1. Einleitung

1.1 Warum dieses Skript?

Designpatterns sind ein Schlagwort, welches – wie so viele Schlagworte – zunächst einmal mehr Fragen aufwirft als beantwortet.

Die erste Frage die sich aufdrängt lautet meistens: ***Was ist es?***
Unmittelbar gefolgt von: ***Ist es was Neues?***
Und schließlich gekrönt von: ***Was bringt es mir?***

Zunächst wiederholt mit dem Schlagwort konfrontiert, geschieht die Annäherung an eine „neue“ Thematik zumeist in drei Phasen:

Am Beginn der Auseinandersetzung mit einem eher theoretischen Thema steht meist eine Phase des eigenen Interesses.
Es folgt ein (manchmal auch größerer) Aha-Effekt, sobald man weiß, dass die Thematik durchaus praktische Bezüge hat.
Und am Ende folgt (bestenfalls) die Einsicht etwas Nützliches gelernt zu haben.

Die Entscheidung dieses Wissen weiter zu vermitteln (und sie in ein Skript wie dieses zu fassen), erfolgt meist erst deutlich später.
Meist erst, wenn man selbst die Erkenntnis gewonnen hat, dass die Thematik nützlich genug ist, die damit verbundene Arbeit auf sich zu nehmen.
So war es auch bei dem vorliegenden Skript, bei dem es von der Idee bis zum Beginn des Niederschreibens fast zwei Jahre gedauert hat.
Ausschlaggebend war aber auch die Einsicht, dass die verfügbare Literatur zu diesem Thema zumeist einen sehr hohen, abstrakt theoretischen Charakter hat – ein Umstand, welcher der Verbreitung eines solchen Themas nicht immer hilfreich ist.

1.2 Was sind eigentlich Designpatterns?

Will man den Begriff Designpatterns in einem Satz beschreiben, so lautet die gängige Antwort:

Entwurfsmuster sind gängige Lösungsansätze für häufig wiederkehrende Probleme der Anwendungsentwicklung.

Es hat sich im Zuge der Verbreitung von Objektorientierten Programmiersprachen gezeigt, dass sich viele Probleme mit immer den gleichen Lösungsmustern bearbeiten lassen.
Jeder Anwendungsentwickler entwickelt mit der Zeit eine ganze Reihe von solchen Lösungsstrategien, die er immer wieder auf ähnlich gelagerte Probleme anwendet, da sie sich in der Praxis bewährt haben.
Es sollte aber auch deutlich gesagt sein, dass nicht jede Anwendung unbedingt alle mögliche Pattern benötigt oder implementieren muss. Es ist völlig normal, dass sich in einer Applikation meist nur eine Auswahl möglicher Designpatterns findet.
Die Verwendung von Entwurfsmuster sind dementsprechend nicht etwas neues, sondern eher ein altbekanntes Phänomen.
Neu ist hingegen der Versuch der Systematisierung. Unter dem Begriff der Designpatterns wurden die Entwurfsmuster erstmalig systematisch von der so

genannten *Gang of Four*[1] (GoF) erfasst, analysiert, benannt und beschrieben. Auf der Arbeit der GoF (und eigener Erfahrung) fußt auch dieses Skript[2].
Da die meisten Designpattern im Laufe der Jahre (vor der Systematisierung) unter diversen Namen verwendet und in der Fachliteratur beschrieben wurden, sind diese alternativen Namen, bei der Beschreibung der einzelnen Pattern mit aufgelistet. Im Verzeichnisteil ist eine Abbildung der gängigen Namen auf die Namen der GoF Designpatterns aufgeführt.
Verbunden mit der systematischen Erfassung durch die GoF ist auch eine Klassifizierung in drei Mustergruppen:

Erzeugungsmuster: Designpattern, welche die Erzeugung von Instanzen betreffen. Sie dienen im Allgemeinen dazu den eigentlichen Erzeugungsprozess vor der Applikation[3] zu verbergen und damit von Änderungen in der Implementierung dieser Klassen zu entkoppeln.

Strukturmuster: Designpattern, die dazu verwendet werden können, größere Objekte aus Teilobjekten aufzubauen, wobei es häufig Ziel ist, die eigentliche Struktur vor der Applikation zu verbergen, um diese vom interner Aufbau der größeren Struktur zu lösen.

Verhaltensmuster: Designpattern, die beschreiben, wie Objekte miteinander agieren und kommunizieren. Ziel ist die Vermeidung von konkretem Wissen über die Methoden der Objekte untereinander.

Die Eingruppierung ist als Ergänzung zum Namen des jeweiligen Designpattern angegeben.

Es sei an dieser Stelle ausdrücklich betont, dass die hier vorliegende Liste keinerlei Anspruch auf Vollständigkeit erhebt. Es handelt sich um eine Auswahl gängiger Designpatterns, die einen Einstieg in die Thematik ermöglicht.
Es sei an dieser Stelle auch betont, dass die Liste der Designpattern nicht auf die durch die *Gang of Four* beschriebenen und klassifizierten Muster beschränkt ist – auch wenn viele Verfechter der Thematik dies glauben und vehement vertreten.

1.3 Was nützen Designpatterns?

Wie bereits betont, erfinden Designpatterns die Programmierung nicht neu, dennoch kann die Kenntnis der Muster sehr nützlich sein. Anstatt selbst zu versuchen eine Problematik zu lösen kann man auf erprobte Verfahren zurückgreifen, deren Stärken und Schwächen bekannt und beschrieben sind. Häufig kann man auf existierende und Beispiel- und Rumpfprogramme (Skeletons) zurückgreifen, die helfen können, Fehler zu vermeiden. Ein bewährtes Rezept abzuschauen (von dem man dann auch sicher sein kann, dass es funktioniert) ist allemal besser als es selbst zu erfinden.
Die Kenntnis der Designpattern erleichtert natürlich auch die Diskussion von Anwendungsentwicklern untereinander, da typische Lösungsstrategien nun über definierte und akzeptierte Namen verfügen. Einige der Designpatterns sind unter

[1] Dr. Erich Gamma, Dr. Richard Helm, Dr. Ralph Johnson und Dr. John Vlissides.
[2] Gamma et al, Entwurfsmuster – Elemente wieder verwendbarer Objektorientierter Software, Addision Wesley 1996
[3] Eigentlich die Gesamtheit des Programms, in diesem Skript verwendet im Sinne eines nicht näher spezifizierten Hauptprogramms, dessen Flexibilität maximiert und Abhängigkeit von Teilmodulen minimiert werden soll.

verschiedenen Namen geläufig, es kann daher sein, das man feststellt, dass man das entsprechende Muster bereits kennt und selbst verwendet, es aber anders bezeichnet.

Die Liste hier beschriebenen Designpattern ist weder vollständig noch inhaltlich umfassend. Sie bildet lediglich einen Einstieg in die bekanntesten und einfachsten Entwurfsmuster.

Nützlicher Nebeneffekt der Beschäftigung mit Designpattern ist wohl auch, dass man vielleicht das eine oder andere Entwurfsmuster neu kennen lernt und feststellt, dass man genau dieses gut gebrauchen kann (oder gut gebraucht hätte, bevor man das Problem irgendwie anders gelöst hat).
Die Festlegung und Beschreibung der einzelnen Designpatterns bedeutet aber nicht automatisch, dass es für jedes Problem eine einfache und eindeutige Lösungsmöglichkeit gibt, für die man nur noch in der entsprechenden Literatur nachsehen muss. Vielmehr überschneiden sich die Möglichkeiten und Anwendungsgebiete der einzelnen Designpatterns zum Teil deutlich, sind komplexere Erweiterungen einfacherer Muster oder nehmen ähnliche Aufgaben wahr, aber betrachtet unter einem anderen Blickwinkel. Die Auswahl und Nutzung des schließlich zur Anwendung gelangenden Musters hängt somit auch von der individuellen Einstellung des Anwendungsentwicklers zu den einzelnen Mustern ab. Manche Muster gefallen einem besser und andere weniger gut, manche mag man als sehr einfach, andere als zu umständlich einschätzen.

1.4 Welche Programmiersprachen kann man verwenden?

Entwurfsmuster lassen sich mit praktisch allen typsicheren, Objektorientierten Programmiersprachen verwenden. Die Beschränkung auf C++ und Java ist pragmatischer Natur und bedeutet natürlich nicht, dass die Muster mit anderen Sprachen nicht verwendet werden können.
Die Aufwand für die entsprechende Transferleistung in eine andere Objektorientierte Programmiersprache sollte nicht allzu hoch sein.

1.5 Was wird für das Verständnis benötigt?

Für die Auseinandersetzung mit den Entwurfsmustern benötigt man ein gutes abstraktes Vorstellungsvermögen und grundlegende Kenntnisse in der OOP[4]. Stichworte wie Vererbung, Polymorphie, statische Attribute und statische Methoden sollten zumindest grundsätzlich geläufig und verstanden sein.

1.6 Das klingt alles recht kompliziert oder?

Ja, einige Muster sind recht kompliziert und schwierig zu verstehen. Wenn man ein solches Muster nicht sofort versteht ist dies durchaus kein Beinbruch, sondern vielmehr ein Indiz dafür, das man ein entsprechendes Designpattern noch nicht benötigt hat. Zur Orientierung dienen die Abschnitte über den Verwendungszweck, es lohnt sich diese zu durchstöbern, wenn man eine Lösung sucht.

1.7 Anmerkung zu den Beispielen

Alle hier gezeigten Beispiele sind in ihrer Komplexität stark reduziert, um die eigentliche Technik hervortreten zu lassen. Daher finden sich z. T. Verweise in den

[4] OOP = Object Orientated Programming – Objektorientierte Programmierung

Kommentaren, die darauf verweisen, dass man an bestimmten Stellen Informationen eigentlich aus anderen Quellen holen müsste.
Da die Komplexität der Beispiele trotz der Vereinfachung z. T. sehr hoch ist, wurde aber an diesen Stellen auf eine Ausprogrammierung verzichtet.

Die Java-Beispiele wurden mit der Entwicklungsumgebung Eclipse (Version 2.1) unter Java 1.4 (Sun VM) erstellt.

Die C++ Beispiele wurden in Microsoft Visual Studio C++ .Net (Version 7.0) erstellt.
Einige C++ Codes zusätzlich auch unter dem Borland C++ Builder Version 4.
Die Beispiele sollten sich ohne großen Aufwand auch an andere C++ Compiler anpassen lassen, sofern diese über eine STL[5]-Implementierung verfügen.

[5] STL = Standard Template Library

2. Model View Control

2.1 Anmerkungen

Das Pattern „Model-View-Control“ (MVC) ist eins der bekanntesten und weitesten verbreiteten Muster für die Architektur interaktiver Systeme. Das MVC-Konzept es geht bereits auf die ersten Smalltalk-Implementierungen zurück und ist bis heute die Grundlage der modernen (Fenster-Basierten) GUI’s.

2.2 Verwendungszweck

Ziel bei der Gliederung einer Anwendung in MVC-Schichten ist die Erhöhung der Flexibilität und Reduktion der Komplexität von Programmen durch eine einheitliche Standard-Struktur.

2.3 Problemstellungen

Wie sollte man eine Applikation grundsätzlich zerlegen, um eine größtmögliche Flexibilität zu erreichen und ggf. Client-Server fähig zu sein.

2.4 Lösung

Das MVC-Pattern ist seit langem anerkannt , bewährt und de facto Stanf´dard für alle größeren Softwaresysteme. Die von MVC vorgesehene Schichtung der Software ist so grundlegend, dass MVC meist gar nicht mehr als Pattern innerhalb einer Applikation sondern als Voraussetzung für größere Programme angesehen wird.

2.5 Vorteile

MVC ist ein sehr leistungsfähiges Konzept, es strukturiert die Rollen einzelner Softwarekomponenten, die üblicherweise in einer Applikation vorkommen, nach einem standardisierten Muster.

Model (Modell) **ist die zu behandelnde Datenbasis.**

- Das Modell enthält keinerlei Logik (Plausibilität) und keinen Programmcode der zur Darstellung dient.
- Das Model besteht nur aus Daten stellt die darauf operierenden Getter- und Setter-Methoden (Lese- und Schreiboperationen) zur Verfügung.
- Einfache MVC Konzepte sehen die Anmeldung und das Abfragen von Views zur Benachrichtigung bei Veränderungen der Datenbasis vor. Es muss daher einen Mechanismus zur Benachrichtigung beliebiger Views nach Änderungen enthalten (Action/Listener).
- Modell ist prinzipiell unabhängig von Views.
- Es muss eine beliebige Zahl von Views möglich sein.
- Es müssen beliebige Arten von Views möglich sein.

Views (Ansichten) **sind Präsentationen des Models.**

- Die Views sind die Ausgabenkomponenten, sie stellen die Daten des Models dar (z.B. graphisch in einem Fenster).

- Die Views erhalten die auszugebenden Daten vom Model, sie verwenden dazu die Leseoperationen des Modells.
- Eine View wird vom Modell über Änderungen am Modell benachrichtigt.
- Es kann mehrere, verschiedene Views zu einem Model geben.
- Die Views müssen laufend (bei Änderung des Models) aktualisiert werden. Eine View ist daher vom Modell abhängig und muss sich diesem registrieren.

Control (Steuerung) **ist die Einflussnahme auf die Daten des Models**

- Der Controller reagiert auf die Eingaben des Benutzers.
- Der Controller liefert die Daten an Benutzer weiter (d.h. an die View).
- Der Controller kümmert sich um die Logik (Plausibilität), er verwendet dazu die Schreiboperationen (ev. auch die Leseoperationen) des Modells.

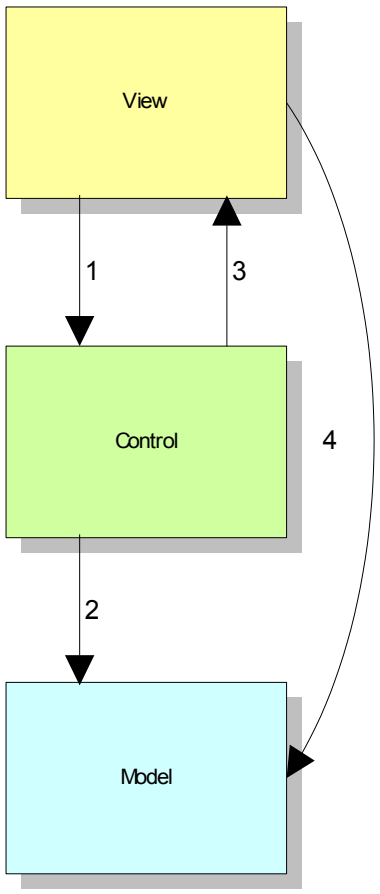

Abbildung 2-1 – ein einfaches MVC Pattern

In der Praxis tritt das MVC Pattern immer wieder in Abstufungen von MVC-Rollen auf. Eine View kann selbst wiederum als Model betrachtet werden, auf dem es (Teil-)Views gibt. So ist ein Geburtstagsdatum eine View vom Model Person, aber das Datum selbst kann verschiedene Formatierungen (d.h. Views) haben

1. Die View erhält ein Dialogereignis vom Benutzer und ruft eine entsprechende Funktion der Controller-Schicht zur Umsetzung des Ereignisses auf.
2. Die Controller-Schicht ändert das Model entsprechend der Eingaben mit passenden Methoden (Getter/Setter), die das Model bereitstellt.
3. Der Controller löst die Änderungen in der Darstellung der View aus, die sich als Folge der Modeländerungen ergeben haben.
4. Die View holt sich je nach Bedarf zusätzliche Daten vom Model, um sie darzustellen.

2.6 Nachteile

Wie immer ist die Wahrheit etwas komplizierter und moderne Applikationssysteme weisen deutlich mehr Schichten auf, auch wenn diese sich grob als MVC Konzept auffassen lassen.
Der Zusatzaufwand für eine saubere MVC-Schichtung ist daher gerade bei überschaubaren Softwaresystemen sehr hoch.

Abb. 1.1 Ein einfaches MVC-Pattern

So enthalten zum Beispiel die meisten Applikationen eine getrennte Persistenz-Schicht, die üblicherweise dem Controller hinzugerechnet oder als separate Schicht betrachtet wird.
Hierdurch kann die unterliegende Datenbank leichter ausgetauscht werden.

Das Model besteht zumeist aus vielen kleinen Teil-Modellen, die zu größeren, logischen Einheiten zusammengefasst werden.

Da der Controller sowohl die View als auch die Logik umfassen soll, man aber die Plausibilität des Models unabhängig von der implementierten View halten möchte, zerfällt der Controller häufig in einen View- und einen Model-Controller.

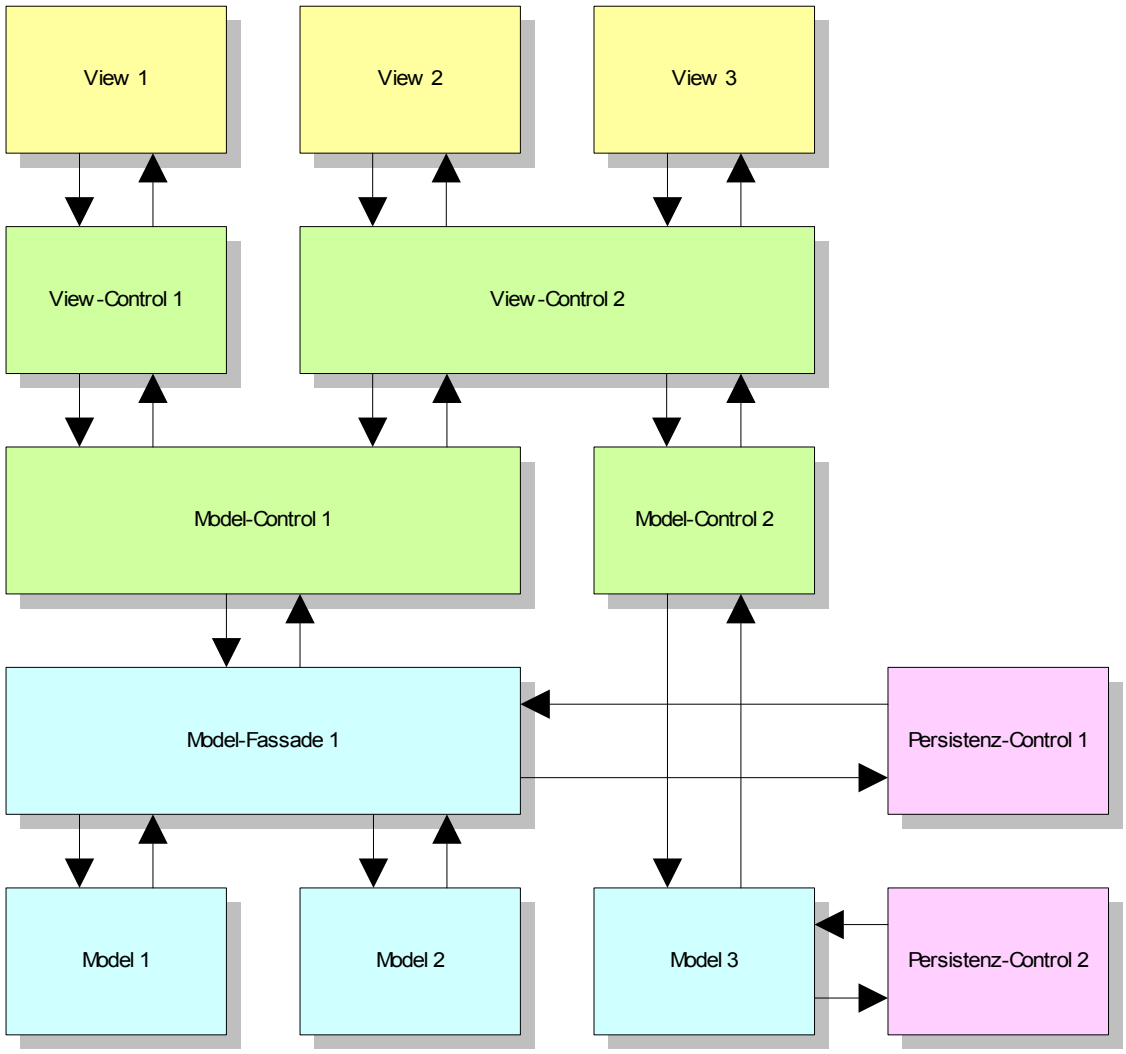

Abbildung 2-2 – ein komplexes MVC-Pattern

Beide Controller-Formen müssen ihre Arbeit ggf. über mehrere Teil-Modelle spannen, was vom eigentlichen Modell abstrahierende Model-Sichten (Fassaden) nahe legt.

Die physikalische Abbildung des Models in Datenbanken (Persistenz) muss dem Model nicht 1:1 entsprechen, auch hier kann die Verwendung logischer Aggregations- oder Separierungsobjekte sinnvoll sein.

In komplexen MVC-Pattern empfiehlt es sich, die Menge der Kommunikationswege so gering wie möglich zu halten. Unter Umständen ist es sinnvoller von der View aus ***nicht*** direkt auf das Model zuzugreifen (wie im einfachen Beispiel), sondern die gesamte Kommunikation über den Controller abzuhandeln. Dies hätte zudem den Vorteil, dass die View stärker vom Model entkoppelt ist und Änderungen im Model nicht auf zwangsläufig auf die View- und Controller-Schicht durchschlagen.

3. Template Method

Alternativnamen: Schablonenmethode
Mustergruppe: Verhaltensmuster

3.1 Anmerkungen

Das Template Method Design Pattern ist wahrscheinlich das einfachste Designpattern überhaupt. Es erfordert nur ein Minimum an abstraktem Denken und ist – wenn Vererbung eingesetzt wird – in praktisch jedem Programm nützlich.

3.2 Verwendungszweck

Bereitstellung eines schematischen Algorithmus, welcher bei Bedarf ganz oder teilweise von abgeleiteten Objekten überschrieben werden kann. Ein sehr einfaches Beispiel wäre z.B. die Sicherung von Objekten in einer Datei. Dazu gehört der gesamte Code, der z.B. feststellt, ob eine Datei des gewünschten Namens bereits existiert, ob diese ggf. überschrieben werden soll, bis hin zum Öffnen der Datei (mit unterschiedlichen Schreibattributen), dem Schreiben selbst und dem Schließen.
Egal welches Objekt in einer Datei gesichert werden soll, bis auf das Schreiben selbst sind die Schritte (wahrscheinlich) vollkommen identisch. Es wäre daher unlogisch diese Schritte mehrfach zu implementieren, andererseits kann man diese Funktionalität auch nicht erben, da ja der Prozess des Schreibens unterschiedlich ist.

3.3 Problemstellungen

Wie kann man ähnliche Vorgehensweisen verallgemeinern, so dass nur die unterschiedlichen Aspekte in spezialisierten (abgeleiteten) Klassen überschrieben werden müssen.

3.4 Lösung

Das Template Method Pattern löst dieses Problem, indem der gewünschte Algorithmus nicht in einer Methode implementiert wird, sondern einzelne Teile des Algorithmus in nichtöffentlichen, aber überschreibbaren Methoden definiert werden.
Bezogen auf das oben beschriebene Beispiel bedeutet dies, das die Methode zum Sichern des Objektinhaltes in einer Datei grob in drei Teile zerfällt (Prüfen und Öffnen, Schreiben und Schließen der Datei). Diese drei Teile werden in virtuelle Methoden ausgegliedert, so dass sie später bei Bedarf von einer abgeleiteten Klasse überschrieben werden können.

3.5 Vorteile

Durch die Verwendung des Template Method Patterns werden Redundanzen vermieden, durch die ggf. schwer auffindbare Copy & Paste Fehler entstehen können. Zugleich wird das algorithmische Vorgehen innerhalb der Template Methoden deutlicher, da die Details der Implementation in die Unterfunktionen delegiert werden.

3.6 Nachteile

Hauptnachteile dieser Vorgehensweise ist eine starke Zersplitterung der einzelnen Algorithmen, deren Implementierung ggf. auf mehrere Hierarchieebenen oberhalb einer abgeleiteten Klasse verteilt ist. Die tatsächliche Verarbeitung ist dadurch schwieriger nachzuvollziehen.

Durch die Aufsplittung in mehrere Teilschritte sind deutlich mehr Methoden zu schreiben und auch das ***private*** Interfaceteil der Klasse wird unübersichtlicher.

3.7 C++ Beispiel

Das Beispiel zeigt eine Template-Method zur Sicherung von Daten in einer Datei. Es zeigt die Verwendung des Template Method Patterns sowohl in Einfach- (TMProdukt) wie in der Mehrfachvererbung (TMKunde).

