Christian-Albrechts-Universität zu Kiel
Sommerstemester 1998
Fortgeschrittenenpraktikum (Java)

Übung 6:

Ausgabetermin: **13. Mai 1998**

Abgabe: **20. Mai 1998**

Aufgabe 1: [Netzwerkprogrammierung]

Schreiben Sie einen Server als Applikation, wenn Sie wollen auch als Applet, der auf Verbindungsaufnahmen wartet und an jeden Klienten zunächst einen Willkommensgruß und dann immer wieder — in Abständen von sagen wir 10 Sekunden — jeweils zwei Textzeilen liefert, die einen kurzen Text und Datum und Uhrzeit enthalten. Dafür verwenden Sie das Standard-Format der aktuellen Uhrzeit (siehe `(new Date() ).toString()`).

Schreiben Sie einen Klienten (als Applikation reicht wieder), der mit diesem Server Verbindung aufnimmt und alle Informationen, die vom Server gesendet werden, Zeile für Zeile auf die Standard-Ausgabe ausgibt.

Für diese Übung müssen Sie die Hostnamen der verwendeten Rechner (innerhalb des lokalen Netzes) kennen und sich auf eine Portnummer einigen.

Aufgabe 2: [Erweiterung von Schnittstellen]

In dieser und der nächsten Aufgabe geht es um die Erweiterung von *Schnittstellen* mittels `extends`, etwas, was wir bisher noch nicht aktiv genutzt haben.

In der vorliegenden Aufgabe gehen wir zurück zu den *booleschen Ausdrücken* des letzten Übungszettels. Um es nicht zu kompliziert zu machen, verwenden wir als Ausgangspunkt für's erste nicht die komplexe Version aus der letzten Aufgabe von Zettel 5, sondern die aus Aufgabe 2.[1] Die booleschen Ausdrücke sollen nachträglich um *Negation* erweitert werden, sodaß folgende Grammatik realisiert wird:

$$expr_B ::= Const \quad | \quad expr_B \vee expr_B \quad | \quad expr_B \wedge expr_B \quad | \quad \neg expr_B \qquad (1)$$

Wenn wir uns anschauen, was wir bereits haben, scheint klar, was zu tun ist: Wir erweitern die abstrakte Klasse `B_Expr_D` einfach um eine weitere Unterklasse für die Negation,

[1] Sobald alle Gruppen ihre Vorschläge abgegeben haben, lege ich meinen Vorschlag auf das Netz.

erweitern ebenso das Interface `B_ExpressionVisitor` und die konkreten Visitorenklassen. Abbildung 1 zeigt die Situation. Die in dieser Aufgabe geforderten Teile sind mit gepunktete Pfeilen dargestellt.[2]

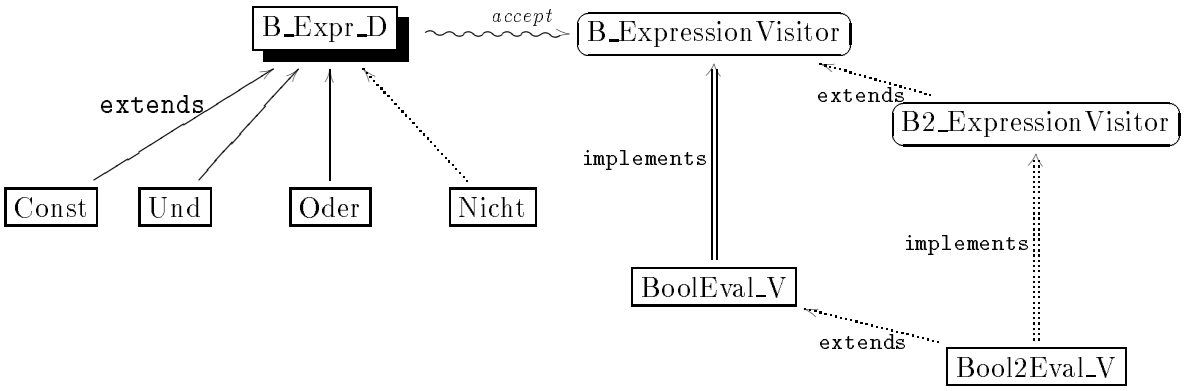

Abbildung 1: Boolesche Ausdrücke

Aufgabe 3: [Erweiterung von Schnittstellen 2]

