Christian-Albrechts-Universität zu Kiel
Sommersemester 2000
Fortgeschrittenenpraktikum (Java)

Übung 7:

Ausgabetermin: 3. Juli 2000

Abgabe: 20. Juli 2000

Aufgabe 1: [Datenbanken]

In dieser Aufgabe soll eine Datenbank zur Verwaltung von Übungsgruppen erstellt werden. Eine postgres-Datenbank ist an der Universität installiert und kann vom Grundausbildungspool aus angesteuert werden. Dazu ist es notwendig, den Classpath wie folgt zu erweitern:

```
export CLASSPATH=$CLASSPATH:/home/unix01/jdbc/postgresql.jar
```

Der Verbindungsaufbau und -abbau zur Datenbank kann dann wie in der Vorlesung besprochen durchgeführt werden, der Zugriff auf die Datenbank `JavaUeb` erfolgt über die URL:

```
String url = "jdbc:postgresql://sokrates/JavaUeb";
```

Jede Gruppe kann nun in dieser Datenbank eine oder auch mehrere eigene Tabellen erstellen (z.B. TabUnixXY) und diese mit den in der Vorlesung vorgestellten Methoden modifizieren.

Für die Aufgabe ist es evtl. nützlich, mehr als eine Tabelle zu erzeugen. Es braucht keine graphische Oberfläche programmiert werden, textuelle Ein- und Ausgabe reichen aus. Folgende Funktionalitäten sollen zur Verfügung stehen:

- Eintragen einer Person mit Daten Name, Vorname, Matrikelnummer, Semester, Email, Telefonnummer, Gruppennummer.
- Löschen einer Person.
- Modifikation eines Personeneintrags.
- Es soll eingegeben werden können, dass eine Gruppe eine bestimmte Serie abgegeben hat.
- Ausgabe aller Namen mit Gruppe, die eine bestimmte (einzugebende) Serie nicht abgegeben haben.
- Ausgabe aller Personen mit Matrikelnummer eines (einzugebenden) Semesters.

Aufgabe 2: [Erweiterung von Schnittstellen 2]

Diese Aufgabe ähnelt der dritten Aufgabe der vorherigen Übungsserie. 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. 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 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 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 constructor*) bezeichnet. Wie die Aufgabe zeigt, kann man (unter anderen) damit erreichen, daß man eine Objekthierarchie *und* ihre Visitoren schmerzlos nachträglich erweitern kann. Diese tauchen sehr häufig in Toolkits auf; eine andere Anwendung sind Parser. Der Pretty Printer dieser Aufgabe ist selbst ausgedacht und ob dieses Muster in existierenden Pretty Printern angewandt wird, habe ich nicht nachgeschaut. Soweit ich am Rande mitbekommen habe, scheinen Typchecker und ähnliche Compilerbestandteile in der Tat als Visitoren programmiert zu werden, was naheliegend ist, sodaß auch bei PrettyPrintern ähnliche Lösungen nicht aus der Luft gegriffen sind.