Funktionale und objektorientierte Programmierkonzepte

Präsenz-Sprechstunde B

Simon Hock, Nhan Huynh, Daniel Mangold

Hinweis

TECHNISCHE UNIVERSITÄT DARMSTADT

12.01.2022 Präsenzsprechstunde B im Raum in Raum S103/**123**!

Überblick

TECHNISCHE UNIVERSITÄT DARMSTADT

Rekursion [25min]
- Vorgehensweise bei rekursiven Aufgaben
- Rekursive Auswertung von Arithmetischen Ausdrücken in Präfix-Notation

Arbeitsphase [75min]

Rekursion

TECHNISCHE UNIVERSITÄT DARMSTADT

„Eine Rekursion ist eine Funktion, die sich selbst aufruft. Dabei wird versucht das Problem auf eine einfachere Variante des **gleichen** Problems rekursiv zu reduzieren, welches dann gelöst wird.“

Vorgehensweise bei rekursiven Aufgaben

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Identifiziere, wie das Problem auf eine einfachere Variante des Problems zerlegt werden kann und direkt gelöst werden kann
   - Teillösungen der Gesamtlösung
2. Das kleinste Problem ist ein Rekursionsanker
3. Überlege, wie die kleinen Problemen kombiniert werden können, um die ursprüngliche Problemstellung zu lösen
4. Kombiniere die Teillösungen zu einer Gesamtlösung

Rekursive Auswertung von Arithmetischen Ausdrücken in Präfix-Notation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Blatt 06 H4
- Prfäix-Notation Beispiele:
  - $+2\ 3 = 5$
  - $*-123 = *(-12)3 = -3$
  - $+4-51 = +4(-51) = 8$

Notation	Addition zweier Zahlen
Präfix	$+\ x\ y$
Infix	$x + y$
Postfix	$x\ y\ +$

Tabelle: Überblick von Notationen

H4.1: Vorbereitung: Klasse `ReturnData`

TECHNISCHE UNIVERSITÄT DARMSTADT

Schreiben Sie zunächst eine eine `public`-Klasse `ReturnData`, welche zwei `public`-Objektattribute vom Typ `int` hat: `result` und `nextIndex`. Diese Klasse muss einen parameterlosen `public`-Konstruktor besitzen. Sie dürfen aber gerne auch andere Konstruktoren implementieren, solange weiterhin ein parameterloser `public`-Konstruktor zur Verfügung steht.

```java
public class ReturnData {

    public int result;
    public int nextIndex;

    public ReturnData() {}

    // Optional
    public ReturnData(int result, int nextIndex) {
        this.result = result;
        this.nextIndex = nextIndex;
    }
}
```

H4.2: Methoden `evaluate` und `evaluateRecursively`

TECHNISCHE UNIVERSITÄT DARMSTADT

Erweitern Sie Klasse `StrangeThings` aus 2 um eine `public`-Klassenmethode `evaluate`, die den Wert eines arithmetischen Ausdrucks in Präfix-Notation (das heißt: zuerst Operator, dann Operanden) berechnet. Es geht also um eine Notation wie in Racket – aber im Gegensatz zu Racket haben wir keine Klammern. Außerdem beschränken wir uns bei den Operanden auf einstellige Zahlen (also $0 \ldots 9$)) und bei dem Operator um die Subtraktion.

Information: „Ein arithmetischer Ausdruck besteht aus einer Folge von arithmetischen Operanden, die durch (arithmetische) Operatoren und Klammern voneinander getrennt (oder miteinander verknüpft) sind.“

Beispiel für arithmetische Ausdrücke

TECHNISCHE UNIVERSITÄT DARMSTADT

$$1 \tag{1}$$

$$-12 \tag{2}$$

$$--123 \tag{3}$$

$$--12-34 \tag{4}$$

$$1 \tag{1}$$

$$\underbrace{-12}_{1-2=-1} \tag{2}$$

$$-\underbrace{-12}_{1-2=-1}3 \tag{3}$$

$$\underbrace{-\underbrace{-12}_{1-2=-1}\underbrace{-34}_{3-4=-1}}_{-1-(-1)=0} \tag{4}$$

