Projektarbeit 4

Ziel, Inhalt

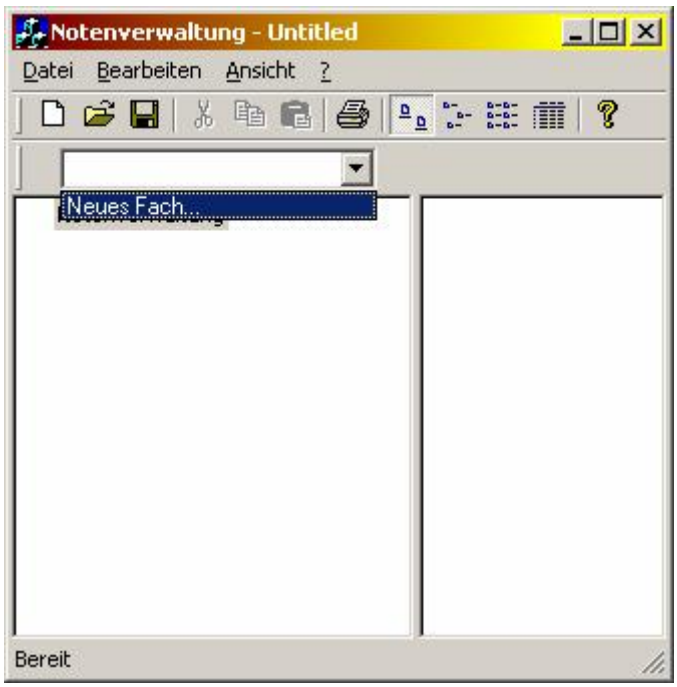
- Beim letzten Zustand, gab es noch ein Problem mit der Darstellung der Fachliste im Dialogbar, was wir als erstes lösen.
- Danach sollten wir auch endlich Noten eintragen können.
- Der rechte Teil mit dem Listview wird im Moment noch nicht verwendet. Mit dem Wissen, das wir über das List-Control (5. Abend) haben, können wir das ändern.

Projektarbeit 4

Fächerliste bereinigen

Der unschöne Effekt

Es gibt nun drei Möglichkeiten neue Fächer zu erzeugen. Das erste ist der spezielle Eintrag in der Drop-Down Liste im Dialog-Bar.



