Universität Potsdam
Institut für Informatik
Professur Informatik II
Prof. Dr. E. Horn
Lehrveranstaltung Softwarebauelemente I

Potsdam, 20. April 2001

Praktikumsdokumentation

Aufgabe: Persistente Verwaltung der Daten einer Dokumentenmappe

Autor: Stephan Brumme, Matrikelnr. 702544

Umfang: 74 Seiten und 1 CD

Seminarbetreuer: Dr. W. Schubert

Bearbeitungsvermerk:

Inhaltsverzeichnis

1 Aufgabenstellung

1.1 Ursprüngliche Aufgabenstellung

Entwickeln Sie eine objektorientierte Lösung für die Problemstellung

Persistente Verwaltung der Daten einer Dokumentenmappe

in den Programmiersprachen C++, Java oder Object Pascal. Die Bausteine des Programmsystems sollen in Teilsystemen abgelegt werden. Bausteine des Programmsystems sind mindestens die Klassen

Abbildung 1: Übersicht Klassen

Klasse	Funktionalität
`CBankDocument`	Wurzelklasse der Dokumentenhierachie
`CForm`	allgemeines Formular
`CStandingOrder`	Dauerauftrag
`CContract`	Vertrag
`CFolder`	Mappe von Dokumenten für einen Kunden

Bestimmen Sie aus der Beschreibung die Klassen- und Teilsystemstruktur sowie die notwendigen Polymorphiebeziehungen. Treffen Sie Implementationsentscheidungen hinsichtlich der Interfaces der Klassen und der Realisierung der Mengenorganisation.

Bestimmen und Beschreiben Sie die Vor- und Nachbedingungen der Methoden.

Zur Absicherung von Softwarequalitätsparametern des Programmsystems ist eine Testumgebung, die eine von Ihnen zu bestimmende Teststrategie unterstützt, zu entwerfen und zu implementieren.

Hinweis: Die genaue Aufgabenstellung ist im Anhang komplett zitiert (insbesondere die Klassenbeschreibungen) im Abschnitt 6.1..

1.2 Einschränkungen

Generell entfallen:

- Persistenzhaltung
- Klasse CFolder

Entfallene Methoden:

- Generate
- ClassInvariant
- EqualKey
- KeyOf

2 Erläuterung der Aufgabenstellung

2.1 Interpretation der Aufgabenstellung

Die Aufgabenstellung verfolgt das Ziel, eine Klassenstruktur zu gewinnen, die die Verwaltung von Kundendaten einer Bank erlaubt. Als Ergebnis bekommt man keine umfassende Lösung, sondern die grundlegenden Bausteine, die in einem größeren System die Rolle von Basismoduln bilden. Sie sind in der Lage, einfache Dokumente (`CBankDocument`), Verträge (`CContract`), Formulare (`CForm`) und Daueraufträge (`CStandingOrder`) zu erzeugen. Zur Umsetzung ist noch eine Datumsstruktur (`CDate`) notwendig.

Die Persistenzhaltung, d.h. Sicherung der Daten auf nicht-flüchtigen Datenspeichern, sollte in `CFolder` implementiert werden, aber dieser Teil der Aufgabenstellung wurde nachträglich gestrichen. Ebenso ist von der Professur auf Typinformationen, die RTTI erweitern sollten, verzichtet worden. Der Grund dafür ist der, dass die Vorlesung noch nicht weit genug im Stoff war, um alle notwendigen Voraussetzungen unterrichtet zu haben. Da ich RTTI bereits in meiner Testumgebung ausgiebig nutze, habe ich sie entsprechend der bisherigen Hausaufgaben und zusätzlicher Literatur programmiert.

Die gegebene CEDL-Beschreibung ist bereits ein Modell, das 1:1 in die Praxis umsetzbar ist. Ich empfand es dennoch als günstiger, einige Erweiterungen in Form zusätzlicher Methoden einzuführen, um die Realisierung zu vereinfachen. Damit meine ich nicht unbedingt den Umfang des Quelltextes, sondern ziele mehr auf eine klarere Klassenstruktur ab. Es ist aber sichergestellt, dass jede einzelne Funktionalität, die in der CEDL-Beschreibung gefordert wird, sich auch in meinem Projekt wiederfindet.

Eine Form der Qualitätssicherung besteht in der analytischen Überprüfung der Klassenstruktur. Zu diesem Zwecke entstand eine eigene Testumgebung, die der Programmierer interaktiv nutzen kann, um alle Klassen in beliebiger Kombination zu erstellen und mit ihnen zu arbeiten

Hinweis: Diese Dokumentation liegt auch in elektronischer Form auf dem beiliegenden Datenträger vor (genaueres steht im Anhang unter dem Punkt 6.2).
Im Falle eventueller Rückfragen etc. bin ich jederzeit per Email unter mail@stephan-brumme.com erreichbar.

2.2 Begriffsklärung

Da es sich um eine objektorientierte Lösung handelt, sind für mich im folgenden die Begriffe *Modul* und *Klasse* synonym.
Ich verwende oft das Wort *Methode*, um eine Funktion einer Klasse zu beschreiben. Ein alternativer Ausdruck ist *Operation*, der aber für mich eine Verwechslungsgefahr mit Operator birgt.
Die Daten, d.h. Variablen einer Klasse heißen *Attribute*.

Die graphische Notation der objektorientierten Beziehungen erfolgt mit Hilfe der *Unified Modelling Language* (UML).

Der Quellcode ist vollständig in englisch geschrieben und kommentiert, weil Englisch quasi *die* Sprache der Informatik ist. Deshalb fällt es des öfteren schwer, für die verwendeten englischen Begriffe deutsche Entsprechungen zu finden. Im Zweifel benutze ich daher den englischen Ausdruck, wenn es sich um einen geläufigen Standard handelt.

2.3 Schriftsatz

In dieser Dokumentation werden Namen aus den Quelltextes in der Schriftart `Courier` gesetzt. Besonders wichtige Wörter sind *kursiv* hervorgehoben. Der Quelltext selber wurde mit Syntax-Highlighting bearbeitet, um die Lesbarkeit zu erhöhen.

Um den Umfang der Dokumentation zu verringern, habe ich große Teile des Quellcodes und Auszüge aus den Testprotokollen in Schriftgröße `8` statt `10` gesetzt. Zusätzlich liegt beides der Dokumentation vollständig in elektronischer Form bei. Trotz aller Vorsichtsmaßnahmen sind gelegentliche ungewollte Zeilenumbrüche im Quellcode möglich. Relevant ist sind daher stets die Original-Dateien und nicht die Versionen im Anhang dieser Dokumentation.

Dateinamen werden in den Schriftart Arial kenntlich gemacht.

3 Darstellung des Lösungsmodells

Ich implementierte die Lösung in C++, da ich darin die meiste Erfahrung besitze und der Sprache auch die größte Flexibilität zuspreche. Ich benutzte keine compiler-spezifischen Eigenheiten, daher sollte jeder ANSI-C++-kompatible Dialekt damit zurecht kommen, auch wenn die Entwicklungsplattform Visual C++ 6 von Microsoft war.

3.1 Darstellung der Lösungsidee

Bei der Betrachtung der CEDL-Beschreibung fällt auf, dass die verschiedenen Klassen viele Gemeinsamkeiten besitzen. Besonders hervorzuheben sind dabei die Methoden `EqualValue`, `Copy` und `Show`. Um mich als Programmierer zu einer sauberen Implementation zu zwingen, wollte ich einen Weg einschlagen, der von mir *verlangt*, dass *jede* Klasse auch wirklich *alle* diese Methode bereitstellt. Unter Java wäre dies ein Interface, da ich aber C++ verwende, erstellte ich eine abstrakte Basisklasse namens `CBasicClass`. *Jede* Klasse wird direkt oder indirekt von ihr abgeleitet.

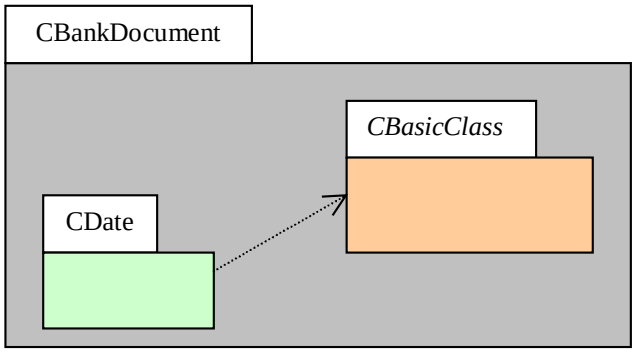
Zusätzlich zu den genannten Methoden empfand ich es als sinnvoll, noch `ShowDebug`, `ClassnameOf` und `Equal` in `CBasicClass` zu deklarieren. Mein Gedankengang ist der, dass jede Klasse auch eine Schnittstelle aufweisen soll, die - zusätzlich zum internen Debugger - zur Laufzeit Informationen über die Klasse bereitstellen kann. Diese Fähigkeit kam mir bei der Entwicklung der Testumgebung dann sehr zur Hilfe.

Diese Testumgebung ist ebenso als eigene Klasse entworfen worden. Sie ist interaktiv bedienbar und daher nicht abhängig von vorher fest codierten Befehlsfolgen. Somit werden auch Personen, die die Klassenstruktur nicht entworfen bzw. programmiert haben, in die Lage versetzt, das System auf mögliche Fehler zu testen.

Die Ausgabe von Daten über `Show` oder `ShowDebug` soll unabhängig vom Darstellungsmedium (Bildschirm, Datei etc.) sein. Ebenso ist es wünschenswert, eine parallele Ausgabe auf min. 2 Medien zu erhalten, wie dies z.B. der Fall ist, wenn interaktiv am Bildschirm getestet wird und gleichzeitig eine Protokolldatei erstellt werden soll. Um dieses Problem zu lösen, wandelte ich `Show`/`ShowDebug` und die Testumgebung derart ab, dass keine direkte Ausgabe erfolgt, sondern nur ein `string` generiert wird. Dieser kann dann mit `CTest::Print(string)` serialisiert werden. In diesem Punkt weiche ich etwas von der CEDL-Beschreibung ab, eine direkte Bildschirmausgabe erfordert somit `cout << Class.Show()`.

Aufgrund der Sprachdefinition von C++ bildet jede Klasse einen eigenen Namensraum, der implizit erzeugt wird. Da mir keine weitere explizite Gruppierung von Klassen sinnvoll erscheint, belasse ich es bei den so generierten Standard-Namensräumen.

Abbildung 2: Ein vereinfachtes Teilsystem



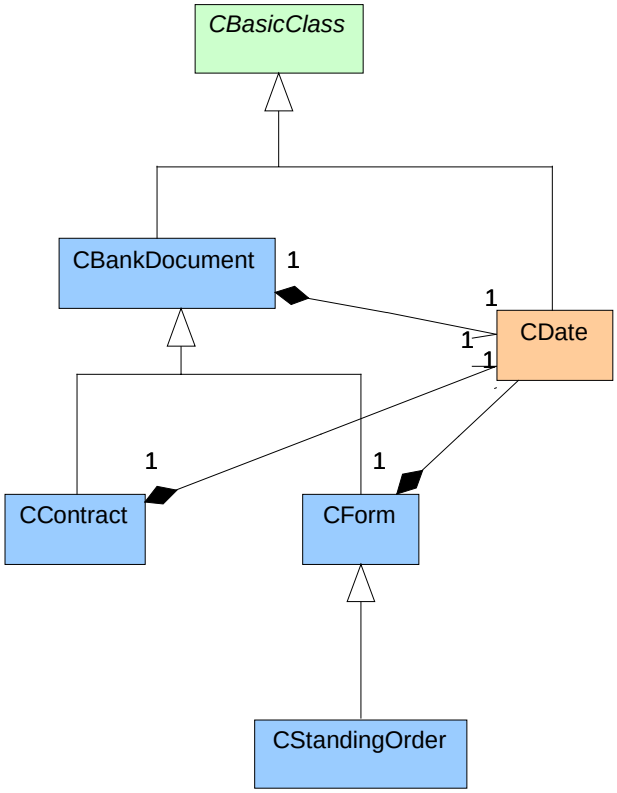
Teilsysteme entstehen automatisch, da jede Verwendung von `CBankDocument`, `CForm`, `CStandingOrder` und `CContract` auch die Benutzung von `CDate` und `CBasicClass` erfordert. Die in der Graphik etwas vereinfachte Struktur ist daher eher hypothetischer Form. Wenn die Aufgabenstellung nicht

gekürzt worden wäre, so hätte das Zusammenspiel mit `CFolder` eine echte, explizit definierte Teilsystemestruktur erforderlich gemacht.

3.2 Darstellung der Architektur des Programmsystems

Um einen schnellen Überblick über das Praktikum zu erhalten, folgt nun eine graphische Veranschaulichung der Klassenstruktur in UML-Notation, wobei ich auf die STL-Bibliothek verzichte:

Abbildung 3: UML-Beschreibung der Klassenstruktur in Kurzform



Von jeder Klasse aus führt ein Pfad von Vererbungsbeziehungen auf die abstrakte Basisklasse `CBasicClass` zurück. Sie kann daher quasi als Fundament bezeichnet werden.

Ein weiterer Grundpfeiler ist `CDate`, das von jeder Klasse benutzt wird (Aggregations-Beziehung), außer natürlich von `CBasicClass`. Bei genauer Code-Inspektion fällt auf, dass `CStandingOrder` nicht direkt auf `CDate` angewiesen ist, da weder ein Attribut, noch eine Methode `CDate` verwendet, aber `CStandingOrder` erbt von `CForm`, welches natürlich `CDate` unbedingt braucht.

Um `CBasicClass` effektiv nutzen zu können, müssen alle von ihr bereitgestellten Methoden polymorph sein. Dies erreicht man in C++ durch das Schlüsselwort `virtual`. Da ich jedoch in dieser Basisklasse keine konkrete Implementation angeben kann, stelle ich nur die Forderung nach einer Umsetzung in allen abgeleiteten Klassen auf. Somit benötige *rein-virtuelle* Methoden, d.h. `virtual returnvalue Method(parameter) = 0;`. Eine Ausnahme bildet `Equal`, da diese Methode lediglich Objektidentität überprüft, was auf einen einfachen Vergleich der Speicheradressen zweier Objekte hinausläuft.

Der große Nutzen der Polymorphie besteht darin, dass trotz *Up-Casting* bei überschriebenen Methoden die korrekte aufgerufen wird. Ein kurzes Beispiel soll dies verdeutlichen:

```
CDate dtDate;
```

```
CBasicClass* pClass = &dtDate;
pClass->Show();
```

Die Klasse `CDate` wurde von `CBasicClass` abgeleitet, ist also eine Tocherklasse. Sie enthält alle Eigenschaft von `CBasicClass` (und einige mehr). Deshalb ist es korrekt, dass `pClass` auf die Adresse von `dtDate` zeigen darf, da Up-Casting generell zulässig ist. Wenn der Compiler jedoch die letzte Zeile verarbeiten will, so ist nicht klar, welches `Show()` benutzt werden soll: `Show()` von `CBasicClass` ist erst gar nicht implementiert, selbst wenn, dann würde es die falsche Bildschirmausgabe liefern. Mit der Benutzung virtuellen Methodentabellen kann `pClass` aber feststellen, dass `Show()` in `CDate` überschrieben wurde und kennt die neue Adresse von `Show()`, was einen nun korrekten Aufruf mit vernünftigen Anzeige erzeugt.

Abbildung 4: Polymorphe Methoden

Methode	Bedeutung
`string Show()`	zeigt Attribute formatiert an
`string ShowDebug()`	zeigt alle Attribute mit Zusatzinformationen an
`string ClassnameOf()`	liefert den Klassennamen zurück
`bool Copy(CBasicClass* pClass)`	kopiert die Attribute eines Objekts
`bool EqualValue(CBasicClass* pClass)`	überprüft Wertidentität zweier Objekte
`bool IsValid()`	überprüft Gültigkeit der Attribute

Eine notwendige Bedingung für polymorphe Methoden ist die Signaturgleichheit. Bei `Show()` ist dies problemlos gegeben, da eh keine Parameter verwendet werden. Etwas komplizierter ist der Fall bei `Copy` und `EqualValue`. Die Signaturgleichheit kann nur erzwungen werden, indem die Parameterliste jeweils `(CBasicClass* pClass)` lautet. Die Gefahr ist nun, dass der Compiler sämtliche Zeiger `CBasicClass*` als Parameter zulässt. Somit ist es für den Compiler zulässig, dass `CDate` mit `CForm` verglichen wird, da beide von `CBasicClass` abstammen. Die Lösung für dieses Problem besteht in der Verwendung von `dynamic_cast`. Unter Verwendung von Laufzeitinformationen (RTTI) werden während der Ausführung des Programmes die Klassentypen verglichen. In `CDate` sieht `Copy` folgendermaßen aus:

Abbildung 5: `dynamic_cast` am Beispiel von `CDate::Copy`

```
// virtual, see operator=
bool CDate::Copy(CBasicClass *pClass)
{
    // cast to CDate
    CDate *date;
    date = dynamic_cast<CDate*>(pClass);

    // invalid class (is NULL when pClass is not a CDate)
    if (date == NULL || date == this)
        return false;

    // use non virtual reference based copy
    // return value isn't needed
    operator=(*date);

    // we're done
    return true;
}
```

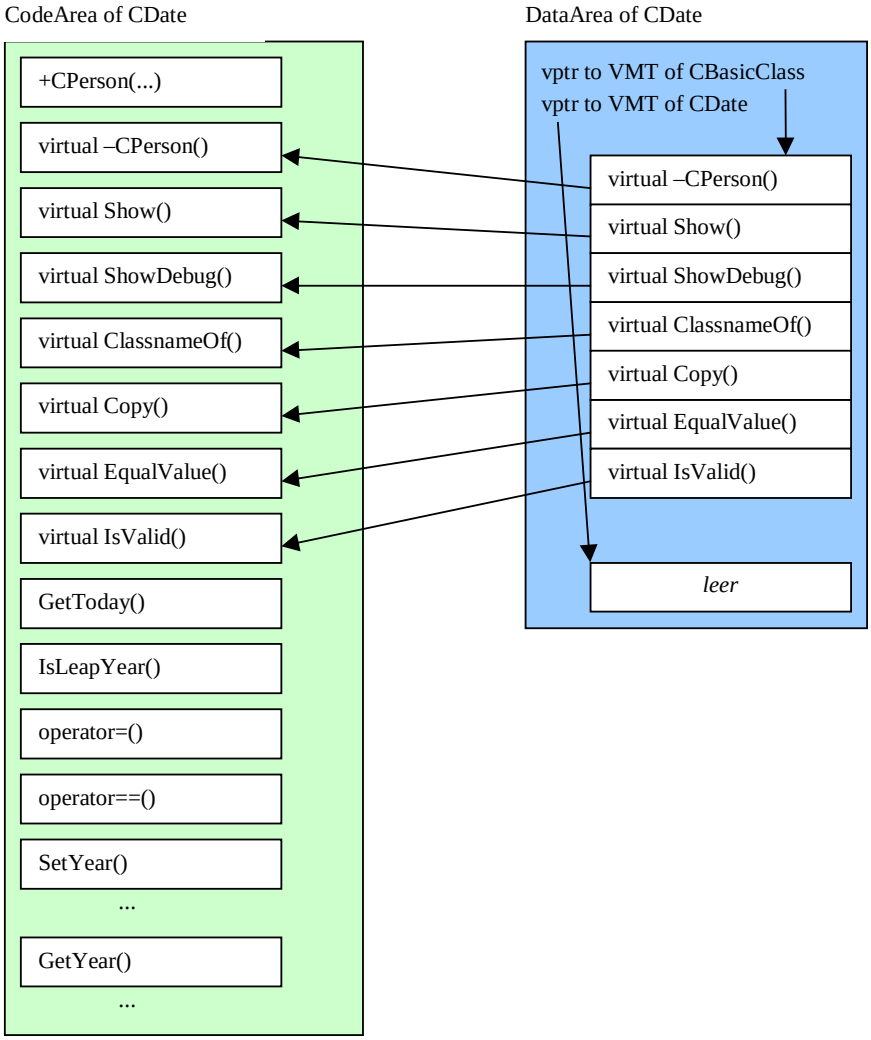
Die Variable `date` enthält als Ergebnis von `dynamic_cast` entweder den korrekten Zeiger von `pClass` oder, wenn `pClass` nicht auf ein `CDate` zeigt, `NULL`. Im weiteren Verlauf ruft `Copy` nur noch `operator=` auf, der den eigentlichen Kopiervorgang durchführt.

