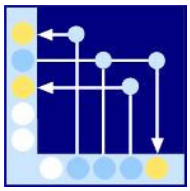
Fakultät Elektronik und Informatik
Studiengang Informatik



Programmieren 2

Vorlesung im Wintersemester 2014/2015
Prof. Dr. habil. Christian Heinlein

8. Übungsblatt (16. Dezember 2014)

Aufgabe 8: Abstrakte Klassen

Implementieren Sie Klassen zur Repräsentation arithmetischer Ausdrücke, die aus Konstanten (Objekte der Klasse `Const`) und den vier Grundrechenarten (Objekte der Klassen `Add`, `Sub`, `Mul` und `Div`) bestehen!

Für irgendeinen Ausdruck `x` soll `x.compute()` seinen Wert berechnen; `x.toString()` soll eine Zeichenkettendarstellung des Ausdrucks `x` liefern; `x.equals(y)` soll genau dann `true` liefern, wenn `y` ebenfalls ein Ausdruck ist, der strukturell und inhaltlich mit `x` übereinstimmt.

Definieren Sie ggf. zusätzliche Hilfsklassen (`Expr` und `Bin`) mit geeigneten abstrakten Hilfsmethoden (`oper` und `combine`), um Gemeinsamkeiten mehrerer Klassen an einer Stelle zusammenzufassen und so Codeverdopplungen so weit wie möglich zu vermeiden (vgl. Hinweise in der Übungsstunde)!

Testen Sie Ihre Klassen z. B. mit folgendem Hauptprogramm, das als Ausgabe

```
((2.0*3.0)+4.0) = 10.0
true
false
false
```

liefern muss:

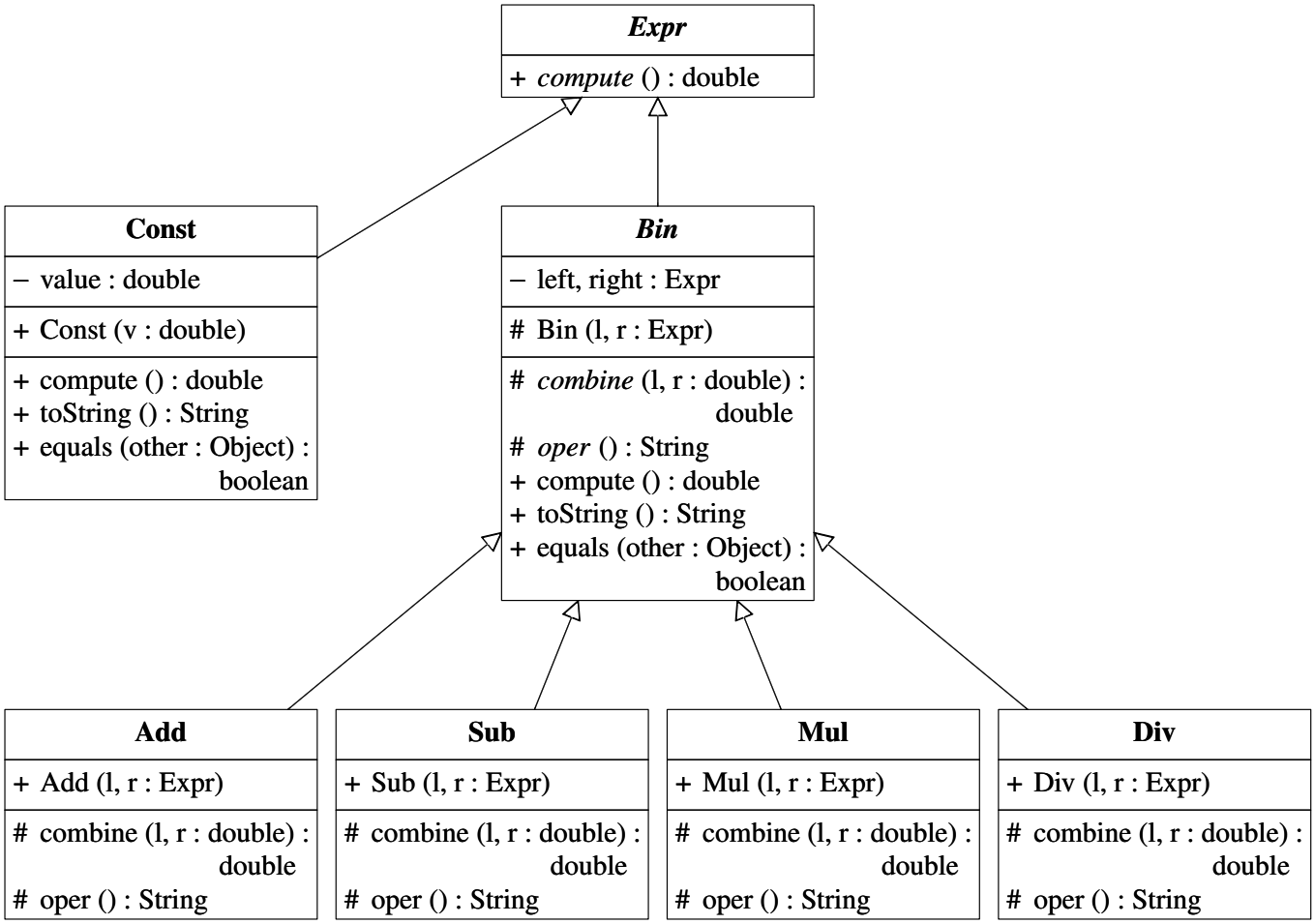
```
// Testprogramm für arithmetische Ausdrücke.
class ExprTest {
    public static void main (String [] args) {
        // Ausdruck 2 * 3 + 4 erzeugen, ausgeben und berechnen.
        Expr x = new Add(new Mul(new Const(2), new Const(3)), new Const(4));
        System.out.println(x + " = " + x.compute());

        // Ausdruck x mit anderen Ausdrücken und Objekten vergleichen.
        Expr y = new Add(new Mul(new Const(2), new Const(3)), new Const(4));
        System.out.println(x.equals(y));

        Expr z = new Add(new Mul(new Const(3), new Const(2)), new Const(4));
        System.out.println(x.equals(z));

        System.out.println(x.equals(x.toString()));
    }
}
```