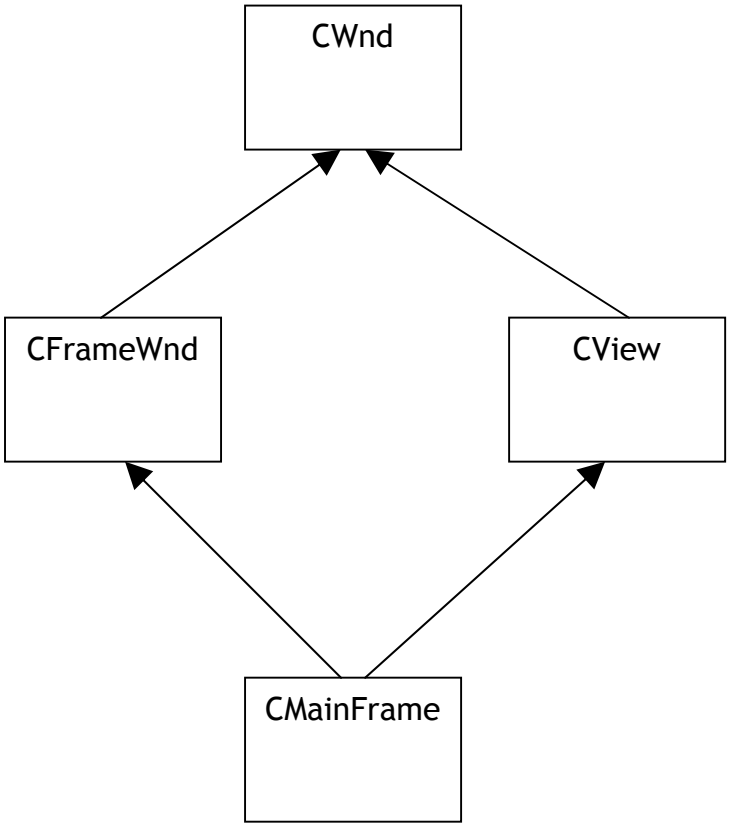
Fach erzeugen mit Drop-Down Liste

Die zweite Möglichkeit führt auf klassische Art über das Bearbeiten Menu und die dritte wird mit einem Rechtsklick auf den Eintrag „Notenverwaltung“ in der Baumansicht genutzt.
Wenn wir ein Fach über das Menu oder durch Rechtklick erzeugen, wird die Drop-Down Liste nicht nachgeführt.

Observer Pattern in der MFC

Eigentlich haben wir alles richtig gemacht, die Daten befinden sich alle im Document (Aktuelles Fach, Fachliste) und sind von den Views auch zugänglich. Zusätzlich werden die Views auch von Änderungen mittels Update unterrichtet. Das Problem liegt daran, dass ein Observer in der MFC auch von CView abgeleitet sein muss. Leider ist CView auch von CWnd abgeleitet, es werden also zwei Dinge auf einmal mitgeerbt. Das ist ein gewisses Problem beim Ableiten, man erbt Dinge mit, die man nicht braucht, die Abhängigkeiten sind häufig recht gross.
In unserem Fall ist das Problem, dass sich der Dialog-Bar im Objekt der Klasse CMainFrame befindet. CMainFrame ist jedoch keine von CView

abgeleitete Klasse. Wir können CMainFrame auch nicht zusätzlich von CView ableiten. Das führt dazu, dass wir in der Vererbungshierarchie mehrmals die gleichen Klassen haben, was mit der MFC schwierig zu lösen ist.



Problematische Mehrfachvererbung in der MFC

Schöner wäre hier eine „reinere“ Implementation des Observer Patterns. Wir werden hier noch ein Observer Pattern selber hinzufügen, was nicht sehr schön ist, da wir dann mehrere Benachrichtigungsmechanismen haben. Für den Augenblick scheint mir das trotzdem die beste Lösung zu sein.

Observer für die Fächerliste

Ich gehe jeweils so vor, dass ich die virtuelle Basisklasse für den Observer direkt über der Klasse des Subjekts deklariere. Das Subjekt ist in unserem Fall die Klasse CNotenverwaltungDoc.
Ich will mich auch beschränken und nur das Problem mit der Fachliste lösen. Das heisst das Observer Interface wird nur Methoden zur Änderung der Fächer enthalten. Sollte später mehr nötig sein, lässt sich das immer noch ergänzen.
Es gibt zwei Ereignisse, die unsere Fächer betreffen können. Einerseits kann ein neues Fach erzeugt werden, andererseits kann ein anderes Fach als aktuelles ausgewählt werden.

```
class FachObserver
{
public:
    virtual void onNeuesFach(const CString& neuesFach) = 0;
    virtual void onNeuesAktuellesFach(const CString&
                                      neuesAktuellesFach) = 0;
};
```

Diese Klasse deklariere ich in einer eigenen Datei „FachObserver.h“. Die Dokumenten-Klasse braucht jetzt noch eine Liste oder einen Container für die Observer. Hier können wir problemlos eine Klasse der Standard Library verwenden, denn die Observer werden wir nicht serialisieren müssen. Ich verwende hierfür gerne ein std::set, da es sehr einfach ist Elemente ein- und auszutragen.

```
class FachObserver
{
public:
    virtual void onNeuesFach(const CString& neuesFach) = 0;
    virtual void onNeuesAktuellesFach(const CString&
                                      neuesAktuellesFach) = 0;
};

Datei CNotenverwaltungDoc.h

#include <set>

typedef std::set<FachObserver*> FachObservers;

class CNotenverwaltungDoc : public CDocument
{
public:
    void fillFachArray(CStringArray& fachnamen);
    void setAktuellesFach(const CString& fachname);
    CString getAktuellesFach() const;
    virtual ~CNotenverwaltungDoc();
    void addFachObserver(FachObserver* fachObserver);
    void removeFachObserver(FachObserver* fachObserver);

protected:

public:
      //{{AFX_MSG(CNotenverwaltungDoc)
      afx_msg void OnBearbeitenNeuesfacherzeugen();
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
private:
      CString         _aktuellesFach;
      CFaecher        _faecher;
      FachObservers   _fachObservers;
};
```