Konkret hat Methode `evaluate` einen Parameter vom formalen Typ „ Array vom primitiven Typ `char` “ und Rückgabetyp `int` . Die Methode `evaluate` darf ohne Überprüfung davon ausgehen, dass das Array mindestens die Länge 1 hat und jede Komponente des Arrays entweder eine Ziffer `'0'`, ..., `'9'` oder das Zeichen `'-'` enthält und der mit der `char`-Kette repräsentierte Präfix-Ausdruck ein korrekt gebildeteter Präfix-Ausdruck ist, in dem jede Subtraktion genau zwei Operanden hat.

Konkret hat Methode `evaluate` einen Parameter vom formalen Typ „ Array vom primitiven Typ `char` “ und Rückgabetyp `int` . Die Methode `evaluate` darf ohne Überprüfung davon ausgehen, dass das Array mindestens die Länge 1 hat und jede Komponente des Arrays entweder eine Ziffer `'0'`, ..., `'9'` oder das Zeichen `'-'` enthält und der mit der `char`-Kette repräsentierte Präfix-Ausdruck ein korrekt gebildeteter Präfix-Ausdruck ist, in dem jede Subtraktion genau zwei Operanden hat.

Information: Beim Parameterwert wird zwischen dem aktualen Wert (der Parameterwert, der eingesetzt wird) und dem formalen Wert (der in der Definition des Parameters in der Methode steht) unterschieden.

Neben der Methode `evaluate` soll in Klasse `StrangeThings` eine `private`- Klassenmethode `evaluateRecursively` mit einem ersten Parameter namens `array` vom Typ „ Array vom primitiven Typ `char` “, einem zweiten Parameter namens `startIndex` vom Typ `int` und Rückgabetyp `ReturnData` existierten.

Die Methode `evaluateRecursively` soll aus dem Array `array` den numerischen (Teil-)Ausdruck ab Index `startIndex` auswerten und ein `ReturnData` -Objekt zurückliefern, dessen Objekt-attribut `result` den Zahlenwert des ausgewerteten numerischen (Teil-) Ausdrucks ab Index `startIndex` und Objektattribut `nextIndex` den ersten Index des nächsten numerischen Teilausdrucks enthält.

Die Methode `evaluate` ruft die Methode `evaluateRecursively` auf, wobei `evaluateRecursively` als ersten aktualen Parameter den `ersten aktualen Parameter von` `evaluate` erhält (beides jeweils „ `Array` von `char` “). Als zweiten aktualen Parameter erhält `evaluateRecursively` den Wert `0` . Nach dem Aufruf von `evaluateRecursively` liefert die Methode `evaluate` den Wert des Objektattributs `result` des von `evaluateRecursively` gelieferten Objekts vom Typ `ReturnData` .

Das Verhalten der Methode `evaluateRecursively` ist vom Zeichen am Index `startIndex` in Array `array` abhängig:

Im Fall, dass der `char`-Wert `array[startIndex]` eine Ziffer darstellt, wurde ein atomarer arithmetischer Ausdruck gefunden und die Methode liefert ein `ReturnData`-Objekt zurück, dessen Objektattribut `result` den durch das Zeichen dargestellten Wert (für `'4'` zum Beispiel 4) und `nextIndex` den Wert `startIndex+1` enthält.

Im anderen Fall, also wenn der `char`-Wert gleich `'-'` ist, müssen die nächsten beiden arithmetischen Teil-Ausdrücke `e1` und `e2` ausgewertet und ein geeignetes `ReturnData`-Objekt zurückgegeben werden.

Implementieren Sie die Methoden `evaluate` und `evaluateRecursively` wie beschrieben.