3.7.1 C++ Klassendiagramm

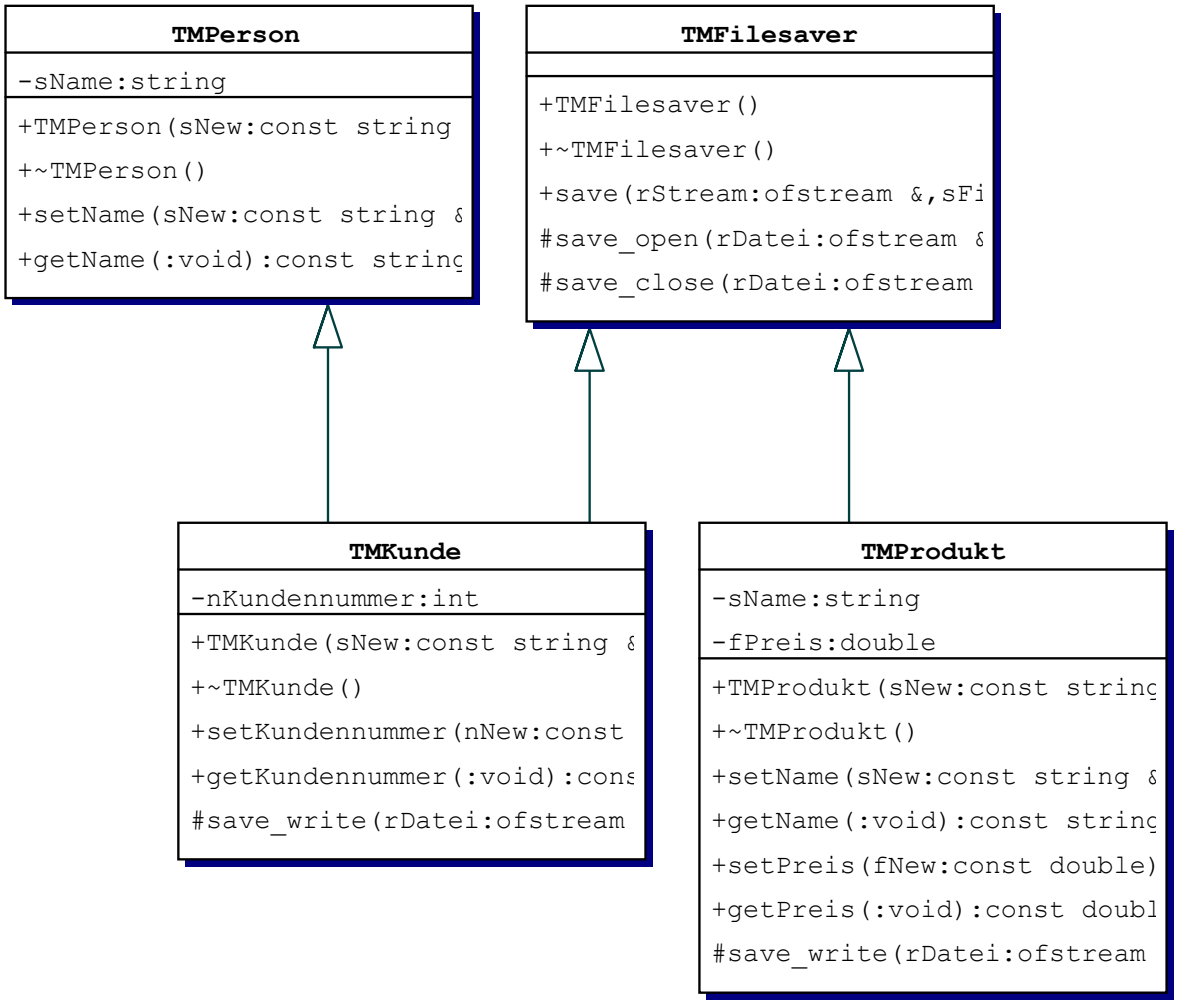

Abbildung 3-1 C++ Klassendiagramm Template Method

3.7.2 C++ Beispielprogramm

```
//============================================================================
// VISUAL STUDIO C++ V7.0
// Borland C++ Builder V4.0
// TemplateMethod.cpp: Definiert den Einstiegspunkt für Konsolenanwendung
//============================================================================

#include "stdafx.h"
#include "TMHauptprogramm.h"

int _tmain(int argc, _TCHAR* argv[])
{
      TemplateMethodTest();
      return 0;
}
```

```
//---------------------------------------------------------------------------
// Borland C++ Builder V4.0
// TemplateMethod.cpp: Definiert den Einstiegspunkt für Konsolenanwendung
//---------------------------------------------------------------------------

#pragma hdrstop
#include <condefs.h>

// ------------- Borland C++ Builder ---------------------------------------
USEUNIT("TMFilesaver.cpp");
USEUNIT("TMPerson.cpp");
USEUNIT("TMProdukt.cpp");
USEUNIT("TMKunde.cpp");
USEUNIT("TMHauptprogramm.cpp");
// -------------------------------------------------------------------------

#include "TMHauptprogramm.h"

#pragma argsused
int main(int argc, char* argv[])
{
      TemplateMethodTest();
      return 0;
}
```

```
//---------------------------------------------------------------------------
// C++ Template Method Designpattern - TMHauptprogramm.h
//---------------------------------------------------------------------------
#ifndef _TMHAUPTPROGRAMM_H_
#define _TMHAUPTPROGRAMM_H_

    void TemplateMethodTest (void);

#endif
```

```
//---------------------------------------------------------------------------
// C++ Template Method Designpattern - TMHauptprogramm.cpp
//---------------------------------------------------------------------------
#include "stdafx.h"    // Entfernen für Borland
#include "TMKunde.h"
#include "TMProdukt.h"
#include <vector>
#include <iostream>

void TemplateMethodTest (void)
{
   vector<TMFilesaver *> aObjekte;
   vector<TMFilesaver *>::const_iterator aI;

   aObjekte.push_back (new TMKunde("Heinz", 12));
   aObjekte.push_back (new TMKunde("Fritz", 44));
   aObjekte.push_back (new TMProdukt("Zahnpasta", 0.59));

   ofstream aStream;
   int      i = 0;
   string   sFilename;
   char     sUmwandlung[100];
   for (aI=aObjekte.begin(); aI<aObjekte.end(); aI++)
   {
      itoa(i, sUmwandlung, 10);
```

```
      sFilename = "c:\\File_";
      sFilename.append(sUmwandlung);
      (*aI)->save (aStream, sFilename);
      i++;
   }

   for (unsigned int i=0; i<aObjekte.size(); i++)
   {
       TMFilesaver *rTemp = aObjekte[i];
       delete rTemp;
   }
}
```

```
//============================================================================
// C++ Template Method Designpattern - TMPerson.h
//============================================================================
#ifndef _TMPERSON_H_
#define _TMPERSON_H_

#include <fstream>
#include <string>

using namespace std;

class TMPerson
{
   //------------------------------------------------------------------------
   // Methoden
   //------------------------------------------------------------------------
   public:
               TMPerson (const string & sNew);
      virtual ~TMPerson ();

              void   setName    (const string & sNew);
      const string & getName    (void);

   //------------------------------------------------------------------------
   // Attribute
   //------------------------------------------------------------------------
   private:
      string sName;
};

#endif
```

```
//============================================================================
// C++ Template Method Designpattern - TMPerson.cpp
//============================================================================
#include "stdafx.h"        // Entfernen für Borland
#include "TMPerson.h"

//----------------------------------------------------------------------------
// Constructor
//----------------------------------------------------------------------------
TMPerson::TMPerson (const string & sNew)
{
   sName = sNew;
}

//----------------------------------------------------------------------------
// Destructor
```

```
//----------------------------------------------------------------------------
TMPerson::~TMPerson ()
{
}

//----------------------------------------------------------------------------
// setName (const string &)
//----------------------------------------------------------------------------
void TMPerson::setName (const string & sNew)
{
   sName = sNew;
}

//----------------------------------------------------------------------------
// getName (void)
//----------------------------------------------------------------------------
const string & TMPerson::getName (void)
{
   return sName;
}
```

```
//============================================================================
// C++ Template Method Designpattern - TMFilesaver.h
//============================================================================
#ifndef _TMFILESAVER_H_
#define _TMFILESAVER_H_

#include <string>
#include <fstream>

using namespace std;

class TMFilesaver
{
   //-------------------------------------------------------------------------
   // Methoden
   //-------------------------------------------------------------------------
   public:
               TMFilesaver ();
      virtual ~TMFilesaver ();

      virtual void save (ofstream & rStream, const string & sFilename);

   protected:
      virtual void save_open    (ofstream & rDatei,
                                 const string & sFilename);
      virtual void save_write   (ofstream & rDatei) = NULL;
      virtual void save_close   (ofstream & rDatei);
};

#endif
```

```
//============================================================================
// C++ Template Method Designpattern - TMFilesaver.cpp
//============================================================================
#include "stdafx.h"        // Entfernen für Borland
#include "TMFilesaver.h"

//----------------------------------------------------------------------------
// Constructor
//----------------------------------------------------------------------------
```

```
TMFilesaver::TMFilesaver ()
{
}

//---------------------------------------------------------------------------
// Destructor
//---------------------------------------------------------------------------
TMFilesaver::~TMFilesaver ()
{
}

//---------------------------------------------------------------------------
// save_open Standardmethode. Kann bei Bedarf geändert werden
//---------------------------------------------------------------------------
void TMFilesaver::save_open (ofstream & rDatei, const string & sFilename)
{
   string sFullname(sFilename);
   sFullname.append(".sav");
   rDatei.open (sFullname.c_str());
}

//---------------------------------------------------------------------------
// Template Method save
//---------------------------------------------------------------------------
void TMFilesaver::save (ofstream & rDatei, const string & sFilename)
{
   save_open (rDatei, sFilename.c_str());
   if (rDatei)
   {
      save_write (rDatei); // pure virtual !
      save_close (rDatei);
   }
}

//---------------------------------------------------------------------------
// save_close Standardmethode. Kann bei Bedarf geändert werden
//---------------------------------------------------------------------------
void TMFilesaver::save_close (ofstream & rDatei)
{
   rDatei.close();
}
```

```
//===========================================================================
// C++ Template Method Designpattern - TMKunde.h
//===========================================================================
#ifndef _TMKUNDE_H_
#define _TMKUNDE_H_

#include <fstream>
#include <string>
#include "tmperson.h"
#include "tmfilesaver.h"

using namespace std;

class TMKunde : public TMPerson,  public TMFilesaver
{
   //---------------------------------------------------------------------------
   // Methoden
   //---------------------------------------------------------------------------
   public:
               TMKunde (const string & sNew, const int nNew);
```

```
      virtual ~TMKunde ();

            void setKundennummer (const int nNew);
      const int  getKundennummer (void);

   protected:
      virtual void save_write (ofstream & rDatei);

   //---------------------------------------------------------------------
   // Attribute
   //---------------------------------------------------------------------
   private:
      int nKundennummer;
};

#endif
```

```
//=====================================================================
// C++ Template Method Designpattern - TMKunde.cpp
//=====================================================================
#include "stdafx.h"        // Entfernen für Borland
#include "TMKunde.h"

//---------------------------------------------------------------------
// Constructor
//---------------------------------------------------------------------
TMKunde::TMKunde (const string & sNew, const int nNew) : TMPerson (sNew)
{
   nKundennummer = nNew;
}

//---------------------------------------------------------------------
// Destructor
//---------------------------------------------------------------------
TMKunde::~TMKunde ()
{
}

//---------------------------------------------------------------------
// Überladen der Template Method save_write
//---------------------------------------------------------------------
void TMKunde::save_write (ofstream & rDatei)
{
   rDatei << "TMKunde:" << endl;
   rDatei << getName() << endl;
   rDatei << nKundennummer << endl;
}

//---------------------------------------------------------------------
// setKundennummer (const int)
//---------------------------------------------------------------------
void TMKunde::setKundennummer (const int nNew)
{
   nKundennummer = nNew;
}

//---------------------------------------------------------------------
// getKundennummer (void)
//---------------------------------------------------------------------
const int TMKunde::getKundennummer (void)
{
   return nKundennummer;
```

```
}
```

```
//============================================================================
// C++ Template Method Designpattern - TMProdukt.h
//============================================================================
#ifndef _TMPRODUKT_H_
#define _TMPRODUKT_H_

#include <fstream>
#include <string>
#include "tmfilesaver.h"

using namespace std;

class TMProdukt : public TMFilesaver
{
   //---------------------------------------------------------------------
   // Methoden
   //---------------------------------------------------------------------
   public:
               TMProdukt (const string & sNew, const double fNew);
      virtual ~TMProdukt ();

              void   setName    (const string & sNew);
      const string & getName    (void);
              void   setPreis   (const double fNew);
      const double   getPreis   (void);

   protected:
      virtual void save_write (ofstream & rDatei);

   //---------------------------------------------------------------------
   // Attribute
   //---------------------------------------------------------------------
   private:
      string sName;
      double fPreis;
};

#endif
```

```
//============================================================================
// C++ Template Method Designpattern - TMProdukt.cpp
//============================================================================
#include "stdafx.h"        // Entfernen für Borland
#include "TMProdukt.h"

//---------------------------------------------------------------------
// Constructor
//---------------------------------------------------------------------
TMProdukt::TMProdukt (const string & sNew, const double fNew)
{
   sName  = sNew;
   fPreis = fNew;
}

//---------------------------------------------------------------------
// Destructor
//---------------------------------------------------------------------
TMProdukt::~TMProdukt ()
{
```

```
}

//---------------------------------------------------------------------------
// Überladen der Template Method save_write
//---------------------------------------------------------------------------
void TMProdukt::save_write (ofstream & rDatei)
{
   rDatei << "TMProdukt:" << endl;
   rDatei << sName  << endl;
   rDatei << fPreis << endl;
}

//---------------------------------------------------------------------------
// setName (const string &)
//---------------------------------------------------------------------------
void TMProdukt::setName (const string & sNew)
{
   sName = sNew;
}

//---------------------------------------------------------------------------
// getName (void)
//---------------------------------------------------------------------------
const string & TMProdukt::getName (void)
{
   return sName;
}

//---------------------------------------------------------------------------
// setPreis (const double)
//---------------------------------------------------------------------------
void TMProdukt::setPreis (const double fNew)
{
   fPreis = fNew;
}

//---------------------------------------------------------------------------
// getPreis (void)
//---------------------------------------------------------------------------
const double TMProdukt::getPreis (void)
{
   return fPreis;
}
```

3.7.3 Anmerkungen zum C++ Beispiel

Um Slicing-Phänomene zu vermeiden, ist die Verwendung von Pointern im Vector zwingend gegeben. Es ist darauf zu achten, dass der Speicherplatz auch wieder freigegeben wird.

3.8 Java Beispiel

Das Beispiel zeigt eine Template-Method zur Sicherung von Daten in einer Datei. Im, Gegensatz zum C++ Beispiel wurde die Vererbungshierarchie linearisiert, da Java die Mehrfachvererbung nicht unterstützt.

3.8.1 Java Klassendiagramm

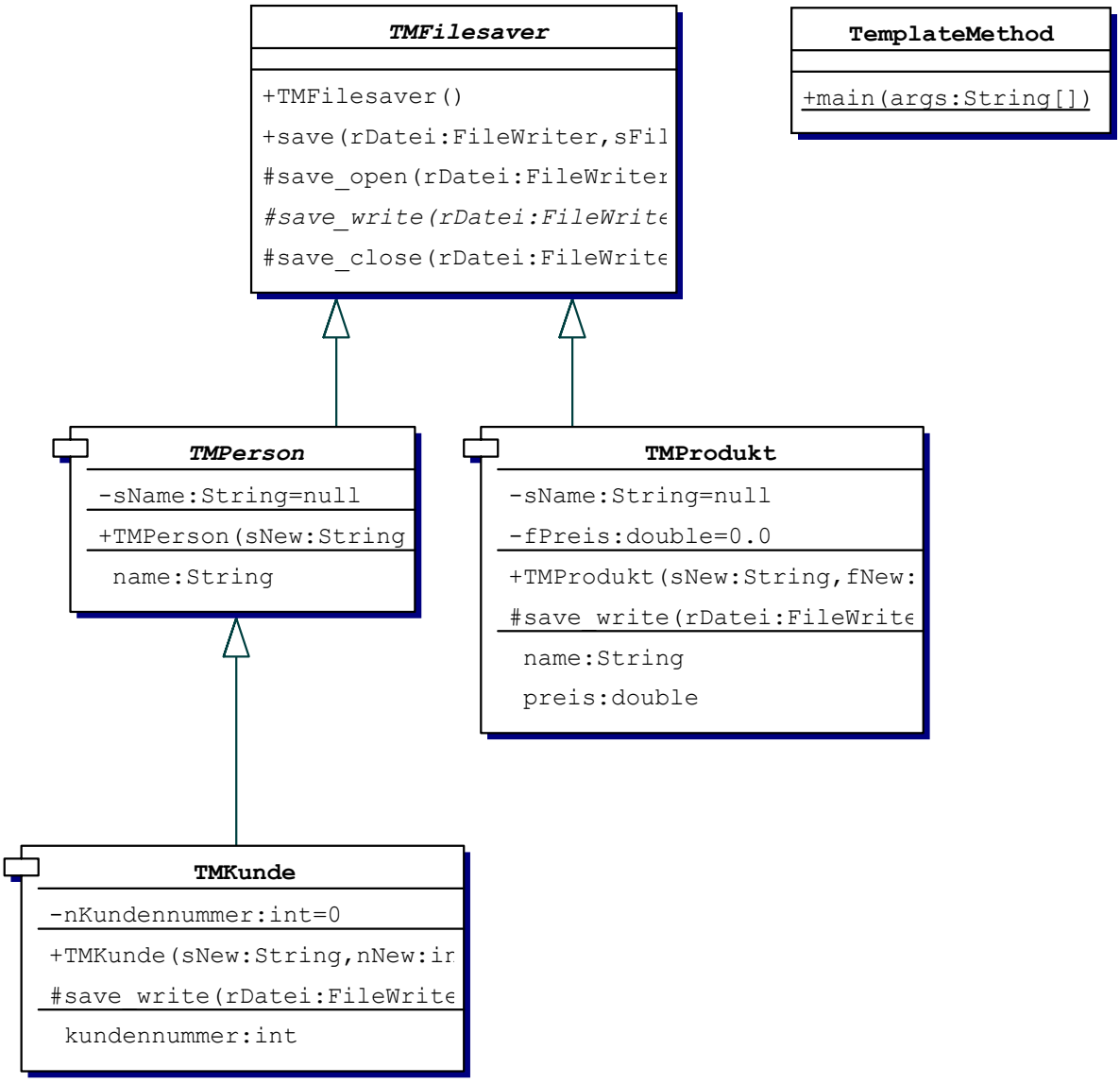

Abbildung 3-2 Java Klassendiagramm Template Method

3.8.2 Java Beispielprogramm

```
//=========================================================================
/** Java Template Method Design Pattern - Testprogramm
 *  Created on 18.04.2003
 */
//=========================================================================
import java.io.FileWriter;
import java.util.Vector;

public class TemplateMethod
```

```
{
   //---------------------------------------------------------------------
   // Hauptprogramm
   //---------------------------------------------------------------------
   public static void main(String[] args)
   {
      Vector aObjekte = new Vector();

      aObjekte.add (new TMKunde("Heinz", 12));
      aObjekte.add (new TMKunde("Fritz", 44));
      aObjekte.add (new TMProdukt("Zahnpasta", 0.59));

      FileWriter   aStream      = null;
      int          i            = 0;
      int          aI           = 0;
      String       sFilename    = null;
      TMFilesaver  rTMFilesaver = null;

      for (i=0; i<aObjekte.size(); i++)
      {
         try
         {
            sFilename = new String ("c:\\File_" + i);
            rTMFilesaver =(TMFilesaver)(aObjekte.elementAt(i));
            aStream = rTMFilesaver.save (aStream, sFilename);
         }
         catch (Exception e)
         {
            e.printStackTrace();
         }
      }
   }
}
```

```
//============================================================================
/** Java Template Method Design Pattern -
 *  TMFilesaver, abstrakte Basisklasse
 *  Created on 18.04.2003
 */
//============================================================================
import java.io.FileWriter;

public abstract class TMFilesaver
{
   //---------------------------------------------------------------------
   // Constructor
   //---------------------------------------------------------------------
   public TMFilesaver ()
   {
   }

   //---------------------------------------------------------------------
   // Template Method save
   //---------------------------------------------------------------------
   public FileWriter save (FileWriter rDatei, String sFilename)
   {
      rDatei = save_open (rDatei, sFilename);
      if (rDatei != null)
      {
         save_write (rDatei); // abstract !
         rDatei = save_close (rDatei);
      }
```

```
      return rDatei;
   }

   //-----------------------------------------------------------------
   // save_open Standardmethode. Kann bei Bedarf geändert werden
   //-----------------------------------------------------------------
   protected FileWriter save_open (FileWriter rDatei, String sFilename)
   {
      StringBuffer sFullname = new StringBuffer (sFilename);
      sFullname.append(".sav");
      try
      {
         rDatei = new FileWriter (sFullname.toString());
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
      return rDatei;
   }

   //-----------------------------------------------------------------
   // save_write muss in nicht-abstrakter Klasse überschrieben werden
   //-----------------------------------------------------------------
   protected abstract void save_write (FileWriter rDatei);

   //-----------------------------------------------------------------
   // save_close Standardmethode. Kann bei Bedarf geändert werden
   //-----------------------------------------------------------------
   protected FileWriter save_close (FileWriter rDatei)
   {
      try
      {
         if (rDatei != null) rDatei.close();
         rDatei = null;
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
      return rDatei;
   }
}
```

```
//=========================================================================
/** Java Template Method Design Pattern - TMProdukt
 *  Erweiterung von TMFilesaver
 *  Created on 18.04.2003
 */
//=========================================================================
import java.io.FileWriter;

public class TMProdukt extends TMFilesaver
{
   private String sName  = null;
   private double fPreis = 0.0;

   //-----------------------------------------------------------------
   // Constructor
   //-----------------------------------------------------------------
   public TMProdukt (String sNew, double fNew)
   {
```

```
      super();
      sName = new String (sNew);
      fPreis = fNew;
   }

   //----------------------------------------------------------------------
   // save_write muss hier überschrieben werden
   //----------------------------------------------------------------------
   protected void save_write (FileWriter rDatei)
   {
      try
      {
         rDatei.write("TMProdukt:");
         rDatei.write("\n");
         rDatei.write(getName());
         rDatei.write("\n");
         rDatei.write(Double.toString(fPreis));
         rDatei.write("\n");
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
   }

   //----------------------------------------------------------------------
   // setName (String)
   //----------------------------------------------------------------------
   public void setName (String sNew)
   {
      sName = new String (sNew);
   }

   //----------------------------------------------------------------------
   // getName ()
   //----------------------------------------------------------------------
   public String getName ()
   {
      return new String (sName);
   }

   //----------------------------------------------------------------------
   // setPreis (double)
   //----------------------------------------------------------------------
   public void setPreis (double fNew)
   {
       fPreis = fNew;
   }

   //----------------------------------------------------------------------
   // getPreis ()
   //----------------------------------------------------------------------
   public double getPreis ()
   {
      return fPreis;
   }
}
```

```
//============================================================================
/** Java Template Method Design Pattern - TMPerson
 *  Erweiterung von TMFilesaver
 *  Created on 18.04.2003
```

```
 */
//============================================================================
public abstract class TMPerson extends TMFilesaver
{
   private String sName = null;

   //------------------------------------------------------------------------
   // Constructor
   //------------------------------------------------------------------------
   public TMPerson (String sNew)
   {
      super();
      sName = new String (sNew);
   }

   //------------------------------------------------------------------------
   // setName (String)
   //------------------------------------------------------------------------
   public void setName (String sNew)
   {
      sName = new String (sNew);
   }

   //------------------------------------------------------------------------
   // getName ()
   //------------------------------------------------------------------------
   public String getName ()
   {
      return new String (sName);
   }
}
```

```
//============================================================================
/** Java Template Method Design Pattern - TMKunde, Erweiterung von TMPerson
 *  Created on 18.04.2003
 */
//============================================================================
import java.io.FileWriter;

public class TMKunde extends TMPerson
{
   private int nKundennummer = 0;
   //------------------------------------------------------------------------
   // Constructor
   //------------------------------------------------------------------------
   public TMKunde (String sNew, int nNew)
   {
      super(sNew);
      nKundennummer = nNew;
   }

   //------------------------------------------------------------------------
   // save_write muss hier überschrieben werden
   //------------------------------------------------------------------------
   protected void save_write (FileWriter rDatei)
   {
      try
      {
         rDatei.write("TMKunde:");
         rDatei.write("\n");
         rDatei.write(getName());
         rDatei.write("\n");
```

```
         rDatei.write(Integer.toString(nKundennummer));
         rDatei.write("\n");
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
   }

   //-----------------------------------------------------------------------
   // setKundennummer (int)
   //-----------------------------------------------------------------------
   public void setKundennummer (int nNew)
   {
      nKundennummer = nNew;
   }

   //-----------------------------------------------------------------------
   // getKundennummer ()
   //-----------------------------------------------------------------------
   public int getKundennummer ()
   {
   return nKundennummer;
   }
}
```

3.8.3 Anmerkungen zum Java Beispiel

Da Java mit Referenzen arbeitet, entfällt eine explizite Freigabe, wie in C++ erforderlich.

4. Singleton

Mustergruppe: Erzeugungsmuster

4.1 Anmerkungen

Singleton gehört zu den einfacheren und bekannteren Designpatterns, die meisten Entwickler sind vermutlich mit diesem Muster vertraut. Das Singleton Designpattern bildet die Grundlage einiger anderer Designpatterns, daher wird es hier als erstes betrachtet.

4.2 Verwendungszweck

Das Singleton Designpattern stellt sicher, das es nur eine einzige (globale) Instanz einer Klasse geben kann.
Dieses Muster ist besonders nützlich um:

- Den Zugang zu bestimmten Ressourcen zu steuern.
  Dazu gehört z.B. der Zugriff auf eine Datenbank, bei der die mehrfache Instanziierung einer Klasse zum Datenbankzugriff ohne automatisches Sperren (Locking) gegebenenfalls zu konkurrierenden Updates („last write wins“) führen kann.
- Die Instanziierung überflüssiger Objekte zu vermeiden.
  Eine Klasse, die alle in einer Anwendung verwendeten Konstanten beinhaltet oder den lesenden Zugriff auf eine Datenbank ermöglicht, welche solcherlei Konstanten bereithält, muss innerhalb einer Anwendung nicht mehrfach existieren.