Die Unterscheidung zwischen `Copy` und `operator=` ist in meinen Augen sinnvoll, da letzteres der intuitivere Weg bei der Programmierung ist. Außerdem kann mir der Compiler bei der Verwendung von `operator=` viel mehr Typsicherheit geben: er kann den per Referenz übergebenen Parameter überprüfen und auswerten, was bei einem Zeiger erst zur Laufzeit möglich ist. So kann es passieren, dass der Code, der Zeiger verwendet, nur 1x pro Jahr aufgerufen wird und daher nur extrem schwer zu entdecken ist. Die Referenz *muss* dagegen *immer* korrekt sein, anderenfalls könnte ich erst gar kein ausführbares Programm erhalten. Im

vorliegenden Praktikum ist dieses Problem zum Glück irrelevant, da Vergleiche von Objekten unterschiedlicher Klassen stets mit dem Resultat `false` enden.

Die Polymorphie erfordert allerdings einen größeren Speicherverbrauch, wie man an folgendem Schema erkennen kann, da die Virtuelle-Methodetabelle (VMT) zusätzlich verwaltet werden muss. Erneut steht `CDate` stellvertretend für alle anderen Klassen:

Abbildung 6: Speicherlayout von CDate



Man erkennt, dass die abgeleiteten Klassen keine zusätzlichen virtuellen Methoden definieren, also lediglich die VMT von `CBasicClass` unverändert übernehmen.

4 Programmtechnische Realisierung des Lösungsmodells

Jede Klasse setzt sich im Quellcode aus zwei Dateien zusammen: einer `.h`-Datei, die die Deklaration der Schnittstelle nach außen darstellt und einer `.cpp`-Datei, die die eigentliche Implementation der Klasse ist. Im Gegensatz zum Klassennamen fehlt das führende `C`, d.h. die Klasse `CDate` findet man in `Date.h` und `Date.cpp`.

Im Verlaufe des Semesters entstehen aufgrund der Hausaufgaben schon mehrere Klassen bzw. Algorithmen, die ich (in abgeänderter Form) für dieses Praktikum wiederverwendete. Dies trifft u.a. auf `CDate` zu, auch die Idee mit der Verwendung von `CBasicClass` stammt dort her. Somit kann ich bereits gesammelte Erfahrungen nutzen und auch auf bereits getätigte Softwarequalitätstest zurückgreifen.

4.1 Namenskonventionen

Ich habe sämtliche Klassen und Methoden entsprechend der CEDL-Vorlage umgesetzt. Bei der Benennung der Variablen ging ich jedoch einen eigenen Weg, der sich im Wesentlichen an meinen bisherigen Erfahrungen mit den Microsoft Foundation Classes (MFC) orientiert, aber nicht allzu stark von der in der Vorlesung dargestellten Vorgehensweise abweicht. Die grundlegende Idee ist die, dass man bereits am Namen einer Variablen ihren Datentyp erkennen kann. Als Sprache kommt Englisch zum Zuge.

Jeder Klassenname beginnt mit `C`, jede selbstdefinierte Datenstruktur mit `T`.

Der Präfix `m_` deutet darauf hin, dass es sich um ein Attribut einer Klasse handelt. Diese bilden zusammen des Zustand einer Instanz und sind deshalb deutlich von lokalen und temporären Variablen zu unterscheiden. Weiterhin besitzt *jede* Variable einen Präfix, der eine Abkürzung eines Datentyps ist:

Abbildung 7: Variablenbenennung

Datentyp	Präfix	C++ Entsprechung	Beispiel
natürliche Zahl	`n`	`unsigned int`	`nInterval`
Wahrheitswert	`b`	`bool`	`bSigned`
Zeiger	`p`	*Datentyp**	`pClass`
Zeichenkette	`str`	`std::string`	`strWording`
Datum/Zeit	`dt`	`CDate` (Eigenentwicklung)	`dtAlterationDate`
reelle Zahl	`f / d`	`float / double`	`fTemperature`
Datei-Handle	`h`	*verschieden*	`hLogFile`
Feld	`ar`	`std::vector<>`	`arPupils`

Die Zeilen unterhalb der etwas dickeren Linie werden in den Haupt-Klassen der Praktikumsarbeit nicht gebraucht, sie finden in der Testumgebung Verwendung.

Die Methodennamen setzen sich i.d.R. aus zwei Teilen zusammen: einem Verb, das die auszuführende Aktion beschreibt und einem Substantiv, das das zu bearbeitende Attribut/Objekt bezeichnet. Die häufigsten Methoden folgen daher dem Muster:

Abbildung 8: Methodenbenennung

Präfix-Verb	Aktion	Beispiel
`Get`	Auslesen eines Attributes	`GetDay`
`Set`	Schreiben eines Attributes	`SetYear`
`Show`	Anzeige	`Show`
`Is`	Zustandsabfrage	`IsValid`

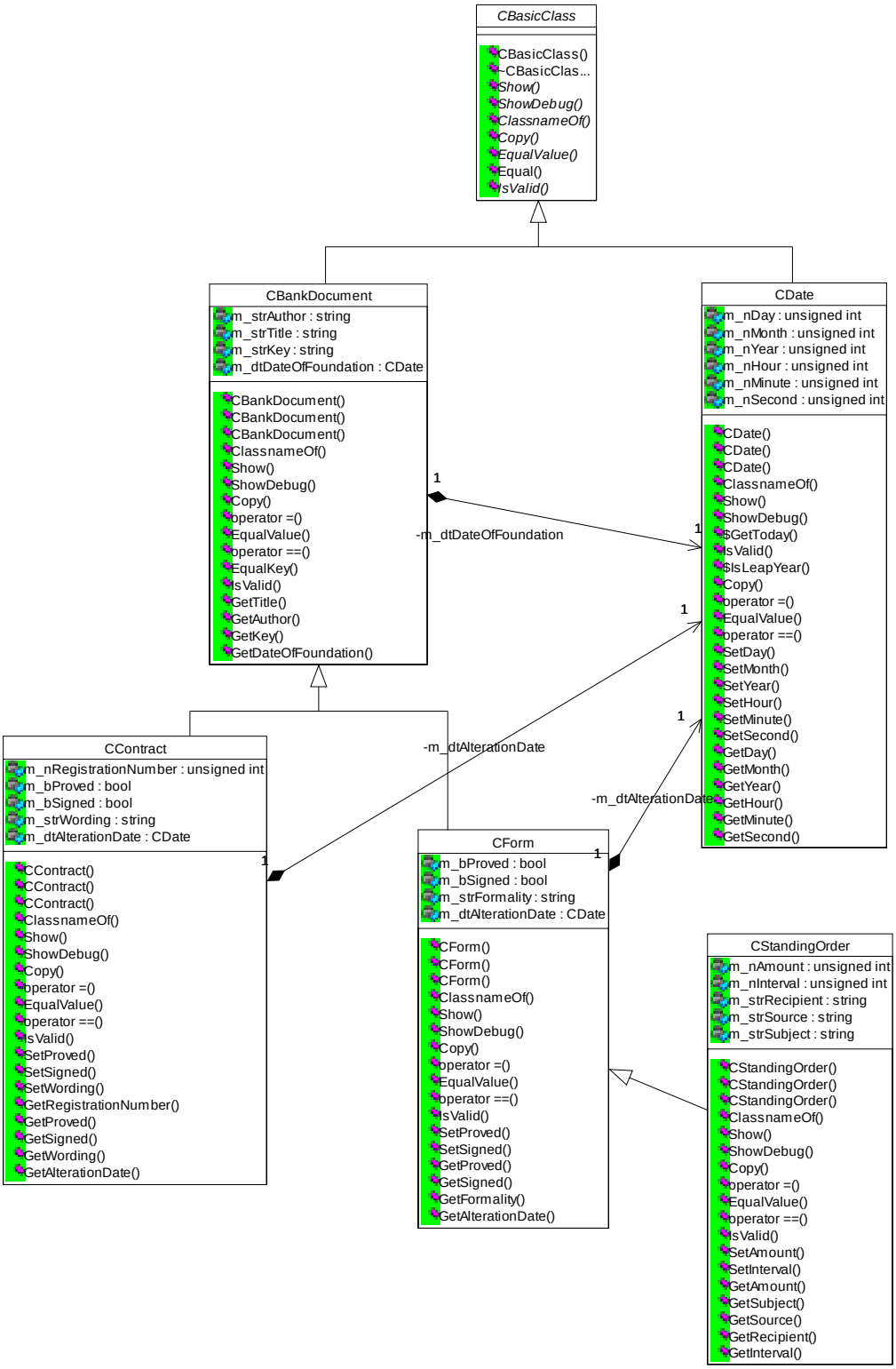
Ich lege sehr großen Wert auf ausgeschriebene Namen, um so die Eindeutigkeit zu wahren und ein schnelleres Verständnis des Quelltextes zu ermöglichen. Diese Aussage bezieht sich sowohl auf Variablen- als auch auf Funktionsnamen.

Konstanten bestehen nur aus Großbuchstaben.

4.2 Realisierungskonzeption

4.2.1 Allgemein

Abbildung 9: Komplette UML-Sicht



Ein Grundprinzip der objektorientierten Programmierung ist die Wiederverwendung von Code. Bei diesem Praktikum ging ich von den teilweise vorhandenen Klassen `CBasicClass` und `CDate` aus, erweiterte sie gemäß den neuen Anforderungen und übertrug das Grunddesign schablonengleich auf die neuen Klassen, wie z.B. `CForm`.

Die Standard Template Library (STL) ist weltweit im Einsatz, da sie für die wichtigsten Probleme Lösungen in Form von Algorithmen und Datenstrukturen bereitstellt. Ihr Einsatz verringerte die von mir zu leistende Programmierarbeit und erhöhte die Stabilität des Projekts, da sie extrem gut auf Korrektheit getestet wurde. Ich benutze die Klasse `string` sehr intensiv, ebenso kommen die Streams `ostringstream`, `cout` und `ofstream` zur Zuge, die beiden letzteren jedoch nur in der Testumgebung. Ich empfand es als bequemer, den Namespace `std` global zu öffnen.

Keine der Klassen benutzt dynamisch erzeugte Datenstrukturen, daher ist für mich die explizite Einführung von Destruktoren überflüssig. Natürlich generiert auch hier der Compiler wieder Standard-Destruktoren, so dass die CEDL-Forderung erfüllt wird. Erst CTest ist auf einen Destruktor angewiesen, ich klammere diese Klasse aber etwas von den anderen aus, da sie von mir komplett selbst entworfen wurde und ihre derzeitige Struktur nicht in einem unbedingten Zusammenhang mit der Aufgabenstellung steht.

Im Punkt 3.2 dieser Dokumentation habe ich das Thema Polymorphie angesprochen. Als Konsequenz sollte so oft wie nur möglich `operator=` einer Klasse benutzt werden, wenn man unbedingt Polymorphieeffekte braucht, so ist `Copy` die bessere Wahl. Beide führen schlussendlich den gleichen Code zum elementeweisen Kopieren aus, haben daher stets die gleiche Semantik, die in `operator=` steckt. Diese Eigenschaft nutze ich in der Testumgebung, wo der Benutzer nur die polymorphe Methode aufrufen kann.

Um meinen Programmierstil in dieser Richtung zu untermauern, werden die Klassen, wenn möglich, auf dem Stack angelegt (z.B. `CDate` als Attribut fast aller anderen Klassen), ihre Lebensdauer entspricht der Ausführungsdauer des umgebenden Blocks. Nur in der Testumgebung musste ich doch auf Zeiger zurückgreifen, um die verschiedenen Klassen in einem `vector` zu speichern.

Als sehr wichtig gilt für mich die größtmögliche Reduzierung der Schreibzugriffe auf Attribute. Dies äußert sich als erstes darin, dass sie `private` deklariert und dadurch verkapselt sind (dies ist bereits eine Forderung der CEDL-Beschreibung) und setzt sich in der intensiven Verwendung von `const` fort. In der Umkehrung kann man sagen, dass wirklich nur die Methoden *nicht* `const` sind, die *unbedingt* Attribute ändern müssen. Zwar ändert diese Vorgehensweise den erzeugten Code nur wenig oder gar nicht, sorgt aber für mehr umfangreichere Hilfestellungen durch die Compiler im Fehlerfall.

In der Aufgabenstellung wird nicht exakt definiert, was unter den Datentypen `Ordinal` und `String` zu verstehen sei. In meiner Lösung benutze ich die in C++ vordefinierten Typen `unsigned int` und `string`, wie auch bereits in den Hausaufgaben.

Ich bitte um die parallele Lesart von dieser Dokumentation und der Quelltext-Kommentare, da sie im Zusammenspiel sicher alle etwaigen Fragen ausräumen.

4.2.2 Beispiel CDate

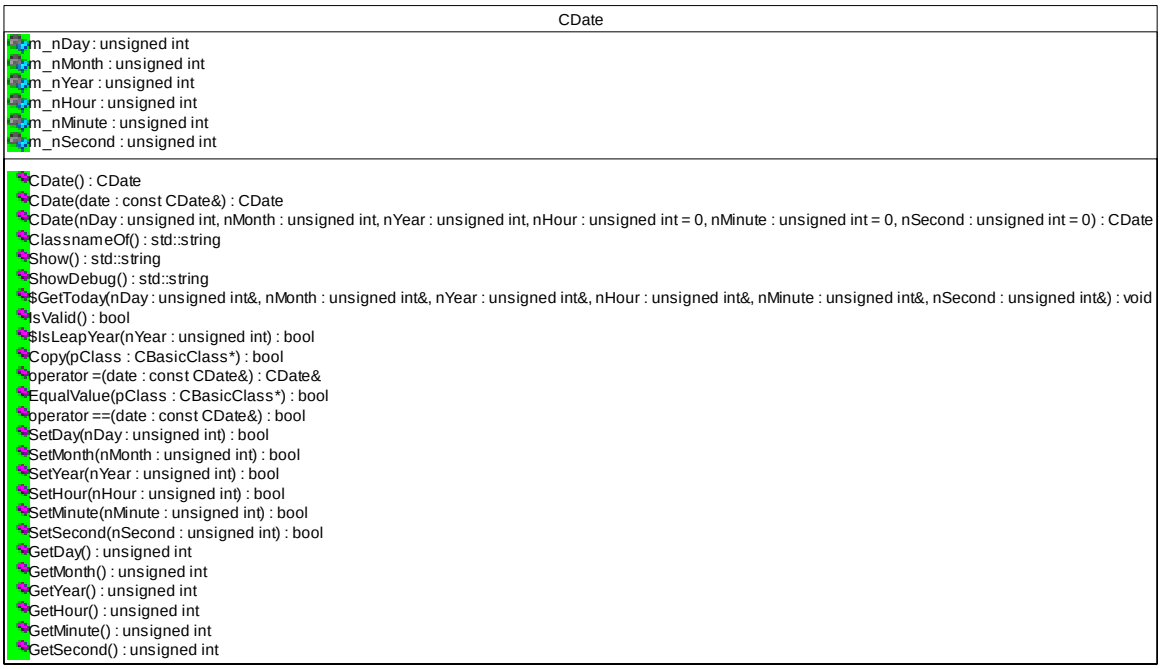
Die mit Abstand komplexeste Klassen ist `CDate`. An ihr lassen sich viele bereits theoretisch erwähnten Probleme und Lösungen sehr gut zeigen. Im wesentlichen sind fast alle Aussagen auf die anderen Klassen übertragbar, da sie sich nur durch die Anzahl an `Set`.../`Get`...-Methoden voneinander unterscheiden. Im Anschluss an die hier zitierten Quelltextzeilen folgen die jeweiligen PRE- und POST-Conditions, die ebenfalls auf alle Klassen übertragbar sind. Ihre genaue Bestimmung ist allerdings nur schwer möglich, da eigentlich fast alle Methoden als `const` deklariert sind und somit keine POST-Condition aufweisen bzw. PRE-Conditions nur bei Zeigern eine echte Notwendigkeit haben. Wäre `CFolder` auch Bestandteil der Aufgabenstellung gewesen, hätte man wesentlich detaillierte und sinnvollere PRE-/POST-Conditions definieren können.

Die Schnittstelle von `CDate` wurde in der Aufgabenstellung recht offen gelassen. Auch das Vorlesungsskript gibt keine scharfen Abgrenzungen vor. Die einzige Forderung liegt in der Bereitstellung der Speicherung von Datums- und Zeitangaben, letzteres mit einer Genauigkeit von 1 Sekunde.

Mit den MFC hätte ich sofort `CTime` oder `COleDateTime` gewählt, allerdings wollte ich darauf verzichten, da die MFC noch nicht Bestandteil der Vorlesung waren. Meine Klasse beruht aber auf ähnlichen Grundprinzipien, z.B. greife ich zur Bestimmung der aktuellen Uhrzeit ebenfalls auf die Windows-API-Funktion `time` zurück.

Einzelne Spezifika und Grenzphänomene von Datums-/Zeitangaben kann ich nicht garantieren, da der Gregorianische Kalender verschiedene Ausnahmen definiert (z.B. wurden in einem Jahr im 16.Jahrhundert bei der Umstellung bewußt 14 Tage ausgelassen). Ebenso konnte ich technische Grenzen der C++ -Bibliotheken bzw. von Windows nicht genau ergründen. Ich gehe aber davon aus, dass diese Einzelheiten nicht Bestandteil des Praktikums sind, sondern mehr Wert auf das generelle Design gelegt wird.

Abbildung 10: Interface von CDate



Jede Klasse umfasst 3 Konstruktoren:

- parameterloser Standard-Konstruktor:

  erzeugt ein neues Objekt, das initialisiert, aber nicht gültig ist

  Hinweis: Aufgrund der Sonderstellung gilt obige Aussage zwar für alle anderen Klassen nicht aber für `CDate`.
  Hier werden die Attribute mit dem aktuellen Datum und der momentanen Uhrzeit initialisiert.

