Vorwort

Die Aufgabe wurde mehr oder weniger von mir dazu missbraucht, als Vorlage für den anstehenden Praktikumsbeleg zu dienen. Daher werde ich kurz die eigentliche Lösung vorstellen und danach noch näher auf die eigens entworfene polymorphe Mengenorganisation eingehen.

Lösung der Aufgabenstellung

Da es im Kern nur um die Menge geht, habe ich die einfachste meiner Klassen als zu speichernde Elemente gewählt: `CDate`. Leider kam ich bei Verwendung dieses Namens mit der MFC in Konflikt, die bereits eine Klasse `CDate` bereitstellt. Meine Klasse soll aber in der Lage sein, nicht nur ein Datum, sondern auch eine dazugehörige Uhrzeit speichern zu können. Mir fiel dazu im Deutschen der Name Zeitpunkt ein, den man ins Englische mit Moment übersetzt. Viele Grundzüge von `CMoment` ähneln `CTime`, ich wollte aber nur bedingt diese vorgefertige Klasse verwenden, um durch das „Nachprogrammieren“ selber weiter Erfahrungen im Umgang mit der Klassenbibliothek zu sammeln.

Die Serialisierung gestaltet sich mit der MFC sehr einfach. Nachdem sowohl die zu serialisierenden Elemente als auch die Menge selbst von `CObject` abgeleitet wurden, müssen beide die Makros `DECLARE_SERIAL` und `IMPLEMENT_SERIAL` verwenden und die Methode `Serialize` überschreiben.

```
// serialization
template<class C> void CPolymorphicSet<C>::Serialize(CArchive &ar)
{
    // do not forget to serialize CObject's data members
    CObject::Serialize(ar);

    // make use of CArchive's overloaded << and >> operators for *CObject

    if (ar.IsStoring())
    {
        // save the set

        // write cardinality
        ar << m_Set.GetCount();

        // write all elements
        POSITION pos = m_Set.GetHeadPosition();
        while (pos != NULL)
            ar << m_Set.GetNext(pos);
    }
    else
    {
        // load set

        // first, delete old set
        ScratchAll();

        int nCount;
        // load cardinality
        ar >> nCount;
        // load all elements
        for (int i=0; i<nCount; i++)
        {
            C* element;
            ar >> element;
            Insert(element);
            delete element;
        }
    }
}
```

Als wesentlichen Ansatzpunkt habe ich ausgenutzt, dass `CArchive` bereits die Operatoren << und >> für `CObject*` (als für einen Zeiger auf `CObject`, nicht für `CObject` selber !) überladen hat. Die in der Vorlesung gezeigte Vorgehensweise, die entweder `Serialize` direkt aufruft oder global die Operatoren für die genaue Klasse, z.B. `CMoment`, überlädt, kann damit umgangen werden, was mir viel Code spart.
Etwas problematisch ist dabei lediglich die Deserialisierung, also das Laden der Daten aus dem Archiv. Da ich wiederum nur mit `CObject*` hantieren kann, muss ich dafür Sorge tragen, dass das Objekt auch wieder korrekt gelöscht wird.

Eine polymorphe Menge mit MFC

Die MFC ist von Haus aus nicht besonders gut auf polymorphe Mengen ausgelegt. Zwar existieren sehr praktische Aufzählungsklassen, die auch typsicher sind, jedoch würde die Notwendigkeit der Polymorphie mehrere davon benötigen (eine pro Klasse), je nachdem, welche Objekte in der Menge gespeichert werden sollen.

Selbst wenn ich nun eine polymorphe Menge hätte, so ist es doch unangemessen, die Basisklasse der Elemente fest zu codieren. Der einzige Ausweg, der sich hier anbietet, ist die Verwendung von Templates. Die einzigen Bedingungen, die die Basisklasse erfüllen muss, ist, dass sie von `CObject` direkt oder indirekt abgeleitet ist und den Operatoren `==` überlädt. Letzteres ist notwendig, um Suchvorgänge zu ermöglichen.

Ich halte es für sinnvoll, wenn die Menge stets Kopien speichern und nicht nur Zeiger auf die Elemente, da sie so resistent gegen externes Löschen oder Verändern ist. Gerade die Freigabe des Speicherplatzes eines Elementes kann bei Verwendung einfacher Zeiger die komplette Menge zerstören. Ein dazu inverses Problem wäre, dass das Anwendungsprogramm ein Element einfügt und sämtliche Verweise darauf löscht. Dann existiert zwar noch der Zeiger in der Menge, mit dem Zerstören derselben geht aber auch diese Information verloren, was in einem Speicherleck resultiert.

Die Speicherung der Elemente erfolgt bei mir in einer echten MFC-Liste vom Typ `CList<*C, *C>`. Da dies aber nur Zeiger sind, muss das Template dafür sorgen, dass die Elemente selbst erzeugt bzw. kopiert werden. Der dafür zuständige Code findet sich in der als `protected` deklarierten Methode `Clone`. Unter Benutzung des MFC-eigenes Mechanismus‘ zur dynamischen Objekterzeugung wird ein neues Objekt der passenden Klasse erzeugt und der Zuweisungsoperator sorgt für die Initialisi erung der Attribute:

```
// creates a new instance and copies all attributes
template<class C> C* CPolymorphicSet<C>::Clone(const C* element) const
{
    // create new instance (care for correct class !)
    C* copy = (C*)element->GetRuntimeClass()->CreateObject();
    // copy attributes
    *copy = *element;

    return copy;
}
```

Das Einfügen eines neuen Elementes in die Menge sieht dann beispielsweise so aus:

```
// inserts an element
template<class C> int CPolymorphicSet<C>::Insert(const C* element)
{
    // don‘t insert an object twice
    if (Find(element))
        return -1;

    // add a copy to our set, change cursor
    m_Cursor = m_Set.AddTail(Clone(element));

    // return index in the set
    return m_Set.GetCount()-1;
}
```