4.3 Problemstellungen

Wie kann verhindert werden, dass mehr als eine Instanz der Klasse erzeugt wird?
Sollte die Singleton-Instanz als globale oder dynamische Variable erzeugt werden?

4.4 Lösung

Es gibt nur einen Weg die Anzahl von Instanzen einer Klasse unter Kontrolle zu bringen – indem der Constructor als ***private*** oder ***protected*** deklariert wird. Da somit die direkte Erzeugung einer Instanz verboten ist, muss ein indirektes Erzeugen ermöglicht werden.
Realisiert wird dies, indem man eine statische Variable deklariert, welche die einzig erlaubte Klasseninstanz enthält. Diese Instanz wird beim ersten Zugriff automatisch erzeugt (siehe unten, ***get()*** Methode). Da die Methode ***get()*** für den Zugriff auf diese Instanz bereits ohne Instanz aufrufbar sein muss, kann diese nur als statische Methode programmiert werden.
Die dynamische Erzeugung der Singleton-Instanz ist aus mehreren Gründen der globalen Definition vorzuziehen.
Erstens wird die Instanz nur dann angelegt, wenn auch tatsächlich ein Element der Singleton-Instanz benötigt wird.
Zweitens ist nicht sicher gestellt, dass die zur Initialisierung notwendigen Ressourcen bereits verfügbar sind (dies trifft insbesondere dann zu, wenn die Konstanten einer externen Quelle wie einer Datenbank entnommen werden).
Drittens ist die Reihenfolge der Erstellung von globalen Objekten nicht definiert und auch nicht beeinflussbar. Gibt es mehrere Singletons, die voneinander abhängig sind

oder voneinander erben, so kann dies aufgrund der Reihenfolge bei der Erstellung zu Fehlern oder Inkonsistenzen führen.
Eine häufig anzutreffende Anforderung erfordert z.B. eine begrenzte Anzahl von Instanzen (z.B. eine Instanz pro angemeldeten Benutzer). In einem solchen Fall wird in der Singleton-Instanz eine Liste (z.B. ein ***Vector***) statt eines Pointer oder einer Referenz verwendet. Bei der Abfrage (Aufruf der ***get()*** Methode) wird nun eine benutzerspezifische Kennung übergeben, die dazu dient, die richtige Instanz herauszusuchen, bzw. bei Bedarf eine neue Instanz anzulegen.
Vor

4.5 Vorteile

Die erforderlichen Instanzen werden nur bei Bedarf angelegt.
Die Anzahl der über die Singleton-Instanz verwalteten Objekte ist meistens genau Eins, muss nicht auf ein einziges Objekt beschränkt sein. Prinzipiell bleibt die Anzahl aber begrenzt.
Es ist möglich von einem Singleton zu erben, d.h. diesen zu spezialisieren. Die Auswahl des korrekten Typs der Instanz kann (anders als bei globalen Variablen) zur Laufzeit erfolgen.

4.6 Nachteile

Es muss sichergestellt werden, dass die Instanz des Singletons auch wieder abgebaut werden kann (z.B. wenn alle zugreifenden Programme beendet sind). Dies ist häufig schwierig zu entscheiden und umzusetzen.
Da die freizugebende Instanz in der singetonklasse gekapselt ist, muss eine selbstgeschriebene Methode (hier: ***remove()***) diese Aufagbe übernehmen.

4.7 C++ Beispiel

Das folgende Beispiel zeigt eine Klasse zur Bereitstellung von Konstanten in Form eines Singletons.

4.7.1 C++ Klassendiagramm

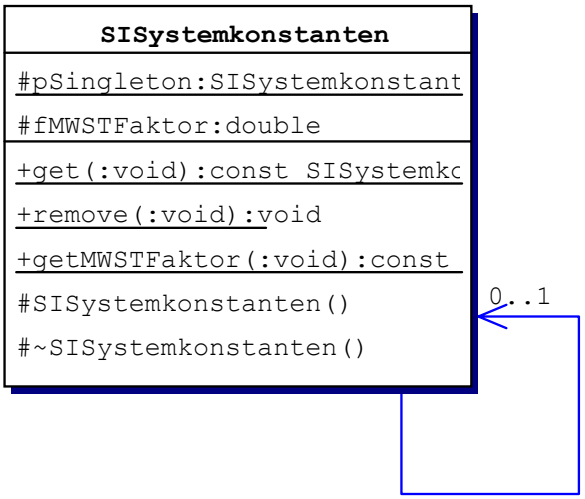

Abbildung 4-1 C++ Klassendiagramm Singleton

4.7.2 C++ Beispielprogramm

```
//============================================================================
// VISUAL STUDIO C++ V7.0
// Singleton.cpp: Definiert den Einstiegspunkt für Konsolenanwendung
//============================================================================

#include "stdafx.h"
#include "SIHauptprogramm.h"

int _tmain(int argc, _TCHAR* argv[])
{
      SingletonTest();
      return 0;
}
```

```
//============================================================================
// Borland C++ Builder V4.0
// Singleton.cpp: Definiert den Einstiegspunkt für Konsolenanwendung
//============================================================================

#pragma hdrstop
#include <condefs.h>

// ------------- Borland C++ Builder ------------------------------------
USEUNIT("SISystemkonstanten.cpp");
USEUNIT("SIHauptprogramm.cpp");
// ----------------------------------------------------------------------

#include "SIHauptprogramm.h"

#pragma argsused
int main(int argc, char* argv[])
```

```
{
      SingletonTest();
      return 0;
}
```

```
//============================================================================
// C++ Template Method Designpattern - SIHauptprogramm.h
//============================================================================
#ifndef _SIHAUPTPROGRAMM_H_
#define _SIHAUPTPROGRAMM_H_

    void SingletonTest (void);

#endif
```

```
//============================================================================
// C++ Singleton Designpattern - SIHauptprogramm.cpp
//============================================================================
#include "stdafx.h"        // Entfernen für Borland
#include <string>
#include <iostream>
#include "SISystemkonstanten.h"

using namespace std;

void SingletonTest (void)
{
   double fEingabe = 0.0;
   double fAusgabe = 0.0;
   string sEingabe;
   char x;

   cout << "Bitte Betrag eingeben: ";
   cin >> sEingabe;
   fEingabe = atof (sEingabe.c_str());

   fAusgabe = fEingabe * SISystemkonstanten::get()->getMWSTFaktor();

   cout << "Die Mehrwertsteuer für " << fEingabe
        << " beträgt "              << fAusgabe;

   cin >> x;
}
```

```
//============================================================================
// C++ Singleton Designpattern - SISystemkonstanten.h
//============================================================================
#ifndef _SISYSTEMKONSTANTEN_H_
#define _SISYSTEMKONSTANTEN_H_

class SISystemkonstanten
{
   //-------------------------------------------------------------------------
   // Methoden
   //-------------------------------------------------------------------------
   public:
      static const SISystemkonstanten * get           (void);
      static       void                 remove        (void);
      static const double               getMWSTFaktor (void);

   protected:
```

```
      SISystemkonstanten  ();
      ~SISystemkonstanten ();

   //------------------------------------------------------------------------
   // Attribute
   //------------------------------------------------------------------------
   protected:
      static SISystemkonstanten * pSingleton;
             double               fMWSTFaktor;
};

#endif
```

```
//============================================================================
// C++ Singleton Designpattern - SISystemkonstanten.cpp
//============================================================================
#include "stdafx.h"        // Entfernen für Borland
#include "SISystemkonstanten.h"

SISystemkonstanten * SISystemkonstanten::pSingleton = NULL;

//----------------------------------------------------------------------------
// Constructor
//----------------------------------------------------------------------------
SISystemkonstanten::SISystemkonstanten ()
{
   // Auslesen der Konstanten aus einer Datenbank oder Konfigurationsdatei
   fMWSTFaktor = 0.17;
}

//----------------------------------------------------------------------------
// Destructor
//----------------------------------------------------------------------------
SISystemkonstanten::~SISystemkonstanten ()
{
}

//----------------------------------------------------------------------------
// Singleton ansprechen (ggf. indirekter Constructoraufruf)
//----------------------------------------------------------------------------
const SISystemkonstanten * SISystemkonstanten::get (void)
{
   if (!pSingleton)
   {
      pSingleton = new SISystemkonstanten();
   }
   return pSingleton;
}

//----------------------------------------------------------------------------
// indirekter Destructoraufruf
//----------------------------------------------------------------------------
void SISystemkonstanten::remove (void)
{
   if (pSingleton)
   {
      delete pSingleton;
      pSingleton = NULL;
   }
}

//----------------------------------------------------------------------------
```

```
// Beispielkonstante abfragen
//---------------------------------------------------------------------------
const double SISystemkonstanten::getMWSTFaktor (void)
{
   return get()->fMWSTFaktor;
}
```

4.7.3 Anmerkungen zum C++ Beispiel

Durch die dynamische Anlage der Singleton-Instanz über eine statische Funktion ist die Einführung einer statischen ***remove()*** Methode notwendig, da sonst der über ***new*** belegte Speicherplatz nicht wieder freigegeben wird.

4.8 Java Beispiel

Das folgende Beispiel zeigt eine Klasse zur Bereitstellung von Konstanten in Form eines Singletons.

4.8.1 Java Klassendiagramm

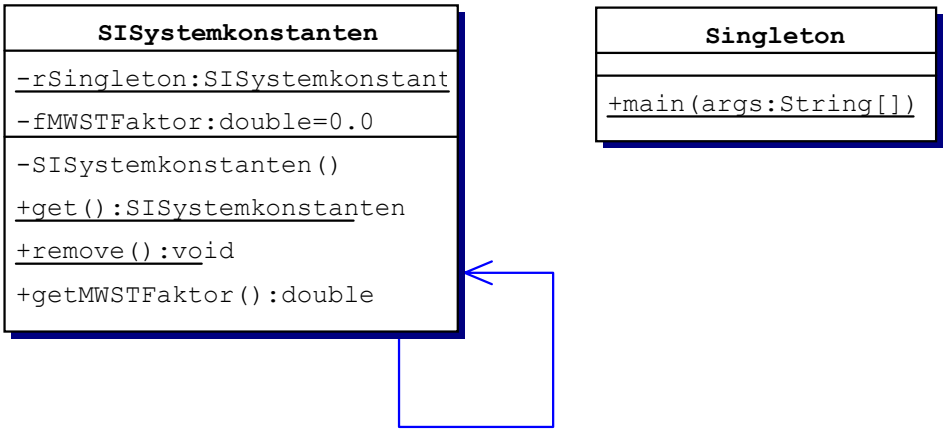

Abbildung 4-2 Java Klassendiagramm Singleton

4.8.2 Java Beispielprogramm

```
//======================================================================
/** Java Singleton Design Pattern - Testprogramm
 */
//======================================================================
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class Singleton
{
   //--------------------------------------------------------------------
   // Hauptprogramm
   //--------------------------------------------------------------------
   public static void main(String[] args)
   {
      InputStreamReader rStream = new InputStreamReader(System.in);
      BufferedReader    rInput  = new BufferedReader(rStream);
      double            fValue  = 0.0;
      double            fCalc   = 0.0;
      try
      {
         System.out.println("Eingabe: ");
         fValue = Double.parseDouble(rInput.readLine());
         fCalc  = fValue * SISystemkonstanten.get().getMWSTFaktor();
         System.out.println("Mehrwertsteuer für " + fValue + " beträgt "
                            + fCalc);
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
   }
}
```

```
//======================================================================
```

```
/** Java Singleton Design Pattern - Singleton Systemkonstanten
 */
//=============================================================================
public final class SISystemkonstanten
{
   private static SISystemkonstanten rSingleton  = null;
   private        double             fMWSTFaktor = 0.0;

   //-----------------------------------------------------------------------
   // Constructor
   //-----------------------------------------------------------------------
   private SISystemkonstanten ()
   {
      // Auslesen der Konstanten aus einer Datenbank oder
      // Konfigurationsdatei
      fMWSTFaktor = 0.17;
   }

   //-----------------------------------------------------------------------
   // Singleton holen
   //-----------------------------------------------------------------------
   static public SISystemkonstanten get ()
   {
      if (rSingleton == null)
      {
         rSingleton = new SISystemkonstanten();
      }
      return rSingleton;
   }

   //-----------------------------------------------------------------------
   // Singleton löschen (bleibt sonst erhalten bis Virtuelle Maschine
   // beendet wird
   //-----------------------------------------------------------------------
   static public void remove ()
   {
      if (rSingleton != null)
      {
         rSingleton = null;
      }
   }

   //-----------------------------------------------------------------------
   // Zugriff auf Wert im Singleton
   //-----------------------------------------------------------------------
   public double getMWSTFaktor ()
   {
      return rSingleton.fMWSTFaktor;
   }
}
```

4.8.3 Anmerkungen zum Java Beispiel

In Java ist eine ***remove()*** Methode entbehrlich, da spätestens mit Beendigung der virtuellen Maschine alle noch belegten Variablen freigegeben werden. wird die virtuelle Maschine des Java Laufzeitsystems jedoch mit Beendigung des Programms heruntergefahren, so empfiehlt sich auch hier die Verwendung einer ***remove()*** Methode, um dem Garbage-Collector die Freigabe des Speichers zu ermöglichen.

5. Adapter

Alternativnamen: Wrapper, Umwickler
Mustergruppe: Strukturmuster

5.1 Anmerkungen

Wie der Singleton gehört auch der Adapter zu den einfacheren und bekannteren Designpatterns. Anders als dieser ist der Adapter aber nicht die Grundlage anderer Patterns, sondern ist vielmehr eine für den Problembereich hinreichend universelle Lösung, so dass bisher keine alternativen Konzepte angeleitet wurden.

5.2 Verwendungszweck

Das Designpattern Adapter dient zur Abbildung inkompatibler Schnittstellen aufeinander, wie der Name es bereits andeutet.
Dieses Muster ist besonders nützlich um:

- Elemente mit unterschiedlichen Methodeninterfaces (Schnittstellen) so zu benutzen (polymorphe Verwendung) als wären Sie von einer gemeinsamen Basisklasse abgeleitet.

5.3 Problemstellungen

Die Schnittstellen der zu verwendenden Klassen sind unterschiedlich, sollten zur besseren Verwendung aber gleich sein. Wie kann ich das erreichen, ohne alle Klassen neu- oder umschreiben zu müssen?

5.4 Lösung

Die Lösung dieses Problems ist relativ trivial. Man kapselt die abweichenden Instanzen in eine Instanz einer Adapterklasse und verwendet diese anstelle des Objektes mit der abweichenden Schnittstelle. Die Auflösung der Abweichung erfolgt (im einfachsten Fall) durch Abbildung der Methoden aufeinander.
Adapter lassen sich in vielfältiger Form implementieren, sei es um die Grenzen (und Unterschiede) von Programmiersprachen zu überbrücken, dynamische Bibliotheken austauschbar zu halten oder externe Module einzubinden.

5.5 Vorteile

Klassen oder Module unterschiedlicher Herkunft lassen sich in bestehende Softwareumgebungen einpassen, ganz als ob sie für diese entwickelt worden wären.
Klassen unterschiedlicher Vererbungsstränge lassen sich über Adapterobjekte polymorph verwalten.

5.6 Nachteile

Die Kapselung der anzupassenden Instanz in eine umgebende Objektinstanz ist grundsätzlich mit (geringen) Laufzeitverlusten behaftet.
Sehr komplexe Kapseln benötigen einen relativ hohen Testaufwand.

5.7 C++ Beispiel

Das folgende Beispiel zeigt den Einsatz einer Adapterklasse anhand einer Klassenfamilie zur Umrechnung von Einheiten.

5.7.1 C++ Klassendiagramm

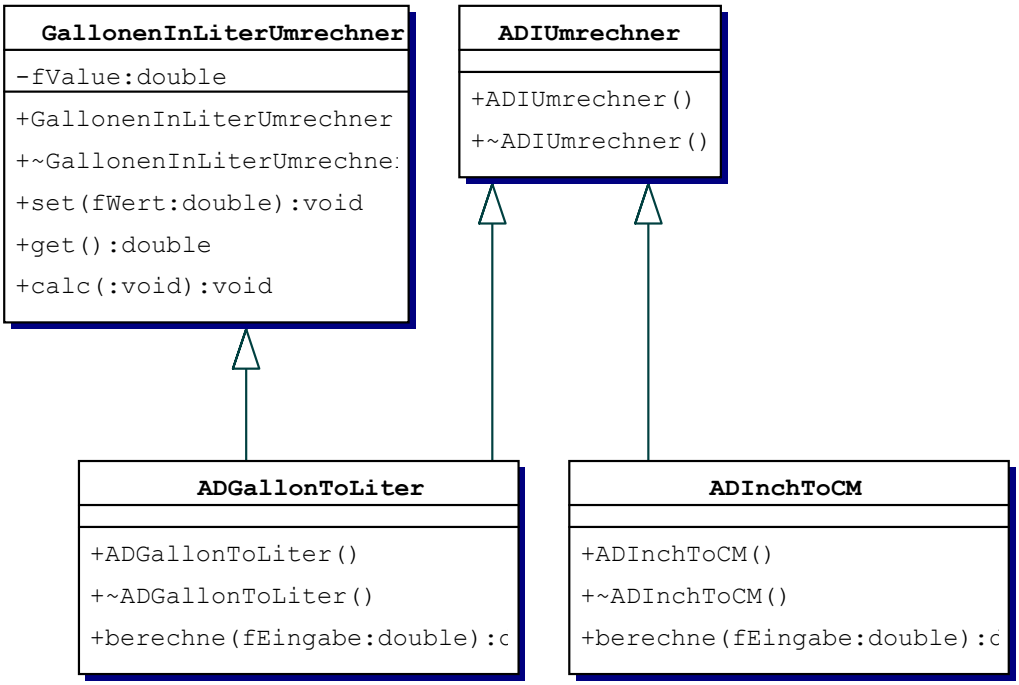

Abbildung 5-1 C++ Klassendiagramm Adapter

5.7.2 C++ Beispielprogramm

```
//=====================================================================
// VISUAL STUDIO C++ V7.0
// Adapter.cpp: Definiert den Einstiegspunkt für Konsolenanwendung
//=====================================================================

#include "stdafx.h"
#include "ADHauptprogramm.h"

int _tmain(int argc, _TCHAR* argv[])
{
      AdapterTest();
      return 0;
}
```

```
//=====================================================================
// Borland C++ Builder V4.0
// Adapter.cpp: Definiert den Einstiegspunkt für Konsolenanwendung
//=====================================================================

#pragma hdrstop
#include <condefs.h>

// ------------- Borland C++ Builder ------------------------------------
USEUNIT("ADIUmrechner.cpp");
USEUNIT("ADHauptprogramm.cpp");
USEUNIT("ADGallonToLiter.cpp");
USEUNIT("ADInchToCM.cpp");
USEUNIT("GallonenInLiterUmrechner.cpp");
// ----------------------------------------------------------------------

#include "ADHauptprogramm.h"
```

```
#pragma argsused
int main(int argc, char* argv[])
{
      AdapterTest ();
      return 0;
}
```

```
//===========================================================================
// C++ Adapter Designpattern - ADHauptprogramm.h
//===========================================================================
#ifndef _ADHAUPTPROGRAMM_H_
#define _ADHAUPTPROGRAMM_H_

    void AdapterTest (void);

#endif
```

```
//===========================================================================
// C++ Adapter Designpattern - ADHauptprogramm.cpp
//===========================================================================
#include "stdafx.h"        // Entfernen für Borland
#include <string>
#include <iostream>
#include "ADIUmrechner.h"
#include "ADInchToCM.h"
#include "ADGallonToLiter.h"

using namespace std;

void AdapterTest (void)
{
   double         fEingabe   = 0.0;
   double         fAusgabe   = 0.0;
   int            nAuswahl   = 0;
   ADIUmrechner * pUmrechner = NULL;
   char           cEnde;

   string sEingabe;
   string sAuswahl;

   cout << "Bitte Umrechnungsart eingeben" << endl
        << "1 = Inch     in cm"            << endl
        << "2 = Gallonen in Liter"         << endl;

   cin >> sAuswahl;
   nAuswahl = atoi (sAuswahl.c_str());

   switch (nAuswahl)
   {
      default: exit(10);
      case 1 : pUmrechner = new ADInchToCM();      break;
      case 2 : pUmrechner = new ADGallonToLiter(); break;
   }

   cout << "Bitte umzurechnenden Wert eingeben" << endl;
   cin >> sEingabe;
   fEingabe = atof (sEingabe.c_str());

   fAusgabe = pUmrechner->berechne (fEingabe);

   cout << "Der umgerechnete Wert beträgt " << fAusgabe;
```

```
   delete pUmrechner;
   cin >> cEnde;
}
```

```
//============================================================================
// C++ Adapter Designpattern - ADIUmrechner.h
// abstrakte (pure virtual) Interfaceklasse
//============================================================================
#ifndef _ADIUMRECHNER_H_
#define _ADIUMRECHNER_H_

class ADIUmrechner
{
   //-------------------------------------------------------------------------
   // Methoden
   //-------------------------------------------------------------------------
   public:
                     ADIUmrechner ();
      virtual       ~ADIUmrechner ();
      virtual double  berechne     (double fEingabe) = NULL;
};

#endif
```

```
//============================================================================
// C++ Adapter Designpattern - ADIUmrechner.cpp
//============================================================================
#include "stdafx.h"        // Entfernen für Borland
#include "ADIUmrechner.h"

ADIUmrechner::ADIUmrechner(void)
{
}

ADIUmrechner::~ADIUmrechner(void)
{
}
```

```
//============================================================================
// C++ Adapter Designpattern - ADInchToCM.h
//============================================================================
#ifndef _ADINCHTOCM_H_
#define _ADINCHTOCM_H_

#include "ADIUmrechner.h"

class ADInchToCM : public ADIUmrechner
{
   //-------------------------------------------------------------------------
   // Methoden
   //-------------------------------------------------------------------------
   public:
             ADInchToCM ();
            ~ADInchToCM ();
      double berechne (double fEingabe);
};

#endif
```

```
//============================================================================
```

```
// C++ Adapter Designpattern - ADInchToCM.cpp
//============================================================================
#include "stdafx.h"        // Entfernen für Borland
#include "ADInchToCM.h"

//----------------------------------------------------------------------------
// Constructor
//----------------------------------------------------------------------------
ADInchToCM::ADInchToCM ()
{
}

//----------------------------------------------------------------------------
// Destructor
//----------------------------------------------------------------------------
ADInchToCM::~ADInchToCM ()
{
}

//----------------------------------------------------------------------------
// Umrechnungsmethode Inch in cm
//----------------------------------------------------------------------------
double ADInchToCM::berechne (double fEingabe)
{
   return fEingabe * 2.54;
}
```

```
//============================================================================
// C++ Adapter Designpattern - GallonenInLiterUmrechner.h
//============================================================================
#ifndef _GALLONENINLITERUMRECHNER_H_
#define _GALLONENINLITERUMRECHNER_H_

class GallonenInLiterUmrechner
{
   //----------------------------------------------------------------------------
   // Methoden
   //----------------------------------------------------------------------------
   public:
           GallonenInLiterUmrechner ();
          ~GallonenInLiterUmrechner ();
      void   set (double fWert);
      double get ();
      void   calc (void);

   //----------------------------------------------------------------------------
   // Attribute
   //----------------------------------------------------------------------------
   private:
      double fValue;
};

#endif
```

```
//============================================================================
// C++ Adapter Designpattern - GallonenInLiterUmrechner.cpp
//============================================================================
#include "stdafx.h"        // Entfernen für Borland
#include "GallonenInLiterUmrechner.h"

//----------------------------------------------------------------------------
```

```
// Constructor
//---------------------------------------------------------------------------
GallonenInLiterUmrechner::GallonenInLiterUmrechner ()
{
   fValue = 0.0;
}

//---------------------------------------------------------------------------
// Destructor
//---------------------------------------------------------------------------
GallonenInLiterUmrechner::~GallonenInLiterUmrechner ()
{
}

//---------------------------------------------------------------------------
// Setze Attribut
//---------------------------------------------------------------------------
void GallonenInLiterUmrechner::set (double fWert)
{
   fValue = fWert;
}

//---------------------------------------------------------------------------
// berechne
//---------------------------------------------------------------------------
void GallonenInLiterUmrechner::calc (void)
{
   fValue = fValue * 3.6;
}

//---------------------------------------------------------------------------
// Hole Attribut
//---------------------------------------------------------------------------
double GallonenInLiterUmrechner::get ()
{
   return fValue;
}
```

```
//===========================================================================
// C++ Adapter Designpattern - ADGallonToLiter.h
// 20031222 - von Vererbung auf eingebettetes Objekt geändert
//===========================================================================
#ifndef _ADGALLONTOLITER_H_
#define _ADGALLONTOLITER_H_

#include "ADIUmrechner.h"
#include "GallonenInLiterUmrechner.h"

class ADGallonToLiter : public ADIUmrechner
{
   //---------------------------------------------------------------------
   // Methoden
   //---------------------------------------------------------------------
   public:
             ADGallonToLiter ();
            ~ADGallonToLiter ();
      double berechne        (double fEingabe);

   //---------------------------------------------------------------------
   // Membervariablen
   //---------------------------------------------------------------------
   private:
```

```
      GallonenInLiterUmrechner rUmrechner;
};

#endif
```

```
//============================================================================
// C++ Adapter Designpattern - ADGallonToLiter.cpp
//============================================================================
#include "stdafx.h"        // Entfernen für Borland
#include "ADGallonToLiter.h"
#include "GallonenInLiterUmrechner.h"

//----------------------------------------------------------------------------
// Constructor
//----------------------------------------------------------------------------
ADGallonToLiter::ADGallonToLiter ()
{
}

//----------------------------------------------------------------------------
// Destructor
//----------------------------------------------------------------------------
ADGallonToLiter::~ADGallonToLiter ()
{
}

//----------------------------------------------------------------------------
// Umrechnungsmethode Gallonen in Liter
//----------------------------------------------------------------------------
double ADGallonToLiter::berechne (double fEingabe)
{
   rUmrechner.set(fEingabe);
   rUmrechner.calc();
   return rUmrechner.get();
}
```

5.7.3 Anmerkungen zum C++ Beispiel

Die Klasse ***IUmrechner*** ist eine Interfaceklasse, die selbst nicht instanziiert werden kann. Sie legt in diesem Beispiel nur fest, dass das alle Umrechnungen mittels des Methodenaufrufes ***berechne(Wert)*** erfolgen.
Exemplarisch sind hier lediglich die Umrechnungsklassen ***InchToCM*** und ***GallonenInLiterUmrechner*** sowie die Adapterklasse ***GallonToLiter*** implementiert. ***InchToCM*** entspricht dem gewünschten Interface, da die Klasse ***public*** von ***IUmrechner*** abgeleitet wird. ***GallonenInLiterUmrechner*** hingegen hat ein gänzlich anderes Interface, welches hier eine Aufrufsequenz ***set(Wert)***, Aufruf der Berechnung mittels ***calc()*** und abschließender Aufruf von ***get()*** voraussetzt. diese etwas umständliche Anwendung wird mittels der Adapterklasse ***GallonToLiter*** auf das Interface ***IUmrechner*** angepasst.
Ein Pointer innerhalb von ***GallonToLiter***, der auf ein dynamisch erzeugtes Element vom Typ ***GallonenInLiterUmrechner*** zeigt, wäre ebenfalls ein denkbarer Lösungsansatz gewesen. Alternativ könnte solch ein Pointer auch durch eine eingebettete ***GallonenInLiterUmrechner*** Instanz ersetzt werden.

