Dr. M. Grabert — Abteilung Angewandte Informationsverarbeitung — 20. Dezember 2001
Johannes Mayer — Blatt 8

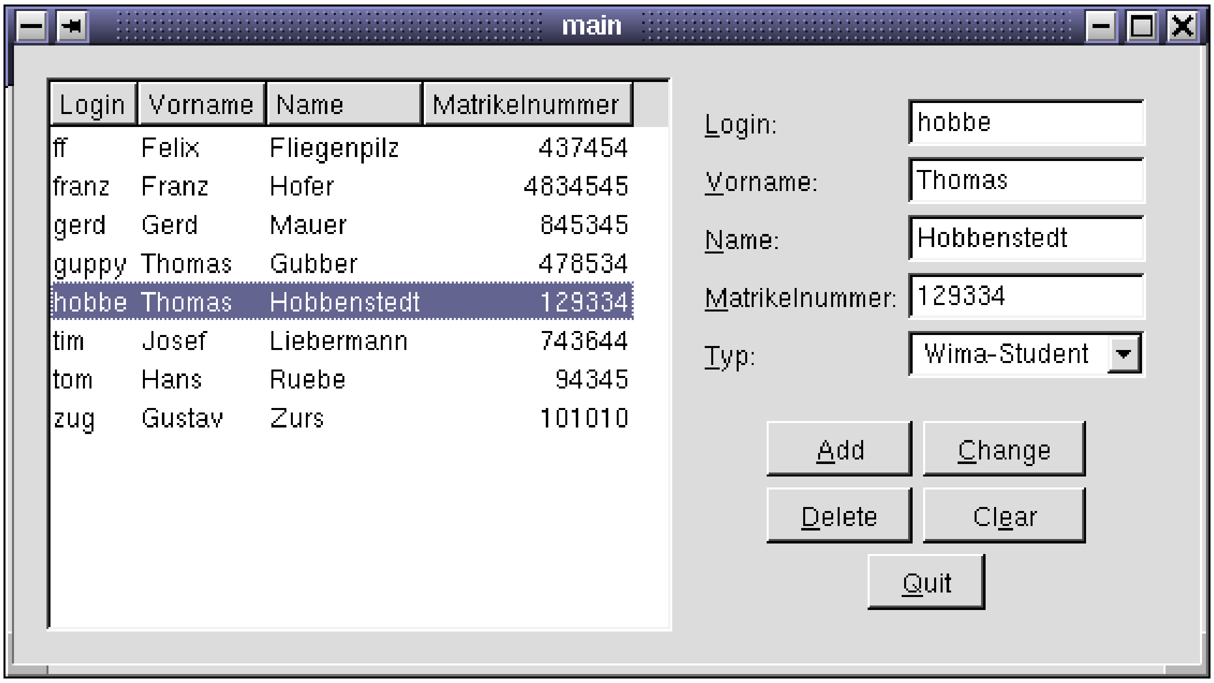
Objektorientierte Programmierung mit C++ (WS 2001/2002)

Abgabetermin: 17. Januar 2002

Beispiellösung

11 Aber bitte komfortabel (10+10 Punkte)

War ja gut und schön, die letzte Übungsaufgabe, aber besonders komfortabel zu bedienen war das Programm nicht. Tja, da bleibt nur eines: Wir basteln uns eine graphische Benutzungs-Schnittstelle (GUI). Diese könnte dann wie folgt aussehen:



Zur Programmierung der GUI verwenden wird die *Qt-Bibliothek* Version 2.3.1 von Trolltech. Diese ist auf Thales, Turing und allen Sun Ultra 5-Rechnern im Verzeichnis `/usr/local/qt-2.3.1/` installiert. Die lokale Dokumentation zu dieser Bibliothek gibt es unter
http://www.mathematik.uni-ulm.de/help/qt-2.3.1/.
Näheres zu der Bibliothek, zum Compilieren und Starten von Programmen und Beispielprogramme gibt es unter