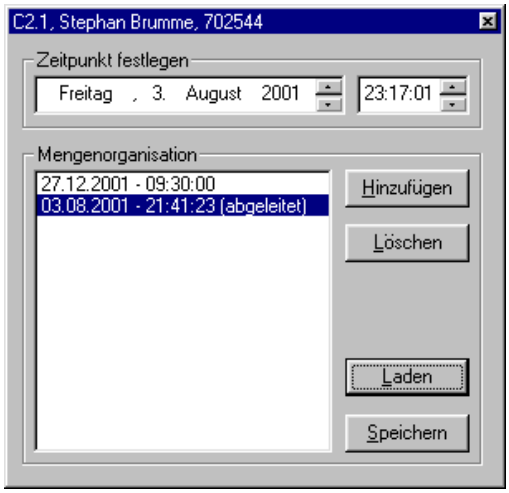
Um durchgängig den MFC-Stil zu verwenden, weiche ich bei einigen Methodennamen von den Originalen ab. Laut MFC wird die Klasseninvarianz in `AssertValid` (vorher `ClassInvariant`) geprüft und eine Debug-Ausgabe der Klassendaten kann mit `Dump` (ehemals `Show`) geschehen.

Außerdem verzichte ich aus Performancegründen auf die Benutzung von Exceptions. Stattdessen übernehmen ASSERT-Makros deren Aufgabe. Da sie nur in der Debug-Einstellung generiert werden, sind sie gerade bei häufigen Zugriffen auf die Menge in einer echten Umgebung nicht mehr bremsend und verbrauchen nicht unnötig Resourcen.

Beispielprogramm

Zur Überprüfung meiner Klassen schrieb ich ein kleines Programm, das Zeitpunkte verwalten kann. Dazu gehört deren Erstellung, Änderung, Löschung und Serialisierung. Um den Code klein zu halten, habe ich mich für eine einfache dialog-basierte Applikation entschieden.

Abbildung 1



Als Kernfunktion kann man über im oberen Fensterbereich einen Zeitpunkt einstellen. Mittels *Hinzufügen* wird sie Bestandteil der Menge. Da hierzu aber keine Polymorphie notwendig ist, musste ich mir einen kleinen Trick einfallen lassen: per Zufall wird nicht die Klasse `CMoment`, sondern die von ihr abgeleitete Klasse `CMomentDerived` benutzt. Ihre Deklaration ist recht einfach gehalten:

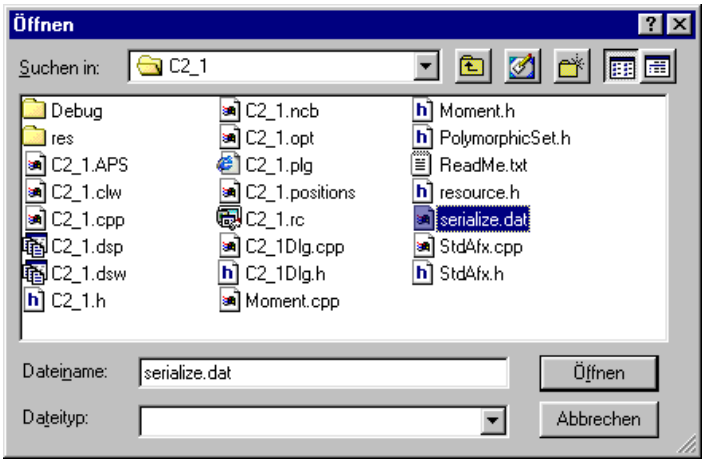
```
// a derived class, just used to show polymorphism
// no new attributes and/or operations are introduced
class CMomentDerived : public CMoment
{
    DECLARE_SERIAL(CMomentDerived)

public:
    CMomentDerived() {};
    ~CMomentDerived() {};
};
IMPLEMENT_SERIAL(CMomentDerived, CMoment, 1)
```

In der Listbox in der obigen Abbildung kann man erkennen, dass das zweite Element von `CMomentDerived` abstammt, weil es den Zusatz *abgeleitet* trägt.

Die Serialisierung greift auf die Standard-Windows-Dialog zurück um eine beliebige Datei zu lesen bzw. zu schreiben. Aus Gründen der Übersichtlichkeit habe ich darauf verzichtet, den Dateityp genau zu überprüfen, beim Lesen kann man also auch ungültige Dateien übergeben.

Abbildung 2


Öffnen
Suchen in:
C2_1
Debug
res
C2_1.APS
C2_1.clw
C2_1.cpp
C2_1.dsp
C2_1.dsw
C2_1.h
C2_1.ncb
C2_1.opt
C2_1.plg
C2_1.positions
C2_1.rc
C2_1Dlg.cpp
C2_1Dlg.h
Moment.cpp
Moment.h
PolymorphicSet.h
ReadMe.txt
resource.h
serialize.dat
StdAfx.cpp
StdAfx.h
Dateiname:
serialize.dat
Öffnen
Dateityp:
Abbrechen