Hier ein Ausschnitt aus meiner Deklaration der Klasse CNotenVerwaltungDoc.

Die Implemetation der beiden Methoden zur Verwaltung sieht so aus:

```
void CNotenverwaltungDoc::addFachObserver(
                          FachObserver* fachObserver)
{
    _fachObservers.insert(fachObserver);
}

void CNotenverwaltungDoc::removeFachObserver(
                          FachObserver* fachObserver)
{
    _fachObservers.erase(fachObserver);
}
```

Das ist das schöne am std::set. Das Einfügen und vor allem das Entfernen von Elementen funktioniert immer.

Es fehlt das Benachrichtigen der Observer:

```
void CNotenverwaltungDoc::setAktuellesFach(const CString &fachname)
{
    _aktuellesFach = fachname;
    UpdateAllViews(0);
    // die Fachobserver zusätzlich
    // benachrichtigen
    FachObservers::iterator it = _fachObservers.begin();
    FachObservers::iterator end = _fachObservers.end();
    for( ; it != end; ++it)
    {
        FachObserver* observer = *it;
        observer->onNeuesAktuellesFach(_aktuellesFach);
    }
}

void CNotenverwaltungDoc::OnBearbeitenNeuesfacherzeugen()
{
    CNeuesFachDialog dlg;
    if(IDOK == dlg.DoModal())
    {
        CString fachname = dlg.getFachname();
        if(0 != fachname.GetLength())
        {
            CString lehrername = dlg.getLehrername();
            _faecher.neuesFach(fachname, lehrername);
            // das neue Fach wird auch gleich
            // zum aktuellen Fach
            _aktuellesFach = fachname;
            // Views über die Änderungen
            // benachrichtigen
            UpdateAllViews(0);
            // die Fachobserver zusätzlich
            // benachrichtigen
            FachObservers::iterator it = _fachObservers.begin();
            FachObservers::iterator end = _fachObservers.end();
            for( ; it != end; ++it)
            {
                FachObserver* observer = *it;
                observer->onNeuesFach(_aktuellesFach);
                observer->onNeuesAktuellesFach(_aktuellesFach);
            }
        }
    }
}
```

Das CMainFrame Objekt als FachObserver

Unser Objekt der Klasse CMainFrame wird sich also beim CDocument Objekt als Observer eintragen, und muss auch von der Klasse FachObserver ableiten und die beiden virtuellen Methoden überschreiben:

```
void CMainFrame::onNeuesAktuellesFach(const CString&
neuesAktuellesFach)
{
    CWnd* comboboxWindow = m_wndDlgBar.GetDlgItem(IDC_FACH);
    if(comboboxWindow)
    {
        CComboBox combobox;
        combobox.Attach(comboboxWindow->m_hWnd);
        combobox.SelectString(-1, neuesAktuellesFach);
        combobox.Detach();
    }
}

void CMainFrame::onNeuesFach(const CString& neuesAktuellesFach)
{
    CWnd* comboboxWindow = m_wndDlgBar.GetDlgItem(IDC_FACH);
    if(comboboxWindow)
    {
        CComboBox combobox;
        combobox.Attach(comboboxWindow->m_hWnd);
        combobox.InsertString(-1, neuesAktuellesFach);
        combobox.Detach();
    }
}
```