ftp://ftp.mathematik.uni-ulm.de/pub/vorlesungen/ws01/cpp/uebungen/8/bsp/.
In den Übungen (am 20. Dezember) gibt es eine Einführung zu Qt.
Programmieren Sie also in Qt 2.3.1 eine GUI für Ihre Klassen `Student`, `BioStudent`, `WimaStudent` und `Verwalter` vom letzten Übungsblatt (oder verwenden Sie diese Klassen aus der Beispiellösung). Die graph. Benutzungsschnittstelle, welche Sie schreiben, soll etwa so aussehen wie die oben abgebildete. Bei Start des Programms wird ein neuer Verwalter angelegt, der alle Studenten einliest. Diese Studenten geben Sie dann alle in einer Listbox aus. Klickt man auf einen Eintrag in der Liste, so sollen diese Werte in die Eingabefelder übernommen werden. Mit dem Button *Clear* kann man alle Eingabefelder wieder löschen. (*Clear* betrifft also nicht den Verwalter, sondern nur die Anzeige. Die Eingabefelder sollen gelöscht werden, damit eine neuer Student schnell angelegt werden kann.) Dabei soll auch eine Auswahl in der Listbox aufgehoben werden. Mittels *Add* wird mit den Werten aus den Eingabefeldern ein neuer Student – mit dem eingestellten Typ (Student, Bio- oder Wima-Student) – erzeugt, falls es noch keinen Studenten mit diesem Login gibt. (Der neu erzeugte Student soll sowohl im Verwalter hinzugefügt werden, wie auch in der Listbox.) Mittels *Change* kann man die Werte des aktuell ausgwählten Listeneintrags ändern (auch den Typ des Studenten). Schließlich kann man mittels *Delete* den aktuell ausgewählten Studenten löschen. Mit *Quit* beendet man die Anwendung, wobei der Verwalter ebenfalls sauber beendet werden muss, damit beim nächsten Start wieder die selben Studenten angezeigt werden.
P.S. Sie müssen entweder Ihre eigenen Klassen `Student`, ... und `Verwalter` oder die der Beispiellösung von Blatt 7 **wiederverwenden**.

Lösung:

makefile

```
QTDIR = /usr/local/qt-2.3.1

CC = g++
CC_QT = $(CC) -I$(QTDIR)/include
CC_QT_LINK = $(CC) -I$(QTDIR)/include -L$(QTDIR)/lib -lqt
MOC = $(QTDIR)/bin/moc

main:           main.o gui.o moc_gui.o student.o verwalter.o
                $(CC_QT_LINK) -o main main.o gui.o moc_gui.o student.o verwalter.o

main.o:         main.cc gui.h
                $(CC_QT) -c main.cc

gui.o:          gui.cc gui.h verwalter.h student.h id.h
                $(CC_QT) -c gui.cc

moc_gui.o:      moc_gui.cc
                $(CC_QT) -c moc_gui.cc

moc_gui.cc:     gui.h
                $(MOC) gui.h -o moc_gui.cc

verwalter.o:    verwalter.cc verwalter.h student.h id.h
```

```
		$(CC) -c verwalter.cc

student.o:	student.cc student.h id.h
		$(CC) -c student.cc
```

main.cc

```
// Uebungsaufgabe: Graph. Frontend mit QT fuer Studenten-Verwalter -- (c) 2001 sai jm
#include <qapplication.h>
#include "gui.h"

void main(int argc, char ** argv) {
    QApplication a(argc, argv);         // Anwendungsobjekt vor Widgets erzeugen

    MainWidget w;                       // eigenes Fenster ...
    a.setMainWidget(&w);                // ... zum Haupt-Fenster machen
    w.show();                           // und anzeigen

    return a.exec();                    // Haupt-Ereignisschleife
}
```

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;
}
```

gui.h

```
// Uebungsaufgabe: Graph. Frontend mit QT fuer Studenten-Verwalter -- (c) 2001 sai jm
#ifndef _GUI_H
#define _GUI_H

#include <qwidget.h>
#include <qobject.h>
#include <qlistview.h>
#include <qlineedit.h>
#include <qcombobox.h>
#include <qpushbutton.h>

#include "verwalter.h"


    // eigene Fensterklasse (wg. Slots) notwendig
class MainWidget : public QWidget
{

    Q_OBJECT

public:
```

```
        // Widgets und Verwalter anlegen
    MainWidget(QWidget *parent = 0, const char *name = 0);
        // Verwalter sauber beenden!
    ~MainWidget();

private slots:

        // wird bei Aenderung der Auswahl in der ListView aufgerufen
    void onSelectionChanged(QListViewItem*);

        // Bearbeiter beim Druecken des jeweiligen Buttons
    void buttonAdd();
    void buttonChange();
    void buttonDelete();
    void buttonClear();

private:

        // initialisiert die ListView mit allen Studenten des Verwalters
    void initListView();
        // einen Eintrag der ListView hinzufuegen
    void addListViewItem(const char*, const char*, const char*, int);

