Dr. M. Grabert
Johannes Mayer



# Objektorientierte Programmierung mit C++ (WS 2001/2002)

**Abgabetermin: 20. Dezember 2001**

## Beispiellösung

## 10 Studenten, wollt ihr ewig leben? (10 Punkte)

Schreiben Sie analog zur Vorlesung Klassen *Student*, *BioStudent* und *WimaStudent*. Die beiden letzten Klassen werden direkt von der Klasse *Student* abgeleitet. Zum Test soll es mindestens eine virtuelle Methode in *Student* geben, die von den beiden abgeleiteten Klassen überschrieben wird. (Zum Beispiel die Methode `printInfo()` aus der Vorlesung.) Soweit gibt's ja noch nichts neues!
Nun sollen die Klasse *Student* und die beiden abgeleiteten Klassen die Möglichkeit bieten, sich in einen Stream zu schreiben und aus einem Stream den gespeicherten Zustand wieder herzustellen. Dazu schreiben Sie zwei Methoden `store()` und `restore()`, die dafür zuständig sind. Außerdem soll es einen Konstruktor geben, der als Parameter einen Input-Stream erhält und aus diesem den Zustand des Objekts (mittels `restore()`) wieder herstellt. (Objekte, die ihren Zustand dauerhaft speichern können und aus dem gespeicherten Zustand wieder hergestellt werden können, heißen *persistente Objekte.*)
Nun brauchen wir noch einen Verwalter. Dazu schreiben Sie eine eigene Klasse, die Objekte vom Typ *Student* (oder einer abgeleiteten Klasse) verwaltet. Diese soll beim Konstruktor den Namen der Datenbank-Datei erhalten und alle Studenten aus der Datenbank einlesen. Dabei soll jeweils der richtige Typ wieder hergestellt werden, also ein gespeicherter Student soll auch wieder ein Student sein, ein Bio-Student wieder ein Bio-Student, und gleiches gilt für einen Wima. Sie sollen also zwei private Methoden schreiben, die alle verwalteten Studenten in die Datenbank schreiben und alle Studenten aus der Datenbank lesen (wobei der Typ der Objekte erhalten bleiben soll, d. h. Student bleibt Student, Bio-Student bleibt Bio-Student und Wima bleibt Wima). Außerdem schreiben Sie öffentliche Methoden zum Anlegen, Ersetzen, Löschen und Finden eines Studenten (über seinen Login-Namen). Außerdem schreiben Sie noch eine Methode mit der man einen Vektor der Login-Namen aller verwalteten Studenten erhält.
Und last but not least: Was bringen die schönsten Klassen, wenn wir sie nicht verwenden. Schreiben Sie also eine kleine Anwendung, die die vorgestellten Klassen verwendet und folgendes kann: Ausgabe der Liste aller verwalteten Studenten (mit und ohne individuelle Ausgabe

`printInfo()` oder einer anderen virtuellen Methode), Anlegen/Ersetzen eines Studenten und Löschen eines Studenten. Diese kann so ähnlich aufgebaut sein wie der Mail-Reader Emilia.
P.S. Die Liste der Studente, die vor dem Beenden des Testprogramms angezeigt werden kann, soll bei einem erneuten Programmstart wieder genauso ausschauen! Auch die Typen der Objekte, die ja angezeit werden können (indirekt via `printInfo()` oder so), sollen noch immer gleich sein. Sonst hat sich wo ein Fehler eingeschlichen. ;-)

**Lösung:**

**makefile**

```
main:   main.o student.o verwalter.o
        g++ -o main main.o student.o verwalter.o

main.o: main.cc student.h verwalter.h id.h
        g++ -c main.cc

verwalter.o:    verwalter.cc verwalter.h student.h id.h
                g++ -c verwalter.cc

student.o:      student.cc student.h id.h
                g++ -c student.cc
```