Quellcode

CPolymorphicSet.h

```
//////////////////////////////////////////////////////
// Softwarebauelemente II, C2.1
//
// A template for polymorphic set - designed in MFC-style
// The only parameter is the base class of all elements
// (at least CObject)
// All elements are copied by the interface methods
// to avoid unexpected behaviour (e.g. external erase)
//
// author:          Stephan Brumme
// last changes:    August 3, 2001

#if !defined(AFX_POLYMORPHICSET_H__BA324DAE_872B_11D5_9BB8_AB7BB57BD00C__INCLUDED_)
#define AFX_POLYMORPHICSET_H__BA324DAE_872B_11D5_9BB8_AB7BB57BD00C__INCLUDED_


#include "stdafx.h"


#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


template<class C> class CPolymorphicSet : public CObject
{
public:
    // default constructor
    CPolymorphicSet();
    // copy constructor
    CPolymorphicSet(const CPolymorphicSet<C>& set);

    // destructor
    virtual ~CPolymorphicSet();

    // serialization
    virtual void Serialize(CArchive &ar);
    // invariance
    virtual void AssertValid() const;
    // dump
    virtual void Dump(CDumpContext& dc) const;

    // copy the whole set
    virtual CPolymorphicSet<C>& operator=(const CPolymorphicSet<C>& set);
    virtual CPolymorphicSet<C>& Copy     (const CPolymorphicSet<C>& set);
    // compare two sets
    // value identity
    virtual bool operator==(const CPolymorphicSet<C>& set) const;
    virtual bool EqualValue(const CPolymorphicSet<C>& set) const;
    // object identity
    virtual bool Equal     (const CPolymorphicSet<C>* set) const;

    // returns number of stored elements
    int Card() const;

    // first stored element
    C* GetFirst();
    // next element
    C* GetNext();
    // gets the element the cursor points to
    C* GetCurrent() const;

    // inserts an element
    int  Insert(const C* element);
    // deletes current element
    void Scratch();
    // finds an element
    bool Find(const C* element);
```

```
protected:
// creates a new instance and copies all attributes
    C* Clone(const C* element) const;

    // deletes the whole set
    void ScratchAll();

    // stores all elements in a list
    CList<C*, C*> m_Set;
    // cursor, may be altered at any time
    POSITION m_Cursor;

};


//////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////


// default constructor
template<class C> CPolymorphicSet<C>::CPolymorphicSet()
{
    // empty set
    m_Cursor = NULL;
}


// copy constructor
template<class C> CPolymorphicSet<C>::CPolymorphicSet(const CPolymorphicSet<C>& set)
{
    operator=(set);
    m_Cursor = m_Set.GetHeadPosition();
}


// destructor
template<class C> CPolymorphicSet<C>::~CPolymorphicSet()
{
    ScratchAll();
}


// serialization
template<class C> void CPolymorphicSet<C>::Serialize(CArchive &ar)
{
    // do not forget to serialize CObject's data members
    CObject::Serialize(ar);

    // make use of CArchive's overloaded << and >> operators for *CObject

    if (ar.IsStoring())
    {
        // save the set

        // write cardinality
        ar << m_Set.GetCount();

        // write all elements
        POSITION pos = m_Set.GetHeadPosition();
        while (pos != NULL)
            ar << m_Set.GetNext(pos);
    }
    else
    {
        // load set

        // first, delete old set
        ScratchAll();

        int nCount;
        // load cardinality
        ar >> nCount;
        // load all elements
```

```
        for (int i=0; i<nCount; i++)
        {
            C* element;
            ar >> element;
            Insert(element);
            delete element;
        }
    }
}


// invariance
template<class C> void CPolymorphicSet<C>::AssertValid() const
{
    CObject::AssertValid();

    // if the set is empty then cursor must be NULL
    if (m_Set.IsEmpty())
    {
        ASSERT(m_Cursor == NULL);
        return;
    }

    // each element may be present in set at most once
    // cursor must point to an element
    POSITION pos = m_Set.GetHeadPosition();
    bool bValidCursor = false;
    while (pos != NULL)
    {
        // element that the cursor points to was found ?
        if (pos == m_Cursor)
            bValidCursor = true;

        // get element
        C* element = m_Set.GetNext(pos);
        POSITION posfind = pos;
        // compare to the remaining elements
        while (posfind != NULL)
        {
            C* compare = m_Set.GetNext(posfind);
            // no two equal elements are allowed
            ASSERT(!(*element == *compare));
        }
    }

    // cursor validated ?
    ASSERT(bValidCursor);
}


// dump
template<class C> void CPolymorphicSet<C>::Dump(CDumpContext& dc) const
{
    CObject::Dump(dc);

    dc << Card() << " elements, cursor points to " << m_Cursor << "\n";

    POSITION pos = m_Set.GetHeadPosition();
    while (pos != NULL)
        dc << m_Set.GetNext(pos) << "\n";
}


// copy the whole set
template<class C> CPolymorphicSet<C>& CPolymorphicSet<C>::operator=(const CPolymorphicSet<C>&
set)
{
    // delete old elements
    m_Set.RemoveAll();

    POSITION pos = set.m_Set.GetHeadPosition();
    while (pos != NULL)
    {
        // save current position
        POSITION cursor = pos;
```

```
        // clone each element
        C* element = set.m_Set.GetNext(pos);
        C* copy    = Clone(element);
        m_Set.AddTail(copy);

        // set cursor
        // we can't use set.m_Cursor directly because we copied all elements
        // and therefore lost their old memory address
        if (cursor == set.m_Cursor)
            m_Cursor = m_Set.GetTailPosition();
    }

    return *this;
}


// see operator=
template<class C> CPolymorphicSet<C>& CPolymorphicSet<C>::Copy(const CPolymorphicSet<C>& set)
{
    return operator=(set);
}


// compare two sets
template<class C> bool CPolymorphicSet<C>::operator==(const CPolymorphicSet<C>& set) const
{
    // the cursors may differ !

    // same number of elements is required
    if (m_Set.GetCount() != set.m_Set.GetCount())
        return false;

    // each element of this set must be found in the other set
    POSITION pos = m_Set.GetHeadPosition();
    while (pos != NULL)
    {
        C* element = m_Set.GetNext(pos);
        if (!set.m_Set.Find(element))
            return false;
    }

    // all elements were found, sets must be equal
    return true;
}


// see operator==
template<class C> bool CPolymorphicSet<C>::EqualValue(const CPolymorphicSet<C>& set) const
{
    return operator==(set);
}


// object identity
template<class C> bool CPolymorphicSet<C>::Equal(const CPolymorphicSet<C>* set) const
{
    return (set == this);
}


// returns number of stored elements
template<class C> int CPolymorphicSet<C>::Card() const
{
    return m_Set.GetCount();
}


// first stored element
template<class C> C* CPolymorphicSet<C>::GetFirst()
{
    // set must not be empty
    ASSERT(!m_Set.IsEmpty());

    // set cursor
```

```
    m_Cursor = m_Set.GetHeadPosition();

    // clone first element
    return GetCurrent();
}


// next element
template<class C> C* CPolymorphicSet<C>::GetNext()
{
    // set must not be empty
    ASSERT(!m_Set.IsEmpty());
    // cursor must not be at the end of the set
    ASSERT(m_Cursor != NULL);

    // set cursor
    m_Set.GetNext(m_Cursor);

    // clone element
    return GetCurrent();
}


// gets the element the cursor points to
template<class C> C* CPolymorphicSet<C>::GetCurrent() const
{
    // cursor must not be at the end of the set
    ASSERT(m_Cursor != NULL);

    // generate copy
    C* copy = Clone(m_Set.GetAt(m_Cursor));

    return copy;
}


// inserts an element
template<class C> int CPolymorphicSet<C>::Insert(const C* element)
{
    // don't insert an object twice
    if (Find(element))
        return -1;

    // add a copy to our set, change cursor
    m_Cursor = m_Set.AddTail(Clone(element));

    // return index in the set
    return m_Set.GetCount()-1;
}


// deletes current element
template<class C> void CPolymorphicSet<C>::Scratch()
{
    // cursor must not be at the end of the set
    ASSERT(m_Cursor != NULL);

    // store current cursor
    POSITION removepos = m_Cursor;
    // go on to the next element (else we lose it after erasing this element)
    m_Set.GetNext(m_Cursor);

    // remove element the cursor pointed to previously
    delete m_Set.GetAt(removepos);
    m_Set.RemoveAt(removepos);
}


// looks for an element
template<class C> bool CPolymorphicSet<C>::Find(const C* element)
{
    // only non-empty sets
    if (m_Set.IsEmpty())
        return false;
```

```
    // iterate through the whole list
    POSITION pos = m_Set.GetHeadPosition();
    while (pos != NULL)
    {
        // store position in case we find a match
        POSITION storepos = pos;

        // get an element we compare to
        C* compare  = m_Set.GetNext(pos);
        // equal ?
        if (*element == *compare)
        {
            // yes, set cursor and return "success !"
            m_Cursor = storepos;
            return true;
        }
    }

    // element is not part of the set
    return false;
}


// creates a new instance and copies all attributes
template<class C> C* CPolymorphicSet<C>::Clone(const C* element) const
{
    // create new instance (care for correct class !)
    C* copy = (C*)element->GetRuntimeClass()->CreateObject();
    // copy attributes
    *copy = *element;

    return copy;
}


// deletes the whole set
template<class C> void CPolymorphicSet<C>::ScratchAll()
{
    while (!m_Set.IsEmpty())
    {
        delete m_Set.GetHead();
        m_Set.RemoveHead();
    }
}


#endif // !defined(AFX_POLYMORPHICSET_H__BA324DAE_872B_11D5_9BB8_AB7BB57BD00C__INCLUDED_)
```