        // ListView erzeugen
    void setupListView(QWidget *parent);
        // Eingabefelder mit Textfeldern erzeugen
    void setupLineEdits(QWidget *parent);
        // Buttons erzeugen
    void setupButtons(QWidget *parent);

        // Studenten-Verwalter
    Verwalter *verw;

        // Fenster-Elemente:
    QListView *listView;
    QLineEdit *login, *vorname, *name, *matr;
    QComboBox *typ;
    QPushButton *add, *change, *del, *clear, *quit;

};

#endif
```

gui.cc

```
// Uebungsaufgabe: Graph. Frontend mit QT fuer Studenten-Verwalter -- (c) 2001 sai jm
#include <qapplication.h>
#include <qwidget.h>
#include <qlayout.h>
```

```
#include <qobject.h>
#include <qlistview.h>
#include <qlabel.h>
#include <qlineedit.h>
#include <qcombobox.h>
#include <qpushbutton.h>
#include <qmessagebox.h>

#include <string>
#include <vector>

#include "id.h"
#include "student.h"
#include "verwalter.h"
#include "gui.h"

    /* Erzeugen aller Fenster-Widgets, Anordnung dieser
     * Widgets und das Erzeugen des Verwalters
     */
MainWidget::MainWidget(QWidget *parent, const char *name)
    : QWidget(parent, name)
{
    QGridLayout *lthis = new QGridLayout(this, 1, 2, 15, 5);

        // ListView links anordnen im Fenster
    setupListView(this);
    lthis->addWidget(listView, 0, 0);

        // rechts im Fenster einen Container anlegen
    QWidget *right = new QWidget(this);
    lthis->addWidget(right, 0, 1);
    QGridLayout *lright = new QGridLayout(right, 3, 1);
    lright->setRowStretch(1, 1);

        // rechts oben im Fenster die Eingabefelder positionieren
    QWidget *top = new QWidget(right);
    lright->addWidget(top, 0, 0);

    setupLineEdits(top);

        // rechts unten im Fenster die Buttons anordnen
    QWidget *bottom = new QWidget(right);
    lright->addWidget(bottom, 2, 0);

    setupButtons(bottom);

        // die Buttons mit ihren Bearbeiter-Methoden verknuepfen
    QObject::connect(add, SIGNAL(pressed()), this, SLOT(buttonAdd()));
    QObject::connect(change, SIGNAL(pressed()), this, SLOT(buttonChange()));
```

```
    QObject::connect(del, SIGNAL(pressed()), this, SLOT(buttonDelete()));
    QObject::connect(clear, SIGNAL(pressed()), this, SLOT(buttonClear()));
    QObject::connect(quit, SIGNAL(pressed()), qApp, SLOT(quit()));

        // die ListView mit der Methode verknuepfen, die immer
        // dann aufgerufen werden soll, wenn sich an der Auswahl in
        // der Listbox etwas aendert
    QObject::connect(listView, SIGNAL(selectionChanged(QListViewItem*)),
                     this, SLOT(onSelectionChanged(QListViewItem*)));

        // einen neuen Studenten-Verwalter anlegen
    verw = new Verwalter("student.db");

        // alle Studenten aus dem Verwalter in die ListView einfuegen
    initListView();
}

    /* den Verwalter der Studenten sauber beenden
     */
MainWidget::~MainWidget()
{
    delete verw;
}

    /* Bearbeiter, falls sich an der Auswahl in der Listbox
     * etwas aendert
     */
void MainWidget::onSelectionChanged(QListViewItem* item)
{
    if (item->isSelected()) {           // falls etwas ausgewaehlt ist
            // Listeneintrag in die Eingabefelder uebernehmen
        login->setText(item->text(0));
        vorname->setText(item->text(1));
        name->setText(item->text(2));
        matr->setText(item->text(3));

            // passenden Studenten (mit diesem Login suchen)
        Student *s = verw->find(item->text(0).latin1());

            // ... und den Studenten-Typ rechts richtig in
            // der ComboBox anzeigen
        typ->setCurrentItem(s->classId());
    }
}

    /* Bearbeiter beim Druecken des Buttons "Add"
     */
void MainWidget::buttonAdd()
{
```

```
        // nur neuen Studenten anlegen, falls das Login nicht leer
        // ist und noch nicht existiert
    if (!login->text().isEmpty() && !verw->find(login->text().latin1())) {
        Student *s;