**main.cc**

```
#include <iostream>
#include <vector>
#include <string>
#include "student.h"
#include "verwalter.h"

void main() {
    Verwalter v("student.db");

    string in;

    system("clear");    // Bildschirm loeschen

        // Liste aller Studenten ausgeben
    vector<string> logins = v.allLogins();
    for (int i = 0; i < logins.size(); i++)
        cout << *v.find(logins[i]) << endl;

    do {
        // auf Eingabe von Kommando warten
        cout << "------------------------------------------------------" << endl;
        cout << "(l)ist (g)reetings (a)dd (r)emove (q)uit: ";
        cin >> in;

        system("clear");        // Bildschirm loeschen
```

```
string login, vorname, name;
int matrikelnr, id;
Student *s;

switch (in.c_str()[0]) {
    case 'l': {
            // Liste der Studenten ausgeben
            vector<string> logins = v.allLogins();
            for (int i = 0; i < logins.size(); i++)
                cout << *v.find(logins[i]) << endl;
        }
        break;
    case 'g': {
            // Liste der Studenten zusammen mit der
            // Begruessung ausgeben
            vector<string> logins = v.allLogins();
            for (int i = 0; i < logins.size(); i++) {
                cout << *v.find(logins[i]) << endl;
                cout << "   => ";
                v.find(logins[i])->greetings();
            }
        }
        break;
    case 'a':
        // Studenten anlegen bzw. aendern
        cout << "Login:          ";
        cin >> login;
        cout << "Vorname:        ";
        cin >> vorname;
        cout << "Name:           ";
        cin >> name;
        cout << "Matrikelnummer: ";
        cin >> matrikelnr;

        do {
            cout << "Typ (0,1,2):    ";
            cin >> id;
        } while (id < 0 || id > 2);

        switch (id) {
            case ID_STUDENT:
                s = new Student(login, vorname, name, matrikelnr); break;
            case ID_BIOSTUD:
                s = new BioStudent(login, vorname, name, matrikelnr); break;
            case ID_WIMASTUD:
                s = new WimaStudent(login, vorname, name, matrikelnr); break;
        }
```

```
                v.replace(s);
                break;
            case 'r':
                // Student loeschen
                cout << "Login:          ";
                cin >> login;
                v.remove(login);
                break;
            case 'q':
                // ... und Schluss
                return 0;
        }
    } while(1);

}
```

**id.h**

```
#ifndef _ID_H
#define _ID_H

#define ID_STUDENT      0
#define ID_BIOSTUD      1
#define ID_WIMASTUD     2

#endif
```

**student.h**

```
#ifndef _STUDENT_H
#define _STUDENT_H

#include <iostream>
#include <string>
#include <strings.h>
#include "id.h"

class Student {
    char login[10];
    char vorname[30];
    char name[50];
    int matrikelnr;
public:
    Student(string login, string vorname, string name, int matrikelnr) {
        strcpy(this->login, login.c_str());
        strcpy(this->vorname, vorname.c_str());
        strcpy(this->name, name.c_str());
        this->matrikelnr = matrikelnr;
    }
        // Student aus DB wiederherstellen
```

```
    virtual void restore(istream &in);
        // Student aus Stream in lesen und wieder herstellen
    Student(istream &in) {
        restore(in);
    }
        // byteweise den Zustand in out abspeichern
    virtual void store(ostream &out);
        // ID fuer die Klasse -- muss von abgeleiteten Klassen
        // ueberschrieben werden und eindeutig die Klasse indentifizieren
    virtual int classId() { return ID_STUDENT; }
        // Gruss -- zur Unterscheidung der Klassen
    virtual void greetings() {
        cout << "Hallo, ich bin ein Student!" << endl;
    }
    string getLogin() { return string(login); }
    string getVorname() { return string(vorname); }
    string getName() { return string(name); }
    int getMatrikelNr() { return matrikelnr; }

    friend ostream &operator<<(ostream &, Student);
};

class BioStudent : public Student {
public:
    BioStudent(string login, string vorname, string name, int matrikelnr)
        : Student(login, vorname, name, matrikelnr) { }
    BioStudent(istream &in) : Student(in) { }
        // hat eigene ID!!!
    virtual int classId() { return ID_BIOSTUD; }
    virtual void greetings() {
        cout << "Hallo, ich bin eine arme Bio-Studentin!" << endl;
    }
};

class WimaStudent : public Student {
public:
    WimaStudent(string login, string vorname, string name, int matrikelnr)
        : Student(login, vorname, name, matrikelnr) { }
    WimaStudent(istream &in) : Student(in) { }
        // hat eigene ID!!!
    virtual int classId() { return ID_WIMASTUD; }
    virtual void greetings() {
        cout << "Tach, jestatten, Wima-Student!" << endl;
    }
};

#endif
```

