Modul 411 – Auswertung arithmetischer Ausdrücke

Ausgangslage

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```

Wenn Sie 4 mit 5 multiplizieren, 3 und 2 addieren, die Ergebnisse multiplizieren und dann 1 hinzufügen, erhalten Sie den Wert 101. Doch wie geht das Java-System bei dieser Berechnung vor? Ohne allzu sehr auf die Details des Java-Systems eingehen zu müssen, können wir die wesentlichen Konzepte erarbeiten, indem wir einfach ein Java-Programm schreiben, das einen String als Eingabe entgegennimmt (den Ausdruck) und als Ausgabe die Zahl erzeugt, die durch den Ausdruck repräsentiert wird. Der Einfachheit halber beginnen wir mit der folgenden expliziten rekursiven Definition: Ein *arithmetischer Ausdruck* ist entweder eine Zahl oder eine linke Klammer gefolgt von einem arithmetischen Ausdruck gefolgt von einem Operator gefolgt von einem weiteren arithmetischen Ausdruck gefolgt von einer rechten Klammer. Um die Aufgabe nicht unnötig zu komplizieren, haben wir hier die Definition für *vollständig geklammerte* arithmetische Ausdrücke gewählt, die klar angibt, welche Operatoren auf welche Operanden anzuwenden sind – im Gegensatz zu der Ihnen sicherlich etwas vertrauteren Schreibweise der Form $1 + 2 * 3$, in der Prioritätsregeln die Klammern ersetzen. Zwar könnten wir mithilfe der gleichen grundlegenden Mechanismen, die wir in der Folge betrachten werden, auch Prioritätsregeln verarbeiten, doch wir wollen das Beispiel bewusst einfach halten. Konkret unterstützen wir die bekannten Binäroperatoren `*`, `+`, `-` und `/` sowie den Quadratwurzeloperator `sqrt`, der nur ein Argument übernimmt. Wir könnten problemlos weitere Operatoren oder Arten von Operatoren zulassen, um eine größere Klasse von bekannten mathematischen Ausdrücken zu unterstützen, einschließlich trigonometrischer, exponentieller und logarithmischer Funktionen. Unser Schwerpunkt liegt aber auf der richtigen Interpretation des Strings aus Klammern, Operatoren und Zahlen, damit wir die arithmetischen Lowlevel-Operationen, die auf jedem Computer zur Verfügung stehen, in der rechten Reihenfolge ausführen.

Aufgabe

Wie können wir also einen arithmetischen Ausdruck – einen String von Zeichen – in den Wert umwandeln, den er repräsentiert? Ein bemerkenswert einfacher Algorithmus, der von Edsgar Dijkstra in den 1960ern entwickelt wurde, verwendet für diese Aufgabe zwei Stapel (einen für Operanden und einen für Operatoren). Ein Ausdruck besteht aus Klammern, Operatoren und Operanden (Zahlen). Wenn wir von links nach rechts vorgehen und uns die Entitäten einzeln vornehmen, setzen wir die Stapel gemäß der vier möglichen Fälle wie folgt ein:

- Wir legen *Operanden* auf dem Operandenstapel ab.
- Wir legen die *Operatoren* auf dem Operatorstapel ab.
- Wir ignorieren die linken Klammern.
- Wenn wir auf eine *rechte* Klammer treffen, löschen wir einen Operator und die dafür erforderliche Anzahl an Operanden und legen auf dem Operandenstapel das Ergebnis ab, nachdem wir den Operator auf die Operanden angewendet haben.

Nachdem die letzte rechte Klammer verarbeitet wurde, liegt nur noch ein Wert auf dem Stapel, und zwar der Wert des Ausdrucks. Diese Methode mag auf dem ersten Blick mysteriös erscheinen, aber Sie können sich leicht davon zu überzeugen, dass sie den korrekten Wert zurückliefert: Jedes Mal, wenn der Algorithmus auf einen Teilausdruck trifft, der aus zwei Operanden besteht, die durch einen Operator getrennt und in Klammern gefasst sind, wird das Ergebnis der Operation auf diese Operanden auf dem Operandenstapel abgelegt. Das Ergebnis ist das gleiche, wie wenn in der Eingabe anstelle des Teilausdrucks dieser Wert gestanden hätte, sodass wir den Teilausdruck gedanklich durch den Wert ersetzen können, um einen Ausdruck zu erhalten, der das gleiche Ergebnis liefert. Wir können diese Ersetzung wiederholt anwenden, bis wir nur noch einen Wert übrig haben. Folglich berechnet der Algorithmus für die folgenden Ausdrücke den immer gleichen Wert:

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
( 1 + ( 5 * ( 4 * 5 ) ) )
( 1 + ( 5 * 20 ) )
( 1 + 100 )
101
```

Erstellen Sie die Klasse Evaluate mit der main()-Methode.