Dadurch, dass das CMainFrame Objekt jedes neue Fach gleich in die Liste einträgt, erübrigt sich der aufwendige Code in der Methode OnSelectFach, die bis anhin immer die ganze Combobox abfüllte.

```
void CMainFrame::OnSelectFach()
{
    CWnd* comboboxWindow = m_wndDlgBar.GetDlgItem(IDC_FACH);
    if(comboboxWindow)
    {
        CString fachname;
        comboboxWindow->GetWindowText(fachname);
        CNotenverwaltungView* pView = GetRightPane();
        CNotenverwaltungDoc* doc = pView->GetDocument();

        if(fachname == neuesFachString)
        {
            doc->OnBearbeitenNeuesfacherzeugen();
        }
        else
        {
            doc->setAktuellesFach(fachname);
        }
    }
}
```

Das CMainFrame Objekt muss sich nun noch als Observer beim Document Objekt eintragen. Eine gute Stelle dafür ist die Methode OnCreateClient. In dieser werden die View Objekte erzeugt und das Document ist bereits vorhanden.

```
BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT /*lpcs*/,
      CCreateContext* pContext)
{
    // create splitter window

!!!    SNIP SNIP EIN TEIL FEHLT HIER

    // typsicherer Zugriff auf das Document Objekt
    // mit Umweg über das View Objekt
    CNotenverwaltungView* pView = GetRightPane();
    CNotenverwaltungDoc* doc = pView->GetDocument();
    // Wir tragen uns als Observer ein
    doc->addFachObserver(this);

      return TRUE;
}
```

Test

Das Programm funktioniert ein wenig besser. Vor allem scheint die Combobox meistens auf dem aktuellen Stand zu sein.
Nur das Wählen eines Faches im TreeView führt noch nicht dazu, dass das aktuelle Fach nachgeführt wird.

Aufgabe 1

Durch Abfangen der linken Maustaste im rechten Fenster, sollte es möglich sein das aktuelle Fach im Document Objekt nachzuführen. Ein Teil des Codes ist ähnlich wie der Code für die rechte Maustaste (OnRClick).
Nützlich könnte die Methode CTreeCtrl.GetParentItem(HTREEITEM) sein, mit der es möglich ist das übergeordnete Element eines Elementes im Baum zu finden.

Aufgabe 2

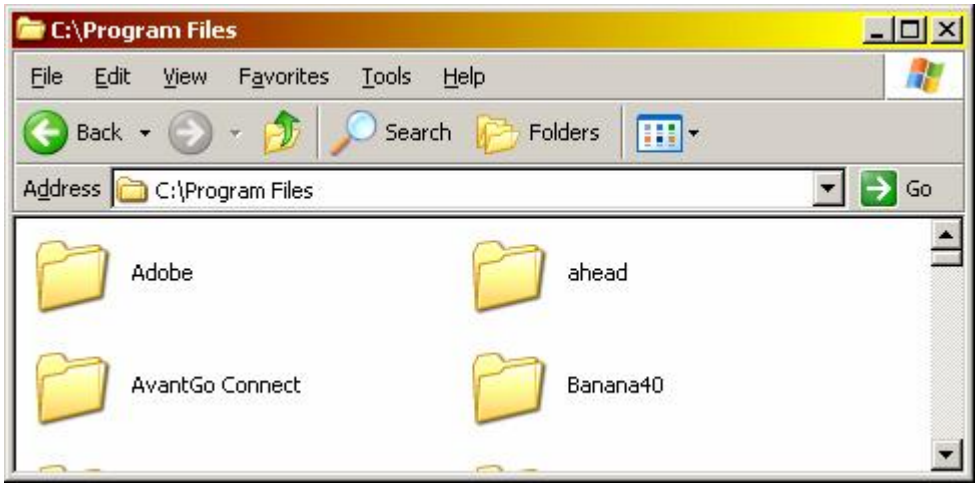
Beim laden eines Dokumentes wird die Listbox immer noch nicht nachgeführt. Erweitere die Klasse FachObserver um eine Methode, die aufgerufen wird, wenn es nötig ist die ganze Liste neu aufzubauen. Das CMainFrame kann sich dann als Reaktion die ganze Fachliste holen und alle Einträge reinmachen.