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

PRE: keine
POST: initialisiert, aber nicht gültig (außer `CDate`)

- Copy-Konstruktor:

  erzeugt ein neues Objekt als Kopie eines existierenden Objekts, beide Objekte sind dann wertidentisch, aber nicht objektidentisch

```
// copy constructor
CDate::CDate(const CDate &date)
{
    operator=(date);
```

```
}
```

PRE: Übergabe eines existierenden Objekts
POST: keine, es wurde eine exakte Kopie erzeugt

- Parameter-Konstruktor

  erzeugt ein neues Objekt anhand der übergebenen Werte

  Hinweis: Für `CDate` ist die Angabe der Uhrzeit fakultativ. Standardwert ist 00:00 Uhr.

```
// set user defined date at construction time
CDate::CDate(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;
}
```

PRE: keine
POST: Objekt ist initialisiert, Gültigkeit hängt von den Parametern ab (wird nicht überprüft)

Diese Konstruktoren sind in der Lage, alle nur erdenklichen Objekte zu erzeugen bzw. zu klonen. Im allgemeinen werden dabei der Copy- und der Parameter-Konstruktor am häufigsten verwendet. Da weder `CDate` noch alle anderen Klassen (ich lasse die Testumgebung außen vor) über dynamisch erzeugte Datenstrukturen verfügen, ist ein Destruktor nicht notwendig.

`ClassnameOf` liefert einen `string` zurück, der den Klassennamen beinhaltet. Diese Eigenschaft ist eine Fähigkeit ala RTTI und für Polymorphie-Zwecke gedacht.

```
// return class name
    virtual string ClassnameOf() const { return "CDate"; }
```

PRE: keine
POST: keine

`Show` ist für eine formatierte Bildschirmausgabe zuständig, wie dies z.B. in einer Bildschirmmaske der Fall ist. Alle für den Benutzer relevanten Attribute werden in lesbarer und verständlicher Form ausgegeben. Für `CDate` bedeutet das, dass die in Deutschland gängige Darstellung von Daten bzw. Uhrzeiten verwendet wird. Sollte die Klasse in einem internationalen Programm Verwendung finden, so ist eine zusätzliche Einführung der englischen/amerikanischen Schreibweisen denkbar. Die Methode stellt selbst keine Zeichen auf dem Bildschirm oder anderen Ausgabemedien dar, sondern liefert nur eine Zeichenkette zurück, die über das Hilfsmittel `ostringstream` erzeugt wurde, was sehr elegante und portable Formatierungen erlaubt. Dieser Weg erleichtert mir mehrfache Ausgaben (siehe Testumgebung mit paralleler Bildschirmausgabe und Protokolldatei) und ist unabhängig vom User Interface.

```
// show attributes
string CDate::Show() const
{
    ostringstream strOutput;

    strOutput << setw(2) << setfill('0') << m_nDay << "."
              << setw(2) << setfill('0') << m_nMonth << "."
              << m_nYear << " - "
              << setw(2) << setfill('0') << m_nHour<< ":"
              << setw(2) << setfill('0') << m_nMinute << ":"
              << setw(2) << setfill('0') << m_nSecond;

    return strOutput.str();
}
```

PRE: keine
POST: keine

`ShowDebug` ist nur für interne Zwecke gedacht und gibt *alle* Attribute nebst ihren Namen in einer Zeichenkette zurück. Diese Methode spielt insbesondere in der Testumgebung eine wichtige Rolle.

```
// shows all internal attributes
string CDate::ShowDebug() const
{
    ostringstream strOutput;

    strOutput << "DEBUG info for '" << ClassnameOf() << "'" << endl
              << "      m_nDay    = " << m_nDay    << endl
              << "      m_nMonth  = " << m_nMonth  << endl
              << "      m_nYear   = " << m_nYear   << endl
              << "      m_nHour   = " << m_nHour   << endl
              << "      m_nMinute = " << m_nMinute << endl
              << "      m_nSecond = " << m_nSecond << endl;

    return strOutput.str();
}
```

PRE: keine
POST: keine

Eine Klassenmethode stellt `GetToday` dar. Sie benötigt kein explizit erzeugtes Objekt, sondern ist direkt nutzbar. Ihre Parameter sind Referenzen, die bisherigen Inhalte werden mit dem aktuellen Datum und der momentanen Uhrzeit überschrieben. Die benutzten Routinen habe ich 1:1 der MSDN entnommen und kurz auf ihre Exaktheit getestet. Der Algorithmus scheint zu funktionieren, ich muss mich dabei aber auf die Aussagen von Microsoft verlassen. Nicht ganz klar sind mir die technischen Grenzwerte, ich halte 2038 als möglichen Schwellwert (ähnlich wie das Y2K-Problem) für möglich, kann das aber nicht überprüfen.

```
// return current date
void CDate::GetToday(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;
}
```

PRE: Übergabe von Referenzen
POST: aktuelles Datum/Uhrzeit

Um eine Überprüfung der Attribute einer Klasse auf ihre Gültigkeit durchzuführen, existiert `IsValid`. Die Notwendigkeit dieser Funktion ergibt sich aus der Tatsache, dass nicht alle technisch möglichen Attributwerte auch eine entsprechende semantische Bedeutung haben. Ein gutes Beispiel dafür ist der 42.April 2001, der zwar in `m_nDay`, `m_nMonth` und `m_nYear` problemlos speicherbar ist, im Gregorianischen Kalender aber nicht definiert wird. Etwas diffiziler ist für `CDate` die korrekte Erkennung von Schaltjahren, deshalb habe ich sie in `IsLeapYear` ausgelagert.

```
// validate a date
bool CDate::IsValid() const
{
    // validate month
    if (m_nMonth < JANUARY || m_nMonth > DECEMBER)
        return false;
```

```
        // days per month, february may vary !!!
        unsigned int nDaysPerMonth[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

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

        // validate day
        if (m_nDay < 1 || m_nDay > nDaysPerMonth[m_nMonth])
            return false;
        // day and month are valid

        // now check the time
        if (m_nHour < 0 || m_nHour > 23)
            return false;
        if (m_nMinute < 0 || m_nMinute > 59)
            return false;
        if (m_nSecond < 0 || m_nSecond > 59)
            return false;

        // instance must be valid now
        return true;
    }
```

PRE: keine
POST: keine

`IsLeapYear` ist, ähnlich `GetToday`, der MSDN entnommen und liefert true im Falle eines Schaltjahres zurück. Es handelt sich auch um eine Klassenmethode (siehe Date.h).

```
    // determine whether it is a leap year
    bool CDate::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;
    }
```

PRE: keine
POST: keine

Der überladene Operator = ist einer der wichtigsten in der Klassenstruktur, da er auch vom Copy-Konstruktor benutzt wird. Ich bin mir dessen bewusst, dass der Compiler automatisch diesen Operator überlädt und auch den korrekten Code erzeugt (der eventuell besser optimiert werden kann als meiner), definiere diese Methode aber trotzdem, um im Vergleich mit == eine Vollständigkeit zu wahren. Rein technisch kopiert diese Methode lediglich alle Attribute, da sie statischer Natur sind, müssen keinerlei besondere Rücksichtsmaßnahmen getroffen werden. Der Parameter ist gezwungenermaßen eine Referenz auf ein `CDate`-Objekt, damit ist keine Polymorphie möglich (Signaturgleichheit verletzt), was bereits im Abschnitt 3.2 ausgiebig diskutiert wurde.

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

```
        return *this;
    }
```

PRE: keine
POST: keine

`Copy` ist das polymorphe Gegenstück zu `operator=`. Hier ist ein dynamisches Casting vonnöten, ansonsten wird `operator=` aufgerufen. Diese genaue Funktionalität habe ich ebenfalls in Abschnitt 3.2 beschrieben.

```
// virtual, see operator=
bool CDate::Copy(CBasicClass *pClass)
{
    // cast to CDate
    CDate *date;
    date = dynamic_cast<CDate*>(pClass);

    // invalid class (is NULL when pClass is not a CDate)
    if (date == NULL || date == this)
        return false;

    // use non virtual reference based copy
    // return value isn't needed
    operator=(*date);

    // we're done
    return true;
}
```

PRE: Übergabe eines Zeigers auf ein anderes `CDate` Objekt
POST: keine

Zusätzlich zu den Kopieraktionen sind auch Vergleiche möglich. Hierbei stehe ich wieder vor dem gleichen Wiederspruch zwischen intuitivem Überladen eines Operators und der Forderung nach Polymorphie. Die Lösung ist die gleiche wie für das Kopieren mittels = bzw. `Copy`. Die Ähnlichkeiten gehen soweit, dass sich `EqualValue` von `Copy` nur in einer Zeile unterscheidet, genauer in einem einzigen Zeichen.

```
// compare two dates
bool CDate::operator ==(const CDate &date) const
{
    // compare date and time attributes
    return (m_nDay    == date.m_nDay    &&
            m_nMonth  == date.m_nMonth  &&
            m_nYear   == date.m_nYear   &&
            m_nHour   == date.m_nHour   &&
            m_nMinute == date.m_nMinute &&
            m_nSecond == date.m_nSecond);
}
```

PRE: keine
POST: keine

```
// virtual, see operator==
bool CDate::EqualValue(CBasicClass *pClass) const
{
    // cast to CDate
    CDate *date;
    date = dynamic_cast<CDate*>(pClass);

    // invalid class (is NULL when pClass is not a CDate)
    if (date == NULL)
        return false;

    // use non virtual reference based copy
    return operator==(*date);
}
```

PRE: Übergabe eines Zeigers auf ein anderes `CDate` Objekt
POST: keine

Eine große Gruppe in jeder Klasse stellen die `Set`.../`Get`... –Methoden dar. Ihr Aufbau ist grundsätzlich trivial: `Set`-Methoden erlauben den Schreibzugriff auf das entsprechende Attribut, dabei wird stets das Ergebnis

einer Überprüfung durch `IsValid` zurückgeliefert. Gemäß den C++ -Konventionen kann dieser Rückgabewert natürlich ignoriert werden. Da überwiegend Lesezugriffe erfolgen, halte ich diese Technik für akzeptabel.

Die `Get`-Methoden sind noch simpler, sie lesen lediglich ein Attribut aus und geben den Wert zurück.

```
// set attributes, returns validility of date
bool CDate::SetDay(unsigned int nDay)
{
    m_nDay = nDay;
    return IsValid();
}
bool CDate::SetMonth(unsigned int nMonth)
{
    m_nMonth = nMonth;
    return IsValid();
}
bool CDate::SetYear(unsigned int nYear)
{
    m_nYear = nYear;
    return IsValid();
}
bool CDate::SetHour(unsigned int nHour)
{
    m_nHour = nHour;
    return IsValid();
}
bool CDate::SetMinute(unsigned int nMinute)
{
    m_nMinute = nMinute;
    return IsValid();
}
bool CDate::SetSecond(unsigned int nSecond)
{
    m_nSecond = nSecond;
    return IsValid();
}

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

Es gibt für alle Get/Set-Methoden keine PRE- oder POST-Conditions.

Etwas Vorsicht ist dennoch bei einigen `Set`-Methoden erforderlich, nämlich wenn die Klasse ein Attribut `m_dtAlterationDate` besitzt. Dieses muss dann auch aktualisiert werden, was dank des Standard-Konstruktors von `CDate` sehr einfach geht:

```
bool CContract::SetWording(string strWording)
{
    m_strWording = strWording;

    // update alteration date
```

```
        CDate dtNow;
        m_dtAlterationDate = dtNow;

        return IsValid();
    }
```

4.3 Testplan und Testergebnisse

4.3.1 Testplan

Das Haupt-Testprinzip war der *Bottom-Up*-Test. Ich halte ihn für dieses Praktikum als besonders geeignet, da alle Klassen komplett neu entwickelt wurden (ich lasse die STL-Container einmal außen vor, da sie aufgrund ihrer enormen Verbreitung als korrekt getestet angesehen werden müssen). Wenn ich also eine Basisklasse weitgehend getestet habe und von ihrer Korrektheit überzeugt bin, so teste ich als nächstes alle von ihr abgeleiteten Klassen und arbeite mich auf diesem Wege zu den komplexen vor.

Der vorliegende Quellcode erlaubt mir, auf Pseudomoduln zu verzichten, weil die Klassen keinen besonders großen Umfang aufweisen und die Verschachtelungstiefe gering ist. Jede Emulation würde komplexer als das Original ausfallen.

Ich unterteile meinen Testplan ferner in zwei Kategorien: Einzeltests und Kombinationstests. Erstere sind als Funktionstests recht trivial, so dass ich sie manuell durchführte, sowohl Codeinspektion als auch Code-Walkthrough geschahen zum Zeitpunkt der initialen Programmierung. Eine Ausnahme davon bilden lediglich die Polymorphiebeziehungen, die ich eher in die Kategorie Kombinationstest einordne, da die Auswirkungen erst im Zusammenspiel mehrerer Klassen auftreten.

Eine Art von Grenzwertanalyse und Äquivalenzklassenanalyse stellen die `IsValid`-Methoden der einzelnen Klassen dar. Die Testumgebung erlaubt aufgrund ihrer Flexibilität ein umfassendes Error Guessing bzw. das gezielte Testen von vermutlich kritischen Punkten. Somit kann ich auch keine starren Graphen oder ähnliches angeben, da der Tester vollkommen frei in konkreten Teststrategie ist. Ich gebe lediglich einen Rahmen vor.

Hinweis: Ich habe aufgrund ihres Umfanges darauf verzichtet, die Testprotokolle im Anhang abzudrucken. Sie befinden sich vollständig auf der beiliegenden CD und sind in reinem ASCII-Textformat mit der Dateiendung .log abgespeichert worden.

4.3.2 Implementation

Die Testumgebung ist als eine eigene Klasse `CTest` umgesetzt worden. Streng genommen müsste man auch für sie eine Testumgebung schreiben, um ihre Korrektheit sicherzustellen. Konsequenterweise sollte dann die Testumgebung der Testumgebung ebenfalls eine Testumgebung besitzen usw. was in eine Endlosschleife führt. Somit bleibt nichts anderes übrig, als die jetzt vorhandene (einzige ...) Testumgebung nach bestem Wissen möglichst fehlerfrei zu codieren. Da dies aber sehr komplex werden kann, wenn man jede Unwägbarkeit überprüft, habe ich mich entschlossen, von möglichst korrekten Eingaben des Benutzers auszugehen. Er hat dafür Sorge zu tragen, dass falsche Eingaben unterbleiben.

Die Ausführung der Testumgebung erfolgt im Textmodus, es werden keinerlei graphische Oberflächen explizit verwendet. Man kann eine Protokolldatei erstellen lassen, die sämtliche Bildschirmausgaben und Tastatureingaben aufzeichnet. Der Dateiname ist frei wählbar, ggf. existierende gleichnamige Dateien werden überschrieben.

4.3.3 Ergebnisse

Im folgenden werde ich einige „Stolpersteine“ aufzählen, die durch den Einsatz der Testumgebung aufgedeckt und eliminiert werden konnten.

4.3.3.1 ClassnameOf

Die Testumgebung war für mich besonders bei der korrekten Implementation der Polymorphiebeziehungen hilfreich. Das folgende Beispiel zeigt den Sachverhalt, indem ich zwei Objekte erzeuge, eines vom Typ `CDate`

und das zweite als `CBankDocument` (siehe auch ClassnameOf.log), die problematische Stelle ist rot markiert, wobei ich die blaue Zeile interessant finde, dort konnte korrekt aufgelöst werden, da nicht `CBasicClass` involviert war:

```
Logfile created: 31.03.2001 - 08:13:43

1 - Create new object
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: (none)
Please choose: 1

1 - CDate
2 - CBankDocument
3 - CContract
4 - CForm
5 - CStandingOrder
0 - none

Please choose the class type: 1
Please choose type of construction (0=default, 1=copy, 2=init): 0

1 - Create new object
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: (none)
Please choose: 1

1 - CDate
2 - CBankDocument
3 - CContract
4 - CForm
5 - CStandingOrder
0 - none

Please choose the class type: 2
Please choose type of construction (0=default, 1=copy, 2=init): 0

1 - Create new object
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: (none)
Please choose: 2

There are 2 classes in use.
  class no. 0
    type = CBankDocument  valid=0
     DEBUG info for 'CDate'
      m_nDay    = 31
      m_nMonth  = 3
      m_nYear   = 2001
      m_nHour   = 10
      m_nMinute = 38
      m_nSecond = 30
  class no. 1
    type = CBasicClass valid=0
     DEBUG info for 'CBankDocument'
      m_strTitle           =
      m_strAuthor          =
      m_strKey             =
```

```
      m_dtDateOfFoundation = DEBUG info for 'CDate'
      m_nDay    = 0
      m_nMonth  = 0
      m_nYear   = 0
      m_nHour   = 0
      m_nMinute = 0
      m_nSecond = 0


1 - Create new object
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: (none)
Please choose: 0
```

In der alten Fassung war `ClassnameOf` eine als `static` deklarierte Methode. Meine Idee hinter dieser Umsetzung war, dass Klassenmethoden generelle Vorteile haben, was ihre Verwendung angeht (wie z.B. dass kein Objekt existieren muss, um diese Methode aufzurufen). Wie man am Protokoll erkennen kann, ist Polymorphie unbedingt notwendig, um den „echten“ Namen eines Objektes zu ermitteln, selbst wenn der Typ durch Casting-Operationen verloren ging. Im neuen Testprotokoll sind die kritischen roten Zeilen dann korrekt (Protokoll aus Platzgründen gekürzt; vollständig nachzulesen in ClassnameOf2.log):

```
Logfile created: 26.03.2001 - 14:24:50

...