5.8 Java Beispiel

Das folgende Beispiel zeigt den Einsatz einer Adapterklasse anhand einer Klassenfamilie zur Umrechnung von Einheiten.

5.8.1 Java Klassendiagramm

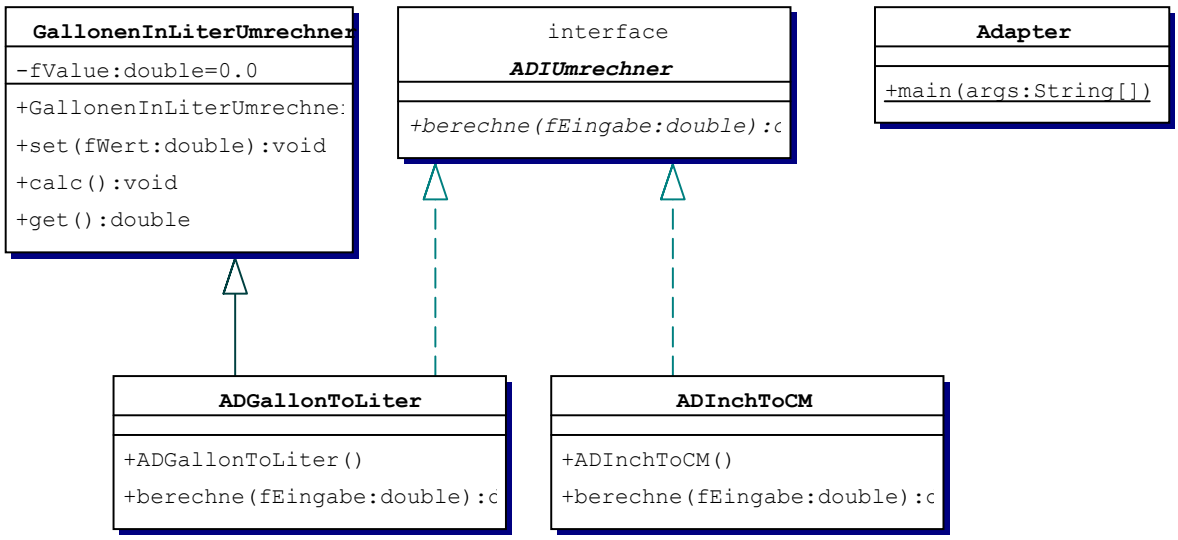

Abbildung 5-2 Java Klassendiagramm Adapter

5.8.2 Java Beispielprogramm

```
//============================================================================
// Java Adapter Design Pattern - Testprogramm
//============================================================================
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class Adapter
{
   //------------------------------------------------------------------------
   // Hauptprogramm
   //------------------------------------------------------------------------
   public static void main(String[] args)
   {
      double            fEingabe   = 0.0;
      double            fAusgabe   = 0.0;
      int               nAuswahl   = 0;
      InputStreamReader rStream    = new InputStreamReader(System.in);
      BufferedReader    rInput     = new BufferedReader(rStream);
      ADIUmrechner      rUmrechner = null;
      char              cEnde;

      try
      {
         System.out.println("Bitte Umrechnungsart eingeben: ");
         System.out.println("1 = Inch     in cm");
         System.out.println("2 = Gallonen in Liter");

         nAuswahl = Integer.parseInt(rInput.readLine());
         switch (nAuswahl)
         {
            case 1 : rUmrechner = new ADInchToCM();      break;
```

```
            case 2 : rUmrechner = new ADGallonToLiter(); break;
         }

         if (rUmrechner != null)
         {
            System.out.println("Bitte umzurechnenden Wert eingeben");
            fEingabe = Double.parseDouble(rInput.readLine());
            fAusgabe = rUmrechner.berechne (fEingabe);
            System.out.println("Der umgerechnete Wert beträgt "
                               + fAusgabe);
         }
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
   }
}
```

```
//============================================================================
// Java Adapter Design Pattern - Umrechner Interface
//============================================================================
public interface ADIUmrechner
{
   double  berechne (double fEingabe);
}
```

```
//============================================================================
// Java Adapter Design Pattern - Umrechner Inch (Zoll) in Zentimeter
//============================================================================
public class ADInchToCM implements ADIUmrechner
{
   //------------------------------------------------------------------
   // Constructor
   //------------------------------------------------------------------
   public ADInchToCM ()
   {
   }

   //------------------------------------------------------------------
   // Berechnungsmethode
   //------------------------------------------------------------------
   public double berechne (double fEingabe)
   {
      return fEingabe * 2.54;
   }
}
```

```
//============================================================================
// Java Adapter Design Pattern - Umrechnungsprogramm Gallonen in Liter
//============================================================================
public class GallonenInLiterUmrechner
{
   private double fValue = 0.0;

   //------------------------------------------------------------------
   // Constructor
   //------------------------------------------------------------------
   public GallonenInLiterUmrechner ()
   {
      super();
```

```
   }

   //-------------------------------------------------------------------------
   // Setze Attribut
   //-------------------------------------------------------------------------
   public void set (double fWert)
   {
      fValue = fWert;
   }

   //-------------------------------------------------------------------------
   // berechne
   //-------------------------------------------------------------------------
   public void calc ()
   {
      fValue = fValue * 3.6;
   }

   //-------------------------------------------------------------------------
   // Hole Attribut
   //-------------------------------------------------------------------------
   public double get ()
   {
      return fValue;
   }
}
```

```
//=============================================================================
// Java Adapter Design Pattern - Umrechner Interface
// 20031222 - von Vererbung auf eingebettetes Objekt geändert
//=============================================================================
public class ADGallonToLiter implements ADIUmrechner
{
   private GallonenInLiterUmrechner rUmrechner =
           new GallonenInLiterUmrechner();

   //-------------------------------------------------------------------------
   // Constructor
   //-------------------------------------------------------------------------
   public ADGallonToLiter ()
   {
   }

   //-------------------------------------------------------------------------
   // Berechnungsmethode
   //-------------------------------------------------------------------------
   public double berechne (double fEingabe)
   {
      rUmrechner.set(fEingabe);
      rUmrechner.calc();
      return rUmrechner.get();
  }
}
```

5.8.3 Anmerkungen zum Java Beispiel

In Java muss ***IUmrechner*** zwangsläufig als Interface deklariert werden, da multiple Vererbung in Java nicht möglich ist.

6. Decorator

Alternativnamen: Dekorierer, Umwickler (gebunden), Wrapper (bounded)
Mustergruppe: Strukturmuster

6.1 Anmerkungen

Wie sich schon bei den Alternativnamen andeutet, ist das Decorator Designpattern stark verwandt mit dem Adapter Pattern (⇨ Adapter). Anders als dieses wird die eingekapselte Objektinstanz aber nicht mit einer neuen, sondern mit einer erweiterten Schnittstelle versehen.

6.2 Verwendungszweck

Das Decorator Pattern dient dazu ein Objekt um zusätzliche Funktionalität zu erweitern ohne dabei die ursprüngliche Schnittstelle zu verändern.
Wenn z.B. mehrere Bitmaps (Pixelgraphiken) geladen werden, so ist es unsinnig alle gleichzeitig in „teure“ Klassen einzubetten, die eine Bearbeitung der Graphiken ermöglichen. In einer Anwendung wird wahrscheinlich immer nur eine Graphik zurzeit bearbeitet werden. Es genügt daher zunächst alle Bitmaps zu laden und nur eine ausgewählte Graphik wird mit der vollen Funktionalität zur Bearbeitung versehen. Ggf. wird sogar das Virtual Proxy Pattern (⇨ Proxy) verwendet und noch nicht einmal die Bitmaps werden vollständig geladen.
Eine andere Verwendungsmöglichkeit ist z.B. die Einbettung gleichartiger Objekte in unterschiedliche Plausibilitäts- oder Berechnungs- Decorator Objekte. So können beispielsweise gleiche Datenobjekte länderspezifisch unterschiedliche Plausibilitäten für gültige Postleitzahlen besitzen oder Preise länderspezifisch umgerechnet werden ohne das Ursprungsobjekt ändern zu müssen.

6.3 Problemstellungen

Wie erweitert man eine Instanz bei Bedarf um zusätzliche Funktionalität ohne eine weitere Spezialisierung (Unterklasse) zu bilden?

6.4 Lösung

Wie in einem Adapter wird die ursprüngliche Instanz in einer neuen Instanz eingekapselt. In C++ erben dabei beide Instanzen von der gleichen Basisklasse, welche die gemeinsame Schnittstelle festlegt. In Java implementieren beide Objekte das gleiche ***interface***. Dazu wird der Zeiger (C++) oder die Referenz der ursprünglichen Instanz an die Instanz der Decorator Klasse übergeben und dort verwaltet. Es entsteht somit keine Kopie des ursprünglichen Objektes, sondern eine temporäre Erweiterung. Der Destructor der Decorator Klasse wird dementsprechend das eingekapselte Objekt auch nicht zerstören.
Decorator Klassen können natürlich wiederum eine Hierarchie von spezialisierten Klassen bilde, alle bezogen auf die gleiche einzukapselnde Klasse.

6.5 Vorteile

Mit dem Decorator Designpattern entstehen keine Kopien des Ursprungsobjektes, der Zugriff auf die eingebettete Instanz über bestehende Zeiger und Referenzen ist weiterhin möglich, so dass nicht versehentlich auf zwei verschiedenen Objekten gearbeitet wird. Ein sonst nötiger Aufwand zur „Synchronisierung“ der Objekte entfällt.

6.6 Nachteile

Aufgrund der Tatsache, dass das eingebettete Objekt nicht mit zerstört werden darf, wird die Speicherverwaltung unter C++ etwas komplizierter, wenn die Decorator Objekte dynamisch erzeugt werden. Es ist zu beachten, dass das zugehörige Decorator Objekt mit zerstört wird, wenn das eingebettete Objekt verworfen wird.

6.7 C++ Beispiel

Das folgende Beispiel zeigt einen Währungs-Decorator. Die Basisklassen enthalten die eigentlichen Waren-Funktionalitäten, da diese weltweit gleich ist.
Die lokalen Besonderheiten (hier die Steuerberechnung) werden durch Decorator Klassen hinzugefügt.

6.7.1 C++ Klassendiagramm

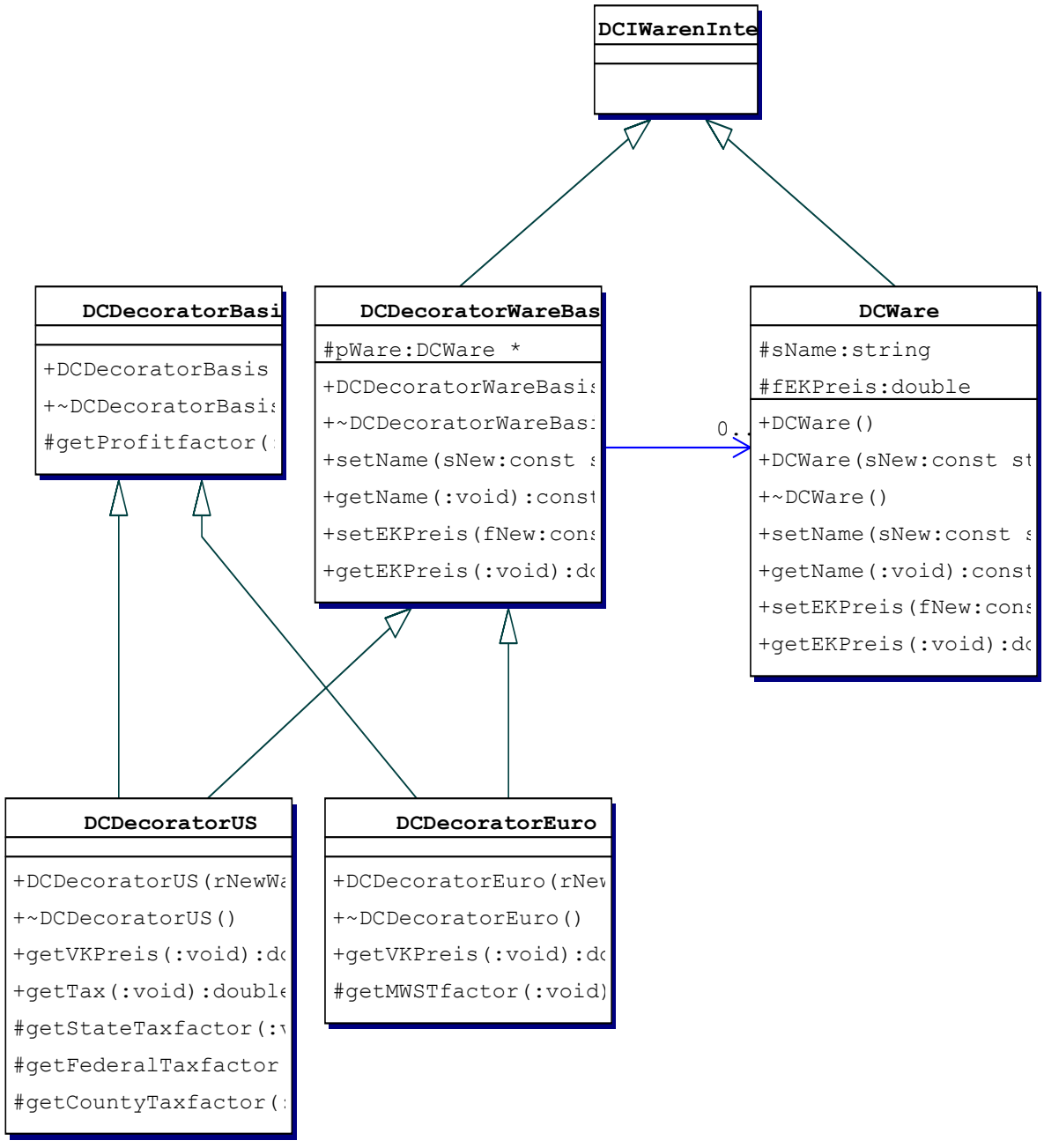

Abbildung 6-1 C++ Klassendiagramm Decorator

6.7.2 C++ Beispielprogramm

```
//===========================================================================
// VISUAL STUDIO C++ V7.0
// Decorator.cpp : Definiert den Einstiegspunkt für die Konsolenanwendung.
```

```
//===========================================================================

#include "stdafx.h"
#include "DCHauptprogramm.h"

int _tmain(int argc, _TCHAR* argv[])
{
      DecoratorTest ();
      return 0;
}
```

```
//===========================================================================
// Borland C++ Builder V4.0
// Decorator.cpp : Definiert den Einstiegspunkt für die Konsolenanwendung.
//===========================================================================

#pragma hdrstop
#include <condefs.h>

// ------------- Borland C++ Builder ------------------------------------
USEUNIT("DCWare.cpp");
USEUNIT("DCDecoratorBasis.cpp");
USEUNIT("DCDecoratorWareBasis.cpp");
USEUNIT("DCDecoratorEuro.cpp");
USEUNIT("DCDecoratorUS.cpp");
USEUNIT("DCHauptprogramm.cpp");
// ----------------------------------------------------------------------

#include "DCHauptprogramm.h"

#pragma argsused
int main(int argc, char* argv[])
{
      DecoratorTest();
      return 0;
}
```

```
//===========================================================================
// C++ Decorator Designpattern - DCHauptprogramm.h
//===========================================================================
#ifndef _DCHAUPTPROGRAMM_H_
#define _DCHAUPTPROGRAMM_H_

    void DecoratorTest (void);

#endif
```

```
//===========================================================================
// C++ Decorator Designpattern - DCHauptprogramm.cpp
//===========================================================================
#include "stdafx.h"        // Entfernen für Borland
#include <string>
#include <iostream>
#include "DCWare.h"
#include "DCDecoratorUS.h"
#include "DCDecoratorEuro.h"

using namespace std;

void DecoratorTest (void)
{
```

```
   char cEnde;
   DCWare aObject ("Vergaserinnenbeleuchtung", 123.45);

   DCDecoratorEuro aEuroZone (aObject);
   DCDecoratorUS   aUSZone   (aObject);

   cout << "E U R O Z O N E" << endl;
   cout << aEuroZone.getName() << ": " << aEuroZone.getVKPreis() << endl;


   cout << endl;
   cout << "U S Z O N E" << endl;
   cout << aUSZone.getName() << ": " << aUSZone.getVKPreis()
        << " (" << aUSZone.getTax() << " Tax)" << endl;
   cin >> cEnde;
}
```

```
//============================================================================
// C++ Decorator Designpattern - DCIWarenInterface.h
//============================================================================
#ifndef _DCIWARENINTERFACE_H_
#define _DCIWARENINTERFACE_H_

#include <string>

using namespace std;

class DCIWarenInterface
{
   //-------------------------------------------------------------------------
   // Methoden
   //-------------------------------------------------------------------------
   public:
      virtual void           setName    (const string & sNew) = NULL;
      virtual const string & getName    (void)                = NULL;
      virtual void           setEKPreis (const double fNew)   = NULL;
      virtual double         getEKPreis (void)                = NULL;
};

#endif
```

```
//============================================================================
// C++ Decorator Designpattern - DCWare.h
//============================================================================
#ifndef _DCWARE_H_
#define _DCWARE_H_

#include "DCIWarenInterface.h"

using namespace std;

class DCWare : public DCIWarenInterface
{
   //-------------------------------------------------------------------------
   // Methoden
   //-------------------------------------------------------------------------
   public:
       DCWare ();
       DCWare (const string & sNew, const double fNew);
      ~DCWare ();
```

```
      virtual void           setName    (const string & sNew);
      virtual const string & getName    (void);
      virtual void           setEKPreis (const double fNew);
      virtual double         getEKPreis (void);

   //------------------------------------------------------------------------
   // Attribute
   //------------------------------------------------------------------------
   protected:
      string sName;
      double fEKPreis;
};

#endif
```

```
//============================================================================
// C++ Decorator Designpattern - DCWare.cpp
//============================================================================
#include "stdafx.h"        // Entfernen für Borland
#include "DCWare.h"

//----------------------------------------------------------------------------
// Constructor
//----------------------------------------------------------------------------
DCWare::DCWare ()
{
}

//----------------------------------------------------------------------------
// Constructor
//----------------------------------------------------------------------------
DCWare::DCWare (const string & sNew, const double fNew)
{
   sName    = sNew;
   fEKPreis = fNew;
}

//----------------------------------------------------------------------------
// Destructor
//----------------------------------------------------------------------------
DCWare::~DCWare ()
{
}

//----------------------------------------------------------------------------
// setName (const string &)
//----------------------------------------------------------------------------
void DCWare::setName (const string & sNew)
{
   sName = sNew;
}

//----------------------------------------------------------------------------
// getName (void)
//----------------------------------------------------------------------------
const string & DCWare::getName (void)
{
   return sName;
}

//----------------------------------------------------------------------------
// setEKPreis (const double fNew)
```

```
//-----------------------------------------------------------------------
void  DCWare::setEKPreis (const double fNew)
{
   fEKPreis = fNew;
}

//-----------------------------------------------------------------------
// getEKPreis (void)
//-----------------------------------------------------------------------
double DCWare::getEKPreis (void)
{
   return fEKPreis;
}
```

```
//=======================================================================
// C++ Decorator Designpattern - DCDecoratorWareBasis.h
//=======================================================================
#ifndef _DCDECORATORWAREBASIS_H_
#define _DCDECORATORWAREBASIS_H_

#include "DCIWarenInterface.h"

using namespace std;

class DCWare;

class DCDecoratorWareBasis : public DCIWarenInterface
{
   //-----------------------------------------------------------------------
   // Methoden
   //-----------------------------------------------------------------------
   public:
                             DCDecoratorWareBasis (DCWare & rNewWare);
                            ~DCDecoratorWareBasis ();
      virtual void           setName    (const string & sNew);
      virtual const string & getName    (void);
      virtual void           setEKPreis (const double fNew);
      virtual double         getEKPreis (void);

   //-----------------------------------------------------------------------
   // Attribute
   //-----------------------------------------------------------------------
   protected:
      DCWare *pWare;
};

#endif
```

```
//=======================================================================
// C++ Decorator Designpattern - DCDecoratorWareBasis.cpp
//=======================================================================
#include "stdafx.h"        // Entfernen für Borland
#include "DCDecoratorWareBasis.h"
#include "DCWare.h"

//-----------------------------------------------------------------------
// Constructor
//-----------------------------------------------------------------------
DCDecoratorWareBasis::DCDecoratorWareBasis (DCWare & rNewWare)
{
   pWare = &rNewWare;
```

```
}

//---------------------------------------------------------------------------
// Destructor
//---------------------------------------------------------------------------
DCDecoratorWareBasis::~DCDecoratorWareBasis ()
{
   // pWare darf nicht mit freigegeben werden !
}

//---------------------------------------------------------------------------
// setName (const string &)
//---------------------------------------------------------------------------
void DCDecoratorWareBasis::setName (const string & sNew)
{
   pWare->setName(sNew);
}

//---------------------------------------------------------------------------
// getName (void)
//---------------------------------------------------------------------------
const string & DCDecoratorWareBasis::getName (void)
{
   return pWare->getName();
}

//---------------------------------------------------------------------------
// setEKPreis (const double fNew)
//---------------------------------------------------------------------------
void DCDecoratorWareBasis::setEKPreis (const double fNew)
{
   pWare->setEKPreis(fNew);
}

//---------------------------------------------------------------------------
// getEKPreis (void)
//---------------------------------------------------------------------------
double DCDecoratorWareBasis::getEKPreis (void)
{
   return pWare->getEKPreis();
}
```

```
//===========================================================================
// C++ Decorator Designpattern - DCDecoratorBasis.h
//===========================================================================
#ifndef _DCDECORATORBASIS_H_
#define _DCDECORATORBASIS_H_

class DCDecoratorBasis
{
   //------------------------------------------------------------------------
   // Methoden
   //------------------------------------------------------------------------
   public:
      DCDecoratorBasis ();
      ~DCDecoratorBasis ();

   protected:
      virtual double getProfitfactor (void);
};

#endif
```

```
//============================================================================
// C++ Decorator Designpattern - DCDecoratorBasis.cpp
//============================================================================
#include "stdafx.h"        // Entfernen für Borland
#include <assert.h>
#include "DCDecoratorBasis.h"

//----------------------------------------------------------------------------
// Constructor
//----------------------------------------------------------------------------
DCDecoratorBasis::DCDecoratorBasis ()
{
}

//----------------------------------------------------------------------------
// Destructor
//----------------------------------------------------------------------------
DCDecoratorBasis::~DCDecoratorBasis ()
{
}

//----------------------------------------------------------------------------
// getGewinnfaktor (void)
//----------------------------------------------------------------------------
double DCDecoratorBasis::getProfitfactor (void)
{
   return 2.0; // BESSER: aus externer Datenbank lesen
               // zB. gemäß Warengruppe
}
```

```
//============================================================================
// C++ Decorator Designpattern - DCDecoratorEuro.h
//============================================================================
#ifndef _DCDECORATOREURO_H_
#define _DCDECORATOREURO_H_

#include "DCDecoratorBasis.h"
#include "DCDecoratorWareBasis.h"

using namespace std;

class    DCDecoratorEuro    :    public    DCDecoratorWareBasis,    public
DCDecoratorBasis
{
   //-------------------------------------------------------------------------
   // Methoden
   //-------------------------------------------------------------------------
   public:
                             DCDecoratorEuro (DCWare & rNewWare);
                            ~DCDecoratorEuro ();
      virtual double         getVKPreis (void);
   protected:
      virtual double         getMWSTfactor (void);
};

#endif
```

```
//============================================================================
// C++ Decorator Designpattern - DCWareEuro.cpp
//============================================================================
```

```
#include "stdafx.h"        // Entfernen für Borland
#include <assert.h>
#include "DCDecoratorEuro.h"
#include "DCWare.h"

//----------------------------------------------------------------------------
// Constructor
//----------------------------------------------------------------------------
DCDecoratorEuro::DCDecoratorEuro (DCWare & rNewWare) : DCDecoratorWareBasis
(rNewWare)
{
}

//----------------------------------------------------------------------------
// Destructor
//----------------------------------------------------------------------------
DCDecoratorEuro::~DCDecoratorEuro ()
{
}

//----------------------------------------------------------------------------
// getVKPreis (void)
//----------------------------------------------------------------------------
double DCDecoratorEuro::getVKPreis (void)
{
   double fVKPreis = pWare->getEKPreis();
   fVKPreis = fVKPreis * getProfitfactor();
   fVKPreis = fVKPreis * getMWSTfactor();
   return fVKPreis;
}

//----------------------------------------------------------------------------
// getMWSTfactor (void)
//----------------------------------------------------------------------------
double DCDecoratorEuro::getMWSTfactor (void)
{
   return 1.17; // BESSER: aus externer Datenbank lesen
}
```

```
//============================================================================
// C++ Decorator Designpattern - DCDecoratorUS.h
//============================================================================
#ifndef _DCDECORATORUS_H_
#define _DCDECORATORUS_H_

#include "DCDecoratorBasis.h"
#include "DCDecoratorWareBasis.h"

using namespace std;

class DCDecoratorUS : public DCDecoratorWareBasis, public DCDecoratorBasis
{
   //-------------------------------------------------------------------------
   // Methoden
   //-------------------------------------------------------------------------
   public:
                            DCDecoratorUS (DCWare & rNewWare);
                           ~DCDecoratorUS ();
      virtual double        getVKPreis (void);
      virtual double        getTax     (void);

   protected:
```

```
      virtual double         getStateTaxfactor (void);
      virtual double         getFederalTaxfactor (void);
      virtual double         getCountyTaxfactor (void);
};

#endif
```

```
//============================================================================
// C++ Decorator Designpattern - DCWareUS.cpp
//============================================================================
#include "stdafx.h"        // Entfernen für Borland
#include <assert.h>
#include "DCDecoratorUS.h"
#include "DCWare.h"

//----------------------------------------------------------------------------
// Constructor
//----------------------------------------------------------------------------
DCDecoratorUS::DCDecoratorUS  (DCWare  &  rNewWare)  :  DCDecoratorWareBasis
(rNewWare)
{
}

//----------------------------------------------------------------------------
// Destructor
//----------------------------------------------------------------------------
DCDecoratorUS::~DCDecoratorUS ()
{
}

//----------------------------------------------------------------------------
// getVKPreis (void)
//----------------------------------------------------------------------------
double DCDecoratorUS::getVKPreis (void)
{
   return pWare->getEKPreis() * getProfitfactor() + getTax();
}

//----------------------------------------------------------------------------
// getTax (void)
//----------------------------------------------------------------------------
double DCDecoratorUS::getTax (void)
{
   return pWare->getEKPreis() * getCountyTaxfactor() +
          pWare->getEKPreis() * getStateTaxfactor()  +
          pWare->getEKPreis() * getFederalTaxfactor();
}

//----------------------------------------------------------------------------
// getStateTaxfactor(void)
//----------------------------------------------------------------------------
double DCDecoratorUS::getStateTaxfactor (void)
{
   return 0.07; // BESSER: aus externer Datenbank lesen
}

//----------------------------------------------------------------------------
// getFederalTaxfactor (void)
//----------------------------------------------------------------------------
double DCDecoratorUS::getFederalTaxfactor (void)
{
   return 0.04; // BESSER: aus externer Datenbank lesen
```

```
}

//---------------------------------------------------------------------------
// getCountyTaxfactor (void)
//---------------------------------------------------------------------------
double DCDecoratorUS::getCountyTaxfactor (void)
{
   return 0.05; // BESSER: aus externer Datenbank lesen
}
```