CMoment.h

```
//////////////////////////////////////////////////////
// Softwarebauelemente II, C2.1
//
// author:          Stephan Brumme
// last changes:    August 3, 2001

#if !defined(AFX_MOMENT_H__BA324DAD_872B_11D5_9BB8_AB7BB57BD00C__INCLUDED_)
#define AFX_MOMENT_H__BA324DAD_872B_11D5_9BB8_AB7BB57BD00C__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CMoment : public CObject
{
    DECLARE_SERIAL(CMoment)

public:
    // constructor, default value is current date and time
    CMoment();
    // copy constructor
    CMoment(const CMoment& moment);
    // set user defined date at construction time
    CMoment(unsigned int nDay,    unsigned int nMonth,    unsigned int nYear,
            unsigned int nHour=0, unsigned int nMinute=0, unsigned int nSecond=0);

    // destructor
    virtual ~CMoment();

    // serialization
    void Serialize(CArchive &ar);
    // validate a moment
    virtual void AssertValid() const;
    // dump
    virtual void Dump(CDumpContext& dc) const;

    // return current date and time
    static void GetCurrent(unsigned int& nDay,  unsigned int& nMonth,  unsigned int& nYear,
                           unsigned int& nHour, unsigned int& nMinute, unsigned int& nSecond);

    // determine whether it is a leap year
    static bool IsLeapYear(unsigned int nYear);

    // copy
    virtual CMoment& operator = (const CMoment& moment);
    virtual CMoment& Copy       (const CMoment& moment);
    // to enhance compatibility to CTime
    virtual CMoment& operator = (const CTime& time);

    // compare two dates
    virtual bool     operator ==(const CMoment& moment) const;
    virtual bool     EqualValue (const CMoment& moment) const;

    // convert to MFC's CTime
    virtual operator CTime() const;

    // set attributes, returns validility of date
    void SetDay   (unsigned int nDay);
    void SetMonth (unsigned int nMonth);
    void SetYear  (unsigned int nYear);
    void SetHour  (unsigned int nHour);
    void SetMinute(unsigned int nMinute);
    void SetSecond(unsigned int nSecond);

    // return attributes
    unsigned int GetDay   () const;
    unsigned int GetMonth () const;
    unsigned int GetYear  () const;
    unsigned int GetHour  () const;
    unsigned int GetMinute() const;
    unsigned int GetSecond() const;

    // constants for month's names
```

```
    enum { JANUARY   =  1, FEBRUARY =  2, MARCH    =  3, APRIL    =  4,
           MAY       =  5, JUNE     =  6, JULY     =  7, AUGUST   =  8,
           SEPTEMBER =  9, OCTOBER  = 10, NOVEMBER = 11, DECEMBER = 12 };

private:
    // attributes
    unsigned int m_nDay;
    unsigned int m_nMonth;
    unsigned int m_nYear;

    unsigned int m_nHour;
    unsigned int m_nMinute;
    unsigned int m_nSecond;
};

#endif // !defined(AFX_MOMENT_H__BA324DAD_872B_11D5_9BB8_AB7BB57BD00C__INCLUDED_)
```

