Fragen, Antworten, Kommentare und Hinweise

Frage: Wie ist das genauer mit den stärkeren Bindungen?

Antwort: Dies kann individuell festgelegt werden, üblich ist, dass „Und“ stärker bindet als „Oder“, also $a \vee b \wedge c$ äquivalent zu $a \vee (b \wedge c)$ ist. Weiterhin erfolgt die Abarbeitung von links nach rechts, was nur relevant ist, wenn es Seiteneffekte gibt, wie es in Programmiersprachen möglich ist. Da diese sogenannten Bindungsregeln in Programmiersprachen unterschiedlich sein können, ist das setzen von Klammern immer eine Alternative. Bei den Bindungsregeln spielen alle Operatoren eine Rolle, z. B. der Cast-Operator () und Bit-Shift >>.

Erinnerung: Entwicklung mit rekursiven Strukturen

Meist gibt es zwei Varianten die Formel-Bibliothek zu erweitern. Die meist bessere Variante ist es, alle speziellen Erweiterungen in die passenden Unterklassen auszugliedern und in der Ober-Klasse Formel die Standardlösung zu schreiben. In der zweiten Variante wird die gesamte Lösung rekursiv in der Klasse Formel umgesetzt. Im folgenden Beispiel sollen Und und Oder gegeneinander ausgetauscht werden.

Lösung 1

in Formel:

```
public Formel undOderTauschen() {
  List<Formel> tmp = new ArrayList<>();
  for (Formel f : this.operanden) {
    tmp.add(f.undOderTauschen());
  }
  this.operanden = tmp;
  return this;
}
```

in Und:

```
@Override
public Formel undOderTauschen() {
      Formel[] opArray = new Formel[super.operanden.size()];
      for(int i = 0; i < super.operanden.size(); i++) {
        opArray[i] = super.operanden.get(i).undOderTauschen();
      }
      return new Oder(opArray);
}
```

in Oder:

```
@Override
public Formel undOderTauschen() {
  Formel[] opArray = new Formel[super.operanden.size()];
  for(int i = 0; i < super.operanden.size(); i++) {
    opArray[i] = super.operanden.get(i).undOderTauschen();
  }
  return new Und(opArray);
}
```

Lösung 2 in Formel:

```
public Formel undOderTauschen2() {
  List<Formel> opera = new ArrayList<>();
  for (int i = 0; i < this.operanden.size(); i++) {
    opera.add(this.operanden.get(i).undOderTauschen2());
  }
  Formel[] farray = new Formel[opera.size()];
  switch (this.typ) {
    case UND: {
      return new Oder(opera.toArray(farray));
    }
    case ODER: {
      return new Und(opera.toArray(farray));
    }
    default: {
      this.operanden = opera;
      return this;
    }
  }
}
```

Beispielnutzung:

```
public static void main(String[] args) {
  Formel f1 = new Und(new Oder(new Atom("a"), new Atom("b")),
      new Und(new Atom("a"), new Atom("b")));
  Formel f2 = (Formel)f1.clone();
  System.out.println(f1.zeigen());
  System.out.println(f1.undOderTauschen().zeigen());
  System.out.println(f2.undOderTauschen2().zeigen());
}
```

Ausgabe:

```
((a OR b) AND (a AND b))
((a AND b) OR (a OR b))
((a AND b) OR (a OR b))
```