            // Studenten-Objekt mit dem richtigen Typ anlegen
        switch (typ->currentItem()) {
            case ID_STUDENT:
                s = new Student(login->text().latin1(),
                                vorname->text().latin1(),
                                name->text().latin1(),
                                matr->text().toInt());
                break;
            case ID_BIOSTUD:
                s = new BioStudent(login->text().latin1(),
                                   vorname->text().latin1(),
                                   name->text().latin1(),
                                   matr->text().toInt());
                break;
            case ID_WIMASTUD:
                s = new WimaStudent(login->text().latin1(),
                                    vorname->text().latin1(),
                                    name->text().latin1(),
                                    matr->text().toInt());
                break;
        }

            // ... beim Verwalter hinzufuegen
        verw->add(s);

            // ... und auch in der Liste hinzufuegen
        addListViewItem(s->getLogin().c_str(), s->getVorname().c_str(),
                        s->getName().c_str(), s->getMatrikelNr());
    }
        // andernfalls eine Warnung ausgeben
    else if (login->text().isEmpty())
        QMessageBox::warning(this, "Warnung", "Login darf nicht leer bleiben!");
    else if (verw->find(login->text().latin1()))
        QMessageBox::warning(this, "Warnung", "Es gibt schon einen Studenten "
                             "mit diesem Login!");
}

    /* Bearbeiter beim Druecken des Buttons "Change"
     */
void MainWidget::buttonChange()
{
    QListViewItem *item = listView->selectedItem();

        // Studenten nur aendern, falls ein Listeneintrag ausgewaehlt,
```

```
        // das Login nicht leer und mit dem aus der Liste uebereinstimmt
    if (item && !login->text().isEmpty() && item->text(0) == login->text()) {
        Student *s;

            // Studenten-Objekt mit dem richtigen Typ anlegen
        switch (typ->currentItem()) {
            case ID_STUDENT:
                s = new Student(login->text().latin1(),
                                vorname->text().latin1(),
                                name->text().latin1(),
                                matr->text().toInt());
                break;
            case ID_BIOSTUD:
                s = new BioStudent(login->text().latin1(),
                                   vorname->text().latin1(),
                                   name->text().latin1(),
                                   matr->text().toInt());
                break;
            case ID_WIMASTUD:
                s = new WimaStudent(login->text().latin1(),
                                    vorname->text().latin1(),
                                    name->text().latin1(),
                                    matr->text().toInt());
                break;
        }

            // Student im Verwalter ersetzen
        verw->replace(s);

            // ... und in der Liste korrigieren
        item->setText(1, vorname->text());
        item->setText(2, name->text());
        item->setText(3, matr->text());
    }
        // ... andernfalls eine Warnung ausgeben
    else if (!item)
        QMessageBox::warning(this, "Warnung", "Es muss ein Listeneintrag "
                             "ausgewaehlt sein!");
    else if (login->text().isEmpty() || item->text(0) != login->text())
        QMessageBox::warning(this, "Warnung", "Der Login muss mit dem "
                             "Listeneintrag uebereinstimmen!");
}

    /* Bearbeiter beim Druecken des Buttons "Delete"
     */
void MainWidget::buttonDelete()
{
    QListViewItem *item = listView->selectedItem();     // ausgew. Listeneintr.
```

```
        // Loeschen geht nur, wenn ein Listeneintrag ausgew. ist
    if (item) {
            // Listeneintrag loeschen
        listView->takeItem(item);

            // und aus dem Verwalter entfernen
        verw->remove(item->text(0).latin1());

            // und dann noch die Eingabefelder loeschen
        buttonClear();
    }
        // andernfalls eine Warnung ausgeben
    else if (!item)
        QMessageBox::warning(this, "Warnung", "Es muss ein Listeneintrag "
                             "ausgewaehlt sein!");
}

    /* Bearbeiter beim Druecken des Buttons "Clear"
     */
void MainWidget::buttonClear()
{
        // die Eingabefelder wieder auf leer setzen
    login->setText("");
    vorname->setText("");
    name->setText("");
    matr->setText("");
        // den Studenten-Typ wieder auf "Student" setzen
    typ->setCurrentItem(0);

        // und die Auswahl in der Listbox deselektieren
    listView->clearSelection();
}

    /* Studenten aus dem Verwalter in die Listbox aufnehmen
     */
void MainWidget::initListView()
{
    vector<string> logins = verw->allLogins();  // Liste aller Logins besorgen

    for (int i = 0; i < logins.size(); i++) {
            // zu jedem Login den Studenten finden
        Student *s = verw->find(logins[i]);

            // ... und diesen in die Listbox einfuegen
        addListViewItem(s->getLogin().c_str(), s->getVorname().c_str(),
                        s->getName().c_str(), s->getMatrikelNr());
    }
}
```

