Übung 10: Visitor Pattern

Abgabetermin: 8. 6. 2017, 8:15

Name: _____ **Matrikelnummer**: _____

Informatik: ☐ G1 (Marr) ☐ G2 (Prähofer) ☐ G3 (Prähofer) ☐ G4 (Löberbauer)

WIN: ☐ G1 (Khalil) ☐ G2 (Hummel) ☐ G3 (Khalil)

Aufgabe	Punkte	abzugeben schriftlich	abzugeben elektronisch	korr.	Punkte
Übung 10	24	Java-Programm inkl. JavaDoc-Kommentare	Java-Programm inkl. JavaDoc-Kommentare	☐	

Assignment 10: Visitor für Expression Trees

Expression-Trees dienen zur Darstellung von arithmetischen Ausdrücken als Baum von Knoten. Abbildung 1 zeigt ein Klassensystem für Expression-Trees. Die Wurzel bildet ein Interface `Expr`. Davon abgeleitet sind Klassen für die Darstellung von Literalen, Variablen, dann abstrakte Klassen für binäre und unäre Ausdrücke und schließlich die konkreten Klassen für die binären Ausdrücke Addition und Multiplikation und für die unären Ausdrücke Minus und Reziprok-Wert. Wie in der Abbildung zu sehen, hat ein `Lit` einen (konstanten) `double`-Wert, eine Variable einen Namen und einen Wert, eine `BinExpr` einen linken und rechten Unterausdruck und `UnExpr` einen Unterausdruck. Daneben gibt es eine Klasse `Exprs` mit statischen Factory-Methoden zum Erzeugen der konkreten Objekte. Mit den Factory-Methoden kann man z.B. für den Ausdruck (x + 0.0) * (y * (-(-1.0))) den Expression-Tree folgend erzeugen

```
mult(add(var("x", 2.0), lit(0.0)), mult(var("y", 3.0), minus(minus(lit(1.0)))))
```

wobei die Variable `x` den Initialwert `2.0` und `y` den Initialwert `3.0` hat.

Dieses Klassensystem finden Sie in der Vorgabe zur Übung.

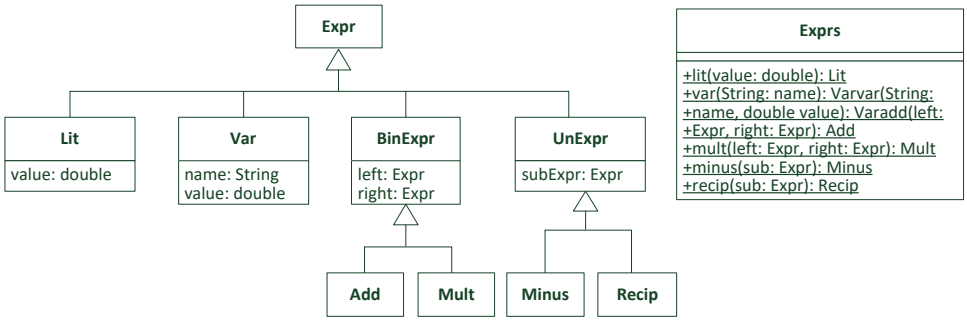

Abbildung 1: Expression-Tree Klassensystem

1) Visitor für Expr (4 Punkte)

Für diese `Expr`-Klassen sollen Sie nun das Visitor-Pattern einführen, wobei die `visit`-Methoden einen generischen Rückgabewert haben sollen. Implementieren Sie dazu zuerst ein generisches Visitor-Interface `ExprVisit<T>` mit den `visit`-Methoden, die alle den gleichen generischen Rückgabewert `T` haben. Führen Sie dann entsprechende `accept`-Methoden in den `Expr`-Klassen ein. Auch diese verwenden den generischen Rückgabewert, d.h., die `accept`-Methoden müssen auch generisch bezüglich ihres Rückgabewertes sein.