Nun kommt die eine sehr ähnliche Aufgabe nochmal. Anstelle des Auswertungsvisitors wollen wir diesmal einen zweiten Visitor nehmen, nämlich einen für den formatierten Ausdruck, den wir `PrettyPrint_V` nennen wollen. Dieser soll in der Lage sein, boolesche Ausdrücke, aufgebaut über „*und*“ „*oder*“ und booleschen Konstanten als formatierten String zurückzugeben.[3] Dies sei (zunächst) fest vorgegeben und es ist naheliegend, das gleiche Kunststück wie in der vorangegangen Aufgabe auch hier zu versuchen. D.h., um auf das erweiterte Interface `B2_ExpressionVisitor` reagieren zu können, müssen wird die Klasse `Pretty_Print_V` um die Funktionalität für die Negation erweitern.

```
public class Pretty_Print2_V
  extends    Pretty_Print_V
  implements B2_ExpressionVisitor {

  Pretty_Print2_V (int _indent){
    super(_indent);
  };

  public Object forNicht(B_Expr_D e) {
    .....
```

Wenn Sie diese Erweiterung in der naheliegenden Weise durchführen, so werden Sie feststellen, daß das Übersetzen zwar noch funktioniert, die Ausführung, also das formatierte Drucken eines booleschen Ausdrucks, allerdings zu einem *Laufzeitfehler* führt, und zwar

[2] Die Form der Kästen und Pfeile ist, soweit ich weiß, aus keiner der verschiedenen graphischen OO-Entwurfsmethodiken entlehnt. Das Bild soll nur zur Veranschaulichung dienen.

[3] Auch diesen Code lege ich beizeiten auf die Kursseite.

wegen einer mißglückten Typumwandlung (*Type cast*). Mit anderen Worten: wirklich flexibel in dem Sinne, daß wir die Klassenhierarchie einfach erweitern und ebenso die Schnittstellen, ist das wieder mal nicht.

Bei dem Auswertungsvisitor der vorangegangenen Aufgabe gab es keine Probleme, d.h., die Schwierigkeit liegt nicht beim Interface, sondern bei dem PrettyPrint-Visitor. Um dieses Problem zu lösen, schauen Sie sich die Ursache des Laufzeitfehlers genau an, und vielleicht hilft es, sich den Visitor `BoolEval_V` zum Vergleich anzuschauen, bei dem der Fehler nicht auftritt. Das Problem ist offensichtlich der Aufruf der Aufruf des Konstruktors `Pretty_Print_V` in den Methoden der Klasse `Pretty_Print_V`. Überlegen Sie, was in der erweiterten Klasse `Pretty_Print2_V` an diesen Stelle passiert und warum das falsch ist. Versuchen Sie nun, mit diesen Hinweisen eine Lösung zu bekommen, bei dieses Problem, der Laufzeitfehler wegen des unpassenden Casts, nicht mehr auftritt und denken sie dabei an das Überschreiben von Methoden.

Exkurs für Interessierte: Das Entwurfsmuster, welches der Lösung dieser Aufgabe zugrunde liegt, wird manchmal als *Factory method* oder auch als *virtueller Konstruktor* (*virtual contructor*) bezeichnet. Wie die Aufgabe zeigt, kann man (unter anderen) damit erreichen, daß man eine Objekthierarchie *und* ihre Visitoren schmerzlos nachträglich erweitern kann. Laut [GHJV95] tauchen sie sehr häufig in Toolkits auf; eine andere Anwendung seien Parser. Der Pretty Printer dieser Aufgabe ist selbst ausgedacht und ob dieses Muster in existierenden Pretty Printern angewandt wird, habe ich nicht nachgeschaut.

Literatur

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.