There are 2 classes in use.
  class no. 0
    type = CDate  valid=0
     DEBUG info for 'CDate'
      m_nDay    = 31
      m_nMonth  = 3
      m_nYear   = 2001
      m_nHour   = 10
      m_nMinute = 38
      m_nSecond = 30
  class no. 1
    type = CBankDocument  valid=0
     DEBUG info for 'CBankDocument'
      m_strTitle           =
      m_strAuthor          =
      m_strKey             =
      m_dtDateOfFoundation = DEBUG info for 'CDate'
      m_nDay    = 0
      m_nMonth  = 0
      m_nYear   = 0
      m_nHour   = 0
      m_nMinute = 0
      m_nSecond = 0
```

Der Einsatz von `virtual` verbietet zwar Klassenmethoden, aber der Verlust wird durch echte Polymorphie mehr als wett gemacht.

4.3.3.2 Show

Die Testumgebung selbst ist ein schönes Beispiel für die Verwendung von Polymorphie. Da beliebig viele Objekte aller zur Verfügung gestellten Klassen angelegt werden können, ist auf den ersten Blick nicht eindeutig, wie diese Objekte auf dem Bildschirm dargestellt werden können. Da aber alle von der gleichen Basisklasse `CBasicClass` abstammen und dort ein rein virtueller Prototyp für `Show()` definiert wurde, kann die Testumgebung auf eben diese Methode zurückgreifen. Die VMT stellt sicher, dass – obwohl nur mit Zeigern vom Typ `CBasicClass*` hantiert wird – die korrekte `Show`-Implementation aufgerufen wird.

4.3.3.3 m_dtAlterationDate

In der Aufgabenstellung wird leider nicht genau die Bedeutung von `m_dtAlterationDate` deutlich abgegrenzt. Meine derzeitige Implementation ändert dieses Attribut jedes Mal, wenn sich der Zustand der Klasse ändert. Leider ist `m_dtAlterationDate` als `private` deklariert, so dass abgeleitete Klassen keinen Schreibzugriff haben, aber jeweils die Möglichkeit des Auslesen per `GetAlterationDate()` besteht. Dieses Problem taucht eigentlich nur bei `CStandingOrder` auf. In meinen Augen sind zwar Intervall und Höhe eines Dauerauftrages Bestandteil eines Vertrages, aber der Vertragskern selbst wird nicht berührt (z.B. der genaue Wortlaut). In Hinsicht auf sich leicht ändernde Daueraufträge, wie sie bei jährlichen Mietsteigerungen normal sind, halte ich die momentane Verhaltensweise meiner Klassen für gerechtfertigt. Die Datei AlterationDate.log demonstriert das Verhalten.

4.3.3.4 Copy/EqualValue

Der Kombinationstest unter Verwendung von `Copy` und `EqualValue` ist für alle Klassen weitgehend identisch, ich habe ein Protokoll für `CBankDocument` exemplarisch unter dem Namen CopyEqualValue.log erstellt.

5 Wertung des erreichten Ergebnisses

Die hier vorgestellte Lösung ist in der Lage, alle Anforderungen der Aufgabenstellung zu erfüllen. Da der Testplan keinerlei Fehler mehr offenbart und die Klassen zum Teil schon in mehreren Hausaufgaben erfolgreich eingesetzt wurden (insbesondere `CDate`) sehe ich das System als korrekt an.

Mit der Verwendung meiner Klassenstruktur ist man in der Lage, als Bank eine Kundenverwaltung mit den geforderten Fähigkeiten durchzuführen. Zwar fehlt noch eine Persistenzhaltung, aber die zusätzlich eingebundenen Debugging-Möglichkeiten erhöhen die Usability für den Programmierer erheblich. Die Klassen wurden im vollen von der Aufgabenstellung geforderten Umfang umgesetzt, die Polymorphiebeziehungen habe ich ebenso erläutert. In meiner Dokumentation finden sich die jeweiligen Vor- und Nachbedingungen der einzelnen Funktionen und ihre Arbeitsweise wird sowohl im Quellcode als auch in der Dokumentation dargelegt. Der interaktive Testrahmen erlaubt eine bequeme Verifizierung, die auch problemlos von Nicht-Programmierern durchführbar ist. Eine umfassende Protokollierung rundet die Testumgebung ab.

6 Anhang

6.1 Vollständige Aufgabenstellung

Abbildung 11: Aufgabenstellung, Seite 1

Universität Potsdam
Institut für Informatik
Professur Software Engineering
Prof. Dr. Erika Horn

Aufgaben für das Praktikum zur Lehrveranstaltung
"Softwarebauelemente I"

Entwickeln Sie eine objektorientierte Lösung für die Problemstellung "persistente Verwaltung der Daten einer Dokumentenmappe" in den Programmiersprachen C++, Java oder Object Pascal.
Die Bausteine des Programmsystem sollen in Teilsystemen abgelegt werden.
Bausteine des Programmsystems sind mindestens die Klassen

CBankDocument	Wurzelklasse der Dokumentenhierarchie
CForm	allgemeines Formular
CStandingOrder	Dauerauftrag
CContract	Vertrag
CFolder	Mappe von Dokumenten für einen Kunden

Die Beschreibung der Klassen ist als Anhang beigefügt.
Bestimmen Sie aus der Beschreibung die Klassen- und Teilsystemstruktur sowie die notwendigen Polymorphiebeziehungen.
Treffen Sie Implementationsentscheidungen hinsichtlich der Interfaces der Klassen und der Realisierung der Mengenorganisation.
Bestimmen und Beschreiben Sie die Vor- und Nachbedingungen der Methoden.
Zur Absicherung von Softwarequalitätsparametern des Programmsystems ist eine Testumgebung, die eine von Ihnen zu bestimmende Teststrategie unterstützt, zu entwerfen und zu implementieren.

Anhang

Prozedurparameternamen und ihre Semantik

Klasse CDate	
dat	date
da	day
mo	mo
ye	year
se	second
mi	minute
hou	hour

Klasse CBankDocument	
bdoc	bankdocument
aut	author
tit	title
key	key
titl	titel

Klasse CForm	
form	form
prv	proved
sig	signed
fty	formality

Klasse CStandingOrder	
sord	standingorder
amo	amount
sub	subject
sou	source
rec	recipient
int	interval

Klasse CContract	
cont	contract
numb	registration number
ada	alteration date
prv	proved
sig	signed
word	wording

Klasse CFolder	
fold	folder
cinf	customerinformation
card	cardinality

Abbildung 12: Aufgabenstellung, Seite 2

Klasse CDate

```
class   CDate    ... classend    // Date&Time
```

Beschreibung der Klasse CBankDocument

```
class CBankDocument
import CDate;

  attributes
    private Author: String,
    private DateOfFoundation: CDate,
    private Title: String,
    private Key: String;

  operations
    public +CBankDocument( ),
    // Constructs an object of 'CBankDocument'.

    public +CBankDocument(in aut: String, in tit: String, in key: String),
    // Constructs an object of 'CBankDocument' and
    // initializes the attributes.

    public +CBankDocument(in bdoc: CBankDocument),
    // Constructs an object of 'CBankDocument' and
    // initializes the attributes with the
    // attributes of bdoc.

    public virtual -CBankDocument( ),
    // Destructs the object.

    public virtual Generate(out bdoc: CBankDocument),
    // Clones an object.

    public virtual ClassInvariant(out t: Boolean),
    // Proofs the invariance. Returns "TRUE", if the invariance
    // is valid (strings are not empty), else returns "FALSE".

    public virtual Equal(in bdoc: CBankDocument, out t: Boolean),
    // Compares two objects. Returns "TRUE", if the names
    // of the Objects are equal, else returns "FALSE".

    public virtual EqualValue(in bdoc: CBankDocument, out t: Boolean),
    // Compares two objects. Returns "TRUE", if the attributes
    // of the Objects are equal, else returns "FALSE".

    public EqualKey(in bdoc: CBankDocument, out t: Boolean),
    // Compares two objects. Returns "TRUE", if the keys
    // of the Objects are equal, else returns "FALSE".

    public virtual KeyOf(out key: String),
    // Delivers the Key of the object.

    public virtual Copy(in bdoc: CBankDocument, out t: Boolean),
    // Copies the object, if it is possible.

    public GetAuthor(out aut: String),

    public GetDateOfFoundation(out dat: CDate),

    public GetTitle(out tit: String),

    public GetKey(out key: String),

    public virtual Show( );
    // Shows the attributes of a BankDocument.

// Valid cases
    //
    // Case 1:
    // require:   The invariance (ClassInvariant) of bdoc2 as an instance of
    //             'CBankDocument' is "TRUE".
    //             The object bdoc1 is an instance of 'CBankDocument'.
    // sequence:
    //            bdoc1.Copy(bdoc2, t1);
    //            bdoc1.ClassInvariant(t2);
    //            bdoc1.EqualValue(bdoc2, t3)
    // ensure:    The values of t1, t2 and t3 are "TRUE".
    //

classend
```

Beschreibung der Klasse CForm

```
class CForm: inherit public CBankDocument;
import CDate, CBankDocument;

  attributes
    private Proved: Boolean,
    private Signed: Boolean,
    private Formality: String,
    private AlterationDate: CDate;


  operations
    public +CForm( ),
    // Constructs an object of 'CForm'.

    public +CForm(in aut: String, in tit: String, in key: String,
                  in prv: Boolean, in sig: Boolean, in fty: String),
    // Constructs an object of 'CForm' and
    // initializes the attributes.

    public +CForm(in form: CForm),
    // Constructs an object of 'CForm' and
    // initializes the attributes with the
    // attributes of form.

    public virtual -CForm( ),
    // Destructs the object.

    public virtual Generate(out form: CForm),
    // Clones an object.
```

Abbildung 13: Aufgabenstellung, Seite 3

```
    public virtual ClassInvariant(out t: Boolean),
    // Proofs the invariance. Returns "TRUE", if the invariance
    // is valid (strings are not empty and the invariance of the
    // inherited class is valid), else returns "FALSE".

    public virtual Equal(in form: CForm, out t: Boolean),
    // Compares two objects. Returns "TRUE", if the names
    // of the Objects are equal, else returns "FALSE".

    public virtual EqualValue(in form: CForm, out t: Boolean),
    // Compares two objects. Returns "TRUE", if the attributes
    // of the Objects are equal, else returns "FALSE".

    public virtual KeyOf(out key: String),
    // Delivers the Key of the object.

    public virtual Copy(in form: CForm, out t: Boolean),
     // Copies the object, if it is possible.

    public GetProved(out prv: Boolean),

    public SetProved(in prv: Boolean),

    public GetSigned(out  sig: Boolean),

    public SetSigned(in  sig: Boolean),

    public GetFormality(out fty: String),

    public GetAlterationDate(out dat: CDate),

    public virtual Show( );

// Valid cases
    //
    // Case 1:
    // require:  The invariance (ClassInvariant) of form2 as an instance of
    //           'CForm' is "TRUE".
    //           The object form1 is an instance of 'CForm'.
    // sequence:
    //           form1.Copy(form2, t1);
    //           form1.ClassInvariant(t2);
    //           form1.EqualValue(form2, t3)
    // ensure:   The values of t1, t2 and t3 are "TRUE".
    //

classend
```

Beschreibung der Klasse CStandingOrder

```
class CStandingOrder: inherit public CForm;
import CDate, CForm;

  attributes
    private Amount: Ordinal,
    private Subject: String,
    private Source: String,
    private Recipient: String,
    private Interval: Ordinal;


  operations
    public +CStandingOrder( ),
    // Constructs an object of 'CStandingOrder'.

    public +CStandingOrder(in aut: String, in tit: String, in key: String,
                           in prv: Boolean, in sig: Boolean, in fty: String,
                           in amo: Ordinal, in sub: String, in sou: String,
                           in rec: String, in int: Ordinal),
    // Constructs an object of 'CStandingOrder' and
    // initializes the attributes.

    public +CStandingOrder(in sord: CStandingOrder),
    // Constructs an object of 'CStandingOrder' and
    // initializes the attributes with the
    // attributes of sordi.

    public virtual -CStandingOrder( ),
    // Destructs the object.

    public virtual Generate(out sord: CStandingOrder),
    // Clones an object.

    public virtual ClassInvariant(out t: Boolean),
    // Proofs the invariance. Returns "TRUE", if the invariance
    // is valid (strings are not empty, the ordinals are not zero
    // and the invariance of the inherited class is valid),
    // else returns "FALSE".

    public virtual Equal(in sord: CStandingOrder, out t: Boolean),
    // Compares two objects. Returns "TRUE", if the names
    // of the Objects are equal, else returns "FALSE".

    public virtual EqualValue(in sord: CStandingOrder, out t: Boolean),
    // Compares two objects. Returns "TRUE", if the attributes
    // of the Objects are equal, else returns "FALSE".

    public virtual KeyOf(out key: String),
    // Delivers the Key of the object.

    public virtual Copy(in sord: CStandingOrder, out t: Boolean),
     // Copies the object, if it is possible.

    public GetAmount(out amo: Ordinal),

    public SetAmount(in amo: Ordinal),
```

Abbildung 14: Aufgabenstellung, Seite 4

```
    public GetSubject(out sub: String),

    public GetSource(out sou: String),

    public GetRecipient(out rec: String),

    public GetInterval(out int: Ordinal),

    public SetInterval(in int: Ordinal),

    public virtual Show( );

// Valid cases
    //
    // Case 1:
    // require:   The invariance (ClassInvariant) of sord2 as an instance of
    //             'CStandingOrder' is "TRUE".
    //             The object sord1 is an instance of 'CStandingOrder'.
    // sequence:
    //            sord1.Copy(sord2, t1);
    //            sord1.ClassInvariant(t2);
    //            sord1.EqualValue(sord2, t3)
    // ensure:    The values of t1, t2 and t3 are "TRUE".
    //

classend
```

Beschreibung der Klasse CContract

```
class CContract: inherit public CBankDocument;
import CDate, CBankDocument;

  attributes
    private RegistrationNumber: Ordinal,
    private AlterationDate: CDate,
    private Proved: Boolean,
    private Signed: Boolean,
    private Wording: String;


  operations
    public +CContract( ),
    // Constructs an object of 'CContract'.

    public +CContract(in aut: String, in tit: String, in key: String,
                      in numb: Ordinal, in prv: Boolean, in sig: Boolean,
                      in word: String),
    // Constructs an object of 'CContract' and
    // initializes the attributes.

    public +CContract(in cont: CContract),
    // Constructs an object of 'CContract' and
    // initializes the attributes with the
    // attributes of cont.

    public virtual -CContract( ),
    // Destructs the object.

    public virtual Generate(out cont: CContract),
    // Clones an object.

    public virtual ClassInvariant(out t: Boolean),
    // Proofs the invariance. Returns "TRUE", if the invariance
    // is valid (strings are not empty and the invariance of the
    // inherited class is valid), else returns "FALSE".

    public virtual Equal(in cont: CContract, out t: Boolean),
    // Compares two objects. Returns "TRUE", if the names
    // of the Objects are equal, else returns "FALSE".

    public virtual EqualValue(in cont: CContract, out t: Boolean),
    // Compares two objects. Returns "TRUE", if the attributes
    // of the Objects are equal, else returns "FALSE".

    public virtual KeyOf(out key: String),
    // Delivers the Key of the object.

    public virtual Copy(in cont: CContract, out t: Boolean),
     // Copies the object, if it is possible.

    public GetRegistrationNumber(out numb: Ordinal),

    public GetAlterationDate(out dat: CDate),

    public GetProved(out prv: Boolean),

    public SetProved(in prv: Boolean),

    public GetSigned(out  sig: Boolean),

    public SetSigned(in  sig: Boolean),

    public GetWording(out word: String),

    public SetWording(in word: String),

    public virtual Show( );

// Valid cases
    //
    // Case 1:
    // require:   The invariance (ClassInvariant) of cont2 as an instance of
    //             'CContract' is "TRUE".
    //             The object cont1 is an instance of CContract.
    // sequence:
    //            cont1.Copy(cont2, t1);
    //            cont1.ClassInvariant(t2);
    //            cont1.EqualValue(cont2, t3)
    // ensure:    The values of t1, t2 and t3 are "TRUE".
    //

classend
```

Abbildung 15: Aufgabenstellung, Seite 5

Beschreibung der Klasse CFolder

```
class CFolder
import CDate, CBankDocument;

  types TSetOfBankDocuments = ordered set of CBankDocument;

  attributes
    private CustomerInformation: String,
    private DateOfFoundation: CDate,
    private Set: TSetOfBankDocuments,
    private Count: Ordinal,
    private Cursor: Ordinal;

  operations
    public +CFolder( ),
    // Constructs an object of 'CFolder'.

    public +CFolder(in cinf: String),
    // Constructs an object of 'CFolder' and
    // initializes the attributes.


    public +CFolder(in fold: CFolder),
    // Constructs an object of 'CFolder' and initializes the attributes
    // with the attributes of fold.

    public virtual -CFolder( ),
    // Destructs the object.

    public virtual Generate(out fold: CFolder),
    // Clones an object.

    public virtual ClassInvariant(out t: Boolean),
    // Proofs the invariance. Returns "TRUE", if the set contains
    // each BankDocument only once and the invariance of each BankDocument
    // is "TRUE", else returns "FALSE".

    public virtual Equal(in fold: CFolder, out t: Boolean),
    // Compares two objects. Returns "TRUE", if the names
    // of the Objects are equal, else returns "FALSE".

    public virtual EqualValue(in fold: CFolder, out t: Boolean),
    // Compares two sets. Returns "TRUE", if both sets contains
    // the same elements, else returns "FALSE".

    public EqualKey(in fold: CFolder, out t: Boolean),
    // Compares two objects. Returns "TRUE", if the keys
    // of the Objects are equal, else returns "FALSE".

    public virtual KeyOf(out key: String),
    // Delivers the Key of the object.

    public virtual Copy(in fold: CFolder, out t: Boolean),
    // Copies the object, if it is possible.

    public CustomerInformation(out cinf: String),
    // Delivers folders's customer information.

    public GetDateOfFoundation(out dat: CDate),
    // Delivers folders's date of foundation.

    public Card(out crd: Ordinal),
    // Delivers the cardinality of set.

    public Insert(in bdoc: CBankDocument, out t: Boolean),
    // Inserts a BankDocument in the set, if it is possible.

    public GetFirst(out bdoc: CBankDocument, out t: Boolean),
    // Gets the first BankDocument in the set, if it is possible.

    public GetNext(in bdoc: CBankDocument, out t: Boolean),
    // Gets the next BankDocument in the set, if it is possible.

    public Find(in bdoc: CBankDocument, out t: Boolean),
    // Sets the cursor on an object in the set, which value is
    // equal to per, if it is possible.

    public GetActual(out bdoc: CBankDocument, out t: Boolean),
    // Delivers the actual object in the set, if it is possible.

    public Scratch(out t: Boolean),
    // Scratches the actual object in the set, if it is possible.

    public virtual Show( );
```

Abbildung 16: Aufgabenstellung, Seite 6

```
// Valid cases
//
// Case 1:
// require:  The invariance (ClassInvariant) of fold2 as an instance of
//           CFolder is "TRUE".
//           The object fold1 is an instance of 'CFolder'.
// sequence:
//           fold1.Copy(fold2, t1);
//           fold1.ClassInvariant(t2);
//           fold1.EqualValue(fold2, t3)
// ensure:   The values of t1, t2 and t3 are "TRUE".
//
// Case 2:
// require:  Object fold as an instance of 'CFolder' is not full
//            and invariance (ClassInvariant) of bdoc1 is "TRUE".
// sequence:
//           fold.Insert(bdoc1, t1);
//           fold.Find(bdoc1, t2);
//           fold.GetActual(bdoc2, t3)
// ensure:   The elements bdoc1 and bdoc2 have the same value
//           (EqualValue).
//           The values of t2 and t3 are "TRUE".
//
// Case 3:
// require:  Object fold as an instance of CFolder contains bdoc (Find).
// sequence:
//           fold.Find(bdoc, t1);
//           fold.Scratch(t2);
//           fold.Find(bdoc, t3)
// ensure:   The value of t2 is "TRUE".
//           The value of t3 is "FALSE".
//
// Case 4:
// require:  Object fold as an instance of 'CFolder'  is not full.
//            It not contains bdoc (Find), which invariance
//            (ClassInvariant) is "TRUE".
//           Object fold has the cardinality (Card) of n1.
// sequence:
//           fold.Insert(bdoc, t1);
//           fold.Card(n2)
// ensure:   The value of n2 is equal n1 + 1.
//
// Case 5:
// require:  Object fold as an instance of 'CFolder' contains bdoc
//           (Find)
//            and has the cardinality (Card) of n1.
// sequence:
//           fold.Find(bdoc, t1);
//           fold.Scratch(t2);
//           fold.Card(n2)
// ensure:   The value of n2 is equal n1 - 1.


