Übung10: Schrittweise Verfeinerung

Abgabetermin: 07.06.19, 15:30 Uhr

Name: _____ Matrikelnummer: _____

Aufgabe	**schriftlich abzugeben**	**elektronisch abzugeben**	**gelöst**
Aufgabe26	Spezifikation nach schrittweiser Verfeinerung, Java Programm, Testausgaben	Java Programm	☐
Aufgabe27	Spezifikation nach schrittweiser Verfeinerung, Java Programm, Testausgaben	Java Programm	☐

Achtung!

Bitte auf diesem Deckblatt:

- **Name** und **Matrikelnummer** ausfüllen,
- gelöste Aufgaben **ankreuzen**

und dann der schriftlichen Abgabe als erste Seite anheften.

Aufgabe26: Textsuche (12 Punkte)

Gesucht ist ein Programm, mit dem man in Texten nach Zeichenketten suchen kann. Dabei soll der Text Zeichen für Zeichen durchlaufen werden. An jeder Position soll der Suchstring neu angesetzt werden, um zu Zählen, wie viele Zeichen im Text ab dieser Position mit den Zeichen im Suchstring übereinstimmen. Das Programm soll jene Position ermitteln, an der die meisten Zeichen des Suchstrings mit den Zeichen des Textes übereinstimmen. Existieren mehrere solche Positionen, soll die erste ausgegeben werden.

Bei einem Treffer soll die Position als Zeilen- und Spaltennummer im Text, sowie die Anzahl der übereinstimmenden Zeichen ausgegeben werden. Weiters soll Groß- und Kleinschreibung bei der Suche irrelevant sein und der zu durchsuchende Text aus einer Datei eingelesen werden. Die Datei `Faust.txt` wird im Moodle bereitgestellt.

Beispieldialog:

```
Name der Textdatei: Faust.txt
Suchen nach: Gretchen
Gefunden: Gretchen (Zeile 4301, Spalte 30, 8/8 Zeichen stimmen überein)
Suchen nach: mephisto
Gefunden: Mephisto (Zeile 609, Spalte 50, 8/8 Zeichen stimmen überein)
Suchen nach: Master
Gefunden: matter (Zeile 109, Spalte 21, 5/6 Zeichen stimmen überein)
Suchen nach: Der Teufel Mephisto
Gefunden: der Teufel kann's n (Zeile 3591, Spalte 8, 12/19 Zeichen stimmen überein)
Suchen nach: .
Ende
```

Hinweis:

Zeilen- und Spaltennummer beginnen mit 1. Das Zeichen `'\n'` markiert eine neue Zeile im Text. Wird so ein Zeichen eingelesen, muss die Zeile erhöht und die Spalte wieder auf 1 zurückgesetzt werden. Ansonsten erfordert das Zeichen keine Sonderbehandlung.

Achtung: Starten Sie nicht direkt mit der Implementierung des Programms. Überlegen Sie zuerst, wie das Problem in kleinere Probleme zerlegt werden kann, die einfach zu lösen sind. Führen Sie eine *schrittweise Verfeinerung* durch, ähnlich zur Vorlesung. Beschreiben Sie, welche Teilaufgaben zu erledigen sind, welche Methoden oder Datenstrukturen dafür benötigt werden und wie die Schnittstellen dazu aussehen. Halten Sie dies schriftlich in Ihrer Ausarbeitung fest und implementieren Sie erst danach das Programm.

Natürlich kann die Spezifikation im Nachhinein korrigiert werden, wenn bei der Implementierung ein unbedachter Fall aufkommt. Versuchen Sie trotzdem, sich über diese Fälle im Vorhinein Gedanken zu machen und diese vorher festzuhalten.

Abzugeben:

- Spezifikation nach schrittweiser Verfeinerung
- Java Programm
- Testausgaben

Aufgabe27: Rechnen mit arithmetischen Ausdrücken (12 Punkte)

Gesucht ist ein Programm zum Rechnen mit arithmetischen Ausdrücken (Expressions). Das Programm soll das Eingeben neuer Expressions in Prefixnotation erlauben. Es soll auch möglich sein, neue Expressions zu bilden, indem zwei definierte Expressions mit einem Operator kombiniert werden. Beim Erzeugen einer neuen Expression wird ihr eine Nummer als Referenzwert zugewiesen und es wird das Ergebnis der Expression evaluiert.

Beispieldialog:

```
Folgende Befehle stehen zur Verfügung:
R   Einlesen einer neuen Expression
+   Summe zweier Expressions
-   Differenz zweier Expressions
*   Produkt zweier Expressions
/   Division zweier Expressions
.   Ende
Kommando (R|+|-|*|/|.): R
Expresssion: + 1 2
[0] (1 + 2) = 3
Kommando (R|+|-|*|/|.): R
Expresssion: - 3 4
[1] (3 - 4) = -1
Kommando (R|+|-|*|/|.): -
1. Operand: 0
2. Operand: 1
[2] ((1 + 2) - (3 - 4)) = 4
Kommando (R|+|-|*|/|.): *
1. Operand: 2
2. Operand: 1
[3] (((1 + 2) - (3 - 4)) * (3 - 4)) = -4
Kommando (R|+|-|*|/|.): .
```

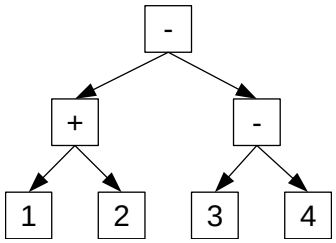

Abbildung 1: Expression `((1 + 2) - (3 - 4))` als Baumstruktur

Hinweis:

In der Prefixnotation steht der Operator einer Expression am Anfang. Zum Beispiel wird die Expression `((2 * 3) + (6 / 7))` in Prefixnotation als `+ * 2 3 / 6 7` eingelesen. Konstante Expressions sollen einen ganzzahligen Wert haben, der mit `In.readInt()` eingelesen werden kann.

Achtung: Führen Sie auch hier eine *schrittweise Verfeinerung* vor der Implementierung durch. Überlegen Sie sich eine geeignete Klassenhierarchie, mit der sich Bäume für verschachtelte Expressions aufbauen lassen, wie es in Abbildung 1 für eine konkrete Expression zu sehen ist. Zum Beispiel könnte die Basisklasse `Expression` Ableitungen für die verschiedenen Ausdrucksarten (`Constant`, `Addition`, `Subtraction`, usw.) besitzen.

Überlegen Sie auch, wie Sie das Berechnen der Ergebniswerte, sowie das Ausgeben und Einlesen der Expressions elegant und einfach umsetzen können. Hierfür eignen sich rekursive Methoden sehr gut.

Abzugeben:

- Spezifikation nach schrittweiser Verfeinerung
- Java Programm
- Testausgaben