```
    /* einen neuen Listeneintrag hinzufuegen
     */
void MainWidget::addListViewItem(const char *login, const char *vorname,
                                 const char *name, int matr)
{
    QListViewItem *item = new QListViewItem(listView);
    item->setText(0, login);
    item->setText(1, vorname);
    item->setText(2, name);
    item->setText(3, QString::number(matr));
}

    /* die ListView erzeugen
     */
void MainWidget::setupListView(QWidget *parent)
{
    listView = new QListView(parent);

        // Titel-Zeile erzeugen
    listView->addColumn("Login");
    listView->addColumn("Vorname");
    listView->addColumn("Name");
    listView->addColumn("Matrikelnummer");

        // Matrikelnr. (in Spalte 3) rechtsbuendig ausgeben
    listView->setColumnAlignment(3, Qt::AlignRight);

        // nur ein Listeneintrag soll ausgewaehlt werden koennen
    listView->setSelectionMode(QListView::Single);

        // Listeneintrag soll komplett markiert sein
        // (nicht nur erste Zelle eines Eintrags)
    listView->setAllColumnsShowFocus(TRUE);
}

    /* Eingabefelder mit Textfeldern erzeugen und anordnen
     */
void MainWidget::setupLineEdits(QWidget *parent)
{
    QGridLayout *lparent = new QGridLayout(parent, 5, 2, 10, 5);

        // Textfeld mit Text "Login:" erzeugen
        // (Alt-L wird wg. & zu Tastenkuerzel)
    QLabel *labelLogin = new QLabel("&Login:", parent);
        // in erster Zeile links anordnen
    lparent->addWidget(labelLogin, 0, 0);
        // Eingabefeld erzeugen
    login = new QLineEdit(parent);
        // ... und in 1. Zeile rechts anordnen
```

```
    lparent->addWidget(login, 0, 1);
        // Focus des Textfeldes wird auf Eingabefeld uebertragen
        // (fuer Alt-L sinnvoll!)
    labelLogin->setBuddy(login);

    QLabel *labelVorname = new QLabel("&Vorname:", parent);
    lparent->addWidget(labelVorname, 1, 0);
    vorname = new QLineEdit(parent);
    lparent->addWidget(vorname, 1, 1);
    labelVorname->setBuddy(vorname);

    QLabel *labelName = new QLabel("&Name:", parent);
    lparent->addWidget(labelName, 2, 0);
    this->name = new QLineEdit(parent);
    lparent->addWidget(this->name, 2, 1);
    labelName->setBuddy(this->name);

    QLabel *labelMatr = new QLabel("&Matrikelnummer:", parent);
    lparent->addWidget(labelMatr, 3, 0);
    matr = new QLineEdit(parent);
    lparent->addWidget(matr, 3, 1);
    labelMatr->setBuddy(matr);

    QLabel *labelTyp = new QLabel("&Typ:", parent);
    lparent->addWidget(labelTyp, 4, 0);
        // ComboBox fuer die Studenten-Typen
    typ = new QComboBox(parent);
    lparent->addWidget(typ, 4, 1);
    labelTyp->setBuddy(typ);

        // Eintraege in die ComboBox schreiben
    typ->insertItem("Student", ID_STUDENT);
    typ->insertItem("Bio-Studentin", ID_BIOSTUD);
    typ->insertItem("Wima-Student", ID_WIMASTUD);
}

    /* Buttons erzeugen und anordnen
     */
void MainWidget::setupButtons(QWidget *parent)
{
    QGridLayout *lparent = new QGridLayout(parent, 3, 4, 10, 5);
        // 0. und 3. Spalte soll beim Vergroessern den
        // kompletten Platz beanspruchen
    lparent->setColStretch(0, 1);
    lparent->setColStretch(3, 1);

        // Button "Add" in der ersten Zeile links pos.
    add = new QPushButton("&Add", parent);
    lparent->addWidget(add, 0, 1);
```

```
    change = new QPushButton("&Change", parent);
    lparent->addWidget(change, 0, 2);

    del = new QPushButton("&Delete", parent);
    lparent->addWidget(del, 1, 1);

    clear = new QPushButton("Cl&ear", parent);
    lparent->addWidget(clear, 1, 2);

        // Button "Quit" anlegen
    quit = new QPushButton("&Quit", parent);
        // ... und in der 3. Zeile von Zelle 1 bis 2 zentriert anordnen
    lparent->addMultiCellWidget(quit, 2, 2, 1, 2, Qt::AlignCenter);
}
```