classend
```

Abbildung 17: Aufgabenstellung, Reduzierung

Reduktion des Inhalts der Praktikumsaufgabe für das Wintersemester

Generell entfallen:

- Persistenzhaltung
- KlasseCFolder

Entfallende Methoden:

- Generate
- ClassInvariant
- EqualKey
- KeyOf

Letzter Termin für die Abgabe der Dokumentation:

20.04.2001

Hausaufgaben

In die ≥ 50% der einzureichenden Hausaufgaben gehen alle Aufgaben der Komplexe modulare (M) und objektorientierte (O) Programmierung der Aufgabensammlung ein.

6.2 Umfang des Praktikums

Das Praktikum besteht im wesentlichen aus 2 Teilen: Dem Quellcode mit einer ausführbaren Testumgebung und einer Dokumentation, welche als PDF-, HTML- und Word-Datei vorliegt.

Auf dem mitgelieferten Datenträger sind folgende Dateien:

Verzeichnis Projekt:

```
.              <DIR>        18.04.01  14:04 .
..             <DIR>        18.04.01  14:04 ..
BANKDO~1 CPP         4.155  18.04.01  13:43 BankDocument.cpp
CONTRACT CPP         5.246  02.04.01  16:06 Contract.cpp
DATE     CPP         6.215  18.04.01  13:43 Date.cpp
FORM     CPP         4.292  02.04.01  16:07 Form.cpp
PRAKTI~1 CPP           914  31.03.01  13:37 Praktikum.cpp
STANDI~1 CPP         5.124  02.04.01  16:07 StandingOrder.cpp
TEST     CPP        15.831  18.04.01  13:44 Test.cpp
PRAKTI~1 DSP         5.238  25.03.01  16:26 Praktikum.dsp
PRAKTI~1 DSW           571  06.03.01  19:42 Praktikum.dsw
BANKDO~1 H           1.975  18.04.01  13:44 BankDocument.h
BASICC~1 H           1.486  18.04.01  13:44 BasicClass.h
CONTRACT H           2.074  18.04.01  13:44 Contract.h
DATE     H           2.979  18.04.01  13:44 Date.h
FORM     H           1.836  18.04.01  13:44 Form.h
STANDI~1 H           2.226  18.04.01  13:44 StandingOrder.h
TEST     H           1.915  18.04.01  13:44 Test.h
ALTERA~1 LOG         3.827  18.04.01  13:49 AlterationDate.log
CLASSN~1 LOG         1.822  31.03.01  10:39 ClassnameOf.log
CLASSN~2 LOG         1.824  31.03.01  10:38 ClassnameOf2.log
COPYEQ~1 LOG         2.059  18.04.01  13:56 CopyEqualValue.log
DEBUG          <DIR>        18.04.01  14:06 Debug
DOKUME~1       <DIR>        18.04.01  14:06 Dokumentation
PRAKTI~1 NCB        99.328  18.04.01  14:05 Praktikum.ncb
PRAKTI~1 OPT        48.640  18.04.01  14:05 Praktikum.opt
RELEASE        <DIR>        18.04.01  14:06 Release
PRAKTI~1 PJT         2.921  27.03.01  10:09 praktikum.pjt
PRAKTI~1 PLG           265  18.04.01  14:04 Praktikum.plg
25 Datei(en)               222.763 Bytes
```

Verzeichnis Dokumentation:

```
.              <DIR>        23.03.01  20:58 .
..             <DIR>        23.03.01  20:58 ..
PRAKTI~1 DOC     2.151.936  18.04.01  14:19 Praktikum.doc
AUFGAB~1 PNG       295.973  23.03.01  21:33 Aufgaben2.png
AUFGAB~3 PNG       304.978  23.03.01  21:34 Aufgaben3.png
AUFGAB~2 PNG       237.910  23.03.01  21:36 Aufgaben1.png
AUFGAB~4 PNG       283.163  23.03.01  21:34 Aufgaben4.png
AUFGAB~5 PNG       266.740  23.03.01  21:35 Aufgaben5.png
AUFGAB~6 PNG       164.991  23.03.01  21:36 Aufgaben6.png
AUFGAB~7 PNG       146.062  23.03.01  21:55 Aufgaben-Reduzierung.png
         9 Datei(en)             3.851.753 Bytes
```

Verzeichnis Release:

```
.              <DIR>        19.03.01  17:27 .
..             <DIR>        19.03.01  17:27 ..
VC60     IDB        50.176  18.04.01  14:04 vc60.idb
PRAKTI~1 PCH     2.667.740  18.04.01  14:03 Praktikum.pch
BANKDO~1 OBJ        56.691  18.04.01  14:03 BankDocument.obj
BASICC~1 OBJ        10.749  19.03.01  17:30 BasicClass.obj
DATE     OBJ        87.814  18.04.01  14:03 Date.obj
PRAKTI~1 OBJ        32.975  18.04.01  14:03 Praktikum.obj
PRAKTI~1 EXE       233.472  18.04.01  14:04 Praktikum.exe
CONTRACT OBJ        94.405  18.04.01  14:03 Contract.obj
FORM     OBJ        89.984  18.04.01  14:03 Form.obj
STANDI~1 OBJ        93.742  18.04.01  14:03 StandingOrder.obj
TEST     OBJ       173.565  18.04.01  14:04 Test.obj
12 Datei(en)             3.591.313 Bytes
```

Verzeichnis Debug:

```
.              <DIR>        06.03.01  19:42 .
..             <DIR>        06.03.01  19:42 ..
VC60     IDB       123.904  18.04.01  14:04 vc60.idb
VC60     PDB       151.552  18.04.01  13:44 vc60.pdb
PRAKTI~1 PCH     2.667.556  18.04.01  13:44 Praktikum.pch
PRAKTI~1 OBJ       115.138  18.04.01  13:44 Praktikum.obj
TEST     OBJ       424.606  18.04.01  13:45 Test.obj
BANKDO~1 OBJ       167.329  18.04.01  13:44 BankDocument.obj
CONTRACT OBJ       263.113  18.04.01  13:44 Contract.obj
DATE     OBJ       250.132  18.04.01  13:44 Date.obj
FORM     OBJ       255.144  18.04.01  13:44 Form.obj
STANDI~1 OBJ       259.376  18.04.01  13:44 StandingOrder.obj
PRAKTI~1 ILK     1.123.840  18.04.01  13:45 Praktikum.ilk
PRAKTI~1 EXE       741.461  18.04.01  13:45 Praktikum.exe
PRAKTI~1 PDB     1.508.352  18.04.01  13:45 Praktikum.pdb
14 Datei(en)             8.051.503 Bytes
```

Hinweis: Um die beiliegende CD benutzerfreundlicher zu machen, befindet sich darauf noch eine PDF- und eine HTML-Version dieser Dokumentation. Leider sind Konvertierungsfehler nicht immer auszuschließen.
Ebenso ist die CD AutoStart-fähig, d.h. nach Einlegen der CD öffnet sich der jeweils installierte Browser und zeigt die drei vorhandenen Typen der Dokumentation zur Auswahl an.

6.3 Abbildungsverzeichnis

6.4 Literaturverzeichnis

[Stroustrup, 1998] Bjarne Stroustrup: *Die C++ Programmiersprache*, 3. Auflage, Addison-Wesley, Reading, Mass. 1998, ISBN 3-8273-1296-5

[MSDN, 2001] diverse Autoren, *Microsoft Developer Network*, http://msdn.microsoft.com

[Balzert, 2001] Helmut Balzert: *Lehrbuch der Software-Technik*, 2.Auflage, Spektrum Akademischer Verlag, Heidelberg 2001, ISBN 3-8274-0480-0

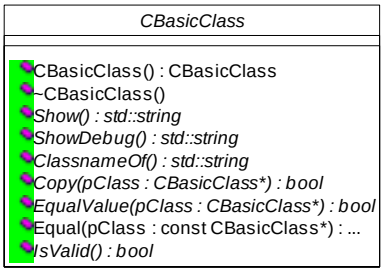
[Willms, 1995] Gerhard Willms: *Das C Grundlagen Buch*, Data Becker, Düsseldorf 1995, ISBN 3-8158-1208-9

[Sedgewick, 1992] Robert Sedgewick: *Algorithmen*, 2.Auflage, Addison-Wesley, Reading, Mass. 1992, ISBN 3-89319-402-9

6.5 Quellcode

6.5.1 CBasicClass

Abbildung 18: Interface von CBasicClass



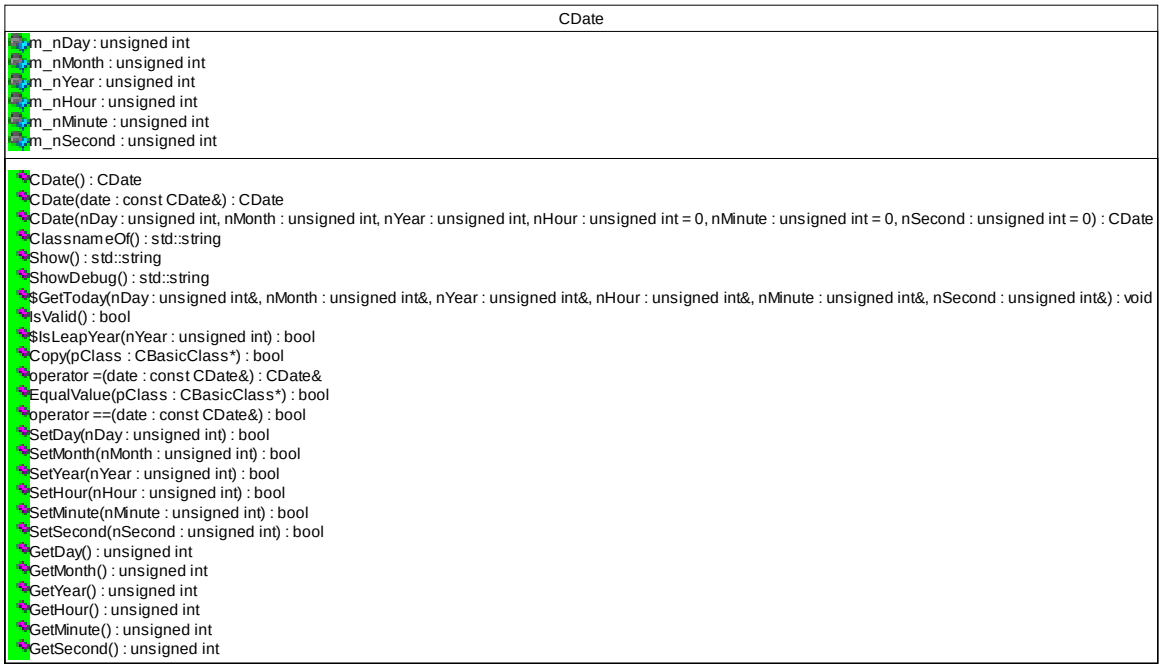
6.5.1.1 BasicClass.h

```
////////////////////////////////////////////////////////
// Softwarebauelemente I, Praktikum
//
// author:          Stephan Brumme
// last changes:    March 27, 2001


#if !defined(AFX_BASICCLASS_H__C0EA0EA0_0BD2_11D5_9BB7_B22A84F06321__INCLUDED_)
#define AFX_BASICCLASS_H__C0EA0EA0_0BD2_11D5_9BB7_B22A84F06321__INCLUDED_

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

// disable compiler warning concerning RTTI
// #pragma warning( disable : 4541 )



#include <string>
#include <sstream>
using namespace std;



class CBasicClass
{
public:
    CBasicClass() {};
    virtual ~CBasicClass() {};

    // show attributes
    virtual string Show() const = 0;
    // shows all internal attributes
    virtual string ShowDebug() const = 0;

    // return class name
    virtual string ClassnameOf() const = 0;


    // copy constructors
    // non-virtual
//  CBasicClass& operator = (const CBasicClass &myclass);
    // virtual
    virtual bool Copy (CBasicClass *pClass) = 0;

    // compare two dates
    // non-virtual
//  bool operator ==(const CBasicClass &myclass) const;
    // virtual
    virtual bool EqualValue(CBasicClass *pClass) const = 0;
```

```
    // compare the address of two objects
    bool Equal(const CBasicClass *pClass) const
        { return (this == pClass); };

    // determine whether object contains valid data
    virtual bool IsValid() const = 0;
};

#endif // !defined(AFX_BASICCLASS_H__C0EA0EA0_0BD2_11D5_9BB7_B22A84F06321__INCLUDED_)
```

6.5.2 CDate

Abbildung 19: Interface von CDate



6.5.2.1 Date.h

```
////////////////////////////////////////////////////////////
// Softwarebauelemente I, Praktikum
//
// author:          Stephan Brumme
// last changes:    March 27, 2001

#if !defined(AFX_DATE_H__A8EA7861_08E1_11D5_9BB7_A8652B9FDD20__INCLUDED_)
#define AFX_DATE_H__A8EA7861_08E1_11D5_9BB7_A8652B9FDD20__INCLUDED_

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


// for basic class behaviour
#include "BasicClass.h"

#include <string>
using namespace std;

////////////////////////////////////////////////////////////
// the CDate class is based upon the Gregorian calender
// it DOES work for year between ca. 1600 and <2^31
// no year validation is performed
//
```

```
// general order of parameters: dd/mm/yyyy hh:mm:ss
class CDate : public CBasicClass
{
public:
    // constructor, default value is current date and time
    CDate();
    // copy constructor
    CDate(const CDate &date);
    // set user defined date at construction time
    CDate(unsigned int nDay,    unsigned int nMonth,    unsigned int nYear,
          unsigned int nHour=0, unsigned int nMinute=0, unsigned int nSecond=0);

    // return class name
    virtual string ClassnameOf() const { return "CDate"; }

    // show attributes
    virtual string Show() const;
    // shows all internal attributes
    virtual string ShowDebug() const;

    // return today's date
    static void GetToday(unsigned int &nDay,  unsigned int &nMonth,  unsigned int &nYear,
                         unsigned int &nHour, unsigned int &nMinute, unsigned int &nSecond);

    // validate a date
    virtual bool IsValid() const;
    // determine whether it is a leap year
    static bool IsLeapYear(unsigned int nYear);

    // copy constructors
    // non virtual
    CDate& operator = (const CDate &date);
    // virtual
    virtual bool Copy (CBasicClass *pClass);

    // compare two dates
    // non virtual
    bool operator == (const CDate &date) const;
    // virtual
    virtual bool EqualValue(CBasicClass *pClass) const;


    // set attributes, returns validility of date
    bool SetDay   (unsigned int nDay);
    bool SetMonth (unsigned int nMonth);
    bool SetYear  (unsigned int nYear);
    bool SetHour  (unsigned int nHour);
    bool SetMinute(unsigned int nMinute);
    bool 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_DATE_H__A8EA7861_08E1_11D5_9BB7_A8652B9FDD20__INCLUDED_)
```

6.5.2.2 Date.cpp

```
//////////////////////////////////////////////////////////
// Softwarebauelemente I, Praktikum
//
// author:          Stephan Brumme
// last changes:    March 31, 2001


#include "Date.h"


// we are using OS-specific date/time operations
#include <time.h>
#include <iomanip>
#include <iostream>
using namespace std;



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


// copy constructor
CDate::CDate(const CDate &date)
{
    operator=(date);
}


// set user defined date at construction time
CDate::CDate(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;
}


// show attributes
string CDate::Show() const
{
    ostringstream strOutput;

    strOutput << setw(2) << setfill('0') << m_nDay << "."
              << setw(2) << setfill('0') << m_nMonth << "."
              << m_nYear << " - "
              << setw(2) << setfill('0') << m_nHour<< ":"
              << setw(2) << setfill('0') << m_nMinute << ":"
              << setw(2) << setfill('0') << m_nSecond;

    return strOutput.str();
}


// shows all internal attributes
string CDate::ShowDebug() const
{
    ostringstream strOutput;

    strOutput << "DEBUG info for 'CDate'" << endl
```

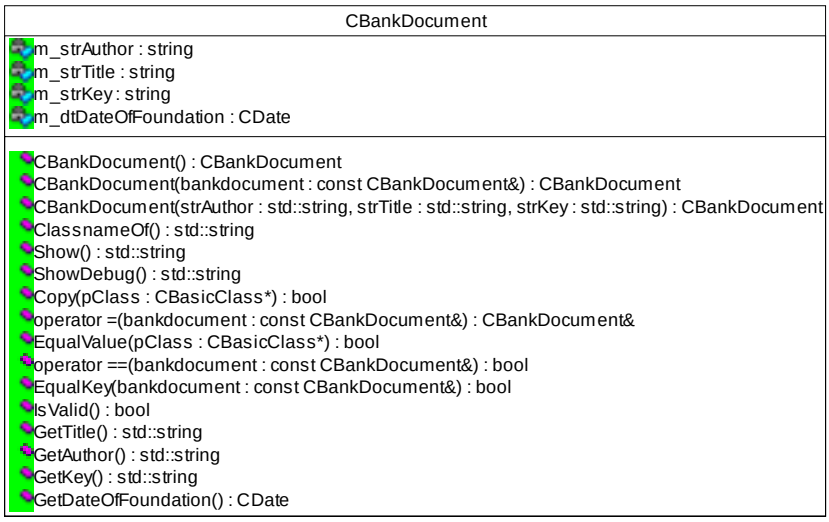
```
                << "      m_nDay    = " << m_nDay    << endl
                << "      m_nMonth  = " << m_nMonth  << endl
                << "      m_nYear   = " << m_nYear   << endl
                << "      m_nHour   = " << m_nHour   << endl
                << "      m_nMinute = " << m_nMinute << endl
                << "      m_nSecond = " << m_nSecond << endl;

    return strOutput.str();
}


// return current date
void CDate::GetToday(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;
}


// validate a date
bool CDate::IsValid() const
{
    // validate month
    if (m_nMonth < JANUARY || m_nMonth > DECEMBER)
        return false;

    // days per month, february may vary, note that array starts with 0, not JANUARY
    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
    if (m_nDay < 1 || m_nDay > nDaysPerMonth[m_nMonth])
        return false;
    // day and month are valid

    // now check the time
    if (m_nHour < 0 || m_nHour > 23)
        return false;
    if (m_nMinute < 0 || m_nMinute > 59)
        return false;
    if (m_nSecond < 0 || m_nSecond > 59)
        return false;

    // instance must be valid now
    return true;
}


// determine whether it is a leap year
bool CDate::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 constructor
CDate& CDate::operator =(const CDate &date)
{
    // date
    m_nDay    = date.m_nDay;
    m_nMonth  = date.m_nMonth;
    m_nYear   = date.m_nYear;
    // time
    m_nHour   = date.m_nHour;
    m_nMinute = date.m_nMinute;
    m_nSecond = date.m_nSecond;

    return *this;
}


// virtual, see operator=
bool CDate::Copy(CBasicClass *pClass)
{
    // cast to CDate
    CDate *date;
    date = dynamic_cast<CDate*>(pClass);

    // invalid class (is NULL when pClass is not a CDate)
    if (date == NULL || date == this)
        return false;

    // use non virtual reference based copy
    // return value isn't needed
    operator=(*date);

    // we're done
    return true;
}


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


// virtual, see operator==
bool CDate::EqualValue(CBasicClass *pClass) const
{
    // cast to CDate
    CDate *date;
    date = dynamic_cast<CDate*>(pClass);

    // invalid class (is NULL when pClass is not a CDate)
    if (date == NULL)
        return false;