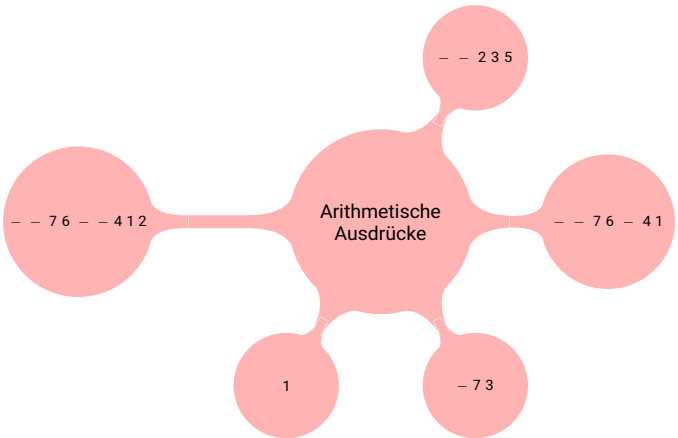
- `array`: Enthält den arithmetischen Ausdruck, bspw. `['-', '1', '2']`
- `startIndex`: Aktualler Index beim Durchlauf durch das Array `array`
- `ReturnData`
  - `result`: Enthält das Ergebnis des bisher berechneten Teilausdrucks
  - `nextIndex`: Gibt den ersten Index des nächsten Teilausdrucks an

```
1 private static ReturnData evaluateRecursively(char[] array, int startIndex) {
2     // TODO
3 }
```

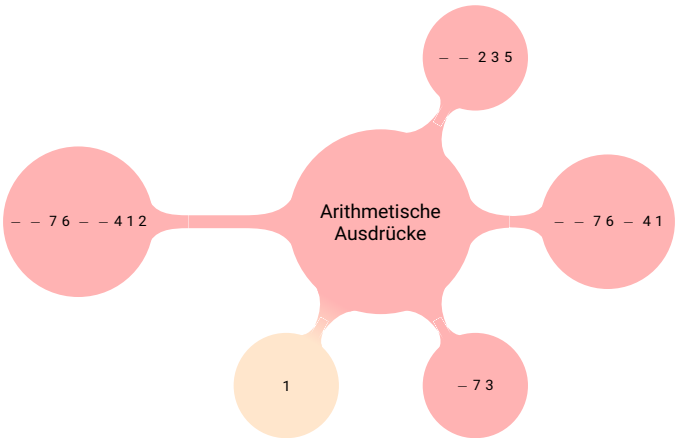
- Hauptrekursion wird in der Hilfsmethode `evaluateRecursively` durchgeführt
- `ReturnData` enthält das Ergebnis der Auswertung

```
1 public static int evaluate(char[] array) {
2     ReturnData data = evaluateRecursively(array, 0);
3     // Alternatively: return evaluateRecursively(array, 0).result;
4     return data.result;
5 }
```

1. Identifiziere, wie das Problem auf eine einfachere Variante des Problems zerlegt werden kann und direkt gelöst werden kann



1. Identifiziere, wie das Problem auf eine einfachere Variante des Problems zerlegt werden kann und direkt gelöst werden kann



2. Das kleinste Problem ist ein Rekursionsanker

Algorithm evaluateRecursively(array, startIndex)

```
1: if array[startIndex] is a digit then
2:     return array[startIndex], startIndex + 1
3: end if
```

```java
private static ReturnData evaluateRecursively(char[] array, int startIndex) {
    char token = array[i];
    if (Character.isDigit(token)) {
        return new ReturnData(
            Character.getNumericValue(token),
            startIndex + 1
        );
    }
    // TODO
}
```

3. Überlege, wie die kleinen Problemen kombiniert werden können, um die ursprüngliche Problemstellung zu lösen

$$\underbrace{\text{1. kleines Problem und 2. kleines Problem}}_{\text{Kombinieren}}$$

- Wann werden die beiden Probleme zusammengefügt?

