Aufgabe 08 – rekursive Auswertung eines Ausdrucks

Es soll eine rekursive Methode `evaluate(expression : String ) : int`
zur Berechnung eins mathematischen Ausdrucks entwickelt werden.

Beispiel: der Methode wird der Text „12+4-3-2“ übergeben und sie gibt den Wert 11 zurück.

Ist der Ausdruck unvollständig – z.B.: „12+4-“(es fehlt der letzte Operand) – oder enthält er ungültige Zeichen (die nicht an dieser Stelle erwartet werden), dann soll eine Exception geworfen werden.

Level 1:

Als Vereinfachungen soll ein Ausdruck zunächst nur aus Ziffern und den Operatoren „+“ und „-“ bestehen dürfen.
D.h. keine Klammern, keine Operatorprioritäten („Punkt- vor Strichrechnung“).

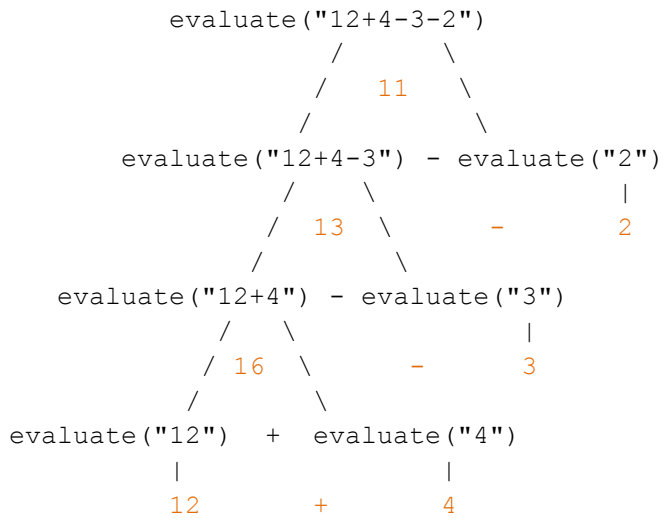
Vorgehensweise:

Man sucht den letzten (binären) Operator und spaltet den Ausdruck an diesem in zwei Teile auf. Jeder Teil wird wiederum der `evaluate(…)` Methode übergeben…
Die Rekursion endet, wenn der übergebene Text nur mehr aus Ziffern besteht.
Den entsprechenden Wert kann man mittels `Integer.parseInt(…)` bestimmen.

Achtung: „+“ und „-“ können auch Vorzeichen (unäre Operatoren) sein! Beginnt ein Ausdruck mit „-“ dann ist der Rückgabewert : - `evaluate`(Text ohne Vorzeichen),
es wird also nicht aufgespalten.

```
int evaluate( String expression )

  wenn:
    expression enthält nur noch Ziffen
               (kann fehlerfrei in int umgewandelt werden)
      return Integer.parseInt(expression)

    expression beginnt mit -
       return -evaluate(expression ohne erstes Zeichen)

  andernfalls aufspalten:
    operator = letztes gefundene + oder - (nicht als erstes Zeichen!)
    expression1 = Text vor dem ersten Operator
    expression2 = Text nach dem ersten Operator

    Rückgabewert abhängig vom Operator berechnen
      + : return evaluate(expression1) + evaluate(expression2)
      - : return evaluate(expression1) - evaluate(expression2)
```

Beispiel: Abfolge der rekursiven Aufrufe beim Ausdruck “12+4-3-2”:



Level 2:

Wir steigern die Schwierigkeit und erlauben auch die Operatoren „*“ und „/“.

Unterschied zu vorher: „Punkt vor Strichrechnung“

Es wird zuerst nach „+“ oder „-“ Operatoren gesucht und an dieser Stelle aufgespalten.
Gibt es keinen solchen Operator mehr, dann wird nach „*“ oder „/“ gesucht…

Level 3:

Jetzt sind auch Klammern, also die Zeichen „(“ und „)“ im Ausdruck erlaubt.

Vorgehensweise:

Suche einen (binären) Operator außerhalb von Klammern und spalte den Ausdruck an dieser Stelle auf. Dabei muss natürlich weiterhin gelten: „Punkt vor Strichrechnung“.

Gibt es keinen solchen Operator außerhalb von Klammern und der Ausdruck beginnt mit „(“, dann entferne die äußeren Klammern und rufe *evaluate* mit dem Text innerhalb dieser Klammern auf… (d.h. Rückgabewert ist: `evaluate(`Text ohne äußere Klammern))

Level 4:

Es sind nicht nur ganze Zahlen erlaubt sondern auch Gleitkommazahlen.
D.h. aus `Integer.parseInt(…)` wird `Double.parseDouble(…)`

Level 5:

Es werden unäre Operatoren erlaubt z.B.: „^2“ oder „^3“ also Ausdruck „hoch 2“ oder „hoch 3“. Oder Berechnen der Quadratwurzel…