CMoment.cpp

```
/////////////////////////////////////////////////////////
// Softwarebauelemente II, C2.1
//
// author:          Stephan Brumme
// last changes:    August 2, 2001


#include "stdafx.h"
#include "Moment.h"
// we are using OS-specific date/time operations
#include <time.h>

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif


// serializable
IMPLEMENT_SERIAL(CMoment, CObject, 1)



// constructor, default value is current date and time
CMoment::CMoment()
{
    GetCurrent(m_nDay, m_nMonth, m_nYear, m_nHour, m_nMinute, m_nSecond);
}


// copy constructor
CMoment::CMoment(const CMoment& moment)
{
    operator=(moment);
}


// set user defined date/time at construction time
CMoment::CMoment(unsigned int nDay,  unsigned int nMonth,  unsigned int nYear,
                 unsigned int nHour, unsigned int nMinute, unsigned int nSecond)
{
    // store date
    m_nDay    = nDay;
    m_nMonth  = nMonth;
    m_nYear   = nYear;

    // store time
    m_nHour   = nHour;
    m_nMinute = nMinute;
    m_nSecond = nSecond;
}


// destructor, nothing left to do ...
CMoment::~CMoment()
{
}


// serialization
void CMoment::Serialize(CArchive &ar)
{
    // do not forget to serialize CObject's data members
    CObject::Serialize(ar);

    if (ar.IsStoring())
        ar << m_nYear << m_nMonth << m_nDay << m_nHour << m_nMinute << m_nSecond;
    else
        ar >> m_nYear >> m_nMonth >> m_nDay >> m_nHour >> m_nMinute >> m_nSecond;
}
```

```
// validate a moment
void CMoment::AssertValid() const
{
    // validate month
    ASSERT(m_nMonth >= JANUARY && m_nMonth <= DECEMBER);

    // days per month, february may vary, note that array starts with 0 (JANUARY=1 !)
    unsigned int nDaysPerMonth[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    // adjust days of february
    if (IsLeapYear(m_nYear))
        nDaysPerMonth[FEBRUARY]++;

    // validate day
    ASSERT(m_nDay > 0 && m_nDay <= nDaysPerMonth[m_nMonth]);
    // day and month are valid

    // now check the time
    ASSERT(m_nHour   >= 0 && m_nHour   <= 23);
    ASSERT(m_nMinute >= 0 && m_nMinute <= 59);
    ASSERT(m_nSecond >= 0 && m_nSecond <= 59);
}


// dump
void CMoment::Dump(CDumpContext& dc) const
{
    CObject::Dump(dc);

    CString strOutput;
    strOutput.Format("date: %02d.%02d.%04d\ntime: %02d:%02d:%02d",
                     m_nDay, m_nMonth, m_nYear, m_nHour, m_nMinute, m_nSecond);
    dc << strOutput;
}


// return current moment
void CMoment::GetCurrent(unsigned int& nDay,  unsigned int& nMonth,  unsigned int& nYear,
                         unsigned int& nHour, unsigned int& nMinute, unsigned int& nSecond)
{
    // the following code is basically taken from MSDN

    // use system functions to get the current date as UTC
    time_t secondsSince1970;
    time(&secondsSince1970);

    // convert UTC to local time zone
    struct tm *localTime;
    localTime = localtime(&secondsSince1970);

    // store retrieved date
    nDay    = localTime->tm_mday;
    nMonth  = localTime->tm_mon + 1;
    nYear   = localTime->tm_year + 1900;
    // store time
    nHour   = localTime->tm_hour;
    nMinute = localTime->tm_min;
    nSecond = localTime->tm_sec;
}


// determine whether it is a leap year
bool CMoment::IsLeapYear(unsigned int nYear)
{
    // used to speed up code, may be deleted
    if (nYear % 4   != 0)
        return false;

    // algorithm taken from MSDN, just converted from VBA to C++
    if (nYear % 400 == 0)
        return true;
    if (nYear % 100 == 0)
        return false;
    if (nYear % 4   == 0)
        return true;
```

```
    // this line won't be executed because of optimization (see above)
    return false;
}


// copy
CMoment& CMoment::operator =(const CMoment& moment)
{
    // date
    m_nDay    = moment.m_nDay;
    m_nMonth  = moment.m_nMonth;
    m_nYear   = moment.m_nYear;
    // time
    m_nHour   = moment.m_nHour;
    m_nMinute = moment.m_nMinute;
    m_nSecond = moment.m_nSecond;

    return *this;
}


// see operator=
CMoment& CMoment::Copy(const CMoment& moment)
{
    return operator=(moment);
}


// accept CTime, too
CMoment& CMoment::operator =(const CTime& time)
{
    // date
    m_nDay    = time.GetDay();
    m_nMonth  = time.GetMonth();
    m_nYear   = time.GetYear();
    // time
    m_nHour   = time.GetHour();
    m_nMinute = time.GetMinute();
    m_nSecond = time.GetSecond();

    return *this;
}


// compare two moments
bool CMoment::operator ==(const CMoment &moment) const
{
    // compare moment and time attributes
    return (m_nDay    == moment.m_nDay    &&
            m_nMonth  == moment.m_nMonth  &&
            m_nYear   == moment.m_nYear   &&
            m_nHour   == moment.m_nHour   &&
            m_nMinute == moment.m_nMinute &&
            m_nSecond == moment.m_nSecond);
}


// see operator==
bool CMoment::EqualValue(const CMoment &moment) const
{
    return operator==(moment);
}


// convert to MFC's CTime
CMoment::operator CTime() const
{
    return CTime(m_nYear, m_nMonth, m_nDay, m_nHour, m_nMinute, m_nSecond);
}


// set attributes, returns validility of moment
void CMoment::SetDay(unsigned int nDay)
{
```

```
    m_nDay = nDay;
}
void CMoment::SetMonth(unsigned int nMonth)
{
    m_nMonth = nMonth;
}
void CMoment::SetYear(unsigned int nYear)
{
    m_nYear = nYear;
}
void CMoment::SetHour(unsigned int nHour)
{
    m_nHour = nHour;
}
void CMoment::SetMinute(unsigned int nMinute)
{
    m_nMinute = nMinute;
}
void CMoment::SetSecond(unsigned int nSecond)
{
    m_nSecond = nSecond;
}

// return attributes
unsigned int CMoment::GetDay() const
{
    return m_nDay;
}
unsigned int CMoment::GetMonth() const
{
    return m_nMonth;
}
unsigned int CMoment::GetYear() const
{
    return m_nYear;
}
unsigned int CMoment::GetHour() const
{
    return m_nHour;
}
unsigned int CMoment::GetMinute() const
{
    return m_nMinute;
}
unsigned int CMoment::GetSecond() const
{
    return m_nSecond;
}
```