Lösung




```
// Allgemeiner arithmetischer Ausdruck.
abstract class Expr {
    // Wert des Ausdrucks berechnen.
    public abstract double compute ();
}

// Konstanter Ausdruck.
class Const extends Expr {
    // Wert des Ausdrucks.
    private double value;

    // Konstanten Ausdruck mit Wert v erzeugen.
    public Const (double v) { value = v; }

    // Wert des konstanten Ausdrucks berechnen.
    public double compute () { return value; }

    // Zeichenkettendarstellung des konstanten Ausdrucks liefern.
    public String toString () { return "" + value; }
```

```
    // Konstanten Ausdruck mit Objekt other vergleichen.
    public boolean equals (Object other) {
        if (!(other instanceof Const)) return false;
        Const that = (Const)other;
        return this.value == that.value;
    }
}

// Binärer Ausdruck.
abstract class Bin extends Expr {
    // Linker und rechter Teilausdruck.
    private Expr left, right;

    // Binären Ausdruck mit Teilausdrücken l und r erzeugen.
    protected Bin (Expr l, Expr r) { left = l; right = r; }

    // Werte l und r mit dem Operator des binären Ausdrucks verknüpfen.
    protected abstract double combine (double l, double r);

    // Operator des binären Ausdrucks liefern.
    protected abstract String oper ();

    // Wert des binären Ausdrucks berechnen.
    public double compute () {
        return combine(left.compute(), right.compute());
    }

    // Zeichenkettendarstellung des binären Ausdrucks liefern.
    public String toString () {
        return "(" + left + oper() + right + ")";
    }

    // Binären Ausdruck mit Objekt other vergleichen.
    public boolean equals (Object other) {
        if (!(other instanceof Bin)) return false;
        Bin that = (Bin)other;
        return this.oper().equals(that.oper())
            && this.left.equals(that.left)
            && this.right.equals(that.right);
    }
}

// Addition.
class Add extends Bin {
    // Addition mit Teilausdrücken l und r erzeugen.
    public Add (Expr l, Expr r) { super(l, r); }

    // Werte l und r durch Addition verknüpfen.
    public double combine (double l, double r) { return l + r; }

    // Operator der Addition liefern.
    protected String oper () { return "+"; }
}
```

```
// Subtraktion.
class Sub extends Bin {
    // Subtraktion mit Teilausdrücken l und r erzeugen.
    public Sub (Expr l, Expr r) { super(l, r); }

    // Werte l und r durch Subtraktion verknüpfen.
    public double combine (double l, double r) { return l - r; }

    // Operator der Subtraktion liefern.
    protected String oper () { return "-"; }
}

// Multiplikation.
class Mul extends Bin {
    // Multiplikation mit Teilausdrücken l und r erzeugen.
    public Mul (Expr l, Expr r) { super(l, r); }

    // Werte l und r durch Multiplikation verknüpfen.
    public double combine (double l, double r) { return l * r; }

    // Operator der Multiplikation liefern.
    protected String oper () { return "*"; }
}

// Division.
class Div extends Bin {
    // Division mit Teilausdrücken l und r erzeugen.
    public Div (Expr l, Expr r) { super(l, r); }

    // Werte l und r durch Division verknüpfen.
    public double combine (double l, double r) { return l / r; }

    // Operator der Division liefern.
    protected String oper () { return "/"; }
}
```