2) Konkrete ExprVisitors (14 Punkte)

Implementieren Sie nun folgende konkrete `ExprVisitor`s:

`InfixReprVisitor`: (3 Punkte)

Soll für eine Expression eine Infix-Darstellung erzeugen, d.h., der Typ der Rückgabewerte der Visitor-Methoden ist `String`. Zum Beispiel soll für den mit obigem Statement erzeugte Expression-Tree die Infixdarstellung

```
((x + 0.0) * (y * (-(-1.0))))
```

erzeugt werden. Beachten Sie die Klammern.

`EvalVisitor`: (3 Punkte)

Soll eine Expression auswerten, d.h., der Typ der Rückgabewerte der Visitor-Methoden ist `Double`. Zum Beispiel soll für den mit obigem Statement erzeugte Expression-Tree die Evaluierung

```
6.0
```

ergeben. Beachten Sie, dass die Variablen `x` und `y` die Werte `2.0` und `3.0` haben.

`SimplifyVisitor`: (8 Punkte)

Soll eine Expression nach folgenden Regeln vereinfachen:

- a + 0.0 = a, 0.0 + a = a
- a * 0.0 = 0.0, 0.0 * a = 0.0
- a * 1.0 = a, 1.0 * a = a
- - (- a) = a
- 1.0 / (1.0 / a) = a

Beachten Sie, dass vor dem Prüfen der Regeln die Unterausdrücke vereinfacht werden sollen und die Regeln dann auf die Unterausdrücke angewendet werden müssen. Zum Beispiel wird (x + (y * 0.0) folgend vereinfacht

x vereinfacht sich zu x (also keine Vereinfachung) aber (y * 0.0) vereinfacht sich zu 0.0

dann kann (x + 0.0) zu x vereinfacht werden

Obiger Expression-Tree für den Ausdruck

```
(x + 0.0) * (y * (-(-1.0)))
```

soll sich daher zu

```
(x * y)
```

vereinfacht werden.

Beachten Sie, dass bei der Vereinfachung die Ausdrücke **nicht** evaluiert werden sollen.

3) Anwendung (2 Punkte)

In der Vorgabe zur Übung finden Sie auch einen einfaches Hauptprogramm expr.app.ExprApp mit einem Dialog zum Arbeiten mit Expressions. Dieses müssen Sie an den mit **`// TODO`** gekennzeichneten Stellen entsprechend ergänzen. Mit dem Programm kann man dann Ausdrücke erzeugen, ausgeben, evaluieren und vereinfachen, wobei jeder Ausdruck unter einem fortlaufenden Index gespeichert wird und für folgende Operationen unter diesem Index abgerufen werden kann. Anbei ist ein Beispieldialog angegeben.

4) JavaDoc (4 Punkte)

Dokumentieren Sie Ihre Klassen mit JavaDoc-Kommentaren.

```
Expression commands:
==========================
l : new literal
v : new variable
= : set variable value
+ : new add expression
* : new mult expression
- : new minus expression
/ : new reciprocal expression
e : eval expression
s : simplify expression
x : exit program
--------------------------
  Command: l
    Double value: 0
[0] 0.0
  Command: l
    Double value: 1
[1] : 1.0
  Command: v
    Variable name: x
   Double value: 2
[2] x
  Command: v
    Variable name: y
    Double value: 3
[3] y
  Command: +
    Index: 2
    Index: 0
[4] (x + 0.0)
  Command: -
    Index: 1
[5] (-1.0)
  Command: -
    Index: 5
[6] (-(-1.0))
  Command: *
    Index: 3
    Index: 6
[7] (y * (-(-1.0)))
  Command: +
    Index: 4
    Index: 7
[8] ((x + 0.0) + (y * (-(-1.0))))
  Command: e
    Index: 8
[9] 5.0
  Command: s
    Index: 8
[10] (x + y)
Command: x
Thank you for using Expressions
```