    // use non virtual reference based copy
    return operator==(*date);
}
```

```
// set attributes, returns validility of date
bool CDate::SetDay(unsigned int nDay)
{
    m_nDay = nDay;
    return IsValid();
}
bool CDate::SetMonth(unsigned int nMonth)
{
    m_nMonth = nMonth;
    return IsValid();
}
bool CDate::SetYear(unsigned int nYear)
{
    m_nYear = nYear;
    return IsValid();
}
bool CDate::SetHour(unsigned int nHour)
{
    m_nHour = nHour;
    return IsValid();
}
bool CDate::SetMinute(unsigned int nMinute)
{
    m_nMinute = nMinute;
    return IsValid();
}
bool CDate::SetSecond(unsigned int nSecond)
{
    m_nSecond = nSecond;
    return IsValid();
}

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

6.5.3 CBankDocument

Abbildung 20: Interface von CBankDocument



6.5.3.1 BankDocument.h

```
////////////////////////////////////////////////////////////
// Softwarebauelemente I, Praktikum
//
// author:          Stephan Brumme
// last changes:    March 27, 2001

#if !defined(AFX_BANKDOCUMENT_H__37B49EC0_195B_11D5_9BB7_B87759331550__INCLUDED_)
#define AFX_BANKDOCUMENT_H__37B49EC0_195B_11D5_9BB7_B87759331550__INCLUDED_

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


// for basic class behaviour
#include "BasicClass.h"
#include <string>

// needed for date of foundation
#include "Date.h"


class CBankDocument : public CBasicClass
{
public:
    // default constructor
    CBankDocument();
    // copy constructor
    CBankDocument(const CBankDocument &bankdocument);
    // constructs an object using given attributes
    CBankDocument(string strAuthor, string strTitle, string strKey);

    // return class name
    virtual string ClassnameOf() const { return "CBankDocument"; }

    // show attributes
    virtual string Show() const;
    // shows all internal attributes
    virtual string ShowDebug() const;

    // copy constructors
    // non virtual
    CBankDocument& operator = (const CBankDocument &bankdocument);
    // virtual
```

```
    virtual bool Copy (CBasicClass *pClass);

    // compare the attributes of two bank documents
    // non virtual
    bool operator == (const CBankDocument &bankdocument) const;
    // virtual
    virtual bool EqualValue(CBasicClass *pClass) const;

    // compare the key of two objects
    bool EqualKey(const CBankDocument &bankdocument) const;

    // determine whether object contains valid data
    virtual bool IsValid() const;

    // return attributes
    string GetTitle() const;
    string GetAuthor() const;
    string GetKey() const;
    CDate  GetDateOfFoundation() const;

private:
    // attributes
    string m_strTitle;
    string m_strAuthor;
    string m_strKey;
    CDate  m_dtDateOfFoundation;
};

#endif // !defined(AFX_BANKDOCUMENT_H__37B49EC0_195B_11D5_9BB7_B87759331550__INCLUDED_)
```

6.5.3.2 BankDocument.cpp

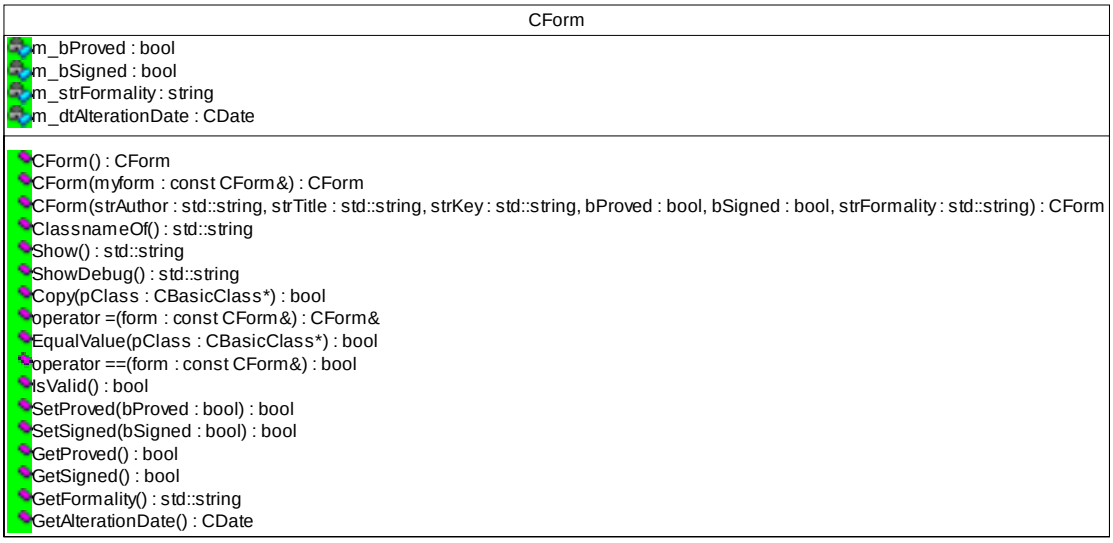
```
//////////////////////////////////////////////////////////
// Softwarebauelemente I, Praktikum
//
// author:          Stephan Brumme
// last changes:    March 27, 2001


#include "BankDocument.h"


// default constructor
CBankDocument::CBankDocument()
{
    m_strAuthor = "";
    m_strTitle  = "";
    m_strKey    = "";
    m_dtDateOfFoundation = CDate(0,0,0);
}


// constructs an object using given attributes
CBankDocument::CBankDocument(string strAuthor, string strTitle, string strKey)
{
    m_strAuthor = strAuthor;
    m_strTitle  = strTitle;
    m_strKey    = strKey;
    // m_dtDateOfFoundation will be set to current date by its constructor
}


// copy constructor
CBankDocument::CBankDocument(const CBankDocument &bankdocument)
{
    operator=(bankdocument);
}


// show attributes
string CBankDocument::Show() const
{
    ostringstream strOutput;
```

```
    strOutput << "Title:  " << m_strTitle  << endl
              << "Author: " << m_strAuthor << endl
              << "Key:    " << m_strKey    << endl
              << "Date:   "
              << m_dtDateOfFoundation.Show() << endl;

    return strOutput.str();
}


// shows all internal attributes
string CBankDocument::ShowDebug() const
{
    ostringstream strOutput;

    strOutput << "DEBUG info for 'CBankDocument'" << endl
              << "      m_strTitle           = " << m_strTitle           << endl
              << "      m_strAuthor          = " << m_strAuthor          << endl
              << "      m_strKey             = " << m_strKey             << endl
              << "      m_dtDateOfFoundation = " << m_dtDateOfFoundation.ShowDebug() << endl;

    return strOutput.str();
}


// copy constructor
CBankDocument& CBankDocument::operator =(const CBankDocument &bankdocument)
{
    m_strTitle           = bankdocument.m_strTitle;
    m_strAuthor          = bankdocument.m_strAuthor;
    m_strKey             = bankdocument.m_strKey;
    m_dtDateOfFoundation = bankdocument.m_dtDateOfFoundation;

    return *this;
}


// virtual, see operator=
bool CBankDocument::Copy(CBasicClass *pClass)
{
    // cast to CBankDocument
    CBankDocument *bankdocument;
    bankdocument = dynamic_cast<CBankDocument*>(pClass);

    // invalid class (is NULL when pClass is not a CBankDocument)
    if (bankdocument == NULL || bankdocument == this)
        return false;

    // use non virtual reference based copy
    // return value isn't needed
    operator=(*bankdocument);

    // we're done
    return true;
}


// compare two bankdocuments
bool CBankDocument::operator ==(const CBankDocument &bankdocument) const
{
    // compare bankdocument attributes
    return (m_strTitle           == bankdocument.m_strTitle  &&
            m_strAuthor          == bankdocument.m_strAuthor &&
            m_strKey             == bankdocument.m_strKey    &&
            m_dtDateOfFoundation == bankdocument.m_dtDateOfFoundation);
}


// virtual, see operator==
bool CBankDocument::EqualValue(CBasicClass *pClass) const
{
    // cast to CBankDocument
    CBankDocument *bankdocument;
    bankdocument = dynamic_cast<CBankDocument*>(pClass);

    // invalid class (is NULL when pClass is not a CBankDocument)
```

```
    if (bankdocument == NULL)
        return false;

    // use non virtual reference based copy
    return operator==(*bankdocument);
}


// compare the key of two objects
bool CBankDocument::EqualKey(const CBankDocument &bankdocument) const
{
    return (m_strKey == bankdocument.m_strKey);
}


// determine whether object contains valid data
bool CBankDocument::IsValid() const
{
    // each attribute must contain some data
    return (m_strTitle .length() > 0 &&
            m_strAuthor.length() > 0 &&
            m_strKey   .length() > 0 &&
            m_dtDateOfFoundation.IsValid());
}


// return attributes
string CBankDocument::GetTitle() const
{
    return m_strTitle;
}

string CBankDocument::GetAuthor() const
{
    return m_strAuthor;
}

string CBankDocument::GetKey() const
{
    return m_strKey;
}

CDate CBankDocument::GetDateOfFoundation() const
{
    return m_dtDateOfFoundation;
}
```

6.5.4 CForm

Abbildung 21: Interface von CForm



6.5.4.1 Form.h

```
////////////////////////////////////////////////////////
// Softwarebauelemente I, Praktikum
//
// author:          Stephan Brumme
// last changes:    March 27, 2001

#if !defined(AFX_FORM_H__087F5DC0_1C94_11D5_9BB7_F9497547F451__INCLUDED_)
#define AFX_FORM_H__087F5DC0_1C94_11D5_9BB7_F9497547F451__INCLUDED_

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

#include "BankDocument.h"
#include "Date.h"

#include <string>


class CForm : public CBankDocument
{
public:
    // default constructor
    CForm();
    // copy constructor
    CForm(const CForm &myform);
    // constructs an object using given attributes
    CForm(string strAuthor, string strTitle, string strKey,
          bool bProved, bool bSigned, string strFormality);

    // return class name
    virtual string ClassnameOf() const { return "CForm"; }

    // show attributes
    virtual string Show() const;
    // shows all internal attributes
    virtual string ShowDebug() const;

    // copy constructors
    // non virtual
    CForm& operator = (const CForm &form);
    // virtual
    virtual bool Copy (CBasicClass *pClass);
```

```
    // compare the attributes of two bank documents
    // non virtual
    bool operator == (const CForm &form) const;
    // virtual
    virtual bool EqualValue(CBasicClass *pClass) const;

    // determine whether object contains valid data
    virtual bool IsValid() const;

    // set attributes
    bool SetProved(bool bProved);
    bool SetSigned(bool bSigned);

    // return attributes
    bool   GetProved() const;
    bool   GetSigned() const;
    string GetFormality() const;
    CDate  GetAlterationDate() const;

private:
    // attributes
    bool m_bProved;
    bool m_bSigned;
    string m_strFormality;
    CDate m_dtAlterationDate;
};

#endif // !defined(AFX_FORM_H__087F5DC0_1C94_11D5_9BB7_F9497547F451__INCLUDED_)
```

6.5.4.2 Form.cpp

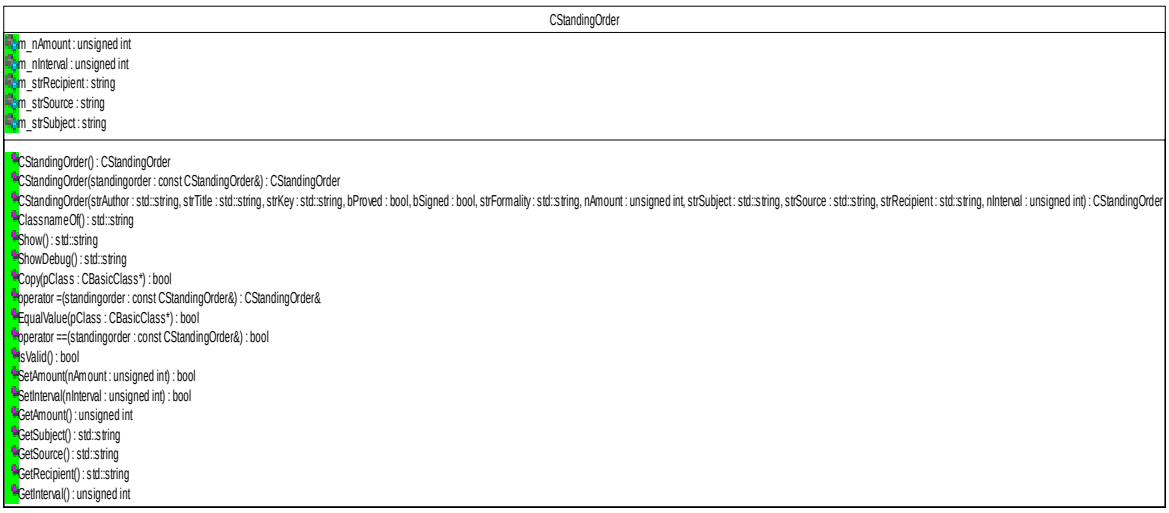
```
/////////////////////////////////////////////////////////
// Softwarebauelemente I, Praktikum
//
// author:          Stephan Brumme
// last changes:    March 27, 2001

#include "Form.h"



// default constructor
CForm::CForm()
      :CBankDocument()
{
    m_bProved = false;
    m_bSigned = false;
    m_strFormality = "";
    // m_dtAlterationDate is initialized by its constructor
}


// copy constructor
CForm::CForm(const CForm &myform)
{
    operator=(myform);
}


// constructs an object using given attributes
CForm::CForm(string strAuthor, string strTitle, string strKey,
             bool bProved, bool bSigned, string strFormality)
      :CBankDocument(strAuthor, strTitle, strKey)
      // call base class constructor
{
    m_bProved = bProved;
    m_bSigned = bSigned;
    m_strFormality = strFormality;
    // m_dtAlterationDate is initialized by its constructor
}
```

```
// show attributes
string CForm::Show() const
{
    ostringstream strOutput;

    strOutput << CBankDocument::Show()
              << "Proved:          " << m_bProved      << endl
              << "Signed:          " << m_bSigned      << endl
              << "Formality:       " << m_strFormality << endl
              << "AlterationDate:  " << m_dtAlterationDate.Show() << endl;

    return strOutput.str();
}


// shows all internal attributes
string CForm::ShowDebug() const
{
    ostringstream strOutput;

    strOutput << CBankDocument::ShowDebug()
              << "DEBUG info for 'CForm'" << endl
              << "      m_bProved          = " << m_bProved      << endl
              << "      m_bSigned          = " << m_bSigned      << endl
              << "      m_strFormality     = " << m_strFormality << endl
              << "      m_dtAlterationDate = " << m_dtAlterationDate.ShowDebug() << endl;

    return strOutput.str();
}


// copy constructor
CForm& CForm::operator =(const CForm &form)
{
    CBankDocument::operator=(form);

    m_bProved = form.m_bProved;
    m_bSigned = form.m_bSigned;
    m_strFormality = form.m_strFormality;
    m_dtAlterationDate = form.m_dtAlterationDate;

    return *this;
}


// virtual, see operator=
bool CForm::Copy(CBasicClass *pClass)
{
    // cast to CForm
    CForm *form;
    form = dynamic_cast<CForm*>(pClass);

    // invalid class (is NULL when pClass is not a CForm)
    if (form == NULL || form == this)
        return false;

    // use non virtual reference based copy
    // return value isn't needed
    operator=(*form);

    // we're done
    return true;
}


// compare two forms
bool CForm::operator ==(const CForm &form) const
{
    // compare form attributes
    return (m_bProved == form.m_bProved &&
            m_bSigned == form.m_bSigned &&
            m_dtAlterationDate == form.m_dtAlterationDate &&
            m_strFormality == form.m_strFormality &&
            CBankDocument::operator==(form));
}
```

```
// virtual, see operator==
bool CForm::EqualValue(CBasicClass *pClass) const
{
    // cast to CForm
    CForm *form;
    form = dynamic_cast<CForm*>(pClass);

    // invalid class (is NULL when pClass is not a CForm)
    if (form == NULL)
        return false;

    // use non virtual reference based copy
    return operator==(*form);
}


// determine whether object contains valid data
bool CForm::IsValid() const
{
    // each attribute must contain some data
    return (CBankDocument::IsValid()    &&
            m_bProved                   &&
            m_bSigned                   &&
            m_strFormality.length() > 0 &&
            m_dtAlterationDate.IsValid());
}


// set attributes
bool CForm::SetProved(bool bProved)
{
    m_bProved = bProved;

    // update alteration date
    CDate dtNow;
    m_dtAlterationDate = dtNow;

    return IsValid();
}
bool CForm::SetSigned(bool bSigned)
{
    m_bSigned = bSigned;

    // update alteration date
    CDate dtNow;
    m_dtAlterationDate = dtNow;

    return IsValid();
}

// return attributes
bool CForm::GetProved() const
{
    return m_bProved;
}
bool CForm::GetSigned() const
{
    return m_bSigned;
}
string CForm::GetFormality() const
{
    return m_strFormality;
}
CDate CForm::GetAlterationDate() const
{
    return m_dtAlterationDate;
}
```

6.5.5 CStandingOrder

Abbildung 22: Interface von CStandingOrder



6.5.5.1 StandingOrder.h

```
//////////////////////////////////////////////////////////
// Softwarebauelemente I, Praktikum
//
// author:          Stephan Brumme
// last changes:    March 27, 2001

#if !defined(AFX_STANDINGORDER_H__087F5DC1_1C94_11D5_9BB7_F9497547F451__INCLUDED_)
#define AFX_STANDINGORDER_H__087F5DC1_1C94_11D5_9BB7_F9497547F451__INCLUDED_

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

#include "Form.h"
#include "Date.h"
#include <string>


class CStandingOrder : public CForm
{
public:
    // default constructor
    CStandingOrder();
    // copy constructor
    CStandingOrder(const CStandingOrder &standingorder);
    // constructs an object using given attributes
    CStandingOrder(string strAuthor, string strTitle, string strKey,
                   bool bProved, bool bSigned, string strFormality,
                   unsigned int nAmount, string strSubject, string strSource,
                   string strRecipient, unsigned int nInterval);

    // return class name
    virtual string ClassnameOf() const { return "CStandingOrder"; }

    // show attributes
    virtual string Show() const;
    // shows all internal attributes
    virtual string ShowDebug() const;

    // copy constructors
    // non virtual
    CStandingOrder& operator = (const CStandingOrder &standingorder);
    // virtual
    virtual bool Copy (CBasicClass *pClass);
```

```
    // compare the attributes of two bank documents
    // non virtual
    bool operator == (const CStandingOrder &standingorder) const;
    // virtual
    virtual bool EqualValue(CBasicClass *pClass) const;

    // determine whether object contains valid data
    virtual bool IsValid() const;

    // set attributes
    bool SetAmount  (unsigned int nAmount);
    bool SetInterval(unsigned int nInterval);

    // return attributes
    unsigned int    GetAmount()    const;
    string GetSubject()   const;
    string GetSource()    const;
    string GetRecipient() const;
    unsigned int GetInterval()  const;

private:
    // attributes
    unsigned int m_nAmount;
    unsigned int m_nInterval;
    string m_strSubject;
    string m_strSource;
    string m_strRecipient;
};

#endif // !defined(AFX_STANDINGORDER_H__087F5DC1_1C94_11D5_9BB7_F9497547F451__INCLUDED_)
```