Zusätzliche Daten im ListControl

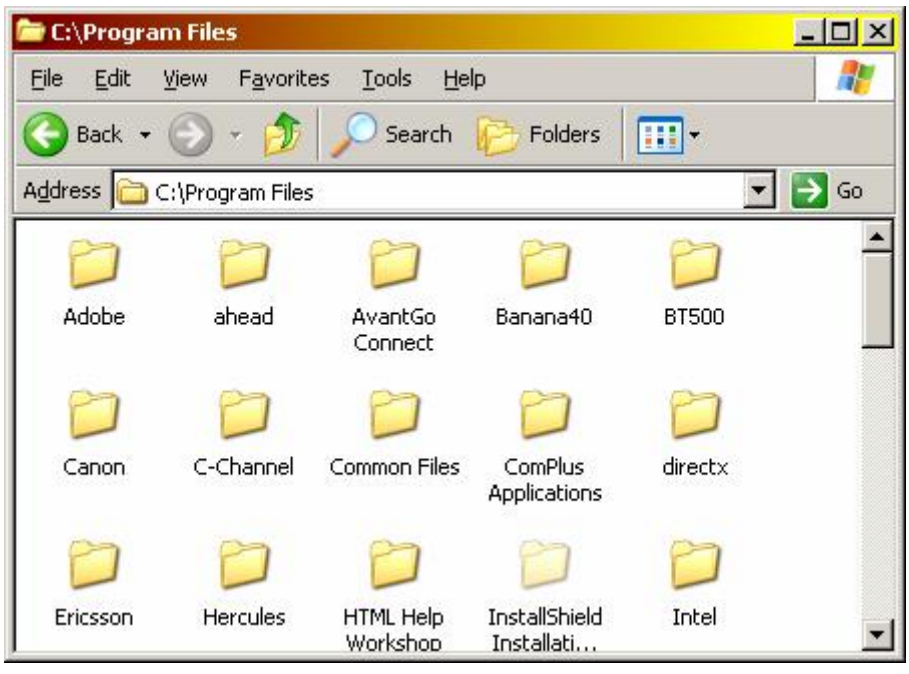
Die rechte Seite in unserer Applikation wird im Moment von einem Fenster mit ListControl belegt. Es wird höchste Zeit auch darin nützliche Daten anzuzeigen.

Das ListControl

Ein ListControl hat verschiedene Anzeigemodi, die man sehr gut im Windows Explorer sieht:

