

Aufgabe 5

Wir betrachten Terme über die Rechenarten $op \in \{+, -, \cdot, \div\}$, die rekursiv definiert sind:

- Jedes Literal ist ein Term, z. B. „4“.
- Jedes Symbol ist ein Term, z. B. „x“.
- Ist t ein Term, so ist „(t)“ ein (geklammerter) Term.
- Sind t_1, t_2 Terme, so ist „ $t_1 op t_2$ “ ebenso ein Term.

Beispiele für gültige Terme sind also „ $4 + 8$ “, „ $4 \cdot x$ “ oder „ $4 + (8 \cdot x)$ “.

- (a) Welches Design-Pattern eignet sich hier am besten zur Modellierung dieses Sachverhalts?

Kompositum

- (b) Nennen Sie drei wesentliche Vorteile von Design-Pattern im Allgemeinen.
- (c) Modellieren Sie eine Klassenstruktur in UML, die diese rekursive Struktur von *Termen* abbildet. Sehen Sie mindestens einzelne Klassen für die *Addition* und *Multiplikation* vor, sowie weitere Klassen für *geklammerte Terme* und *Literale*, welche ganze Zahlen repräsentieren. Gehen Sie bei der Modellierung der Klassenstruktur davon aus, dass eine objektorientierte Programmiersprache wie Java zu benutzen ist.
- (d) Erstellen Sie ein Objektdiagramm, welches den Term $t := 4 + (3 \cdot 2) + (12 \cdot y / (8 \cdot x))$ entsprechend Ihres Klassendiagramms repräsentiert.
- (e) Überprüfen Sie, ob das Objektdiagramm für den in Teilaufgabe d) gegebenen Term eindeutig definiert ist. Begründen Sie Ihre Entscheidung.
- (f) Die gegebene Klassenstruktur soll mindestens folgende Operationen unterstützen:
- das Auswerten von Termen,
 - das Ausgeben in einer leserlichen Form,
 - das Auflisten aller verwendeten Symbole. Welches Design-Pattern ist hierfür am besten geeignet?
- (g) Erweitern Sie Ihre Klassenstruktur um die entsprechenden Methoden, Klassen und Assoziationen, um die in Teilaufgabe f) genannten zusätzlichen Operationen gemäß dem von Ihnen genannten Design Pattern zu unterstützen.

```
3 public class Addition extends Rechenart {
4     public Addition(Term a, Term b ) {
5         super(a, b, "+");
6     }
7
8     public double auswerten() {
9         return a.auswerten() * b.auswerten();
10    }
11 }
```

github: raw

```
3 public class Divison extends Rechenart {
4     public Divison(Term a, Term b ) {
5         super(a, b, "/");
6     }
7
8     public double auswerten() {
9         return a.auswerten() / b.auswerten();
10    }
11 }
```

github: raw

```
3 public class GeklammelterTerm extends Term {
4     Term term;
5
6     public GeklammelterTerm (Term term) {
7         this.term = term;
8     }
9
10    public double auswerten() {
11        return term.auswerten();
12    }
13
14    public void ausgeben() {
15        term.ausgeben();
16    }
17 }
```

github: raw

```
3 public class Literal extends Term {
4     int wert;
5
6     public Literal(int wert) {
7         this.wert = wert;
8     }
9
10    public double auswerten() {
11        return wert;
12    }
13
14    public void ausgeben() {
15        System.out.print(wert);
16    }
17 }
```

github: raw

```
3 public class Multiplikation extends Rechenart {
4     public Multiplikation(Term a, Term b ) {
5         super(a, b, "*");
6     }
7
8     public double auswerten() {
9         return a.auswerten() * b.auswerten();
10    }
11 }
```

github: raw

```

3  abstract class Rechenart extends Term {
4      Term a;
5      Term b;
6      String operatorZeichen;
7
8      public Rechenart (Term a, Term b, String operatorZeichen) {
9          this.a = a;
10         this.b = b;
11         this.operatorZeichen = operatorZeichen;
12     }
13
14     public void ausgeben () {
15         a.ausgeben();
16         System.out.print(" " + operatorZeichen + " ");
17         b.ausgeben();
18     }
19
20     abstract public double auswerten();
21 }

```

[github: raw](#)

```

3  public class Subtraktion extends Rechenart {
4      public Subtraktion(Term a, Term b) {
5          super(a, b, "-");
6      }
7
8      public double auswerten() {
9          return a.auswerten() - b.auswerten();
10     }
11 }

```

[github: raw](#)

```

3  public class Symbol {
4      String name;
5      public Symbol(String name) {
6          this.name = name;
7      }
8
9      public double auswerten() {
10         return 0;
11     }
12
13     public void ausgeben() {
14         System.out.print(name);
15     }
16 }

```

[github: raw](#)

```

3  abstract class Term {
4      abstract public double auswerten();
5
6      abstract public void ausgeben();
7  }

```

[github: raw](#)