6.5.5.2 StandingOrder.cpp

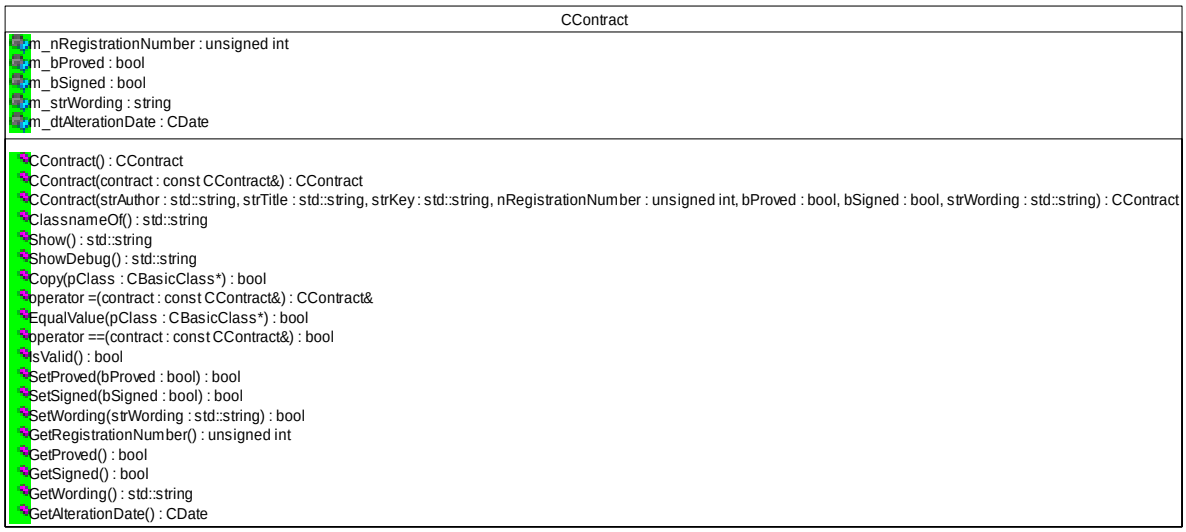
```
//////////////////////////////////////////////////////////
// Softwarebauelemente I, Praktikum
//
// author:          Stephan Brumme
// last changes:    March 27, 2001

#include "StandingOrder.h"


// default constructor
CStandingOrder::CStandingOrder()
               :CForm()
{
    m_nAmount   = 0;
    m_nInterval = 0;
    m_strRecipient = "";
    m_strSource    = "";
    m_strSubject   = "";
}


// copy constructor
CStandingOrder::CStandingOrder(const CStandingOrder &standingorder)
{
    operator=(standingorder);
}


// constructs an object using given attributes
CStandingOrder::CStandingOrder(string strAuthor, string strTitle, string strKey,
                               bool bProved, bool bSigned, string strFormality,
                               unsigned int nAmount, string strSubject, string strSource,
                               string strRecipient, unsigned int nInterval)
               :CForm(strAuthor, strTitle, strKey, bProved, bSigned, strFormality)
{
    m_nAmount      = nAmount;
    m_strSubject   = strSubject;
    m_strSource    = strSource;
    m_strRecipient = strRecipient;
    m_nInterval    = nInterval;
}
```

```
// show attributes
string CStandingOrder::Show() const
{
    ostringstream strOutput;

    strOutput << CForm::Show()
              << "Amount:    " << m_nAmount << endl
              << "Subject:   " << m_strSubject << endl
              << "Source:    " << m_strSource << endl
              << "Recipient: " << m_strRecipient << endl
              << "Interval:  " << m_nInterval << endl;

    return strOutput.str();
}


// shows all internal attributes
string CStandingOrder::ShowDebug() const
{
    ostringstream strOutput;

    strOutput << CForm::ShowDebug()
              << "DEBUG info for 'CStandingOrder'"  << endl
              << "      m_nAmount      = " << m_nAmount      << endl
              << "      m_strSubject   = " << m_strSubject   << endl
              << "      m_strSource    = " << m_strSource    << endl
              << "      m_strRecipient = " << m_strRecipient << endl
              << "      m_nInterval    = " << m_nInterval    << endl;

    return strOutput.str();
}


// copy constructor
CStandingOrder& CStandingOrder::operator =(const CStandingOrder &standingorder)
{
    CForm::operator=(standingorder);

    m_nAmount      = standingorder.m_nAmount;
    m_nInterval    = standingorder.m_nInterval;
    m_strRecipient = standingorder.m_strRecipient;
    m_strSource    = standingorder.m_strSource;
    m_strSubject   = standingorder.m_strSubject;

    return *this;
}


// virtual, see operator=
bool CStandingOrder::Copy(CBasicClass *pClass)
{
    // cast to CStandingOrder
    CStandingOrder *standingorder;
    standingorder = dynamic_cast<CStandingOrder*>(pClass);

    // invalid class (is NULL when pClass is not a CStandingOrder)
    if (standingorder == NULL || standingorder == this)
        return false;

    // use non virtual reference based copy
    // return value isn't needed
    operator=(*standingorder);

    // we're done
    return true;
}


// compare two standingorders
bool CStandingOrder::operator ==(const CStandingOrder &standingorder) const
{
    // compare standingorder attributes
    return (m_nAmount == standingorder.m_nAmount           &&
            m_nInterval == standingorder.m_nInterval       &&
```

```
            m_strRecipient == standingorder.m_strRecipient &&
            m_strSource == standingorder.m_strSource       &&
            m_strSubject == standingorder.m_strSubject     &&
            CForm::operator==(standingorder));
}


// virtual, see operator==
bool CStandingOrder::EqualValue(CBasicClass *pClass) const
{
    // cast to CStandingOrder
    CStandingOrder *standingorder;
    standingorder = dynamic_cast<CStandingOrder*>(pClass);

    // invalid class (is NULL when pClass is not a CStandingOrder)
    if (standingorder == NULL)
        return false;

    // use non virtual reference based copy
    return operator==(*standingorder);
}


// determine whether object contains valid data
bool CStandingOrder::IsValid() const
{
    // each attribute must contain some data
    return (CForm::IsValid()            &&
            m_nAmount   > 0             &&
            m_nInterval > 0             &&
            m_strRecipient.length() > 0 &&
            m_strSource   .length() > 0 &&
            m_strSubject  .length() > 0);
}



// set attributes
bool CStandingOrder::SetAmount(unsigned int nAmount)
{
    m_nAmount = nAmount;
    return IsValid();
}
bool CStandingOrder::SetInterval(unsigned int nInterval)
{
    m_nInterval = nInterval;
    return IsValid();
}

// return attributes
unsigned int CStandingOrder::GetAmount() const
{
    return m_nAmount;
}
string CStandingOrder::GetSubject() const
{
    return m_strSubject;
}
string CStandingOrder::GetSource() const
{
    return m_strSource;
}
string CStandingOrder::GetRecipient() const
{
    return m_strRecipient;
}
unsigned int CStandingOrder::GetInterval() const
{
    return m_nInterval;
}
```

6.5.6 CContract

Abbildung 23: Interface von CContract



6.5.6.1 Contract.h

```
/////////////////////////////////////////////////////////
// Softwarebauelemente I, Praktikum
//
// author:          Stephan Brumme
// last changes:    March 27, 2001

#if !defined(AFX_CONTRACT_H__EE08A560_1CB5_11D5_9BB7_B36F5BC31F50__INCLUDED_)
#define AFX_CONTRACT_H__EE08A560_1CB5_11D5_9BB7_B36F5BC31F50__INCLUDED_

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

#include "BankDocument.h"
#include "Date.h"
#include <string>


class CContract : public CBankDocument
{
public:
    // default constructor
    CContract();
    // copy constructor
    CContract(const CContract &contract);
    // constructs an object using given attributes
    CContract(string strAuthor, string strTitle, string strKey,
              unsigned int nRegistrationNumber, bool bProved, bool bSigned, string
strWording);

    // return class name
    virtual string ClassnameOf() const { return "CContract"; }

    // show attributes
    virtual string Show() const;
    // shows all internal attributes
    virtual string ShowDebug() const;

    // copy constructors
    // non virtual
    CContract& operator = (const CContract &contract);
    // virtual
    virtual bool Copy (CBasicClass *pClass);
```

```
    // compare the attributes of two bank documents
    // non virtual
    bool operator == (const CContract &contract) const;
    // virtual
    virtual bool EqualValue(CBasicClass *pClass) const;

    // determine whether object contains valid data
    virtual bool IsValid() const;

    // set attributes
    bool SetProved(bool bProved);
    bool SetSigned(bool bSigned);
    bool SetWording(string strWording);

    // return attributes
    unsigned int GetRegistrationNumber() const;
    bool   GetProved() const;
    bool   GetSigned() const;
    string GetWording() const;
    CDate  GetAlterationDate() const;

private:
    // attributes
    unsigned int m_nRegistrationNumber;
    CDate  m_dtAlterationDate;
    bool   m_bProved;
    bool   m_bSigned;
    string m_strWording;
};

#endif // !defined(AFX_CONTRACT_H__EE08A560_1CB5_11D5_9BB7_B36F5BC31F50__INCLUDED_)
```

6.5.6.2 Contract.cpp

```
//////////////////////////////////////////////////////////
// Softwarebauelemente I, Praktikum
//
// author:          Stephan Brumme
// last changes:    March 27, 2001

#include "Contract.h"


// default constructor
CContract::CContract()
          :CBankDocument()
{
    m_bProved = false;
    m_bSigned = 0;
    m_nRegistrationNumber = 0;
    m_strWording = "";
    // m_dtAlterationDate is initialized by its constructor
}


// copy constructor
CContract::CContract(const CContract &contract)
{
    operator=(contract);
}


// constructs an object using given attributes
CContract::CContract(string strAuthor, string strTitle, string strKey,
                     unsigned int nRegistrationNumber, bool bProved, bool bSigned, string
strWording)
          :CBankDocument(strAuthor, strTitle, strKey)
{
    m_nRegistrationNumber = nRegistrationNumber;
    m_bProved = bProved;
    m_bSigned = bSigned;
    m_strWording = strWording;
    // m_dtAlterationDate is initialized by its constructor
}
```

```
// show attributes
string CContract::Show() const
{
    ostringstream strOutput;

    strOutput << CBankDocument::Show()
              << "Reg-Number:     " << m_nRegistrationNumber << endl
              << "Proved:         " << m_bProved             << endl
              << "Signed:         " << m_bSigned             << endl
              << "Wording:        " << m_strWording          << endl
              << "AlterationDate: " << m_dtAlterationDate.Show() << endl;

    return strOutput.str();
}


// shows all internal attributes
string CContract::ShowDebug() const
{
    ostringstream strOutput;

    strOutput << CBankDocument::ShowDebug()
              << "DEBUG info for 'CContract'" << endl
              << "      m_nRegistrationNumber = " << m_nRegistrationNumber << endl
              << "      m_bProved             = " << m_bProved    << endl
              << "      m_bSigned             = " << m_bSigned    << endl
              << "      m_strWording          = " << m_strWording << endl
              << "      m_dtAlterationDate    = " << m_dtAlterationDate.ShowDebug() << endl;

    return strOutput.str();
}


// copy constructor
CContract& CContract::operator =(const CContract &contract)
{
    CBankDocument::operator=(contract);

    m_nRegistrationNumber = contract.m_nRegistrationNumber;
    m_bProved = contract.m_bProved;
    m_bSigned = contract.m_bSigned;
    m_strWording = contract.m_strWording;
    m_dtAlterationDate = contract.m_dtAlterationDate;

    return *this;
}


// virtual, see operator=
bool CContract::Copy(CBasicClass *pClass)
{
    // cast to CContract
    CContract *contract;
    contract = dynamic_cast<CContract*>(pClass);

    // invalid class (is NULL when pClass is not a CContract)
    if (contract == NULL || contract == this)
        return false;

    // use non virtual reference based copy
    // return value isn't needed
    operator=(*contract);

    // we're done
    return true;
}


// compare two contracts
bool CContract::operator ==(const CContract &contract) const
{
    // compare contract attributes
    return (m_bProved == contract.m_bProved &&
            m_bSigned == contract.m_bSigned &&
```

```
            m_nRegistrationNumber == contract.m_nRegistrationNumber &&
            m_dtAlterationDate    == contract.m_dtAlterationDate &&
            m_strWording          == contract.m_strWording &&
            CBankDocument::operator==(contract));
}


// virtual, see operator==
bool CContract::EqualValue(CBasicClass *pClass) const
{
    // cast to CContract
    CContract *contract;
    contract = dynamic_cast<CContract*>(pClass);

    // invalid class (is NULL when pClass is not a CContract)
    if (contract == NULL)
        return false;

    // use non virtual reference based copy
    return operator==(*contract);
}


// determine whether object contains valid data
bool CContract::IsValid() const
{
    // each attribute must contain some data
    return (CBankDocument::IsValid()  &&
            m_bProved                 &&
            m_bSigned                 &&
            m_nRegistrationNumber > 0 &&
            m_strWording.length() > 0 &&
            m_dtAlterationDate.IsValid());
}


// set attributes
bool CContract::SetProved(bool bProved)
{
    m_bProved = bProved;

    // update alteration date
    CDate dtNow;
    m_dtAlterationDate = dtNow;

    return IsValid();
}
bool CContract::SetSigned(bool bSigned)
{
    m_bSigned = bSigned;

    // update alteration date
    CDate dtNow;
    m_dtAlterationDate = dtNow;

    return IsValid();
}
bool CContract::SetWording(string strWording)
{
    m_strWording = strWording;

    // update alteration date
    CDate dtNow;
    m_dtAlterationDate = dtNow;

    return IsValid();
}

// return attributes
unsigned int CContract::GetRegistrationNumber() const
{
    return m_nRegistrationNumber;
}
bool CContract::GetProved() const
{
    return m_bProved;
```

```
}
bool CContract::GetSigned() const
{
    return m_bSigned;
}
string CContract::GetWording() const
{
    return m_strWording;
}
CDate CContract::GetAlterationDate() const
{
    return m_dtAlterationDate;
}
```

6.5.7 CTest

6.5.7.1 Test.h

```
//////////////////////////////////////////////////////////
// Softwarebauelemente I, Praktikum
//
// author:          Stephan Brumme
// last changes:    March 31, 2001


#if !defined(AFX_TEST_H__FC385B01_2136_11D5_9BB7_ABD2649BA651__INCLUDED_)
#define AFX_TEST_H__FC385B01_2136_11D5_9BB7_ABD2649BA651__INCLUDED_

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


#include "BasicClass.h"

#include <string>
#include <vector>
#include <fstream>
using namespace std;



class CTest
{
public:
    // constructor, builds a new test environment
    CTest(string strLogFile="");
    // destructor, free any used memory
    virtual ~CTest();

    // run the test environment
    void Run();

    // determines whether log file was successfully created
    bool IsLogFileUsed();

private:
    // create a new CBasicClass-based object
    void CreateObject();
    // delete an object
    void DeleteObject();
    // select an object
    void SelectObject();
    // compare two objects or copy one to another
    void CopyCompareObject();
    // set some attributes of an object
    void ChangeObject();

    // read a number from console
    int ReadNumber();
```

```
    // show debug of all stored objects
    void Show();
    // only show object's types
    string ShowTypes();
    // print a specified string, able to write the output to a log file
    void Print(const string& strOutput);


    // currently selected object
    int      m_nSelected;
    // determines whether log file was successfully created
    bool     m_bUseLogFile;
    // file handle for log file
    ofstream m_hLogFile;

    // stored objects, taken from STL
    vector<CBasicClass*> m_arClasses;
    // define new data type for to ease access to m_arClasses
    typedef vector<CBasicClass*>::iterator TIterator;
};

#endif // !defined(AFX_TEST_H__FC385B01_2136_11D5_9BB7_ABD2649BA651__INCLUDED_)
```