3. Überlege, wie die kleinen Problemen kombiniert werden können, um die ursprüngliche Problemstellung zu lösen

$$\underbrace{\text{1. kleines Problem und 2. kleines Problem}}_{\text{Kombinieren}}$$

- Wann werden die beiden Probleme zusammengefügt? $\Rightarrow$ Wenn ein Operator gelesen wird

Algorithm evaluateRecursively(array, startIndex)

```
1: if array[startIndex] is a digit then
2:     return array[startIndex], startIndex + 1
3: else
4:     leftOperand ← evaluateRecursively(array, ...)
5:     rightOperand ← evaluateRecursively(array, ...)
6:     return leftOperand.result + rightOperand.result, ...
7: end if
```

- Von welchen Indizes starten die Operanden bzw. was ist der nächste Index nach der Kombination?

Algorithm evaluateRecursively(array, startIndex)

```
1: if array[startIndex] is a digit then
2:     return array[startIndex], startIndex + 1
3: else
4:     leftOperand ← evaluateRecursively(array, ...)
5:     rightOperand ← evaluateRecursively(array, ...)
6:     return leftOperand.result + rightOperand.result, ...
7: end if
```

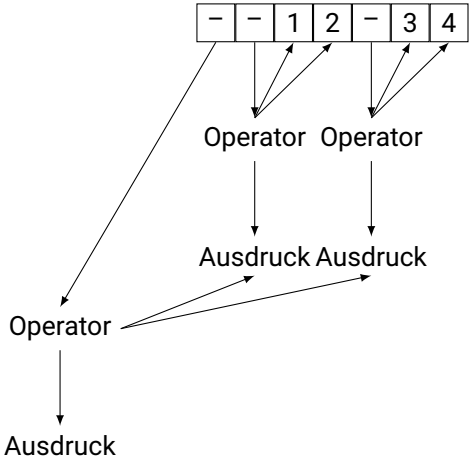


Abbildung: Visualisierung der Auswertung

Linker Operand

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wir müssen rekursiv den nächsten Index abprüfen

Algorithm evaluateRecursively(array, startIndex)

```
1: if array[startIndex] is a digit then
2:     return array[startIndex], startIndex + 1
3: else
4:     leftOperand ← evaluateRecursively(array, startIndex + 1)
5:     rightOperand ← evaluateRecursively(array, . . . )
6:     return leftOperand.result + rightOperand.result, . . .
7: end if
```

Rechter Operand

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wir müssen rekursiv nach dem letzten besuchten Index von dem linken Operand abprüfen

Algorithm evaluateRecursively(array, startIndex)

```
1: if array[startIndex] is a digit then
2:     return array[startIndex], startIndex + 1
3: else
4:     leftOperand ← evaluateRecursively(array, startIndex + 1)
5:     rightOperand ← evaluateRecursively(array, leftOperand .nextIndex)
6:     return leftOperand.result + rightOperand.result, . . .
7: end if
```

Kombination

TECHNISCHE UNIVERSITÄT DARMSTADT

- Der nächste Index nach der Kombination ist der Index nach der Auswertung von dem rechten Teilausdruck

Algorithm evaluateRecursively(array, startIndex)

```
1: if array[startIndex] is a digit then
2:     return array[startIndex], startIndex + 1
3: else
4:     leftOperand ← evaluateRecursively(array, startIndex + 1)
5:     rightOperand ← evaluateRecursively(array, leftOperand .nextIndex)
6:     return leftOperand.result + rightOperand.result, rightOperand.nextIndex
7: end if
```

```java
private static ReturnData evaluateRecursively(char[] array, int startIndex) {
    char token = array[i];
    if (Character.isDigit(token)) {
        return new ReturnData(
            Character.getNumericValue(token), startIndex + 1);
    }
    ReturnData leftOperand = evaluateRecursively(array, startIndex + 1);
    ReturnData rightOperand = evaluateRecursively(array,
        leftOperand.nextIndex);
    return new ReturnData(
        leftOperand.result + rightOperand.result,
        rightOperand.nextIndex);
}
```

Selbstständiges Arbeiten