6.7.3 Anmerkungen zum C++ Beispiel

Die Erstellung der Decorator Klassen kann in diesem Beispiel vereinfacht werden, indem eine Zwischenschicht eingezogen wird, die genau das benötigte Interface implementiert. Von dieser Klasse erben dann sowohl ***DCWareEuro*** als auch ***DCWareUS***.
Dies hätte den Vorteil, dass durch die C++ Referenzen sichergestellt ist, dass das im Constructor übergebene Objekt auch wirklich existiert (in C++ sind ***null*** Referenzen wie in Java nicht erlaubt).

6.8 Java Beispiel

Das folgende Beispiel zeigt einen Währungs-Decorator. Die Basisklassen enthalten die eigentlichen Waren-Funktionalitäten, da diese weltweit gleich ist.
Die lokalen Besonderheiten (hier die Steuerberechnung) werden durch Decorator Klassen hinzugefügt.

6.8.1 Java Klassendiagramm

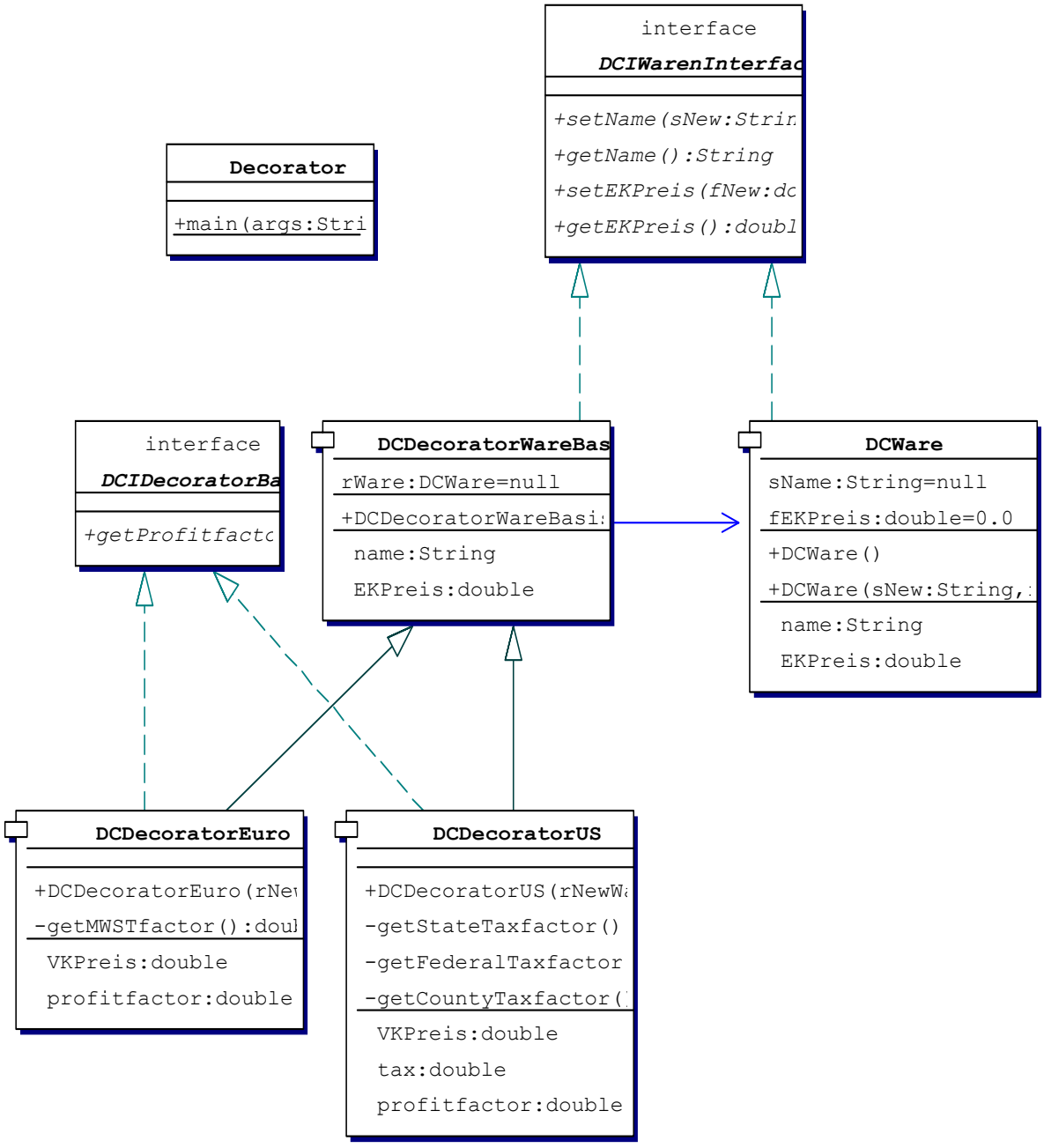

Abbildung 6-2 Java Klassendiagramm Decorator

6.8.2 Java Beispielprogramm

```
//==========================================================================
/** Java Decorator Design Pattern - Testprogramm
 */
```

```
//============================================================================
public class Decorator
{
   //----------------------------------------------------------------------
   // Hauptprogramm
   //----------------------------------------------------------------------
   public static void main(String[] args)
   {
      DCWare aObject = new DCWare ("Vergaserinnenbeleuchtung", 123.45);

      DCDecoratorEuro aEuroZone = new DCDecoratorEuro (aObject);
      DCDecoratorUS   aUSZone   = new DCDecoratorUS   (aObject);

      System.out.println("E U R O Z O N E");
      System.out.println(aEuroZone.getName() + ": " +
                         aEuroZone.getVKPreis());

      System.out.println();

      System.out.println("U S Z O N E");
      System.out.println(aUSZone.getName() + ": " + aUSZone.getVKPreis() +
                         " (" + aUSZone.getTax() + " Tax)");
   }
}
```

```
//============================================================================
/** Java Decorator Pattern - Interface für verwaltete Ware
 */
//============================================================================
public interface DCIWarenInterface
{
   abstract void   setName    (String sNew);
   abstract String getName    ();
   abstract void   setEKPreis (double fNew);
   abstract double getEKPreis ();
}
```

```
//============================================================================
/** Java Decorator Pattern - Ware
 */
//============================================================================
public class DCWare implements DCIWarenInterface
{
   String sName    = null;
   double fEKPreis = 0.0;

   //----------------------------------------------------------------------
   //  Constructor
   //----------------------------------------------------------------------
   public DCWare ()
   {
      sName = new String ();
   }

   //----------------------------------------------------------------------
   //  Constructor
   //----------------------------------------------------------------------
   public DCWare (String sNew, double fNew)
   {
      sName    = sNew;
      fEKPreis = fNew;
```

```
   }

   //------------------------------------------------------------------------
   // setName (String)
   //------------------------------------------------------------------------
   public void setName (String sNew)
   {
      sName = new String (sNew);
   }

   //------------------------------------------------------------------------
   // getName ()
   //------------------------------------------------------------------------
   public String getName ()
   {
      return sName;
   }

   //------------------------------------------------------------------------
   // setEKPreis (const double fNew)
   //------------------------------------------------------------------------
   public void setEKPreis (double fNew)
   {
      fEKPreis = fNew;
   }

   //------------------------------------------------------------------------
   // getEKPreis ()
   //------------------------------------------------------------------------
   public double getEKPreis ()
   {
      return fEKPreis;
   }
}
```

```
//============================================================================
/** Java Decorator Pattern - Decorator-BasisInterface
 */
//============================================================================
public interface DCIDecoratorBasis
{
   public double getProfitfactor ();
}
```

```
//============================================================================
/** Java Decorator Pattern - Decorator-Basisklasse für verwaltete Ware
 */
//============================================================================
public class DCDecoratorWareBasis implements DCIWarenInterface
{
   DCWare rWare = null;

   //------------------------------------------------------------------------
   // Constructor
   //------------------------------------------------------------------------
   public DCDecoratorWareBasis (DCWare rNewWare)
   {
      rWare = rNewWare;
   }

   //------------------------------------------------------------------------
```

```
   // setName (String)
   //-----------------------------------------------------------------------
   public void setName (String sNew)
   {
      rWare.setName(sNew);
   }

   //-----------------------------------------------------------------------
   // getName ()
   //-----------------------------------------------------------------------
   public String getName ()
   {
      return rWare.getName();
   }

   //-----------------------------------------------------------------------
   // setEKPreis (double fNew)
   //-----------------------------------------------------------------------
   public void setEKPreis (double fNew)
   {
      rWare.setEKPreis(fNew);
   }

   //-----------------------------------------------------------------------
   // getEKPreis ()
   //-----------------------------------------------------------------------
   public double getEKPreis ()
   {
      return rWare.getEKPreis();
   }
}
```

```
//=============================================================================
/** Java Decorator Pattern - Decorator für Euro-Zone
 */
//=============================================================================
public class DCDecoratorEuro extends DCDecoratorWareBasis
implements DCIDecoratorBasis
{
   //-----------------------------------------------------------------------
   // Constructor
   //-----------------------------------------------------------------------
   public DCDecoratorEuro (DCWare rNewWare)
   {
      super (rNewWare);
   }

   //-----------------------------------------------------------------------
   // getVKPreis ()
   //-----------------------------------------------------------------------
   public double getVKPreis ()
   {
      double fVKPreis = rWare.getEKPreis();
      fVKPreis = fVKPreis * getProfitfactor();
      fVKPreis = fVKPreis * getMWSTfactor();
      return fVKPreis;
   }

   //-----------------------------------------------------------------------
   // getMWSTfactor ()
   //-----------------------------------------------------------------------
   private double getMWSTfactor ()
```

```
   {
      return 1.17; // BESSER: aus externer Datenbank lesen
   }

   //-------------------------------------------------------------------------
   // getProfitfactor ()
   //-------------------------------------------------------------------------
   public double getProfitfactor ()
   {
      return 2.0; // BESSER: aus externer Datenbank lesen
                  // zB. gemäß Warengruppe
   }
}
```

```
//=============================================================================
/** Java Decorator Pattern - Decorator für Dollar-Zone
 */
//=============================================================================
public class DCDecoratorUS extends DCDecoratorWareBasis
implements DCIDecoratorBasis
{
   //-------------------------------------------------------------------------
   // Constructor
   //-------------------------------------------------------------------------
   public DCDecoratorUS (DCWare rNewWare)
   {
      super (rNewWare);
   }

   //-------------------------------------------------------------------------
   // getVKPreis ()
   //-------------------------------------------------------------------------
   public double getVKPreis ()
   {
      return rWare.getEKPreis() * getProfitfactor() + getTax();
   }

   //-------------------------------------------------------------------------
   // getTax ()
   //-------------------------------------------------------------------------
   public double getTax ()
   {
      return rWare.getEKPreis() * getCountyTaxfactor() +
             rWare.getEKPreis() * getStateTaxfactor()  +
             rWare.getEKPreis() * getFederalTaxfactor();
   }

   //-------------------------------------------------------------------------
   // getStateTaxfactor ()
   //-------------------------------------------------------------------------
   private double getStateTaxfactor ()
   {
      return 0.07; // BESSER: aus externer Datenbank lesen
   }

   //-------------------------------------------------------------------------
   // getFederalTaxfactor ()
   //-------------------------------------------------------------------------
   private double getFederalTaxfactor ()
   {
      return 0.04; // BESSER: aus externer Datenbank lesen
   }
```

```
   //-----------------------------------------------------------------------
   // getCountyTaxfactor ()
   //-----------------------------------------------------------------------
   private double getCountyTaxfactor ()
   {
      return 0.05; // BESSER: aus externer Datenbank lesen
   }

   //-----------------------------------------------------------------------
   // getProfitfactor ()
   //-----------------------------------------------------------------------
   public double getProfitfactor ()
   {
      return 2.0; // BESSER: aus externer Datenbank lesen
                  // zB. gemäß Warengruppe
   }
}
```

6.8.3 Anmerkungen zum Java Beispiel

Im Gegensatz zur C++ Version, die mit Mehrfach-Vererbung arbeitet, kann Java nur Interfaces implementieren, was einen etwas höheren Aufwand in der Implementation erfordert. Außerdem sollten (ergänzend zum Beispiel) mögliche ***null*** Referenzen abgefangen werden.

7. Façade

Alternativnamen: **Fassade**
Mustergruppe: **Strukturmuster**

7.1 Anmerkungen

Das Façade Pattern ist ein sehr einfaches Designpattern. Es ist in seiner Grundstruktur und Anwendung so offensichtlich, dass es in vielen Applikationsprogrammen eingesetzt wird (oftmals ohne als Façade Pattern erkannt bzw. ausgewiesen zu werden).

7.2 Verwendungszweck

Das Façade Designpattern dient dazu, der Applikation gegenüber die Komplexität eines Subsystems zu verstecken. Unter einem Subsystem ist hier eine Menge von Klassen zu verstehen, die sich – um ihre Aufgabe zu erfüllen – gegenseitig kennen bzw. per Referenz kennen. Eine Applikation, die ein solches Subsystem nutzen will muss zwangsläufig über detaillierte Kenntnisse der Beziehungen der Elemente des Subsystems untereinander verfügen.
Ein Kundendatensatz z.B. kann eine Klasse für die Personendaten enthalten, die wiederum auf einen Adressdatensatz verweist. Dieser Adressdatensatz wiederum verweist auf eine Reihe von Kommunikationsverbindungen. Diese Form von Subsystem spiegelt üblicherweise den Aufbau einer dahinter liegenden relationalen Datenbank wider.
Eine Applikation müsste nun, um alle Daten zu laden oder speichern, detaillierte Kenntnisse über die zugrunde liegenden Klassenbeziehungen zu haben.
Besser wäre es jedoch, diese Kenntnis nicht in die Applikation zu verlagern, da diese sonst bei jeder Strukturänderung des Subsystems ebenfalls betroffen wäre.
Das Façade Pattern wird häufig mit dem Singleton Pattern (⇨ Singleton) kombiniert, da bei Subsystemen, die keine Datenbeziehungen spiegeln sondern z.B. ein System zur Datenauswertung oftmals nur eine einzelne Instanz notwendig ist.

7.3 Problemstellungen

Wie können komplexe Subsysteme einer Applikation gegenüber vereinfacht und die Implementation des Subsystems verborgen werden, so dass bei Änderung der Struktur des Subsystems die Applikation nicht ebenfalls geändert werden muss?

7.4 Lösung

Das Façade Pattern verbirgt die eigentliche Struktur des Subsystems hinter einer Klasse, die alle Zugriffe entgegennimmt und dann an die entsprechenden Objekte des Subsystems weiterleitet.

7.5 Vorteile

Nur die Façade Klasse kennt die Struktur des Subsystems. Die Komplexität und Zerlegung des Subsystems wird somit in der Applikation nicht sichtbar und führt auch nicht zu Anpassungsaufwand, wenn die Struktur des Subsystems geändert wird.
Ein gegebenenfalls nötiger Anpassungsaufwand wird in einer Klasse gebündelt und nicht über die Applikation verstreut.

7.6 Nachteile

Außer dem zusätzlichen Aufwand für die Façade Klasse ist kein Nachteil zu verzeichnen. Die Errichtung der Façade verhindert direkte Durchgriffe auf das Subsystem oder Teile davon nicht, so dass es den Programmierern der Applikation obliegt diese Grenze einzuhalten.

7.7 C++ Beispiel

Das hier gezeigte Fassade-Pattern vereinfacht den Zugriff auf eine komplexe Kundenstruktur, indem es die wirkliche Hierarchie der Implementation verbirgt.