6.5.7.2 Test.cpp

```
//////////////////////////////////////////////////////////
// Softwarebauelemente I, Praktikum
//
// author:          Stephan Brumme
// last changes:    March 31, 2001


#include "Test.h"

#include "Date.h"
#include "BankDocument.h"
#include "Form.h"
#include "StandingOrder.h"
#include "Contract.h"

#include <iostream>
#include <fstream>
#include <sstream>



// constructor, builds a new test environment
CTest::CTest(string strLogFile)
{
    // only use log file if any is specified
    if (!strLogFile.empty())
        m_hLogFile.open(strLogFile.c_str());

    // file successfully opened ?
    m_bUseLogFile = m_hLogFile.is_open();

    if (m_bUseLogFile)
    {
        // insert date and time of creation
        CDate dtNow;
        m_hLogFile << "Logfile created: " << dtNow.Show() << endl << endl;
    }
}


// destructor, free any used memory
CTest::~CTest()
{
    // close logfile
    m_hLogFile.close();

    // delete all allocated objects
    TIterator myIterator = m_arClasses.begin();
    // loop through vector
```

```
    while (myIterator != m_arClasses.end())
    {
        delete *myIterator;
        myIterator++;
    }
}


// run the test environment
void CTest::Run()
{
    // menu selection
    int nInput;

    // repeat until user finishes (nInput=0)
    do
    {
        // show menu
        ostringstream strOutput;
        strOutput << "1 - Create new object"   << endl
                  << "2 - Show stored classes" << endl
                  << "3 - Select an object"    << endl
                  << "4 - Copy/CompareObjects" << endl
                  << "5 - Set attributes"      << endl
                  << "6 - Delete object"       << endl
                  << "0 - Quit"        << endl << endl;

        // show currently selected object
        strOutput << "Currently selected object: ";
        if(m_nSelected >= 0)
           strOutput << m_nSelected;
        else
           strOutput << "(none)";

        // ask user
        strOutput << endl << "Please choose: ";
        Print(strOutput.str());

        // get user's input
        nInput = ReadNumber();
        Print("\n");

        // call method depending upon user's input
        switch (nInput)
        {
        case 1: CreateObject(); break;
        case 2: Show(); break;
        case 3: SelectObject(); break;
        case 4: CopyCompareObject(); break;
        case 5: ChangeObject(); break;
        case 6: DeleteObject(); break;
        case 0: break;

        default: Print ("???\n");
        }

        Print("\n");
    }
    // finish when 0 is pressed
    while (nInput != 0);
}


// determines whether log file was successfully created
bool CTest::IsLogFileUsed()
{
    return m_bUseLogFile;
}


// create a new CBasicClass-based object
void CTest::CreateObject()
{
    // prepare pointer to hold a new object
    CBasicClass *pClass = NULL;
```

```
    // display choose list
    ostringstream strOutput;
    strOutput << "1 - CDate"          << endl;
    strOutput << "2 - CBankDocument"  << endl;
    strOutput << "3 - CContract"      << endl;
    strOutput << "4 - CForm"          << endl;
    strOutput << "5 - CStandingOrder" << endl;
    strOutput << "0 - none"           << endl;
    strOutput << endl;

    strOutput << "Please choose the class type: ";
    Print(strOutput.str());

    // read selected menu item
    int nClass = ReadNumber();
    if (nClass <= 0 || nClass > 5)
        return;

    // there 3 constructors for every object
    Print("Please choose type of construction (0=default, 1=copy, 2=init): ");
    int nType = ReadNumber();

    // for cloning
    int nSource;
    CBasicClass *pSource = NULL;

    // parameters (different use, therefore these simple names)
    int    n1, n2, n3, n4, n5, n6;
    string s1, s2, s3, s4, s5, s6, s7;
    bool   b1, b2;



    switch(nType)
    {
    case 0: // create an object using default constructor
            switch (nClass)
            {
            case 1: pClass = new CDate();
                    break;
            case 2: pClass = new CBankDocument();
                    break;
            case 3: pClass = new CContract();
                    break;
            case 4: pClass = new CForm();
                    break;
            case 5: pClass = new CStandingOrder();
                    break;
            }
            break;

    case 1: // create an object using copy constructor
            Print(ShowTypes());

            // get number of object which will be copied
            Print("Which object should be cloned: ");
            nSource = ReadNumber();

            // object which should be copied
            pSource = m_arClasses[nSource];

            // cast using dynamic_cast before actually copying !!!
            switch(nClass)
            {
            case 1: pClass = new CDate(*dynamic_cast<CDate*>(pSource));
                    break;

            case 2: pClass = new CBankDocument(*dynamic_cast<CBankDocument*>(pSource));
                    break;

            case 3: pClass = new CContract(*dynamic_cast<CContract*>(pSource));
                    break;

            case 4: pClass = new CForm(*dynamic_cast<CForm*>(pSource));
                    break;
```

```
            case 5: pClass = new CStandingOrder(*dynamic_cast<CStandingOrder*>(pSource));
                    break;
            }
            break;

    case 2: // create an object using parameters
            // ask user to type in all parameters
            switch (nClass)
            {
            case 1: Print("nDay    = "); n1 = ReadNumber();
                    Print("nMonth  = "); n2 = ReadNumber();
                    Print("nYear   = "); n3 = ReadNumber();
                    Print("nHour   = "); n4 = ReadNumber();
                    Print("nMinute = "); n5 = ReadNumber();
                    Print("nSecond = "); n6 = ReadNumber();
                    pClass = new CDate(n1, n2, n3, n4, n5, n6);
                    break;

            case 2: Print("strAuthor = "); getline(cin, s1);
                    Print("strTitle  = "); getline(cin, s2);
                    Print("strKey    = "); getline(cin, s3);
                    pClass = new CBankDocument(s1, s2, s3);
                    break;

            case 3: Print("strAuthor = "); getline(cin, s1);
                    Print("strTitle  = "); getline(cin, s2);
                    Print("strKey    = "); getline(cin, s3);
                    Print("nRegistrationNumber = "); n1 = ReadNumber();
                    Print("bProved (true=1)    = "); b1 = (ReadNumber()==1);
                    Print("bSigned (true=1)    = "); b2 = (ReadNumber()==1);
                    Print("strWording          = "); getline(cin, s4);
                    pClass = new CContract(s1, s2, s3, n1, b1, b2, s4);
                    break;

            case 4: Print("strAuthor = "); getline(cin, s1);
                    Print("strTitle  = "); getline(cin, s2);
                    Print("strKey    = "); getline(cin, s3);
                    Print("bProved (true=1)    = "); b1 = (ReadNumber()==1);
                    Print("bSigned (true=1)    = "); b2 = (ReadNumber()==1);
                    Print("strFormality        = "); getline(cin, s4);

                    pClass = new CForm(s1, s2, s3, b1, b2, s4);
                    break;

            case 5: Print("strAuthor = "); getline(cin, s1);
                    Print("strTitle  = "); getline(cin, s2);
                    Print("strKey    = "); getline(cin, s3);
                    Print("bProved (true=1)    = "); b1 = (ReadNumber()==1);
                    Print("bSigned (true=1)    = "); b2 = (ReadNumber()==1);
                    Print("strFormality        = "); getline(cin, s4);
                    Print("nAmount             = "); n1 = ReadNumber();
                    Print("strSubject          = "); getline(cin, s5);
                    Print("strSource           = "); getline(cin, s6);
                    Print("strRecipient        = "); getline(cin, s7);
                    Print("nInterval           = "); n2 = ReadNumber();

                    pClass = new CStandingOrder(s1, s2, s3, b1, b2, s4, n1, s5, s6, s7, n2);
                    break;
            }
            break;


    // invalid input
    default:
        Print("... aborted\n");
    }

    // push to the tail
    if (pClass != NULL)
        m_arClasses.push_back(pClass);

}


// delete an object
void CTest::DeleteObject()
```

```
{
    // there must be at least one object
    if (m_arClasses.empty() || m_nSelected < 0)
        return;

    // delete currently selected object
    delete m_arClasses[m_nSelected];

    // remove pointer to that object from array
    m_arClasses.erase(m_arClasses.begin() + m_nSelected);

    // print action
    Print("Object successfully removed.\n");
}


// select an object
void CTest::SelectObject()
{
    // there must be at least one object
    if (m_arClasses.empty())
        return;

    // show all stored objects
    Show();

    // ask user to select one
    Print("Please choose an object: ");
    int nInput = ReadNumber();

    // must be a valid object number
    if (nInput >= 0 && nInput < m_arClasses.size())
        m_nSelected = nInput;
    else
        Print("... aborted");
}


// compare two objects or copy one to another
void CTest::CopyCompareObject()
{
    ostringstream strOutput;

    // show all object types and ask user to select the destination object
    // source object is the currently selected one
    strOutput << "Source is object no. " << m_nSelected << endl;
    strOutput << ShowTypes();
    strOutput << "Which object should be the destination object: ";
    Print(strOutput.str());

    // get destination object, verify validity
    int nDestiny = ReadNumber();
    if (nDestiny < 0 || nDestiny >= m_arClasses.size())
        return;

    // select action
    Print("1 - Copy, 2 - EqualValue, 3 - Equal\n");
    int nAction = ReadNumber();
    if (nAction < 1 || nAction > 3)
        return;

    // create pointers for the source and destination objects
    CBasicClass *pSource  = m_arClasses[m_nSelected];
    CBasicClass *pDestiny = m_arClasses[nDestiny];

    switch(nAction)
    {
    case 1: // copy source to destination
            pDestiny->Copy(pSource); break;

    case 2: // compare attributes of both objects (value identity)
            Print("Comparing: ");
            if (pSource->EqualValue(pDestiny))
                Print("equal\n");
            else
                Print("different\n");
```

```
            break;

    case 3: // compare memory address of both objects (object identity)
            Print("Comparing: ");
            if (pSource->Equal(pDestiny))
                Print("equal\n");
            else
                Print("different\n");
            break;
    }
}


// set some attributes of an object
void CTest::ChangeObject()
{
    // there must be at least one object
    if (m_arClasses.empty() || m_nSelected < 0)
        return;

    // get pointer to the object
    CBasicClass *pClass = m_arClasses[m_nSelected];

    // build pointers to all possible classes, some will fail
    CDate          *pDate          = dynamic_cast<CDate*>         (pClass);
    CBankDocument  *pBankDocument  = dynamic_cast<CBankDocument*> (pClass);
    CForm          *pForm          = dynamic_cast<CForm*>         (pClass);
    CStandingOrder *pStandingOrder = dynamic_cast<CStandingOrder*>(pClass);
    CContract      *pContract      = dynamic_cast<CContract*>     (pClass);

    // show all attributes
    Print(pClass->ShowDebug());

    // there may be some pointers (e.g. more than 1!) valid
    if (pContract != NULL)
    {
        bool b1, b2;
        string s1;

        Print("bProved (true=1)    = "); b1 = (ReadNumber()==1);
        Print("bSigned (true=1)    = "); b2 = (ReadNumber()==1);
        Print("strWording          = "); getline(cin, s1);

        pContract->SetProved(b1);
        pContract->SetSigned(b2);
        pContract->SetWording(s1);
    }

    if (pStandingOrder != NULL)
    {
        int n1, n2;

        Print("nAmount             = "); n1 = ReadNumber();
        Print("nInterval           = "); n2 = ReadNumber();

        pStandingOrder->SetAmount(n1);
        pStandingOrder->SetInterval(n2);
    }

    if (pForm != NULL)
    {
        bool b1, b2;
        string s1;

        Print("bProved (true=1)    = "); b1 = (ReadNumber()==1);
        Print("bSigned (true=1)    = "); b2 = (ReadNumber()==1);

        pForm->SetProved(b1);
        pForm->SetSigned(b2);
    }

    if (pBankDocument != NULL)
    {
        // no attributes may be changed after construction
    }
```

```
    if (pDate != NULL)
    {
        int n1, n2, n3, n4, n5, n6;

        Print("nDay    = "); n1 = ReadNumber();
        Print("nMonth  = "); n2 = ReadNumber();
        Print("nYear   = "); n3 = ReadNumber();
        Print("nHour   = "); n4 = ReadNumber();
        Print("nMinute = "); n5 = ReadNumber();
        Print("nSecond = "); n6 = ReadNumber();

        pDate->SetDay(n1);
        pDate->SetMonth(n2);
        pDate->SetYear(n3);
        pDate->SetHour(n4);
        pDate->SetMinute(n5);
        pDate->SetSecond(n6);
    }
}


// display stored objects and their debug info
void CTest::Show()
{
    // display stream
    ostringstream strOutput;

    // get number of objects
    int nCount = m_arClasses.size();
    // index counter
    int nIndex = 0;

    strOutput << "There are " << nCount << " classes in use." << endl;

    // process each object
    TIterator myIterator = m_arClasses.begin();
    while (myIterator != m_arClasses.end())
    {
        strOutput << "  class no. " << (nIndex++) << endl;

        // display type and debug info
        CBasicClass* pClass = *myIterator;
        strOutput << "    type = " << (pClass->ClassnameOf()) << "  valid="
                  << (pClass->IsValid()) << endl
                  << "     " << pClass->ShowDebug();

        // next object
        myIterator++;
    }

    // done
    Print(strOutput.str());
}


// only show object's types
string CTest::ShowTypes()
{
    ostringstream strOutput;

    // get amount of stored objects
    int nCount = m_arClasses.size();
    int nIndex = 0;

    // use an iterator to go through the array
    TIterator myIterator = m_arClasses.begin();
    while (myIterator != m_arClasses.end())
    {
        CBasicClass* pClass = *myIterator;

        // print an index counter an class type of each object
        strOutput << (nIndex++) << " - " << (pClass->ClassnameOf()) << endl;

        myIterator++;
    }
```

```
    return strOutput.str();
}


// print a string, use log file if neccessary
void CTest::Print(const string& strOutput)
{
    cout << strOutput.c_str();

    if (m_bUseLogFile)
        m_hLogFile << strOutput.c_str();
}


// read a number from console
int CTest::ReadNumber()
{
    // typed in number
    int nInput;

    // read number
    string strInput;

    do
    {
        // get string
        cin >> strInput;
    }
    while (strInput.empty());

    // convert string to integer, zero if failed
    nInput = atoi(strInput.c_str());

    // write explicitely to logfile because it is not routed through Print()
    if (m_bUseLogFile)
        m_hLogFile << nInput << endl;

    // return typed in number
    return nInput;
}
```

6.5.7.3 Praktikum.cpp

```
//////////////////////////////////////////////////////////
// Softwarebauelemente I, Praktikum
//
// author:          Stephan Brumme
// last changes:    March 27, 2001


#include "Test.h"

#include <iostream>
#include <string>


// main routine
void main()
{
    // ask user to type in log file name
    cout << "Welcome to the Test environment !" << endl << endl;
    cout << "Please enter the log file name (press [ESC]+[Return] for none): ";

    // get log file name
    string strLogFile;
    cin >> strLogFile;

    // create new test environment using log file name
    CTest myTest(strLogFile);

    // determine whether log file should be used
    if (myTest.IsLogFileUsed())
        cout << "Actions will be logged to '" << strLogFile << "'." << endl << endl;
    else
```

```
        cout << "Log file won't be used." << endl << endl;

    // run test environment
    myTest.Run();
}
```

6.6 Testprotokolle

6.6.1 ClassnameOf.log

```
Logfile created: 31.03.2001 - 08:13:43

1 - Create new object
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: (none)
Please choose: 1

1 - CDate
2 - CBankDocument
3 - CContract
4 - CForm
5 - CStandingOrder
0 - none

Please choose the class type: 1
Please choose type of construction (0=default, 1=copy, 2=init): 0

1 - Create new object
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: (none)
Please choose: 1

1 - CDate
2 - CBankDocument
3 - CContract
4 - CForm
5 - CStandingOrder
0 - none

Please choose the class type: 2
Please choose type of construction (0=default, 1=copy, 2=init): 0

1 - Create new object
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: (none)
Please choose: 2

There are 2 classes in use.
  class no. 0
    type = CDate  valid=0
     DEBUG info for 'CBasicClass'
      m_nDay    = 31
      m_nMonth  = 3
      m_nYear   = 2001
```

```
      m_nHour   = 10
      m_nMinute = 38
      m_nSecond = 30
  class no. 1
    type = CBankDocument  valid=0
     DEBUG info for 'CBasicClass'
      m_strTitle           =
      m_strAuthor          =
      m_strKey             =
      m_dtDateOfFoundation = DEBUG info for 'CDate'
      m_nDay    = 0
      m_nMonth  = 0
      m_nYear   = 0
      m_nHour   = 0
      m_nMinute = 0
      m_nSecond = 0


1 - Create new object
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: (none)
Please choose: 0
```

6.6.2 ClassnameOf2.log

```
Logfile created: 31.03.2001 - 10:38:23

1 - Create new object
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: (none)
Please choose: 1

1 - CDate
2 - CBankDocument
3 - CContract
4 - CForm
5 - CStandingOrder
0 - none

Please choose the class type: 1
Please choose type of construction (0=default, 1=copy, 2=init): 0

1 - Create new object
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: (none)
Please choose: 1

1 - CDate
2 - CBankDocument
3 - CContract
4 - CForm
5 - CStandingOrder
0 - none

Please choose the class type: 2
Please choose type of construction (0=default, 1=copy, 2=init): 0
```

```
1 - Create new object
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: (none)
Please choose: 2

There are 2 classes in use.
  class no. 0
    type = CDate  valid=0
     DEBUG info for 'CDate'
      m_nDay    = 31
      m_nMonth  = 3
      m_nYear   = 2001
      m_nHour   = 10
      m_nMinute = 38
      m_nSecond = 30
  class no. 1
    type = CBankDocument  valid=0
     DEBUG info for 'CBankDocument'
      m_strTitle           =
      m_strAuthor          =
      m_strKey             =
      m_dtDateOfFoundation = DEBUG info for 'CDate'
      m_nDay    = 0
      m_nMonth  = 0
      m_nYear   = 0
      m_nHour   = 0
      m_nMinute = 0
      m_nSecond = 0


1 - Create new object
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: (none)
Please choose: 0
```

6.6.3 AlterationDate.log

```
Logfile created: 18.04.2001 - 13:48:35

1 - Create new object
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: (none)
Please choose: 1

1 - CDate
2 - CBankDocument
3 - CContract
4 - CForm
5 - CStandingOrder
0 - none

Please choose the class type: 5
Please choose type of construction (0=default, 1=copy, 2=init): 0

1 - Create new object
```

```
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: (none)
Please choose: 3

There are 1 classes in use.
  class no. 0
    type = CStandingOrder  valid=0
     DEBUG info for 'CBankDocument'
      m_strTitle           =
      m_strAuthor          =
      m_strKey             =
      m_dtDateOfFoundation = DEBUG info for 'CDate'
      m_nDay    = 0
      m_nMonth  = 0
      m_nYear   = 0
      m_nHour   = 0
      m_nMinute = 0
      m_nSecond = 0

DEBUG info for 'CForm'
      m_bProved          = 0
      m_bSigned          = 0
      m_strFormality     =
      m_dtAlterationDate = DEBUG info for 'CDate'
      m_nDay    = 18
      m_nMonth  = 4
      m_nYear   = 2001
      m_nHour   = 13
      m_nMinute = 48
      m_nSecond = 41

DEBUG info for 'CStandingOrder'
      m_nAmount      = 0
      m_strSubject   =
      m_strSource    =
      m_strRecipient =
      m_nInterval    = 0
Please choose an object: 0

1 - Create new object
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: 0
Please choose: 5

DEBUG info for 'CBankDocument'
      m_strTitle           =
      m_strAuthor          =
      m_strKey             =
      m_dtDateOfFoundation = DEBUG info for 'CDate'
      m_nDay    = 0
      m_nMonth  = 0
      m_nYear   = 0
      m_nHour   = 0
      m_nMinute = 0
      m_nSecond = 0

DEBUG info for 'CForm'
      m_bProved          = 0
      m_bSigned          = 0
      m_strFormality     =
      m_dtAlterationDate = DEBUG info for 'CDate'
      m_nDay    = 18
      m_nMonth  = 4
      m_nYear   = 2001
```

```
      m_nHour   = 13
      m_nMinute = 48
      m_nSecond = 41

DEBUG info for 'CStandingOrder'
      m_nAmount      = 0
      m_strSubject   =
      m_strSource    =
      m_strRecipient =
      m_nInterval    = 0
nAmount             = 10
nInterval           = 20
bProved (true=1)    = 1
bSigned (true=1)    = 1

1 - Create new object
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: 0
Please choose: 2

There are 1 classes in use.
  class no. 0
    type = CStandingOrder  valid=0
     DEBUG info for 'CBankDocument'
      m_strTitle           =
      m_strAuthor          =
      m_strKey             =
      m_dtDateOfFoundation = DEBUG info for 'CDate'
      m_nDay    = 0
      m_nMonth  = 0
      m_nYear   = 0
      m_nHour   = 0
      m_nMinute = 0
      m_nSecond = 0

DEBUG info for 'CForm'
      m_bProved          = 1
      m_bSigned          = 1
      m_strFormality     =
      m_dtAlterationDate = DEBUG info for 'CDate'
      m_nDay    = 18
      m_nMonth  = 4
      m_nYear   = 2001
      m_nHour   = 13
      m_nMinute = 48
      m_nSecond = 56

DEBUG info for 'CStandingOrder'
      m_nAmount      = 10
      m_strSubject   =
      m_strSource    =
      m_strRecipient =
      m_nInterval    = 20

1 - Create new object
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: 0
Please choose: 0
```

6.6.4 CopyEqualValue.log

```
Logfile created: 18.04.2001 - 13:56:05
```

```
1 - Create new object
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: (none)
Please choose: 1

1 - CDate
2 - CBankDocument
3 - CContract
4 - CForm
5 - CStandingOrder
0 - none

Please choose the class type: 2
Please choose type of construction (0=default, 1=copy, 2=init): 0

1 - Create new object
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: (none)
Please choose: 3

There are 1 classes in use.
  class no. 0
    type = CBankDocument  valid=0
     DEBUG info for 'CBankDocument'
      m_strTitle           =
      m_strAuthor          =
      m_strKey             =
      m_dtDateOfFoundation = DEBUG info for 'CDate'
      m_nDay    = 0
      m_nMonth  = 0
      m_nYear   = 0
      m_nHour   = 0
      m_nMinute = 0
      m_nSecond = 0

Please choose an object: 0

1 - Create new object
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: 0
Please choose: 1

1 - CDate
2 - CBankDocument
3 - CContract
4 - CForm
5 - CStandingOrder
0 - none

Please choose the class type: 2
Please choose type of construction (0=default, 1=copy, 2=init): 1
0 - CBankDocument
Which object should be cloned: 0

1 - Create new object
2 - Show stored classes
3 - Select an object
```

```
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: 0
Please choose: 4

Source is object no. 0
0 - CBankDocument
1 - CBankDocument
Which object should be the destination object: 1
1 - Copy, 2 - EqualValue, 3 - Equal
2
Comparing: equal

1 - Create new object
2 - Show stored classes
3 - Select an object
4 - Copy/CompareObjects
5 - Set attributes
6 - Delete object
0 - Quit

Currently selected object: 0
Please choose: 0
```