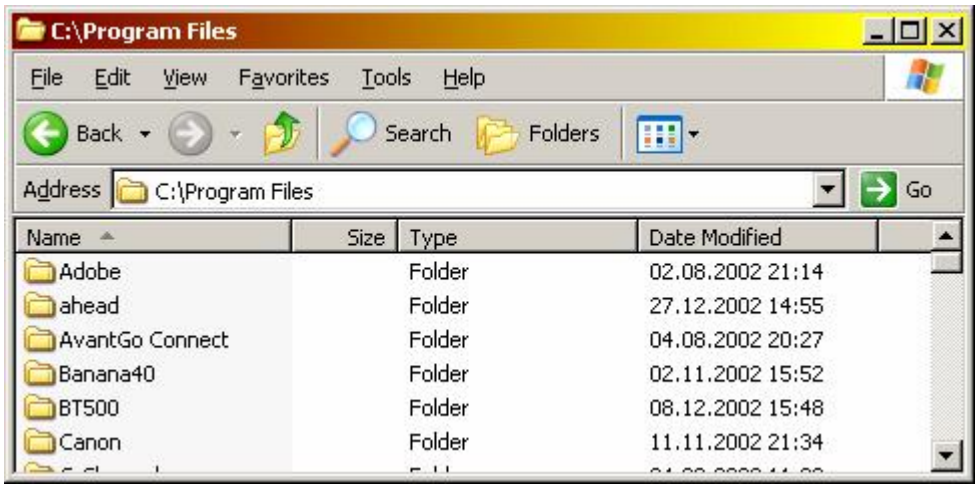


Ansicht „Grosse Symbole“



Ansicht „Kleine Symbole“

Ansicht „Liste“



Ansicht „Details“

Kolonnen für Detail-Ansicht

Wir können Kolonnen einfügen, indem wir bei einem CListCtrl InsertColumn aufrufen. Durch eine kleine Änderung in der Klasse CLeftView können wir das aktuelle Fach auf einen leeren String setzen, was bedeutet, dass kein Fach gewählt ist.

```
void CLeftView::OnNMClick(NMHDR *pNMHDR, LRESULT *pResult)
{
    // Hier die Position
    // des Mauscursors bestimmen
    CPoint point;
    GetCursorPos(&point);

    // Die Koordianten müssen
    // in Korrdinaten innerhalb des
    // Fensters umgerechnet werden
    CPoint clientPoint(point);
    ScreenToClient(&clientPoint);


    CTreeCtrl& treeCtrl = GetTreeCtrl();
    HTREEITEM root = treeCtrl.GetRootItem();
    // geckl icktes Item holen
    HTREEITEM hit = treeCtrl.HitTest(clientPoint);
    // Uebergeordnetes Item holen, es
    // könnte das Root Item sein
    HTREEITEM parent = treeCtrl.GetParentItem(hit);

    CNotenverwaltungDoc* document = GetDocument();
    if(parent == root)
    {
        CString fach = treeCtrl.GetItemText(hit);
        document->setAktuellesFach(fach);
    }
    else if(hit == root) // es ist das Root Item
    {
        CString empty;
        document->setAktuellesFach(empty);
    }

    *pResult = 0;
}
```

In der CNotenVerwaltungView können wir in der OnUpdate Methode je nach gewähltem Item verschiedene Kolonnen einfügen. Auf der nächsten Seite kommt mein Vorschlag.

```
void CNotenverwaltungView::OnUpdate(CView* /*pSender*/, LPARAM
/*lHint*/, CObject* /*pHint*/)
{
    CListCtrl& listCtrl = GetListCtrl();
    listCtrl.DeleteAllItems();
    // auch noch alle Kolonnen löschen
    // Ein ListCtrl enthält ein
    // HeaderCtrl!
    CHeaderCtrl* headerCtrl = listCtrl.GetHeaderCtrl();
    if(headerCtrl)
    {
        int columnCount = headerCtrl->GetItemCount();
        for(int columnIndex = 0;
            columnIndex < columnCount;
            ++columnIndex)
        {
            listCtrl.DeleteColumn(0);
        }
    }

    CNotenverwaltungDoc* document = GetDocument();
    CString aktuellesFach = document->getAktuellesFach();

    // falls das Aktuelle Fach
    // leer ist, stellen wir alle
    // Fächer dar
    if(aktuellesFach.IsEmpty())
    {
        // Die Kolonnen einfügen,
        // drittes Argument : Linksbündig, Mittig, etc.
        // viertes Argument : Breite in Pixel
        // fünftes Argument : Subitem index
        listCtrl.InsertColumn(0, "Fach", LVCFMT_LEFT, 60, 0);
        listCtrl.InsertColumn(1, "Lehrer", LVCFMT_LEFT, 60, 1);
        listCtrl.InsertColumn(2, "Durchschnitt",
                              LVCFMT_CENTER, 110, 2);
    }
    else
    {
        listCtrl.InsertColumn(0, aktuellesFach, LVCFMT_LEFT, 100);
    }
}
```

Aufgabe

In dieser Methode ist es nötig auf die einzelnen Fächer zuzugreifen und die Information wie den Namen, den Lehrer und die Durchschnittsnote abzufragen. Die Methoden, die hier verlangt sind fehlen uns noch. Ergänze was gebraucht wird, um unsere Ansicht zu vervollständigen.