StdAfx.h

```
// stdafx.h : Include-Datei für Standard-System-Include-Dateien,
//  oder projektspezifische Include-Dateien, die häufig benutzt, aber
//      in unregelmäßigen Abständen geändert werden.
//

#if !defined(AFX_STDAFX_H__BA324DA7_872B_11D5_9BB8_AB7BB57BD00C__INCLUDED_)
#define AFX_STDAFX_H__BA324DA7_872B_11D5_9BB8_AB7BB57BD00C__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define VC_EXTRALEAN        // Selten verwendete Teile der Windows-Header nicht einbinden

#include <afxwin.h>         // MFC-Kern- und -Standardkomponenten
#include <afxext.h>         // MFC-Erweiterungen
#include <afxdtctl.h>       // MFC-Unterstützung für allgemeine Steuerelemente von Internet
Explorer 4
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>         // MFC-Unterstützung für gängige Windows-Steuerelemente
#endif // _AFX_NO_AFXCMN_SUPPORT

#include <afxtempl.h>       // MFC-Vorlagenklassen
#include <afxcoll.h>

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ fügt unmittelbar vor der vorhergehenden Zeile zusätzliche
Deklarationen ein.

#endif // !defined(AFX_STDAFX_H__BA324DA7_872B_11D5_9BB8_AB7BB57BD00C__INCLUDED_)
```

StdAfx.cpp

```
// stdafx.cpp : Quelltextdatei, die nur die Standard-Includes einbindet
//  C2_1.pch ist die vorcompilierte Header-Datei
//  stdafx.obj enthält die vorcompilierte Typinformation

#include "stdafx.h"
```

resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by C2_1.rc
//
#define IDD_C2_1_DIALOG                 102
#define IDR_MAINFRAME                   128
#define IDC_LIST                        1000
#define IDC_DATEPICKER                  1001
#define IDC_BUTTON_LOAD                 1009
#define IDC_BUTTON_SAVE                 1010
#define IDC_TIMEPICKER                  1011
#define IDC_BUTTON_DELETE               1012

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        129
#define _APS_NEXT_COMMAND_VALUE         40001
#define _APS_NEXT_CONTROL_VALUE         1013
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

C2_1.h

```
////////////////////////////////////////////////////////
// Softwarebauelemente II, C2.1
//
// author:          Stephan Brumme
// last changes:    August 2, 2001


#if !defined(AFX_C2_1_H__BA324DA3_872B_11D5_9BB8_AB7BB57BD00C__INCLUDED_)
#define AFX_C2_1_H__BA324DA3_872B_11D5_9BB8_AB7BB57BD00C__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // Hauptsymbole

/////////////////////////////////////////////////////////////////////////////
// CC2_1App:
// Siehe C2_1.cpp für die Implementierung dieser Klasse
//

class CC2_1App : public CWinApp
{
public:
    CC2_1App();

// Überladungen
    // Vom Klassenassistenten generierte Überladungen virtueller Funktionen
    //{{AFX_VIRTUAL(CC2_1App)
    public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

// Implementierung

    //{{AFX_MSG(CC2_1App)
        // HINWEIS - An dieser Stelle werden Member-Funktionen vom Klassen-Assistenten
eingefügt und entfernt.
        //    Innerhalb dieser generierten Quelltextabschnitte NICHTS VERÄNDERN!
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ fügt unmittelbar vor der vorhergehenden Zeile zusätzliche
Deklarationen ein.

#endif // !defined(AFX_C2_1_H__BA324DA3_872B_11D5_9BB8_AB7BB57BD00C__INCLUDED_)
```

