Stephan Brumme, SST, 3.FS, Matrikelnr. 70 25 44
mail@stephan-brumme.com

Aufgabe O1.1

Ich habe das Modul *MRoom* lediglich als Vorlage genommen, den gesamten Quellcode, da er nicht sehr umfangreich war, jedoch komplett neu geschrieben. Eine Ausnahme hiervon bildet die Headerdatei *PrimitiveTypes.h*, die die CEDL-Datentypen zur Verfügung stellt.

Die Klasse *CRoom* besteht aus zwei als privat geschützten Variablen, `m_nNumberOfRoom` und `m_nArea`, die beide vom CEDL-Typ `Ordinal` sind. Um diese herum scharen sich die Funktionen:

```
// constructor (former Init !)
CRoom(Ordinal nNumberOfRoom, Ordinal nArea) :
       m_nArea(nArea), m_nNumberOfRoom(nNumberOfRoom) {}

// destructor (not neccessary)
virtual ~CRoom() {}

// compare two rooms
virtual Boolean operator==(const CRoom& room) const;
virtual Boolean EqualValue(const CRoom& room) const;

// copy one room to another one
virtual CRoom&  operator=(const CRoom& room);
virtual Boolean Copy     (const CRoom& room);

// access m_nNumberOfRoom
Ordinal GetNumberOfRoom() const;
void    SetNumberOfRoom(const Ordinal nNumberOfRoom);

// retrieve covered area
Ordinal GetArea() const;

// display the attributes
// only for internal purposes !
virtual void Show() const;
```

Ich habe Konstruktor/Destruktor farblich hervorgehoben. Von ihrer Funktionalität sind `operator==` und `EqualValue` sowie `operator=` und `Copy` gleich.

Die überladene Operation == wird implementiert als:

```
Boolean CRoom::operator==(const CRoom& room) const
{
       return ((room.m_nArea         == m_nArea) &&
               (room.m_nNumberOfRoom == m_nNumberOfRoom));
}
```

Für = lautet der entsprechende Code:

```
CRoom& CRoom::operator=(const CRoom& room)
{
       // cannot copy an object to itself
       if (this != &room)
       {
              // copy all variables
              m_nArea         = room.m_nArea;
              m_nNumberOfRoom = room.m_nNumberOfRoom;
       }

       return *this;
}
```

Als kleines Feature sorge ich noch dafür, dass ein Objekt sich nicht in sich selbst kopiert.

CRoom.h:

```
////////////////////////////////////////////////////////
// Softwarebauelemente I, Aufgabe O1.1
//
```

```
// author:            Stephan Brumme
// last changes:      January 13, 2001

#if !defined(AFX_ROOM_H__34138CE0_E97C_11D4_9BB7_8BA1BD2C3421__INCLUDED_)
#define AFX_ROOM_H__34138CE0_E97C_11D4_9BB7_8BA1BD2C3421__INCLUDED_

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


// use the CEDL names for data types
#include "PrimitiveTypes.h"


// declare class CRoom
class CRoom
{
private:
       // protect the member variables
       Ordinal m_nNumberOfRoom;
       Ordinal m_nArea;

public:
       // constructor (former Init !)
       CRoom(Ordinal nNumberOfRoom, Ordinal nArea) :
              m_nArea(nArea), m_nNumberOfRoom(nNumberOfRoom) {}

       // destructor (not neccessary)
       virtual ~CRoom() {}

       // compare two rooms
       virtual Boolean operator==(const CRoom& room) const;
       virtual Boolean EqualValue(const CRoom& room) const;

       // copy one room to another one
       virtual CRoom&  operator=(const CRoom& room);
       virtual Boolean Copy     (const CRoom& room);

       // access m_nNumberOfRoom
       Ordinal GetNumberOfRoom() const;
       void    SetNumberOfRoom(const Ordinal nNumberOfRoom);

       // retrieve covered area
       Ordinal GetArea() const;

       // display the attributes
       // only for internal purposes !
       virtual void Show() const;
};

#endif // !defined(AFX_ROOM_H__34138CE0_E97C_11D4_9BB7_8BA1BD2C3421__INCLUDED_)
```

CRoom.cpp:

```
////////////////////////////////////////////////////////
// Softwarebauelemente I, Aufgabe O1.1
//
// author:            Stephan Brumme
// last changes:      January 13, 2001

#include "Room.h"
#include <iostream>


// compare the room with another one
Boolean CRoom::operator==(const CRoom& room) const
{
       return ((room.m_nArea         == m_nArea) &&
               (room.m_nNumberOfRoom == m_nNumberOfRoom));
}
```

```
// compare the room with another one
// routes call down to "==" operator
Boolean CRoom::EqualValue(const CRoom& room) const
{
	return (operator==(room));
}



// copy one room to another one
// prevents user from copying an object to itself
CRoom& CRoom::operator=(const CRoom& room)
{
	// cannot copy an object to itself
	if (this != &room)
	{
		// copy all variables
		m_nArea         = room.m_nArea;
		m_nNumberOfRoom = room.m_nNumberOfRoom;
	}

	return *this;
}


// copy one room to another one
// prevents user from copying an object to itself
// calls "=" operator internally
Boolean CRoom::Copy(const CRoom &room)
{
	// cannot copy an object to itself
	if (this == &room)
		return false;

	// use "=" operator
	operator=(room);

	return true;
}


// retrieve the private value of m_nNumberOfRoom
Ordinal CRoom::GetNumberOfRoom() const
{
	return m_nNumberOfRoom;
}


// change private m_nNumberOfRoom
void CRoom::SetNumberOfRoom(const Ordinal nNumberOfRoom)
{
	m_nNumberOfRoom = nNumberOfRoom;
}


// retrieve the private value of m_nArea
Ordinal CRoom::GetArea() const
{
	return m_nArea;
}


// display the attributes
// only for internal purposes !
void CRoom::Show() const
{
	// open namespace
	using namespace std;

	// display the crap using default output
	cout << "The current room has the room number "<<GetNumberOfRoom()
		<<" and covers "<<GetArea()<<" acres !"<<endl;
}
```

O1_1.cpp:

```
//////////////////////////////////////////////////////
// Softwarebauelemente I, Aufgabe O1.1
//
// author:            Stephan Brumme
// last changes:      January 13, 2001


#include "Room.h"
#include <iostream>


using namespace std;


void main()
{
       CRoom myRoom(2,3);
       CRoom myRoom2(4,5);

       myRoom.Show();
       myRoom2.Show();

       // I wrote some lines twice to verify the overloaded operators

       cout<<"Compare rooms ... "<<(myRoom==myRoom2)<<endl;
//     cout<<"Compare rooms ... "<<myRoom.EqualValue(myRoom2)<<endl;

       cout<<"Now copy myRoom to myRoom2 ..."<<endl;
       myRoom2 = myRoom;
//     myRoom2.Copy(myRoom);

       cout<<"Compare again ... "<<(myRoom==myRoom2)<<endl;
//     cout<<"Compare again ... "<<myRoom.EqualValue(myRoom2)<<endl;

       cout<<"Verify the Copy-To-Itself protection ..."<<myRoom.Copy(myRoom)<<endl;
}
```