7.7.1 C++ Klassendiagramm

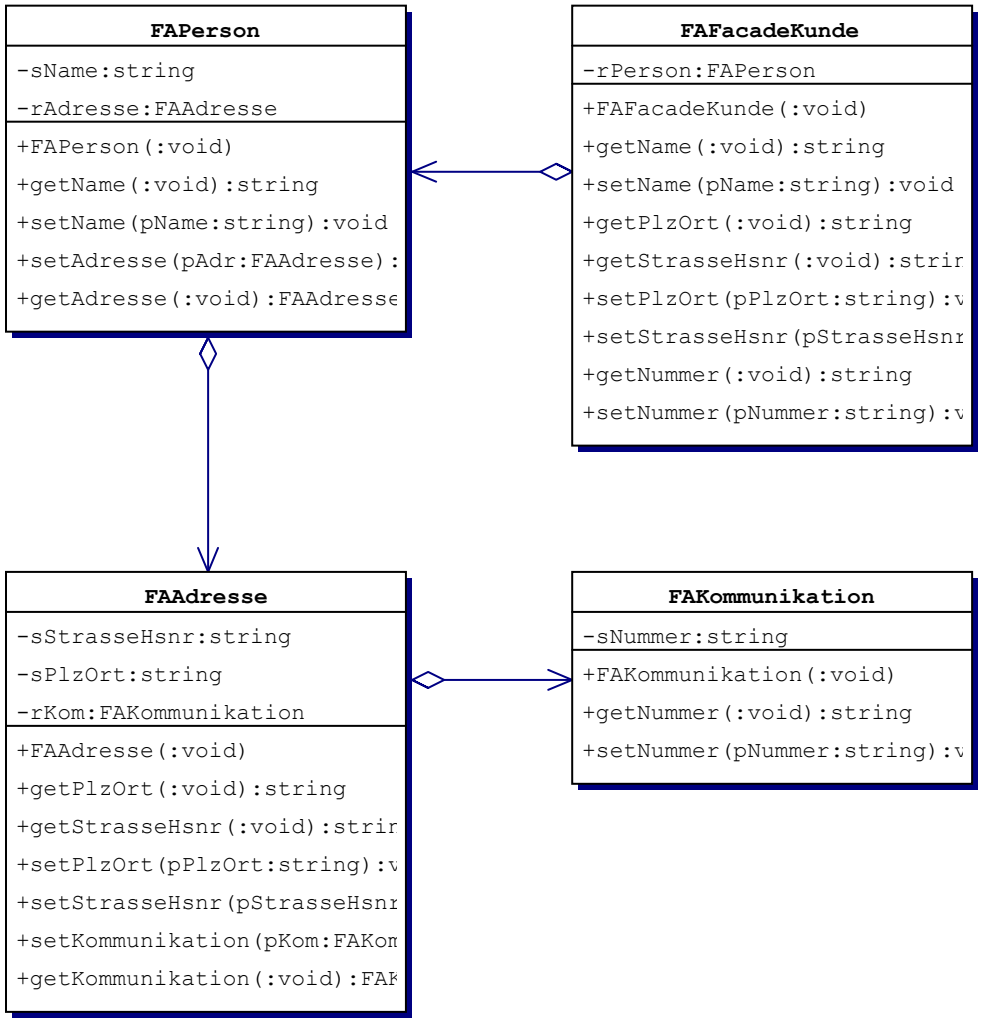

Abbildung 7-1 C++ Klassendiagramm Façade

7.7.2 C++ Beispielprogramm

```
//============================================================================
// C++ Facade Designpattern - Facade.cpp
//============================================================================
#include "stdafx.h"
#include "FAPerson.h"
#include "FAFacadeKunde.h"
#include <iostream>

int _tmain(int argc, _TCHAR* argv[])
{
   //---------------------------------------------------------------------
   // Ohne Facade-Pattern
   //---------------------------------------------------------------------
   FAPerson rP;
```

```
   rP.setName("Donald Duck");
   rP.getAdresse()->setStrasseHsnr("Entengasse 13");
   rP.getAdresse()->setPlzOrt("1313 Entenhausen");
   rP.getAdresse()->getKommunikation()->setNummer("013 - 13 13 13");

   cout << "Name....: " << rP.getName() << endl;
   cout << "Strasse.: " << rP.getAdresse()->getStrasseHsnr() << endl;
   cout << "Ort.....: " << rP.getAdresse()->getPlzOrt() << endl;
   cout << "Telephon: " << rP.getAdresse()->getKommunikation()->getNummer()
        << endl << endl;

   //---------------------------------------------------------------------
   // Mit Facade-Pattern
   //---------------------------------------------------------------------
   FAFacadeKunde rF;
   rF.setName("Dagobert Duck");
   rF.setStrasseHsnr("Geldspeicher 1");
   rF.setPlzOrt("3333 Entenhausen");
   rF.setNummer("012 - 34 56 78");

   cout << "Name....: " << rF.getName() << endl;
   cout << "Strasse.: " << rF.getStrasseHsnr() << endl;
   cout << "Ort.....: " << rF.getPlzOrt() << endl;
   cout << "Telephon: " << rF.getNummer() << endl;
   return 0;
}
```

```
//============================================================================
// C++ Facade Designpattern - FAPerson.h
//============================================================================
#ifndef _FAPERSON_H_
#define _FAPERSON_H_

#include <string>
#include "FAAdresse.h"

using namespace std;

class FAPerson
{
   public:
      FAPerson (void);

      string     getName    (void);
      void       setName    (string pName);
      void       setAdresse (FAAdresse pAdr);
      FAAdresse* getAdresse (void);

   private:
      string    sName;
      FAAdresse rAdresse;
};

#endif
```

```
//============================================================================
// C++ Facade Designpattern - FAPerson.cpp
//============================================================================
#include "stdafx.h"
#include "FAPerson.h"
```

```
//-----------------------------------------------------------------------
// Constructor
//-----------------------------------------------------------------------
FAPerson::FAPerson (void)
{
}

//-----------------------------------------------------------------------
string FAPerson::getName (void)
{
   return sName;
}

//-----------------------------------------------------------------------
void FAPerson::setName (string pName)
{
   sName = pName;
}

//-----------------------------------------------------------------------
void FAPerson::setAdresse (FAAdresse pAdr)
{
   rAdresse = pAdr;
}

//-----------------------------------------------------------------------
FAAdresse* FAPerson::getAdresse (void)
{
   return &rAdresse;
}
```

```
//=======================================================================
// C++ Facade Designpattern - FAAdresse.h
//=======================================================================
#ifndef _FAADRESSE_H_
#define _FAADRESSE_H_

#include <string>
#include "FAKommunikation.h"

using namespace std;

class FAAdresse
{
   public:
      FAAdresse (void);

      string           getPlzOrt        (void);
      string           getStrasseHsnr   (void);
      void             setPlzOrt        (string pPlzOrt);
      void             setStrasseHsnr   (string pStrasseHsnr);
      void             setKommunikation (FAKommunikation pKom);
      FAKommunikation* getKommunikation (void);

   private:
      string          sStrasseHsnr;
      string          sPlzOrt;
      FAKommunikation rKom;
};

#endif
```

```
//============================================================================
// C++ Facade Designpattern - FAAdresse.cpp
//============================================================================
#include "stdafx.h"
#include "FAAdresse.h"

//----------------------------------------------------------------------
// Constructor
//----------------------------------------------------------------------
FAAdresse::FAAdresse (void)
{
}

//----------------------------------------------------------------------
string FAAdresse::getPlzOrt (void)
{
   return sPlzOrt;
}

//----------------------------------------------------------------------
string FAAdresse::getStrasseHsnr (void)
{
   return sStrasseHsnr;
}

//----------------------------------------------------------------------
void FAAdresse::setPlzOrt (string pPlzOrt)
{
   sPlzOrt = pPlzOrt;
}

//----------------------------------------------------------------------
void FAAdresse::setStrasseHsnr (string pStrasseHsnr)
{
   sStrasseHsnr = pStrasseHsnr;
}

//----------------------------------------------------------------------
void FAAdresse::setKommunikation (FAKommunikation pKom)
{
   rKom = pKom;
}

//----------------------------------------------------------------------
FAKommunikation* FAAdresse::getKommunikation (void)
{
   return &rKom;
}
```

```
//============================================================================
// C++ Facade Designpattern - FAKommunikation.h
//============================================================================
#ifndef _FAKOMMUNIKATION_H_
#define _FAKOMMUNIKATION_H_

#include <string>

using namespace std;

class FAKommunikation
{
   public:
```

```
      FAKommunikation (void);

      string getNummer (void);
      void setNummer   (string pNummer);

   private:
       string sNummer;
};

#endif
```

```
//============================================================================
// C++ Facade Designpattern - FAKommunikation.cpp
//============================================================================
#include "stdafx.h"
#include "FAKommunikation.h"

//----------------------------------------------------------------------------
// Constructor
//----------------------------------------------------------------------------
FAKommunikation::FAKommunikation (void)
{
}

//----------------------------------------------------------------------------
string FAKommunikation::getNummer (void)
{
   return sNummer;
}

//----------------------------------------------------------------------------
void FAKommunikation::setNummer (string pNummer)
{
   sNummer = pNummer;
}
```

```
//============================================================================
// C++ Facade Designpattern - FAKunde.h
//============================================================================
#ifndef _FAFACADEKUNDE_H_
#define _FAFACADEKUNDE_H_

#include <string>
#include "FAPerson.h"

using namespace std;

class FAFacadeKunde
{
   public:
      FAFacadeKunde (void);

      string getName        (void);
      void   setName        (string pName);
      string getPlzOrt      (void);
      string getStrasseHsnr (void);
      void   setPlzOrt      (string pPlzOrt);
      void   setStrasseHsnr (string pStrasseHsnr);
      string getNummer      (void);
      void   setNummer      (string pNummer);
```

```
   private:
      FAPerson rPerson;
};

#endif
```

```
//============================================================================
// C++ Facade Designpattern - FAKunde.cpp
//============================================================================
#include "stdafx.h"
#include "FAFacadeKunde.h"

//----------------------------------------------------------------------------
// Constructor
//----------------------------------------------------------------------------
FAFacadeKunde::FAFacadeKunde (void)
{
}

//----------------------------------------------------------------------------
string FAFacadeKunde::getName (void)
{
   return rPerson.getName();
}

//----------------------------------------------------------------------------
void FAFacadeKunde::setName (string pName)
{
   rPerson.setName(pName);
}

//----------------------------------------------------------------------------
string FAFacadeKunde::getPlzOrt (void)
{
   return rPerson.getAdresse()->getPlzOrt();
}

//----------------------------------------------------------------------------
string FAFacadeKunde::getStrasseHsnr (void)
{
   return rPerson.getAdresse()->getStrasseHsnr();
}

//----------------------------------------------------------------------------
void FAFacadeKunde::setPlzOrt (string pPlzOrt)
{
   rPerson.getAdresse()->setPlzOrt(pPlzOrt);
}

//----------------------------------------------------------------------------
void FAFacadeKunde::setStrasseHsnr (string pStrasseHsnr)
{
   rPerson.getAdresse()->setStrasseHsnr(pStrasseHsnr);
}

//----------------------------------------------------------------------------
string FAFacadeKunde::getNummer (void)
{
   return rPerson.getAdresse()->getKommunikation()->getNummer();
}

//----------------------------------------------------------------------------
```

```
void FAFacadeKunde::setNummer (string pNummer)
{
   rPerson.getAdresse()->getKommunikation()->setNummer(pNummer);
}
```

7.7.3 Anmerkungen zum C++ Beispiel

Das Beispiel ist bewusst sehr einfach gehalten, in der derzeitigen Komplexität macht das Façade Pattern nur wenig Sinn. Stellt man sich aber vor, dass man z.B. einen Teil der Kommunikationsverbindungen
An der Person hält (z.B. für Handynummern) und einen weiteren Teil an der Adresse (Festnetznummern), so wird bei einer Komplettdarstellung aller vorhandenen Kommunikationsverbindungen die Komplexität auf Klassen abgewälzt, welche die innere Struktur und Aufteilung des Kundenobjekts eigentlich nicht interessieren sollte.

7.8 Java Beispiel

Das hier gezeigte Fassade-Pattern vereinfacht den Zugriff auf eine komplexe Kundenstruktur, indem es die wirkliche Hierarchie der Implementation verbirgt.

7.8.1 Java Klassendiagramm

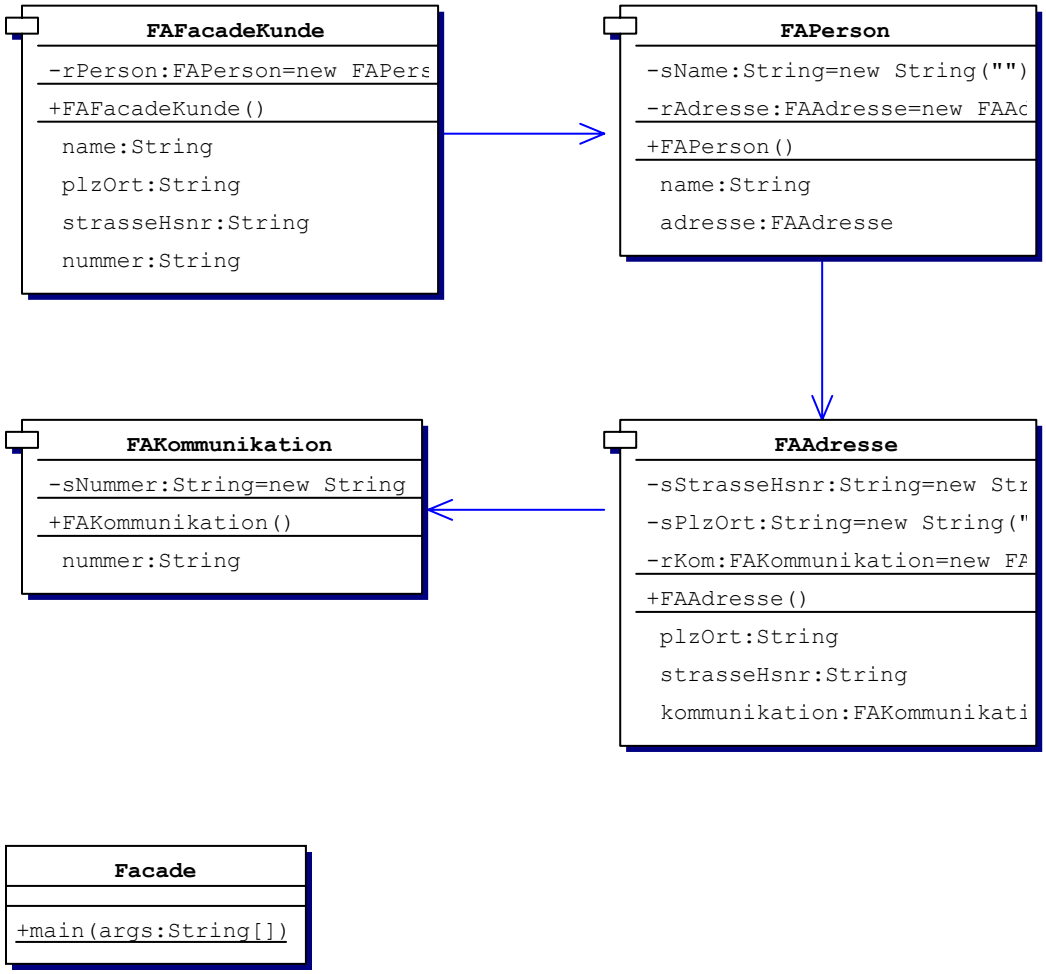

Abbildung 7-2 Java Klassendiagramm Façade

7.8.2 Java Beispielprogramm

```
//======================================================================
/*
 * Java Facade Design Pattern - Testprogramm
 * Created on 15.11.2003
 */
//======================================================================
ppublic class Facade
{
   //----------------------------------------------------------------
   // Hauptprogramm
   //----------------------------------------------------------------
   public static void main(String[] args)
   {
      //-------------------------------------------------------------
      // Ohne Facade-Pattern
      //-------------------------------------------------------------
```

```
      FAPerson        rP = new FAPerson();

      rP.setName("Donald Duck");
      rP.getAdresse().setStrasseHsnr("Entengasse 13");
      rP.getAdresse().setPlzOrt("1313 Entenhausen");
      rP.getAdresse().getKommunikation().setNummer("013 - 13 13 13");

      System.out.println("Name....: " + rP.getName());
      System.out.println("Strasse.: " + rP.getAdresse().getStrasseHsnr());
      System.out.println("Ort.....: " + rP.getAdresse().getPlzOrt());
      System.out.println("Telephon: " +
                         rP.getAdresse().getKommunikation().getNummer());

      System.out.println("");

      //---------------------------------------------------------------------
      // Mit Facade-Pattern
      //---------------------------------------------------------------------
      FAFacadeKunde rF = new FAFacadeKunde();
      rF.setName("Dagobert Duck");
      rF.setStrasseHsnr("Geldspeicher 1");
      rF.setPlzOrt("3333 Entenhausen");
      rF.setNummer("012 - 34 56 78");

      System.out.println("Name....: " + rF.getName());
      System.out.println("Strasse.: " + rF.getStrasseHsnr());
      System.out.println("Ort.....: " + rF.getPlzOrt());
      System.out.println("Telephon: " + rF.getNummer());
   }
}
```

```
//============================================================================
/*
 * Java Facade Design Pattern - Personendaten
 * Created on 15.11.2003
 */
//============================================================================
public class FAPerson
{
   private String    sName     = new String("");
   private FAAdresse rAdresse  = new FAAdresse();

   //---------------------------------------------------------------------
   /**
    * Constructor
    */
   //---------------------------------------------------------------------
   public FAPerson()
   {
      super();
   }

   //---------------------------------------------------------------------
   public String getName()
   {
      return sName;
   }

   //---------------------------------------------------------------------
   public void setName(String pName)
   {
      sName = new String (pName);
```

```
   }

   //-----------------------------------------------------------------------
   public void setAdresse(FAAdresse pAdr)
   {
      rAdresse = pAdr;
   }

   //-----------------------------------------------------------------------
   public FAAdresse getAdresse()
   {
      return rAdresse;
   }
}
```

```
//===========================================================================
/*
 * Java Facade Design Pattern - Adresse
 * Created on 15.11.2003
 */
//===========================================================================
public class FAAdresse
{
   private String sStrasseHsnr  = new String("");
   private String sPlzOrt       = new String("");
   private FAKommunikation rKom = new FAKommunikation();

   //-----------------------------------------------------------------------
   /**
    * Constructor
    */
   //-----------------------------------------------------------------------
   public FAAdresse()
   {
      super();
   }

   //-----------------------------------------------------------------------
   public String getPlzOrt()
   {
      return sPlzOrt;
   }

   //-----------------------------------------------------------------------
   public String getStrasseHsnr()
   {
      return sStrasseHsnr;
   }

   //-----------------------------------------------------------------------
   public void setPlzOrt(String pPlzOrt)
   {
      sPlzOrt = pPlzOrt;
   }

   //-----------------------------------------------------------------------
   public void setStrasseHsnr(String pStrasseHsnr)
   {
      sStrasseHsnr = pStrasseHsnr;
   }

   //-----------------------------------------------------------------------
```

```
   public void setKommunikation(FAKommunikation pKom)
   {
      rKom = pKom;
   }

   //-------------------------------------------------------------------------
   public FAKommunikation getKommunikation()
   {
      return rKom;
   }
}
```

```
//=============================================================================
/*
 * Java Facade Design Pattern - Kommunikationsverbindung
 * Created on 15.11.2003
 */
//=============================================================================
public class FAKommunikation
{
   private String sNummer = new String("");

   //-------------------------------------------------------------------------
   /**
    * Constructor
    */
   //-------------------------------------------------------------------------
   public FAKommunikation()
   {
      super();
   }

   //-------------------------------------------------------------------------
   public String getNummer()
   {
      return sNummer;
   }

   //-------------------------------------------------------------------------
   public void setNummer(String pNummer)
   {
      sNummer = pNummer;
   }
}
```

```
//=============================================================================
/*
 * Java Facade Design Pattern - Facade Kunde, verbirgt Struktur
 * Person/Adresse/Kommunikationsverbindung
 * Created on 15.11.2003
 */
//=============================================================================
public class FAFacadeKunde
{
   private FAPerson rPerson = new FAPerson();

   //-------------------------------------------------------------------------
   /**
    * Constructor
    */
   //-------------------------------------------------------------------------
```

```
   public FAFacadeKunde()
   {
      super();
   }

   //-----------------------------------------------------------------------
   public String getName()
   {
      return rPerson.getName();
   }

   //-----------------------------------------------------------------------
   public void setName(String pName)
   {
      rPerson.setName(pName);
   }

   //-----------------------------------------------------------------------
   public String getPlzOrt()
   {
      return rPerson.getAdresse().getPlzOrt();
   }

   //-----------------------------------------------------------------------
   public String getStrasseHsnr()
   {
      return rPerson.getAdresse().getStrasseHsnr();
   }

   //-----------------------------------------------------------------------
   public void setPlzOrt(String pPlzOrt)
   {
      rPerson.getAdresse().setPlzOrt(pPlzOrt);
   }

   //-----------------------------------------------------------------------
   public void setStrasseHsnr(String pStrasseHsnr)
   {
      rPerson.getAdresse().setStrasseHsnr(pStrasseHsnr);
   }

   //-----------------------------------------------------------------------
   public String getNummer()
   {
      return rPerson.getAdresse().getKommunikation().getNummer();
   }

   //-----------------------------------------------------------------------
   public void setNummer(String pNummer)
   {
      rPerson.getAdresse().getKommunikation().setNummer(pNummer);
   }
}
```

7.8.3 Anmerkungen zum Java Beispiel

Das Beispiel ist bewusst sehr einfach gehalten, in der derzeitigen Komplexität macht das Façade Pattern nur wenig Sinn. Stellt man sich aber vor, dass man z.B. einen Teil der Kommunikationsverbindungen
An der Person hält (z.B. für Handynummern) und einen weiteren Teil an der Adresse (Festnetznummern), so wird bei einer Komplettdarstellung aller vorhandenen

Kommunikationsverbindungen die Komplexität auf Klassen abgewälzt, welche die innere Struktur und Aufteilung des Kundenobjekts eigentlich nicht interessieren sollte.

8. Prototype

Alternativnamen: **Prototyp**
Mustergruppe: **Erzeugungsmuster**

8.1 Anmerkungen

Die Designpatterns Factory Method (⇨ Factory Method) und Prototyp erscheinen im ersten Moment konkurrierend zu sein. In einigen Fällen sind sie es sicherlich auch, sie können sich jedoch auch hervorragend ergänzen.

8.2 Verwendungszweck

Das Pattern Prototype konstruiert neue Instanzen eines Objektes, indem eine vorhandene Instanz geklont wird. Die Applikation benötigt dazu – anders als bei Aufruf eines Copy-Constructors – keinerlei Kenntnis über den Typ des Objektes.
Das Pattern ist auch hervorragend dazu geeignet den Programmieraufwand gering zu halten, wenn eine Vielzahl von Instanzen zu erzeugen ist, die sich nur geringfügig voneinander unterscheiden. In diesem Fall ist es einfacher ein bekanntermaßen ähnliches Objekt zu kopieren und anschließend nur die notwendigen Änderungen vorzunehmen. Ein externes Kopieren (Anlage einer neuen Instanz und schrittweise lesen aus dem Vorbild und setzen in die Kopie) ist immer verbunden mit der teilweisen Offenlegung der Struktur (wenn ein zu kopierendes Attribut plötzlich entfällt, muss auch die Kopierfunktion geändert werden).

8.3 Problemstellungen

Wie kann man ein Objekt kopieren, dessen Typ man nicht kennt, weil es aus einer Factory oder aus einer dynamisch gebundenen Bibliothek stammt?

8.4 Lösung

Die Verwendung eines Copy-Constructors aus der Applikation heraus ist nicht möglich, da der Typ der Instanz unbekannt (bei dynamischer Anlage in einer externen Bibliothek) oder nicht ermittelbar ist (durch Factory erzeugt). Letztlich weiß in dieser Situation nur eine Stelle von welchem Typ eine Instanz ist: die Instanz selbst.
Um nun eine – bezogen auf den Datentyp – „anonyme“ Instanz zu kopieren wird ein Ansatzpunkt benötigt, um die Instanz dazu zu bringen den eigenen Copy-Constructor aufzurufen. Für diesen Zweck verwendet man üblicherweise den Methodennamen ***clone()***. In C++ führt man dazu eine ***clone()*** Methode (meist ***pure virtual***) an der Basisklasse einer Hierarchie ein, die z.B. von einer Factory erzeugt wird. In Java hingegen versieht man die Basisklasse mit dem Interface ***cloneable***.
Jede abgeleitete Klasse ist nun verpflichtet die ***clone()*** Methode zu überschreiben und einen Zeiger (C++) oder eine Referenz (Java) vom Typ der gemeinsamen Basisklasse zurückzugeben.

8.5 Vorteile

Der Typ einer Instanz muss nicht mehr bekannt sein, um eine Kopie davon zu erzeugen. Die Applikation ist von der Kenntnis aller verwendeten Klassenspezialisierungen befreit.

8.6 Nachteile

Jede Klasse der Hierarchie muss eine zusätzliche Methode ***clone()*** bereitstellen, dies ist zusätzlicher Aufwand in der Implementation. Unter C++, wo jede Klasse ohnehin einen Copy-Constructor definieren sollte um STL-fähig zu sein, ist der Aufwand relativ gering. Java hat mit dem ***cloneable*** Interface den Vorteil eine vordefinierte Schnittstelle für diesen Zweck zu besitzen, deren Funktionalität allen Entwicklern bekannt ist.

8.7 C++ Beispiel

Das Beispiel zeigt ein typisches Beispiel aus einem Objektorientierten Programm (hier ein Graphikprogramm), bei dem eine Point-and-Click-Oberfläche realisiert wird.