C2_1.cpp

```
/////////////////////////////////////////////////////////
// Softwarebauelemente II, C2.1
//
// author:          Stephan Brumme
// last changes:    August 2, 2001


#include "stdafx.h"
#include "C2_1.h"
#include "C2_1Dlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CC2_1App

BEGIN_MESSAGE_MAP(CC2_1App, CWinApp)
    //{{AFX_MSG_MAP(CC2_1App)
        // HINWEIS - Hier werden Mapping-Makros vom Klassen-Assistenten eingefügt und
entfernt.
        //    Innerhalb dieser generierten Quelltextabschnitte NICHTS VERÄNDERN!
    //}}AFX_MSG
    ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CC2_1App Konstruktion

CC2_1App::CC2_1App()
{
    // ZU ERLEDIGEN: Hier Code zur Konstruktion einfügen
    // Alle wichtigen Initialisierungen in InitInstance platzieren
}

/////////////////////////////////////////////////////////////////////////////
// Das einzige CC2_1App-Objekt

CC2_1App theApp;

/////////////////////////////////////////////////////////////////////////////
// CC2_1App Initialisierung

BOOL CC2_1App::InitInstance()
{
    // Standardinitialisierung
    // Wenn Sie diese Funktionen nicht nutzen und die Größe Ihrer fertigen
    //  ausführbaren Datei reduzieren wollen, sollten Sie die nachfolgenden
    //  spezifischen Initialisierungsroutinen, die Sie nicht benötigen, entfernen.

#ifdef _AFXDLL
    Enable3dControls();         // Diese Funktion bei Verwendung von MFC in gemeinsam
genutzten DLLs aufrufen
#else
    Enable3dControlsStatic();   // Diese Funktion bei statischen MFC-Anbindungen aufrufen
#endif

    CC2_1Dlg dlg;
    m_pMainWnd = &dlg;
    int nResponse = dlg.DoModal();
    if (nResponse == IDOK)
    {
        // ZU ERLEDIGEN: Fügen Sie hier Code ein, um ein Schließen des
        //  Dialogfelds über OK zu steuern
    }
    else if (nResponse == IDCANCEL)
    {
        // ZU ERLEDIGEN: Fügen Sie hier Code ein, um ein Schließen des
        //  Dialogfelds über "Abbrechen" zu steuern
    }
```

```
    // Da das Dialogfeld geschlossen wurde, FALSE zurückliefern, so dass wir die
    //  Anwendung verlassen, anstatt das Nachrichtensystem der Anwendung zu starten.
    return FALSE;
}
```

C2_1Dlg.h

```
//////////////////////////////////////////////////////
// Softwarebauelemente II, C2.1
//
// author:          Stephan Brumme
// last changes:    August 2, 2001


#if !defined(AFX_C2_1DLG_H__BA324DA5_872B_11D5_9BB8_AB7BB57BD00C__INCLUDED_)
#define AFX_C2_1DLG_H__BA324DA5_872B_11D5_9BB8_AB7BB57BD00C__INCLUDED_

#include "Moment.h" // Hinzugefügt von der Klassenansicht
#include "PolymorphicSet.h"

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

/////////////////////////////////////////////////////////////////////////////
// CC2_1Dlg Dialogfeld

class CC2_1Dlg : public CDialog
{
// Konstruktion
public:
    CC2_1Dlg(CWnd* pParent = NULL); // Standard-Konstruktor

// Dialogfelddaten
    //{{AFX_DATA(CC2_1Dlg)
    enum { IDD = IDD_C2_1_DIALOG };
    CListBox    m_List;
    CTime   m_Date;
    CTime   m_Time;
    //}}AFX_DATA

    // Vom Klassenassistenten generierte Überladungen virtueller Funktionen
    //{{AFX_VIRTUAL(CC2_1Dlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV-Unterstützung
    //}}AFX_VIRTUAL

// Implementierung
protected:
    HICON m_hIcon;

    // Generierte Message-Map-Funktionen
    //{{AFX_MSG(CC2_1Dlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    virtual void OnOK();
    afx_msg void OnDelete();
    afx_msg void OnLoad();
    afx_msg void OnSave();
    afx_msg void OnDblclkList();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()

private:
    void UpdateListBox();

    CMoment m_Moment;
    CPolymorphicSet<CMoment> m_Set,
                             m_CopiedSet;
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ fügt unmittelbar vor der vorhergehenden Zeile zusätzliche
Deklarationen ein.

#endif // !defined(AFX_C2_1DLG_H__BA324DA5_872B_11D5_9BB8_AB7BB57BD00C__INCLUDED_)
```