**student.cc**

```
#include <iostream>
#include <iomanip>
#include "student.h"

void Student::restore(istream &in) {
    if (in) {
        // Attribute (byteweise) in den Stream in schreiben
        in.read(login, sizeof(login));
        in.read(vorname, sizeof(vorname));
        in.read(name, sizeof(name));
        in.read((char*) &matrikelnr, sizeof(matrikelnr));
    }
}

void Student::store(ostream &out) {
    if (out) {
        // Attribute (byteweise) aus dem Stream out lesen
        out.write(login, sizeof(login));
        out.write(vorname, sizeof(vorname));
        out.write(name, sizeof(name));
        out.write((char*) &matrikelnr, sizeof(matrikelnr));
    }
}

ostream &operator<<(ostream &out, Student s) {
    out.setf(ios::left);
    out << setw(9) << s.login << ":" << setw(29) << s.name << ":" << setw(19) << s.vorname;
    out.setf(ios::right);
    return out << ":" << setw(9) << s.matrikelnr;
}
```

**verwalter.h**

```
#ifndef _VERWALTER_H
#define _VERWALTER_H

#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
#include <map>
#include <vector>
#include <strings.h>
#include "student.h"
#include "id.h"

class Verwalter {
    string db;
    map<string,Student*> studis;
```

```
        // alle Studenten aus der DB einlesen
    void readDB();
        // alle verwalteten Studenten in die DB schreiben
    void writeDB();
public:
    Verwalter(string db) : db(db) {
        readDB();
    }
        // alle Studenten in die DB schreiben und Speicher freigeben
    ~Verwalter() {
        writeDB();

        for (map<string,Student*>::iterator i = studis.begin(); i != studis.end(); i++)
            delete i->second;

        studis.clear();
    }
        // Studenten neu anlegen
    void add(Student *s);
        // Studenten anlegen und ggf. ersetzen
    void replace(Student *s);
        // Studenten ersetzen
    void remove(string login);
        // Studenten suchen und evtl. finden :-))
    Student *find(string login);
        // Liste aller verwalteten Logins liefern
    vector<string> allLogins();
};

#endif
```

**verwalter.cc**

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
#include <map>
#include <vector>
#include <strings.h>
#include "verwalter.h"
#include "id.h"

// alle Studenten aus der DB einlesen
void Verwalter::readDB() {
    ifstream in(db.c_str());
    if (in)
        while (!in.eof()) {
            int id;
            in.read((char*) &id, sizeof(id));
```

```
            // die Studenten-Klasse abhaengig von id waehlen
            Student *s;
            switch (id) {
                case ID_STUDENT:
                    s = new Student(in); break;
                case ID_BIOSTUD:
                    s = new BioStudent(in); break;
                case ID_WIMASTUD:
                    s = new WimaStudent(in); break;
            }
            if (in.eof())
                delete s;
            else
                studis[s->getLogin()] = s;
        }
}

// alle Studenten in die DB schreiben
void Verwalter::writeDB() {
    ofstream out(db.c_str());
    if (out)
        for (map<string,Student*>::iterator i = studis.begin(); i != studis.end(); i++) {
            // die ID gibt die Klasse des Studenten an => notw. fuer readDB()
            int id = i->second->classId();
            out.write((char*) &id, sizeof(id));
            i->second->store(out);
        }
}

// neuen Studenten anlegen
void Verwalter::add(Student *s) {
    if (!studis[s->getLogin()])
        studis[s->getLogin()] = s;
}

// Studenten (mit geg. Login) ggf. ersetzen
void Verwalter::replace(Student *s) {
    if (studis[s->getLogin()])
        delete studis[s->getLogin()];

    studis[s->getLogin()] = s;
}

// Studenten entfernen
void Verwalter::remove(string login) {
    if (studis.find(login) != studis.end()) {
        Student *s = studis.find(login)->second;
        studis.erase(studis.find(login));
```

```
        delete s;
    }
}

// Studenten suchen und ggf. finden :-))
Student *Verwalter::find(string login) {
    if (studis.find(login) != studis.end())
        return studis[login];
    else
        return 0;
}

// Liste aller Logins der verwalteten Studenten
vector<string> Verwalter::allLogins() {
    vector<string> logins;

    for (map<string,Student*>::iterator i = studis.begin(); i != studis.end(); i++)
        logins.insert(logins.end(), i->first);

    return logins;
}
```