8.7.1 C++ Klassendiagramm

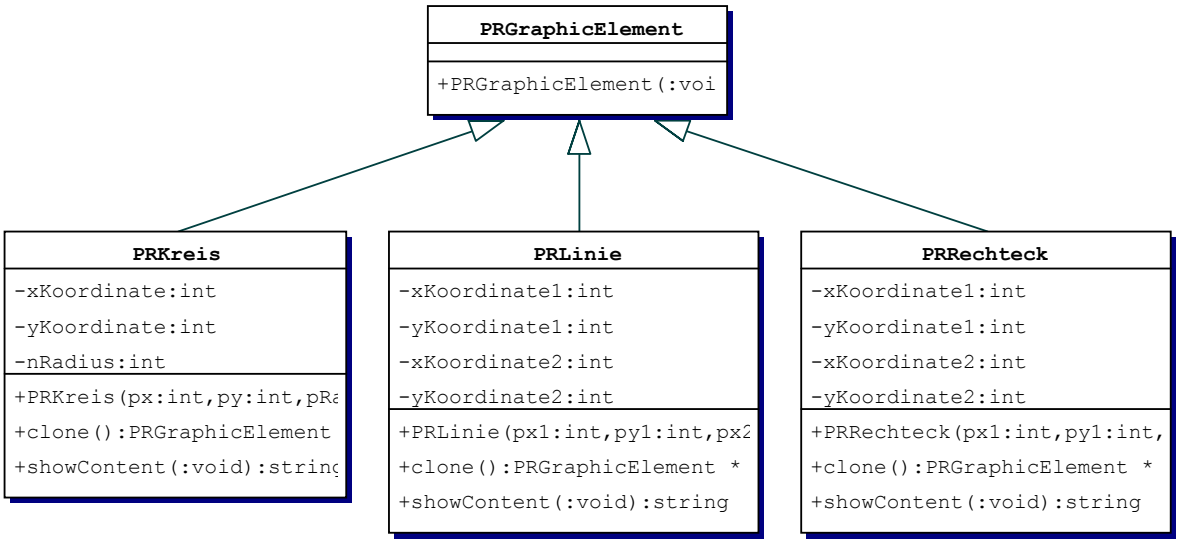

Abbildung 8-1 C++ Klassendiagramm Prototype

8.7.2 C++ Beispielprogramm

```
//============================================================================
// C++ Prototype Designpattern - Prototype.cpp
//============================================================================
#include "stdafx.h"
#include "PRKreis.h"
#include "PRLinie.h"
#include "PRRechteck.h"
#include <time.h>
#include <stdlib.h>

PRGraphicElement* aElements [10];
PRGraphicElement* aCopy     [10];

int _tmain(int argc, _TCHAR* argv[])
{
   srand ((unsigned)time(NULL));

   // 10 zufällige Elemente erzeugen
   for (int i=0; i<10; i++)
   {
      switch (rand() % 3)
      {
         case  0:
               aElements [i] = new PRKreis(rand(), rand(), rand());
               break;
         case  1:
               aElements [i] = new PRLinie(rand(), rand(),
                                           rand(), rand());
               break;
         case  2:
               aElements [i] = new PRRechteck(rand(), rand(),
```

```
                                                rand(), rand());
                break;
         }
      }

      // 10 erzeugte Elemente ausgeben
      for (int i=0; i<10; i++)
      {
         cout << aElements[i]->showContent() << endl;
      }
      cout << "-----------------------" << endl;

      // Die 10 zufällig erzeugten Elemente kopieren
      for (int i=0; i<10; i++)
      {
         aCopy[i] = aElements[i]->clone();
      }

      // Die 10 kopierten Elemente ausgeben
      for (int i=0; i<10; i++)
      {
         cout << aCopy[i]->showContent() << endl;
      }

      // 10 Elemente wieder freigeben
      for (int i=0; i<10; i++)
      {
         delete aElements[i];
         aElements[i] = NULL;
         delete aCopy[i];
         aCopy[i] = NULL;
      }

      return 0;
}
```

```
//===========================================================================
// C++ Prototype Designpattern - PRGraphicElement.h
//===========================================================================
#ifndef _PRGRAPHICELEMENT_H_
#define _PRGRAPHICELEMENT_H_

#include <string>

using namespace std;

class PRGraphicElement
{
   public:
      PRGraphicElement(void) {};

      virtual PRGraphicElement* clone() = NULL;
      virtual string showContent(void)  = NULL;
};

#endif
```

```
//===========================================================================
// C++ Prototype Designpattern - PRLinie.h
//===========================================================================
#ifndef _PRLINIE_H_
```

```
#define _PRLINIE_H_

#include <iostream>
#include "PRGraphicElement.h"
#include <string>

using namespace std;

class PRLinie : public PRGraphicElement
{
   public:
      PRLinie(int px1, int py1, int px2, int py2);

      virtual PRGraphicElement* clone();
      virtual string showContent(void);

   private:
      int xKoordinate1;
      int yKoordinate1;
      int xKoordinate2;
      int yKoordinate2;
};

#endif
```

```
//============================================================================
// C++ Prototype Designpattern - PRLinie.cpp
//============================================================================
#include "stdafx.h"
#include "PRLinie.h"
#include <iostream>

using namespace std;

//----------------------------------------------------------------------------
PRLinie::PRLinie(int px1, int py1, int px2, int py2)
{
   xKoordinate1 = px1;
   yKoordinate1 = py1;
   xKoordinate2 = px2;
   yKoordinate2 = py1;
}

//----------------------------------------------------------------------------
PRGraphicElement* PRLinie::clone()
{
   return new PRLinie(xKoordinate1, yKoordinate1,
                      xKoordinate2, yKoordinate2);
}

//----------------------------------------------------------------------------
string PRLinie::showContent(void)
{
   char s[1000] = "";
   sprintf (s, "[#%p] Linie: %d, %d, %d, %d",
            this, xKoordinate1, yKoordinate1,
            xKoordinate2, yKoordinate1);
   string sRC = s;
   return sRC;
}
```

```
//============================================================================
// C++ Prototype Designpattern - PRRechteck.h
//============================================================================
#ifndef _PRRECHTECK_H_
#define _PRRECHTECK_H_

#include <iostream>
#include <string>
#include "PRGraphicElement.h"

using namespace std;

class PRRechteck : public PRGraphicElement
{
   public:
      PRRechteck(int px1, int py1, int px2, int py2);

      virtual PRGraphicElement* clone();
      virtual string showContent(void);

   private:
      int xKoordinate1;
      int yKoordinate1;
      int xKoordinate2;
      int yKoordinate2;
};

#endif
```

```
//============================================================================
// C++ Prototype Designpattern - PRRechteck.cpp
//============================================================================
#include "stdafx.h"
#include "PRRechteck.h"
#include <iostream>

using namespace std;

//----------------------------------------------------------------------------
PRRechteck::PRRechteck(int px1, int py1, int px2, int py2)
{
   xKoordinate1 = px1;
   yKoordinate1 = py1;
   xKoordinate2 = px2;
   yKoordinate2 = py1;
}

//----------------------------------------------------------------------------
PRGraphicElement* PRRechteck::clone()
{
   return new PRRechteck(xKoordinate1, yKoordinate1,
                         xKoordinate2, yKoordinate2);
}

//----------------------------------------------------------------------------
string PRRechteck::showContent(void)
{
   char s[1000] = "";
   sprintf (s, "[#%p] Rechteck: %d, %d, %d, %d",
            this, xKoordinate1, yKoordinate1,
                  xKoordinate2, yKoordinate1);
   string sRC = s;
```

```
   return sRC;
}
```

```
//============================================================================
// C++ Prototype Designpattern - PRKreis.h
//============================================================================
#ifndef _PRKREIS_H_
#define _PRKREIS_H_

#include <iostream>
#include "PRGraphicElement.h"
#include <string>

using namespace std;

class PRKreis : public PRGraphicElement
{
   public:
      PRKreis(int px, int py, int pRad);

      virtual PRGraphicElement* clone();
      virtual string showContent(void);

   private:
      int xKoordinate;
      int yKoordinate;
      int nRadius;
};

#endif
```

```
//============================================================================
// C++ Prototype Designpattern - PRKreis.cpp
//============================================================================
#include "stdafx.h"
#include "PRKreis.h"
#include <iostream>

using namespace std;

//----------------------------------------------------------------------------
PRKreis::PRKreis(int px, int py, int pRad)
{
   xKoordinate = px;
   yKoordinate = py;
   nRadius     = pRad;
}

//----------------------------------------------------------------------------
PRGraphicElement* PRKreis::clone()
{
   return new PRKreis(xKoordinate, yKoordinate, nRadius);
}

//----------------------------------------------------------------------------
string PRKreis::showContent(void)
{
   char s[1000] = "";
   sprintf (s, "[#%p] Kreis: %d, %d, %d",
            this, xKoordinate, yKoordinate,
            nRadius);
```

```
    string sRC = s;
    return sRC;
}
```

8.7.3 Anmerkungen zum C++ Beispiel

Durch die Anonymität der Objekte bei Aufruf der ***clone()*** Methode wird eine hohe Abstraktion und Erweiterbarkeit erreicht.

8.8 Java Beispiel

Das Beispiel zeigt ein typisches Beispiel aus einem Objektorientierten Programm (hier ein Graphikprogramm), bei dem eine Point-and-Click-Oberfläche realisiert wird.

8.8.1 Java Klassendiagramm

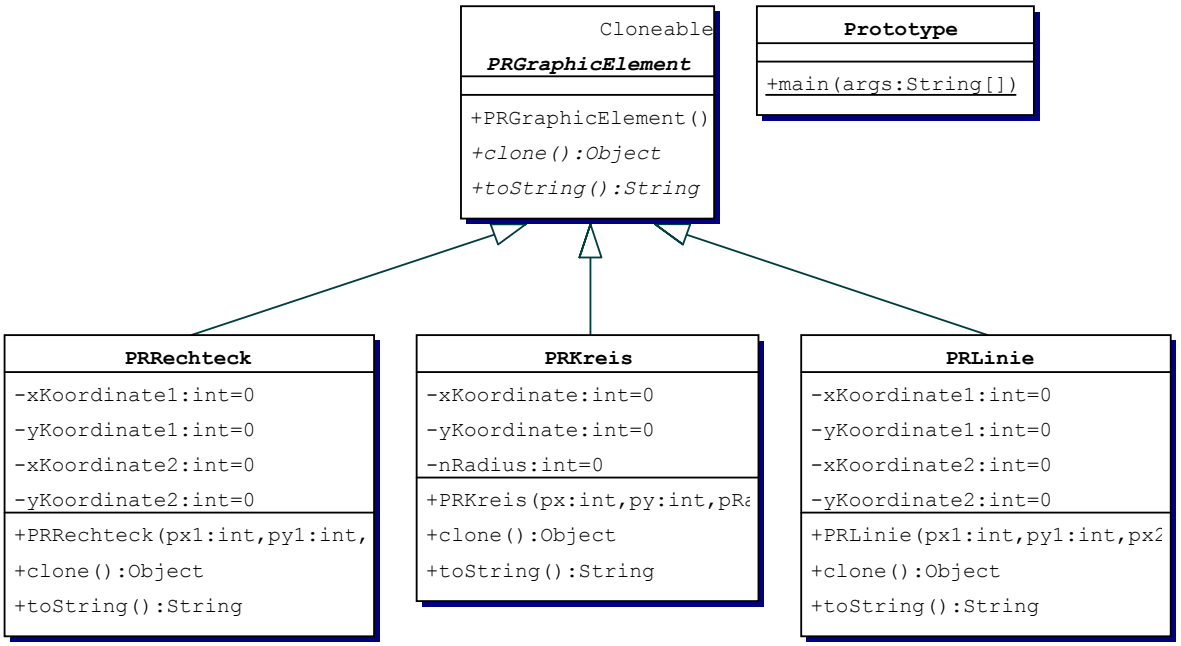

Abbildung 8-2 Java Klassendiagramm Prototype

8.8.2 Java Beispielprogramm

```
//=============================================================================
/** Java Prototype Design Pattern - Testprogramm
 */
//=============================================================================
import java.util.Random;
import java.util.Vector;

public class Prototype
{
   //-------------------------------------------------------------------------
   // Hauptprogramm
   //-------------------------------------------------------------------------
   public static void main(String[] args)
   {
      Random rRand     = new Random (System.currentTimeMillis());
      Vector vElements = new Vector();
      Vector vCopy     = new Vector();

      // 10 zufällige Elemente erzeugen
      for (int i=0; i<10; i++)
      {
         switch (Math.abs(rRand.nextInt()) % 3)
         {
            case  0:
                  vElements.add(new PRKreis(rRand.nextInt(),
                                            rRand.nextInt(),
                                            rRand.nextInt()));
```

```
                  break;
            case  1:
                  vElements.add(new PRLinie(rRand.nextInt(),
                                            rRand.nextInt(),
                                            rRand.nextInt(),
                                            rRand.nextInt()));
                  break;
            case  2:
                  vElements.add(new PRRechteck(rRand.nextInt(),
                                               rRand.nextInt(),
                                               rRand.nextInt(),
                                               rRand.nextInt()));
                  break;
         }
      }

      // 10 zufällig erzeugte Elemente ausgeben
      for (int i=0; i<10; i++)
      {
         System.out.println (vElements.elementAt(i).toString());
      }
      System.out.println ("-----------------------");

      // 10 zufällig erzeugte Elemente kopieren
      for (int i=0; i<10; i++)
      {
         vCopy.add (((PRGraphicElement)vElements.elementAt(i)).clone());
      }

      // 10 zufällig erzeugte Elemente ausgeben
      for (int i=0; i<10; i++)
      {
         System.out.println (vCopy.elementAt(i).toString());
      }
   }
}
```

```
//============================================================================
/** Java Prototype Design Pattern - Graphikelement, abstrakte Basisklasse
 */
//============================================================================
public abstract class PRGraphicElement implements Cloneable
{
   //-------------------------------------------------------------------------
   public PRGraphicElement()
   {
      super();
   }

   public abstract Object clone();
   public abstract String toString();
}
```

```
//============================================================================
/** Java Prototype Design Pattern - Linie
 */
//============================================================================
public class PRLinie extends PRGraphicElement
{
   private int xKoordinate1 = 0;
   private int yKoordinate1 = 0;
```

```
   private int xKoordinate2 = 0;
   private int yKoordinate2 = 0;

   //------------------------------------------------------------------------
   public PRLinie(int px1, int py1, int px2, int py2)
   {
      super();
      xKoordinate1 = px1;
      yKoordinate1 = py1;
      xKoordinate2 = px2;
      yKoordinate2 = py1;
   }

   //------------------------------------------------------------------------
   public Object clone()
   {
      PRLinie rRC = new PRLinie(xKoordinate1, yKoordinate1,
                                xKoordinate2, yKoordinate2);
      return rRC;
   }

   //------------------------------------------------------------------------
   public String toString()
   {
      StringBuffer sb = new StringBuffer();
      sb.append ("[#"+ hashCode() + "] Linie: " + xKoordinate1 + ", "
                     + yKoordinate1 + ", " + xKoordinate2 + ", "
                     + yKoordinate2);
      return sb.toString();
   }
}
```

```
//============================================================================
/** Java Prototype Design Pattern - Kreis
 */
//============================================================================
public class PRKreis extends PRGraphicElement
{
   private int xKoordinate = 0;
   private int yKoordinate = 0;
   private int nRadius     = 0;

   //------------------------------------------------------------------------
   public PRKreis(int px, int py, int pRad)
   {
      super();
      xKoordinate = px;
      yKoordinate = py;
      nRadius     = pRad;
   }

   //------------------------------------------------------------------------
   public Object clone()
   {
      PRKreis rRC = new PRKreis(xKoordinate, yKoordinate, nRadius);
      return rRC;
   }

   //------------------------------------------------------------------------
   public String toString()
   {
      StringBuffer sb = new StringBuffer();
```

```
      sb.append ("[#"+ hashCode() + "] Kreis: " + xKoordinate + ", "
                     + yKoordinate + ", rad " + nRadius);
      return sb.toString();
   }
}
```

```
//============================================================================
/** Java Prototype Design Pattern - Rechteck
 */
//============================================================================
public class PRRechteck extends PRGraphicElement
{
   private int xKoordinate1 = 0;
   private int yKoordinate1 = 0;
   private int xKoordinate2 = 0;
   private int yKoordinate2 = 0;

   //------------------------------------------------------------------------
   public PRRechteck(int px1, int py1, int px2, int py2)
   {
      super();
      xKoordinate1 = px1;
      yKoordinate1 = py1;
      xKoordinate2 = px2;
      yKoordinate2 = py1;
   }

   //------------------------------------------------------------------------
   public Object clone()
   {
      PRRechteck rRC = new PRRechteck(xKoordinate1, yKoordinate1,
                                      xKoordinate2, yKoordinate2);
      return rRC;
   }

   //------------------------------------------------------------------------
   public String toString()
   {
      StringBuffer sb = new StringBuffer();
      sb.append ("[#"+ hashCode() + "] Rechteck: " + xKoordinate1 + ", "
                     + yKoordinate1 + ", " + xKoordinate2 + ", "
                     + yKoordinate2);
      return sb.toString();
   }
}
```

8.8.3 Anmerkungen zum Java Beispiel

Durch die Anonymität der Objekte bei Aufruf der ***clone()*** Methode wird eine hohe Abstraktion und Erweiterbarkeit erreicht.

9. Bridge

Alternativnamen: Brücke, Handle, Body
Mustergruppe: Strukturmuster

9.1 Anmerkungen

Die Bridge ist ein relativ einfach zu verstehendes Designpatterns. Obwohl nicht Grundlage für komplexere Designpatterns ist es häufig in Kombination mit anderen zu finden. Dies liegt primär daran, dass das Bridge Designpattern zumeist in relativ großen Anwendungen mit hohem Skalierungsbedarf eingesetzt wird.
Die Bridge ähnelt dem Adapter (⇨ Adapter), denn beide sorgen für eine einheitliche, abstrakte Schnittstelle zu sonst inkompatiblen Codeteilen. Der Adapter ist aber eher ein nachträgliches Korrektiv, während die Bridge ein meist bereits in der Frühphase bewusst gewähltes Muster ist. Auch handelt es sich bei der Bridge meist um eine hierarchische, erweiterbare Klassenfamilie, beim Adapter hingegen um eine singuläre Klasse.

9.2 Verwendungszweck

Das Bridge Designpattern stellt sicher, dass Programmteile, von denen man weiß, dass sie plattform- oder Anwenderabhängig sind, so aus der Applikation isoliert werden, dass in der Applikation selbst bei Plattformwechsel kein (oder nur geringer) Änderungsaufwand entsteht.
Dieses Muster ist besonders nützlich um:

- Codebereiche mit hohem Wartungs- und Änderungspotential aus dem Applikationsprogramm zu isolieren.
  Dazu gehört z.B. der Zugriff GUI[6]-Bibliotheken auf verschiedenen Betriebssystemen, Zugriffsschnittstellen auf Datenbanken (wenn unterschiedliche Datenbanken anzubinden sind) oder länderspezifische Einstellungen.
- Implementation verschiedener Sichten auf gleiche Datenbestände.
- Isolierung von Schnittstelle und Implementierung.
  Die Trennung von Implementierung und Schnittstelle entfernt die Details der Implementation aus der Applikation. Die spezifischen Teile der Implementation werden meist als externe API[7] hinzugefügt (z.B. in Form einer DLL[8] oder eines JAR[9]-Files). Die Implementation kann dann verändert oder ausgetauscht werden, ohne die Applikation ändern zu müssen.

9.3 Problemstellungen

Wie kann man absehbar nicht portabel programmierbare Programmteile so von der Applikation entkoppeln, dass in der Applikation selbst kein Änderungsaufwand entsteht, wenn ein eine andere Plattform bedient werden muss.

[6] GUI = Graphical User Interface – Graphische Benutzungsschnittstelle. Hierunter werden heutzutage primär die auf Fenster basierten Systeme verstanden wie z.B. Windows, XWindows, McOS, NeXt und andere.
[7] API = Application Programmers Interface – Programmschnittstelle
[8] DLL = Dynamic Link Library – dynamisch geladene Programmbibliothek)
[9] JAR = Java Archive – zu einer Datei zusammengefasste, zuvor compilierte Javaklassen

9.4 Lösung

Es werden zwei Klassenhierarchien (im einfachsten Fall mit identischem Aufbau) gebildet. Die erste Hierarchie, ohne Implementation wird in die Applikation eingebunden und enthält in ihren Klassen nur Aufrufe in die identische zweite Hierarchie. Die zweite Hierarchie enthält die eigentliche, spezifische Implementation. Die eigentliche Brücke entsteht in der Wurzelklasse beider Hierarchien, wo von Seiten der Applikation ein Verweis in Form eines Zeigers (C++) oder einer Referenz (Java) auf die eigentliche Implementation gehalten wird. Wird dieser Verweis über dynamische Bindung zur Laufzeit erzeugt (z.B. einer DLL entnommen), so ist die Entkopplung vollständig.

9.5 Vorteile

Die Implementation wird vollständig vor der Applikation verborgen und ist unabhängig von dieser. Änderungen in Applikation haben keine Auswirkungen auf die entkoppelte Implementierung und umgekehrt.
Die Programmlogik der Applikation ist unabhängig von der Plattform auf der sie eingesetzt wird. Der Änderungsaufwand für eine weitere Plattform wird minimiert und findet sich konzentriert in einem einzigen Modul anstatt über das gesamte Programm verstreut.
Die Schnittstellen sind frühzeitig in der Programmentwicklung klar, so dass die Implementierung gegebenenfalls von unabhängigen Teams mit minimalem Aufwand zur Synchronisation betrieben werden kann.
Das Bridge Designpattern ist gut durch zusätzliche Unterklassen erweiterbar, da beide Stränge separat erweitert werden können, ohne dass zwangsläufig der andere mit geändert werden muss.

9.6 Nachteile

Durch die Verwendung von zwei oder mehr (ggf. sogar gleichartigen) Hierarchien steigt die Anzahl der zu implementierenden Klassen gegebenenfalls erheblich.
Die Schnittstellen der beiden Hierarchien müssen frühzeitig definiert werden, jede nachträgliche Änderung bestehender Schnittstellen führt zumeist zu recht hohem Änderungsaufwand.

9.7 C++ Beispiel

Das folgende Beispiel zeigt den Einsatz des Bridge-Patterns zur Entkopplung von Details der Implementation. Die Anwendung besteht aus einer Model-Klasse, welche die Daten hält (hier ***BRModel***) – in diesem Fall eine Preisliste. Es gibt zu diesem Model allerdings mehr als eine Sicht (View). Die Sicht des Verkäufers, dem es nicht gestattet ist, Preisnachlässe zu gewähren und die Sicht des Verkaufsleiters, der Preisnachlässe bis zu einem rechnerischen Minimalverkaufspreis einräumen kann.
Es wäre natürlich möglich, dies auch ohne die Bridge zu implementieren, dann hätten aber alle View Klassen Kenntnis über den Aufbau (d.h. die Datentypen) des Models. Bei einer Änderung des Models müssten ggf. alle drei View Klassen korrigiert werden, durch den Einsatz der Bridge ist dies nur in der Bridgeklasse (***BRBridge***) nötig.