C2_1Dlg.cpp

```
////////////////////////////////////////////////////////
// Softwarebauelemente II, C2.1
//
// author:          Stephan Brumme
// last changes:    August 2, 2001


#include "stdafx.h"
#include "C2_1.h"
#include "C2_1Dlg.h"
#include "PolymorphicSet.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CC2_1Dlg Dialogfeld

CC2_1Dlg::CC2_1Dlg(CWnd* pParent /*=NULL*/)
    : CDialog(CC2_1Dlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CC2_1Dlg)
    m_Date = 0;
    m_Time = 0;
    //}}AFX_DATA_INIT
    // Beachten Sie, dass LoadIcon unter Win32 keinen nachfolgenden DestroyIcon-Aufruf
benötigt
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CC2_1Dlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CC2_1Dlg)
    DDX_Control(pDX, IDC_LIST, m_List);
    DDX_DateTimeCtrl(pDX, IDC_DATEPICKER, m_Date);
    DDX_DateTimeCtrl(pDX, IDC_TIMEPICKER, m_Time);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CC2_1Dlg, CDialog)
    //{{AFX_MSG_MAP(CC2_1Dlg)
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_BUTTON_DELETE, OnDelete)
    ON_BN_CLICKED(IDC_BUTTON_LOAD, OnLoad)
    ON_BN_CLICKED(IDC_BUTTON_SAVE, OnSave)
    ON_LBN_DBLCLK(IDC_LIST, OnDblclkList)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CC2_1Dlg Nachrichten-Handler

BOOL CC2_1Dlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Symbol für dieses Dialogfeld festlegen. Wird automatisch erledigt
    //  wenn das Hauptfenster der Anwendung kein Dialogfeld ist
    SetIcon(m_hIcon, TRUE);         // Großes Symbol verwenden
    SetIcon(m_hIcon, FALSE);        // Kleines Symbol verwenden

    // ZU ERLEDIGEN: Hier zusätzliche Initialisierung einfügen

    // initialize random generator
    srand((unsigned)time(NULL));

    // set timer to refresh m_Time
    SetTimer(1, 1000, NULL);
```

```
    m_Date = m_Time = m_Moment;
    UpdateData(FALSE);

    return TRUE;  // Geben Sie TRUE zurück, außer ein Steuerelement soll den Fokus erhalten
}

// Wollen Sie Ihrem Dialogfeld eine Schaltfläche "Minimieren" hinzufügen, benötigen Sie
//  den nachstehenden Code, um das Symbol zu zeichnen. Für MFC-Anwendungen, die das
//  Dokument/Ansicht-Modell verwenden, wird dies automatisch für Sie erledigt.

void CC2_1Dlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // Gerätekontext für Zeichnen

        SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

        // Symbol in Client-Rechteck zentrieren
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Symbol zeichnen
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }
}

// Die Systemaufrufe fragen den Cursorform ab, die angezeigt werden soll, während der Benutzer
//  das zum Symbol verkleinerte Fenster mit der Maus zieht.
HCURSOR CC2_1Dlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}


// a derived class, just used to show polymorphism
// no new attributes and/or operations are introduced
class CMomentDerived : public CMoment
{
    DECLARE_SERIAL(CMomentDerived)

public:
    CMomentDerived() {};
    ~CMomentDerived() {};
};
IMPLEMENT_SERIAL(CMomentDerived, CMoment, 1)



// insert element
void CC2_1Dlg::OnOK()
{
    CMomentDerived derived;
    CMoment* moment;

    // use derived class at random
    if (rand()<RAND_MAX/3)
        moment = &derived;
    else
        moment = &m_Moment;

    // actually set time/date
    UpdateData(TRUE);
    moment->SetYear  (m_Date.GetYear());
    moment->SetMonth (m_Date.GetMonth());
    moment->SetDay   (m_Date.GetDay());
```

```
    moment->SetHour  (m_Time.GetHour());
    moment->SetMinute(m_Time.GetMinute());
    moment->SetSecond(m_Time.GetSecond());

    // add to the set
    if (m_Set.Insert(moment) != -1)
        UpdateListBox();
    else
        AfxMessageBox("Zeitpunkt schon in der Menge enthalten !", MB_ICONSTOP);
}


// delete a entry from the listbox
void CC2_1Dlg::OnDelete()
{
    // save current selection
    int nSelected = m_List.GetCurSel();
    if (nSelected < 0)
        return;

    // move cursor to the selection
    delete m_Set.GetFirst();
    while (nSelected-- > 0)
        delete m_Set.GetNext();

    // and delete
    m_Set.Scratch();

    // redraw the listbox
    UpdateListBox();
}


// deserialize
void CC2_1Dlg::OnLoad()
{
    CFileDialog dlg(TRUE);
    dlg.DoModal();

    CFile    file(dlg.GetPathName(), CFile::modeRead);
    CArchive ar  (&file, CArchive::load);

    m_Set.Serialize(ar);

    // redraw the listbox
    UpdateListBox();
}


// serialize
void CC2_1Dlg::OnSave()
{
    CFileDialog dlg(FALSE);
    dlg.DoModal();

    CFile    file(dlg.GetPathName(), CFile::modeCreate|CFile::modeWrite);
    CArchive ar  (&file, CArchive::store);

    m_Set.Serialize(ar);
}


// redraw listbox
void CC2_1Dlg::UpdateListBox()
{
    // save index of selected item
    int nSelected = m_List.GetCurSel();

    // delete all elements
    m_List.ResetContent();

    int nElements = m_Set.Card();

    // empty list ?
    if (nElements == 0)
```

```
        return;

    CMoment* obj = m_Set.GetFirst();
    while (nElements > 0)
    {
        // display some info
        CString info;
        info.Format("%02d.%02d.%04d - %02d:%02d:%02d",
                    obj->GetDay(),  obj->GetMonth(),  obj->GetYear(),
                    obj->GetHour(), obj->GetMinute(), obj->GetSecond());

        // look for derived classes (they prove polymorphism !)
        if (obj->IsKindOf(RUNTIME_CLASS(CMomentDerived)))
            info += " (abgeleitet)";

        delete obj;
        m_List.AddString(info);

        nElements--;
        // if the end is not reached then get next element
        if (nElements > 0)
            obj = m_Set.GetNext();
    }

    // set selection
    if (nSelected == -1 || nSelected > m_List.GetCount())
        nSelected = m_List.GetCount()-1;
    m_List.SetCurSel(nSelected);
}


// change both pickers
void CC2_1Dlg::OnDblclkList()
{
    // save current selection
    int nSelected = m_List.GetCurSel();
    if (nSelected < 0)
        return;

    // move cursor to the selection, delete all generated elements
    delete m_Set.GetFirst();
    while (nSelected-- > 0)
        delete m_Set.GetNext();

    // even CMomentDerived can be handled this way
    CMoment* moment = m_Set.GetCurrent();
    m_Date = *moment;
    m_Time = *moment;
    delete moment;

    // update both pickers
    UpdateData(FALSE);

#ifdef _DEBUG
    afxDump << m_Set;
#endif
}
```