Einzelprüfung „Datenbanksysteme / Softwaretechnologie (vertieft)“

Einzelprüfungsnummer 66116 / 2020 / Herbst

Thema 2 / Teilaufgabe 1 / Aufgabe 5 (*Terme über die Rechenarten*)

Stichwörter: Entwurfsmuster

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?

Lösungsvorschlag

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.

Lösungsvorschlag

```java
public class Addition extends Rechenart {
  public Addition(Term a, Term b) {
    super(a, b, "+");
  }

  public double auswerten() {
    return a.auswerten() + b.auswerten();
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/rechenarten/Addition.java`

```java
public class Divison extends Rechenart {
  public Divison(Term a, Term b ) {
    super(a, b, "/");
  }

  public double auswerten() {
    return a.auswerten() / b.auswerten();
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/rechenarten/Divison.java`

```java
public class GeklammerterTerm extends Term {
  Term term;

  public GeklammerterTerm(Term term) {
    this.term = term;
  }

  public double auswerten() {
    return term.auswerten();
  }

  public void ausgeben() {
    System.out.print("(");
    term.ausgeben();
    System.out.print(")");
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/rechenarten/GeklammerterTerm.java`

```java
public class Literal extends Term {
  int wert;

  public Literal(int wert) {
    this.wert = wert;
  }

  public double auswerten() {
    return wert;
  }

  public void ausgeben() {
    System.out.print(wert);
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/rechenarten/Literal.java`

```java
public class Multiplikation extends Rechenart {
  public Multiplikation(Term a, Term b) {
    super(a, b, "*");
  }

  public double auswerten() {
    return a.auswerten() * b.auswerten();
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/rechenarten/Multiplikation.java`

```java
abstract class Rechenart extends Term {
  Term a;
  Term b;
  String operatorZeichen;

  public Rechenart (Term a, Term b, String operatorZeichen) {
    this.a = a;
    this.b = b;
    this.operatorZeichen = operatorZeichen;
  }

  public void ausgeben () {
    a.ausgeben();
    System.out.print(" " + operatorZeichen + " ");
    b.ausgeben();
  }

  abstract public double auswerten();
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/rechenarten/Rechenart.java`

```java
public class Subtraktion extends Rechenart {
  public Subtraktion(Term a, Term b) {
    super(a, b, "-");
  }

  public double auswerten() {
    return a.auswerten() - b.auswerten();
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/rechenarten/Subtraktion.java`

```java
public class Symbol extends Term {
  String name;
  public Symbol(String name) {
    this.name = name;
  }

  public double auswerten() {
    return Klient.symbole.get(name);
  }

  public void ausgeben() {
    System.out.print(name);
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/rechenarten/Symbol.java`

```java
abstract class Term {
  abstract public double auswerten();

  abstract public void ausgeben();
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/rechenarten/Term.java`