreeNode<Character> parse(){
    TreeNode<Character> left = term();
    if(isBracket(charArray[index])){
        next();
    }

    if(index < charArray.length && operation(index)){
        Character symbol = charArray[index];
        next();
        TreeNode<Character> right = term();
        return parentNode = new TreeNode<Character>(symbol, left, right);
    } else {
        return parentNode = left;
    }

}
```

Aufgabe 5

Die Ausgabe in den jeweiligen Formen erfolgt ähnlich. Lediglich die Reihenfolge der auszugebenen Nodes variiert.
In Pre-Order wird zunächst das Element ausgegeben und dann die weiterführenden linken oder rechten Knoten, wenn jene existieren.

```
public void printTreePreOrder(TreeNode<Character> parentNode){
    System.out.print(parentNode.getElement());

    if(parentNode.getLeft() != null){
        printTreePreOrder(parentNode.getLeft());
    }

    if(parentNode.getRight() != null){
        printTreePreOrder(parentNode.getRight());
    }
```

In In-Order zuerst der linke Knoten, dann das Element und schlussendlich der Rechte.

```
public void printTreeInOrder(TreeNode<Character> parentNode){
    if(parentNode.getLeft() != null){
        printTreeInOrder(parentNode.getLeft());
    }

    System.out.print(parentNode.getElement());

    if(parentNode.getRight() != null){
        printTreeInOrder(parentNode.getRight());
    }
}
```

Und in Post-Order erst die Child-Knoten und dann das Element.

```
public void printTreePostOrder(TreeNode<Character> parentNode){
    if(parentNode.getLeft() != null){
        printTreePostOrder(parentNode.getLeft());
    }

    if(parentNode.getRight() != null){
        printTreePostOrder(parentNode.getRight());
    }

    System.out.print(parentNode.getElement());
}
```

In der Main-Klasse werden dann die Tests für die unterschiedlichen Fälle eines möglichen Terms ausprobiert.
Die Fälle sind:

- 1*(3+5)-3*2
- ((x+y)*(3+(7-2)))
- x+y*3+7-2
- x+y*(3+7-2)
- 2*(3+7)
- (3+7)*2

Das Ergebnis wird in Pre-, In- und Post-Order ausgegeben.

```
-*1+35*32
1*3+5-3*2
135+*32*-
next:
*+xy+3-72
x+y*3+7-2
xy+372-+*
next:
++x*y3-72
x+y*3+7-2
xy3*+72-+
next:
*+xy+3-72
x+y*3+7-2
xy+372-+*
next:
*2+37
2*3+7
237+*
next:
*+372
3+7*2
37+2*
```