```
% java Evaluate
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
101.0

% java Evaluate
( ( 1 + sqrt ( 5.0 ) ) / 2.0 )
1.618033988749895
```

Erwartete Ergebnisse:

- Struktogramme
- Saubere Implementierung des Programms inklusive inline Dokumentation
- Dokumentation des Projektablaufs inklusive Verantwortlichkeiten
- Kurze Präsentation des Programms und der Struktogramme (max. 10 Minuten)

Hilfestellung:

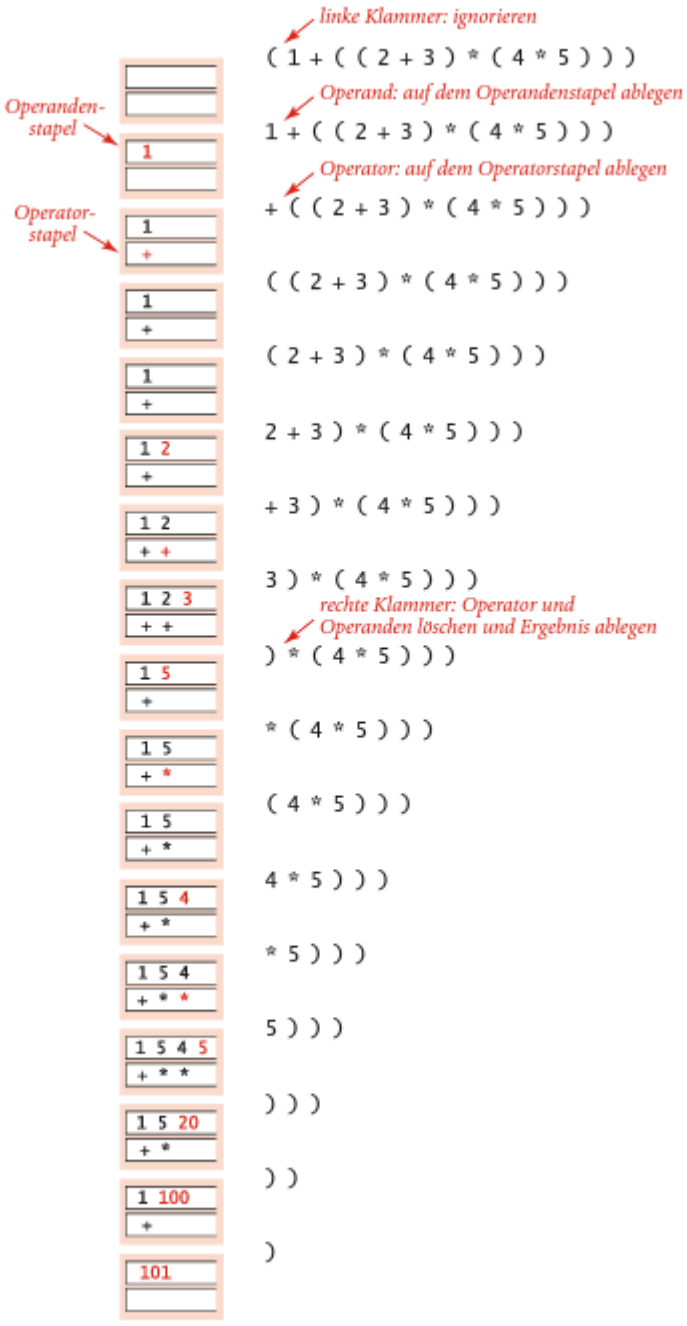

Abbildung 1.33: Ablaufprotokoll von Dijkstras Algorithmus zur Auswertung arithmetischer Ausdrücke mittels