9.7.1 C++ Klassendiagramm

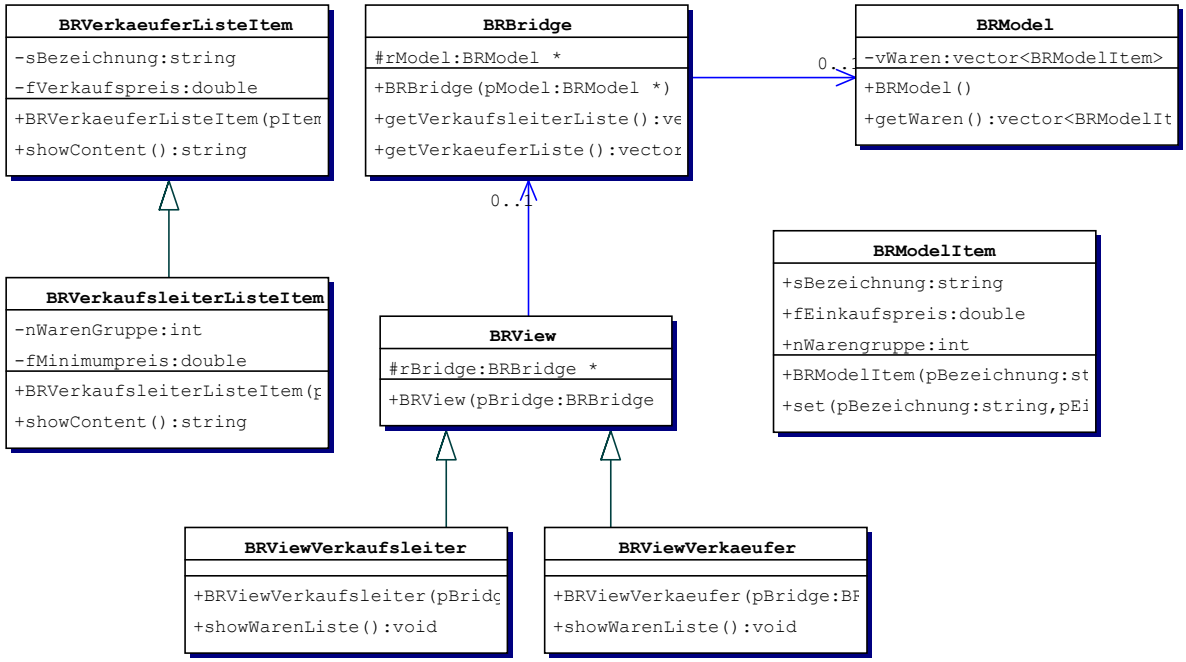

Abbildung 9-1 C++ Klassendiagramm Bridge

9.7.2 C++ Beispielprogramm

```
//============================================================================
// C++ Bridge Designpattern - Bridge.cpp
//============================================================================
#include "stdafx.h"

#include "BRBridge.h"
#include "BRViewVerkaeufer.h"
#include "BRViewVerkaufsleiter.h"

int _tmain(int argc, _TCHAR* argv[])
{
   BRModel              rModel;
   BRBridge             rBridge(&rModel);
   BRViewVerkaeufer     rView1(&rBridge);
   BRViewVerkaufsleiter rView2(&rBridge);

   rView1.showWarenListe();
   rView2.showWarenListe();
      return 0;
}
```

```
//============================================================================
// C++ Bridge Designpattern - BRModelItem.h
//============================================================================
#ifndef _BRMODELITEM_H_
#define _BRMODELITEM_H_

#include <string>

using namespace std;

class BRModelItem
{
```

```
   public:
      BRModelItem(string pBezeichnung, double pEinkaufspreis,
                  int pWarengruppe);

      void set (string pBezeichnung, double pEinkaufspreis,
                int pWarengruppe);

      string sBezeichnung;
      double fEinkaufspreis;
      int    nWarengruppe;
};

#endif
```

```
//============================================================================
// C++ Bridge Designpattern - BRModelItem.cpp
//============================================================================
#include "stdafx.h"

#include "BRModelItem.h"

using namespace std;

//----------------------------------------------------------------------------
/**
 * Constructor
 */
//----------------------------------------------------------------------------
BRModelItem::BRModelItem(string pBezeichnung, double pEinkaufspreis,
                         int pWarengruppe)
{
   set(pBezeichnung, pEinkaufspreis, pWarengruppe);
}

//----------------------------------------------------------------------------
/**
 * set
 */
//----------------------------------------------------------------------------
void BRModelItem::set (string pBezeichnung, double pEinkaufspreis,
                       int pWarengruppe)
{
   sBezeichnung   = pBezeichnung;
   fEinkaufspreis = pEinkaufspreis;
   nWarengruppe   = pWarengruppe;
}
```

```
//============================================================================
// C++ Bridge Designpattern - BRModel.h
//============================================================================
#ifndef _BRMODEL_H_
#define _BRMODEL_H_

#include <vector>
#include "BRModelItem.h"

using namespace std;

class BRModel
{
   public:
```

```
      BRModel();
      vector<BRModelItem> getWaren();

   private:
      vector<BRModelItem> vWaren;
};

#endif
```

```
//============================================================================
// C++ Bridge Designpattern - BRModel.cpp
//============================================================================
#include "stdafx.h"

#include "BRModel.h"

using namespace std;

//----------------------------------------------------------------------------
/**
 * Standard-Constructor
 */
//----------------------------------------------------------------------------
BRModel::BRModel()
{
   BRModelItem Temp("Lampe",   63.45, 0);
   vWaren.push_back(Temp);

   Temp.set("Sessel", 110.95, 1);
   vWaren.push_back(Temp);

   Temp.set("Sofa",   221.15, 1);
   vWaren.push_back(Temp);

   Temp.set("Tisch",  177.15, 3);
   vWaren.push_back(Temp);

   Temp.set("Stuhl",   58.55, 3);
   vWaren.push_back(Temp);
}

//----------------------------------------------------------------------------
/**
 * @return Vector, bestehend aus BRModelItems
 */
//----------------------------------------------------------------------------
vector<BRModelItem> BRModel::getWaren()
{
   return vWaren;
}
```

```
//============================================================================
// C++ Bridge Designpattern - BRBridge.h
//============================================================================
#ifndef _BRBRIDGE_H_
#define _BRBRIDGE_H_

#include <vector>
#include "BRModel.h"
#include "BRVerkaufsleiterListeItem.h"
#include "BRVerkaeuferListeItem.h"
```

```
using namespace std;

class BRBridge
{
   public:
      BRBridge(BRModel *pModel);

      vector<BRVerkaufsleiterListeItem> getVerkaufsleiterListe();
      vector<BRVerkaeuferListeItem> getVerkaeuferListe();

   protected:
      BRModel *rModel;
};

#endif
```

```
//============================================================================
// C++ Bridge Designpattern - BRBridge.cpp
//============================================================================
#include "stdafx.h"

#include "BRBridge.h"

using namespace std;

//----------------------------------------------------------------------------
/**
 * Constructor
 */
//----------------------------------------------------------------------------
BRBridge::BRBridge(BRModel *pModel)
{
   rModel = pModel;
}

//----------------------------------------------------------------------------
/**
 * getVerkaeuferListe
 */
//----------------------------------------------------------------------------
vector<BRVerkaeuferListeItem> BRBridge::getVerkaeuferListe()
{
   vector<BRVerkaeuferListeItem> vListe;
   vector<BRModelItem> vModelListe = rModel->getWaren();

   for (unsigned int i=0; i<vModelListe.size(); i++)
   {
      BRVerkaeuferListeItem rTemp(vModelListe[i]);
      vListe.push_back(rTemp);
   }
   return vListe;
}

//----------------------------------------------------------------------------
/**
 * getVerkaufsleiterListe
 */
//----------------------------------------------------------------------------
vector<BRVerkaufsleiterListeItem> BRBridge::getVerkaufsleiterListe()
{
   vector<BRVerkaufsleiterListeItem> vListe;
```

```
   vector<BRModelItem> vModelListe = rModel->getWaren();

   for (unsigned int i=0; i<vModelListe.size(); i++)
   {
      BRVerkaufsleiterListeItem rTemp(vModelListe[i]);
      vListe.push_back(rTemp);
   }
   return vListe;
}
```

```
//============================================================================
// C++ Bridge Designpattern - BRVerkaeuferListeItem.h
//============================================================================
#ifndef _BRVERKAEUFERLISTEITEM_H_
#define _BRVERKAEUFERLISTEITEM_H_

#include <string>
#include "BRModelItem.h"

using namespace std;

class BRVerkaeuferListeItem
{
   public:
      BRVerkaeuferListeItem(BRModelItem pItem);

      string showContent();

   private:
      string sBezeichnung;
      double fVerkaufspreis;
};

#endif
```

```
//============================================================================
// C++ Bridge Designpattern - BRVerkaeuferListeItem.cpp
//============================================================================
#include "stdafx.h"

#include "BRVerkaeuferListeItem.h"

using namespace std;

//----------------------------------------------------------------------------
/**
 * Constructor
 */
//----------------------------------------------------------------------------
BRVerkaeuferListeItem::BRVerkaeuferListeItem(BRModelItem pItem)
{
   sBezeichnung   = pItem.sBezeichnung;
   fVerkaufspreis = pItem.fEinkaufspreis + pItem.fEinkaufspreis * 1.2;
   fVerkaufspreis = fVerkaufspreis * 1.16;
}

//----------------------------------------------------------------------------
/**
 * showContent
 */
//----------------------------------------------------------------------------
```

```
string BRVerkaeuferListeItem::showContent()
{
   char s[1000] = "";
   sprintf (s, "%-15s: %10.2lf Euro",
            sBezeichnung.c_str(), fVerkaufspreis);
   string sRC = s;
   return sRC;
}
```

```
//===========================================================================
// C++ Bridge Designpattern - BRVerkaufsleiterListeItem.h
//===========================================================================
#ifndef _BRVERKAUFSLEITERLISTEITEM_H_
#define _BRVERKAUFSLEITERLISTEITEM_H_

#include <string>
#include "BRVerkaeuferListeItem.h"

using namespace std;

class BRVerkaufsleiterListeItem : public BRVerkaeuferListeItem
{
   public:
      BRVerkaufsleiterListeItem(BRModelItem pItem);

      string showContent();

   private:
      int    nWarenGruppe;
      double fMinimumpreis;
};

#endif
```

```
//===========================================================================
// C++ Bridge Designpattern - BRVerkaufsleiterListeItem.cpp
//===========================================================================
#include "stdafx.h"

#include "BRVerkaufsleiterListeItem.h"

using namespace std;

//---------------------------------------------------------------------------
/**
 * Constructor
 */
//---------------------------------------------------------------------------
BRVerkaufsleiterListeItem::BRVerkaufsleiterListeItem(BRModelItem pItem)
: BRVerkaeuferListeItem(pItem)
{
   nWarenGruppe = pItem.nWarengruppe;
   switch (nWarenGruppe)
   {
    case 0:  fMinimumpreis = pItem.fEinkaufspreis+pItem.fEinkaufspreis*1.2;
             fMinimumpreis = fMinimumpreis * 1.16;
             break;
    case 1:  fMinimumpreis = pItem.fEinkaufspreis+pItem.fEinkaufspreis*1.0;
             fMinimumpreis = fMinimumpreis * 1.16;
             break;
    default: fMinimumpreis = pItem.fEinkaufspreis+pItem.fEinkaufspreis*0.5;
```

```
            fMinimumpreis = fMinimumpreis * 1.16;
            break;
   }
}

//---------------------------------------------------------------------------
/**
 * showContent
 */
//---------------------------------------------------------------------------
string BRVerkaufsleiterListeItem::showContent()
{
   char s[1000] = "";
   sprintf (s, "%-15s  Warengruppe: %d  Mindestpreis: %10.2lf Euro",
            BRVerkaeuferListeItem::showContent().c_str(),
            nWarenGruppe, fMinimumpreis);
   string sRC = s;
   return sRC;
}
```

```
//===========================================================================
// C++ Bridge Designpattern - BRView.h
//===========================================================================
#ifndef _BRVIEW_H_
#define _BRVIEW_H_

#include "BRBridge.h"

using namespace std;

class BRView
{
   public:
      BRView(BRBridge *pBridge);
      virtual void showWarenListe() = NULL;

   protected:
      BRBridge *rBridge;
};

#endif
```

```
//===========================================================================
// C++ Bridge Designpattern - BRView.cpp
//===========================================================================
#include "stdafx.h"

#include "BRView.h"

using namespace std;

//---------------------------------------------------------------------------
/**
 * Constructor
 */
//---------------------------------------------------------------------------
BRView::BRView(BRBridge *pBridge)
{
   rBridge = pBridge;
}
```

```
//============================================================================
// C++ Bridge Designpattern - BRViewVerkaeufer.h
//============================================================================
#ifndef _BRVIEWVERKAEUFER_H_
#define _BRVIEWVERKAEUFER_H_

#include <vector>
#include "BRBridge.h"
#include "BRView.h"

using namespace std;

class BRViewVerkaeufer : public BRView
{
   public:
      BRViewVerkaeufer(BRBridge *pBridge);
      void showWarenListe();
};

#endif
```

```
//============================================================================
// C++ Bridge Designpattern - BRViewVerkaeufer.cpp
//============================================================================
#include "stdafx.h"

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

using namespace std;

//----------------------------------------------------------------------------
/**
 * Constructor
 */
//----------------------------------------------------------------------------
BRViewVerkaeufer::BRViewVerkaeufer(BRBridge *pBridge) : BRView(pBridge)
{
}

//----------------------------------------------------------------------------
/**
 * showWarenListe
 */
//----------------------------------------------------------------------------
void BRViewVerkaeufer::showWarenListe()
{
   vector<BRVerkaeuferListeItem> vListe = rBridge->getVerkaeuferListe();
   cout << "****** Verkaeufer ******" << endl;
   for (unsigned int i=0; i<vListe.size(); i++)
   {
      cout << vListe[i].showContent() << endl;
   }
   cout << endl;
}
```

```
//============================================================================
// C++ Bridge Designpattern - BRViewVerkaufsleiter.h
//============================================================================
#ifndef _BRVIEWVERKAUFSLEITER_H_
#define _BRVIEWVERKAUFSLEITER_H_
```

```
#include <vector>
#include "BRBridge.h"
#include "BRView.h"

using namespace std;

class BRViewVerkaufsleiter : public BRView
{
   public:
      BRViewVerkaufsleiter(BRBridge *pBridge);
      void showWarenListe();
};

#endif
```

```
//============================================================================
// C++ Bridge Designpattern - BRViewVerkaufsleiter.cpp
//============================================================================
#include "stdafx.h"

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

using namespace std;

//----------------------------------------------------------------------------
/**
 * Constructor
 */
//----------------------------------------------------------------------------
BRViewVerkaufsleiter::BRViewVerkaufsleiter(BRBridge       *pBridge)       :
BRView(pBridge)
{
}

//----------------------------------------------------------------------------
/**
 * showWarenListe
 */
//----------------------------------------------------------------------------
void BRViewVerkaufsleiter::showWarenListe()
{
   vector<BRVerkaufsleiterListeItem> vListe =
               rBridge->getVerkaufsleiterListe();
   cout << "****** Verkaufsleiter ******" << endl;
   for (unsigned int i=0; i<vListe.size(); i++)
   {
      cout << vListe[i].showContent() << endl;
   }
   cout << endl;
}
```

9.7.3 Anmerkungen zum C++ Beispiel

Das Beispiel zeigt ein Bridge-Pattern zur Darstellung zweier verschiedener Datensichten. Dabei kapselt die die eigentliche Bridgeklasse (***BRView***) die beiden Views gemeinsame Zugriffstechnik auf das Datenmodell.

9.8 Java Beispiel

Das folgende Beispiel zeigt den Einsatz des Bridge-Patterns zur Entkopplung von Details der Implementation. Die Anwendung besteht aus einer Model-Klasse, welche die Daten hält (hier ***BRModel***) – in diesem Fall eine Preisliste. Es gibt zu diesem Model allerdings mehr als eine Sicht (View). Die Sicht des Verkäufers, dem es nicht gestattet ist, Preisnachlässe zu gewähren und die Sicht des Verkaufsleiters, der Preisnachlässe bis zu einem rechnerischen Minimalverkaufspreis einräumen kann.
Es wäre natürlich möglich, dies auch ohne die Bridge zu implementieren, dann hätten aber alle View Klassen Kenntnis über den Aufbau (d.h. die Datentypen) des Models. Bei einer Änderung des Models müssten ggf. alle drei View Klassen korrigiert werden, durch den Einsatz der Bridge ist dies nur in der Bridgeklasse (***BRBridge***) nötig.

9.8.1 Java Klassendiagramm

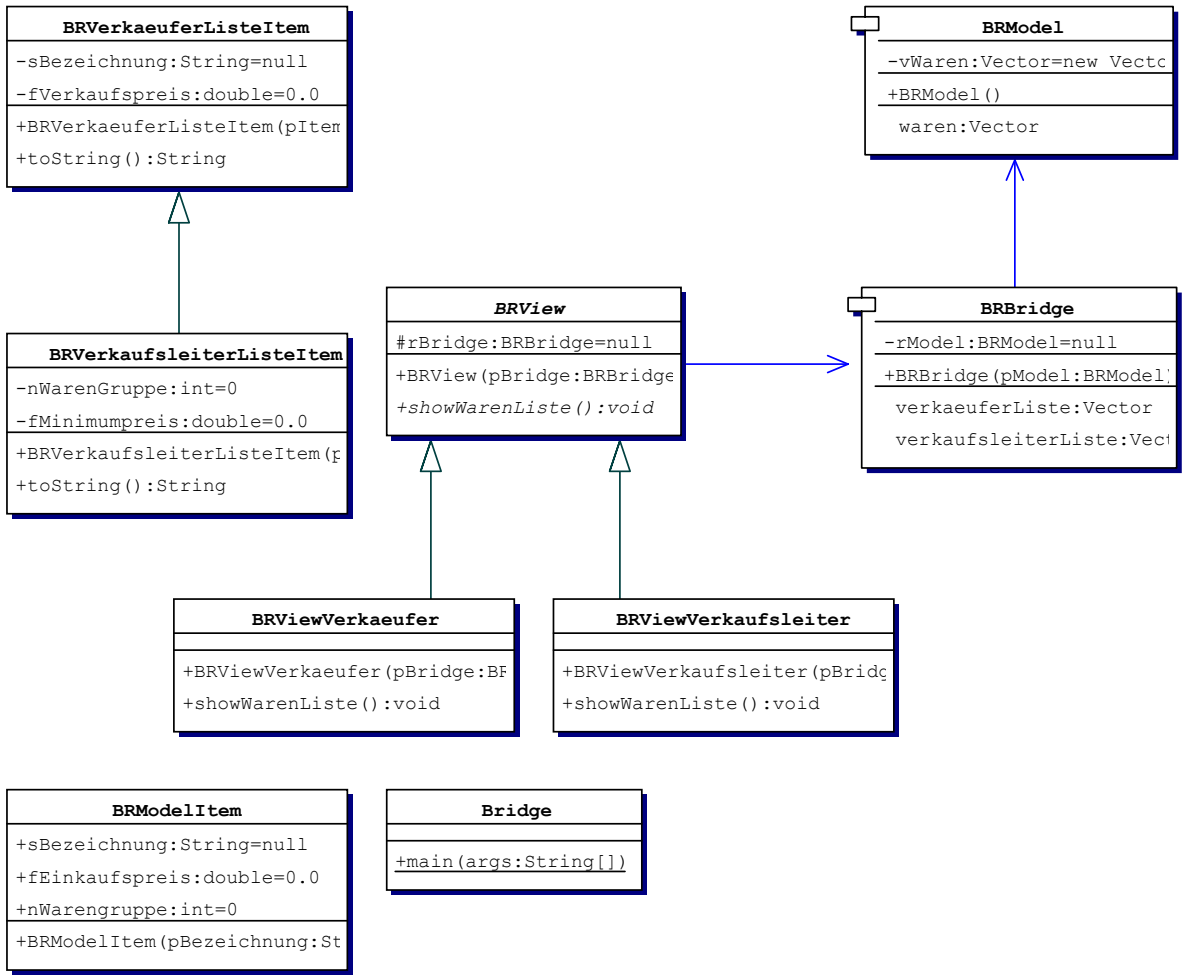

Abbildung 9-2 Java Klassendiagramm Bridge

9.8.2 Java Beispielprogramm

```
//=============================================================================
/** Java Bridge Design Pattern - Testprogramm
 *  Created on 15.11.2003
 */
```

```
//============================================================================
public class Bridge
{
   //----------------------------------------------------------------------
   // Hauptprogramm
   //----------------------------------------------------------------------
   public static void main(String[] args)
   {
      BRModel              rModel  = new BRModel();
      BRBridge             rBridge = new BRBridge(rModel);
      BRViewVerkaeufer     rView1  = new BRViewVerkaeufer(rBridge);
      BRViewVerkaufsleiter rView2  = new BRViewVerkaufsleiter(rBridge);

      rView1.showWarenListe();
      rView2.showWarenListe();
   }
}
```

```
//============================================================================
/** Java Bridge Design Pattern - einzelne Ware
 *  Created on 15.11.2003
 */
//============================================================================
public class BRModelItem
{
   public String sBezeichnung   = null;
   public double fEinkaufspreis = 0.0;
   public int    nWarengruppe   = 0;

   //----------------------------------------------------------------------
   /**
    * Constructor
    */
   //----------------------------------------------------------------------
   public BRModelItem(String pBezeichnung, double pEinkaufspreis,
                      int pWarengruppe)
   {
      super();
      sBezeichnung   = new String (pBezeichnung);
      fEinkaufspreis = pEinkaufspreis;
      nWarengruppe   = pWarengruppe;
   }
}
```

```
//============================================================================
/** Java Bridge Design Pattern - Datenmodell - enthält diverse Waren
 *  Created on 15.11.2003
 */
//============================================================================
import java.util.Vector;

public class BRModel
{
   private Vector vWaren = new Vector();

   //----------------------------------------------------------------------
   /**
    * Standard-Constructor
    */
   //----------------------------------------------------------------------
   public BRModel()
```

```
   {
      super();
      vWaren.add(new BRModelItem("Lampe",   63.45, 0));
      vWaren.add(new BRModelItem("Sessel", 110.95, 1));
      vWaren.add(new BRModelItem("Sofa",   221.15, 1));
      vWaren.add(new BRModelItem("Tisch",  177.15, 3));
      vWaren.add(new BRModelItem("Stuhl",   58.55, 3));
   }

   //------------------------------------------------------------------------
   /**
    * @return Vector, bestehend aus BRModelItems
    */
   //------------------------------------------------------------------------
   public Vector getWaren()
   {
      return vWaren;
   }
}
```

```
//============================================================================
/** Java Bridge Design Pattern - einzelne Ware, Sicht des Verkaufsleiters
 *  Created on 15.11.2003
 */
//============================================================================
import java.text.DecimalFormat;

public class BRVerkaeuferListeItem
{
   private String sBezeichnung   = null;
   private double fVerkaufspreis = 0.0;

   //------------------------------------------------------------------------
   /**
    * Constructor
    */
   //------------------------------------------------------------------------
   public BRVerkaeuferListeItem(BRModelItem pItem)
   {
      super();
      sBezeichnung   = new String (pItem.sBezeichnung);
      fVerkaufspreis = pItem.fEinkaufspreis + pItem.fEinkaufspreis * 1.2;
      fVerkaufspreis = fVerkaufspreis * 1.16;
   }

   //------------------------------------------------------------------------
   /**
    * toString
    */
   //------------------------------------------------------------------------
   public String toString()
   {
      DecimalFormat df = new DecimalFormat("###,###,##0.00");
      StringBuffer  sb = new StringBuffer(80);

      sb.append(sBezeichnung);
      for (int i=sBezeichnung.length(); i<30; i++)
      {
         sb.append(".");
      }
      sb.append(": ");
      sb.append(df.format (fVerkaufspreis));
```

```
      sb.append(" €");

      return sb.toString();
   }
}
```

```
//============================================================================
/** Java Bridge Design Pattern - einzelne Ware, Sicht des Verkäufers
 *  Created on 15.11.2003
 */
//============================================================================
import java.text.DecimalFormat;

public class BRVerkaufsleiterListeItem extends BRVerkaeuferListeItem
{
   private int    nWarenGruppe  = 0;
   private double fMinimumpreis = 0.0;

   //-------------------------------------------------------------------------
   /**
    * Constructor
    */
   //-------------------------------------------------------------------------
   public BRVerkaufsleiterListeItem(BRModelItem pItem)
   {
      super(pItem);
      nWarenGruppe = pItem.nWarengruppe;
      switch (nWarenGruppe)
      {
         case 0:  fMinimumpreis = pItem.fEinkaufspreis +
                                  pItem.fEinkaufspreis * 1.2;
                  fMinimumpreis = fMinimumpreis * 1.16;
                  break;
         case 1:  fMinimumpreis = pItem.fEinkaufspreis +
                                  pItem.fEinkaufspreis * 1.0;
                  fMinimumpreis = fMinimumpreis * 1.16;
                  break;
         default: fMinimumpreis = pItem.fEinkaufspreis +
                                  pItem.fEinkaufspreis * 0.5;
                  fMinimumpreis = fMinimumpreis * 1.16;
                  break;
      }
   }

   //-------------------------------------------------------------------------
   /**
    * toString
    */
   //-------------------------------------------------------------------------
   public String toString()
   {
      DecimalFormat df = new DecimalFormat("###,###,##0.00");
      StringBuffer  sb = new StringBuffer(80);

      sb.append(super.toString());
      sb.append("   Warengruppe: ");
      sb.append(nWarenGruppe);
      sb.append("   Mindestpreis: ");
      sb.append(df.format(fMinimumpreis));
      sb.append(" €");

      return sb.toString();
```

```
   }
}
```

```
//============================================================================
/** Java Bridge Design Pattern - Bridge, Brückenschlag zwischen den
 *  Ausgabevarianten
 *  Created on 15.11.2003
 */
//============================================================================
import java.util.Vector;

public class BRBridge
{
   private BRModel rModel = null;

   //--------------------------------------------------------------------------
   /**
    * Constructor
    */
   //--------------------------------------------------------------------------
   public BRBridge(BRModel pModel)
   {
      super();
      rModel = pModel;
   }

   //--------------------------------------------------------------------------
   /**
    * getVerkaeuferListe
    */
   //--------------------------------------------------------------------------
   public Vector getVerkaeuferListe()
   {
      Vector vListe = new Vector();
      Vector vModelListe = (Vector)rModel.getWaren();

      for (int i=0; i<vModelListe.size(); i++)
      {
         vListe.add (new BRVerkaeuferListeItem
                ((BRModelItem)vModelListe.elementAt(i)));
      }
      return vListe;
   }

   //--------------------------------------------------------------------------
   /**
    * getVerkaufsleiterListe
    */
   //--------------------------------------------------------------------------
   public Vector getVerkaufsleiterListe()
   {
      Vector vListe = new Vector();
      Vector vModelListe = (Vector)rModel.getWaren();

      for (int i=0; i<vModelListe.size(); i++)
      {
         vListe.add (new BRVerkaufsleiterListeItem
                ((BRModelItem)vModelListe.elementAt(i)));
      }
      return vListe;
   }
}
```

```
//============================================================================
/** Java Bridge Design Pattern - Basisklasse der Sichten
 *  Created on 15.11.2003
 */
//============================================================================
public abstract class BRView
{
   protected BRBridge rBridge = null;

   //------------------------------------------------------------------------
   /**
    * @param pBridge
    */
   //------------------------------------------------------------------------
   public BRView(BRBridge pBridge)
   {
      super();
      rBridge = pBridge;
   }

   //------------------------------------------------------------------------
   /**
    * showWarenListe
    */
   //------------------------------------------------------------------------
   public abstract void showWarenListe();
}
```

```
//============================================================================
/** Java Bridge Design Pattern - Verkäufersicht
 *  Created on 15.11.2003
 */
//============================================================================
import java.util.Vector;

public class BRViewVerkaeufer extends BRView
{
   //------------------------------------------------------------------------
   /**
    * Constructor
    */
   //------------------------------------------------------------------------
   public BRViewVerkaeufer(BRBridge pBridge)
   {
      super(pBridge);
   }

   //------------------------------------------------------------------------
   /**
    * showWarenListe
    */
   //------------------------------------------------------------------------
   public void showWarenListe()
   {
      Vector vListe = rBridge.getVerkaeuferListe();
      System.out.println("****** Verkäufer ******");
      for (int i=0; i<vListe.size(); i++)
      {
         System.out.println(vListe.elementAt(i).toString());
      }
      System.out.println("");
```

```
   }
}
```

```
//============================================================================
/** Java Bridge Design Pattern - Verkaufsleitersicht
 *  Created on 15.11.2003
 */
//============================================================================
import java.util.Vector;

public class BRViewVerkaufsleiter extends BRView
{
   //--------------------------------------------------------------------------
   /**
    * Constructor
    */
   //--------------------------------------------------------------------------
   public BRViewVerkaufsleiter(BRBridge pBridge)
   {
      super(pBridge);
   }

   //--------------------------------------------------------------------------
   /**
    * showWarenListe
    */
   //--------------------------------------------------------------------------
   public void showWarenListe()
   {
      Vector vListe = rBridge.getVerkaufsleiterListe();
      System.out.println("****** Verkaufsleiter ******");
      for (int i=0; i<vListe.size(); i++)
      {
         System.out.println(vListe.elementAt(i).toString());
      }
      System.out.println("");
   }
}
```

9.8.3 Anmerkungen zum Java Beispiel

Das Beispiel zeigt ein Bridge-Pattern zur Darstellung zweier verschiedener Datensichten. Dabei kapselt die die eigentliche Bridgeklasse (***BRView***) die beiden Views gemeinsame Zugriffstechnik auf das Datenmodell.

10. Proxy

Alternativnamen: Surrogat, Platzhalter, Stellvertreter, Ambassador, Botschafter
Mustergruppe: Strukturmuster

10.1 Anmerkungen

Das Proxy Designpattern ist im Prinzip sehr einfach, kann aber sehr viele Ausprägungen und Aufgaben übernehmen. Wie Adapter (⇨ Adapter) und Bridge (⇨ Bridge) handelt es sich beim Proxy um ein isoliertes Pattern, d.h. es bildet nicht Grundlage für komplexere Muster. Andererseits ähnelt es in einigen Einsatzgebieten dem Bridge Pattern.

10.2 Verwendungszweck

Die Aufgaben für die ein Proxy Designpattern geeignet ist, sind recht vielfältig:

- **Virtual Proxy Klassen** werden verwendet, um „teure“ (bezogen auf Zeit oder Platz) Objektinstanzen erst dann aufzubauen, wenn sie wirklich benötigt werden. So können beispielsweise Anwendungen mit sehr vielen Bildschirmmasken, die vermutlich vom Anwender nicht alle aufgerufen werden, diese erst auf Anforderung (statt wie sonst zum Start der Anwendung) aufbauen und dann im Hintergrund halten (ausblenden) wenn sie nicht benötigt werden. Somit wird die notwendige Startzeit der Anwendung scheinbar verkürzt, weil sie auf mehrere Teilschritte verteilt wird. Ein typisches auf Speicherplatz bezogenes Beispiel ist die Verwendung von Platzhaltern anstelle von aufwendigen Graphiken in Textverarbeitungsprogrammen (diese werden dann als leerer, durchkreuzter Rahmen dargestellt). Diese Form des Proxy-Patterns ähnelt ggf. dem Adapter-Pattern.
  Ein typischer Anwendungsfall von **Virtual Proxy Klassen** sind z.B. Vorschaudateien[10] welche die in einem Pfad vorhandenen Bilddateien als kleine Vorschaubilder anzeigen. Diese Programme halten natürlich nicht alle Graphikdateien gleichzeitig im Speicher.
- **Remote Proxy Klassen** dienen als Stellvertreter für einen fremden Adressraum. Sie sind häufig in Client-Server-Anwendungen zu finden. Dort dienen die **Remote Proxy Klassen** zur Kapselung der technischen RMI[11]-Anbindung (meist über TCP/IP[12]). Diese Form des Proxy-Patterns ähnelt dem Bridge-Pattern.
- **Security Proxy Klassen** sind die Form von Proxy-Klassen, die man heutzutage mit dem Begriff Proxy am ehesten in Verbindung bringt. Sie realisieren Zugriffs- bzw. Berechtigungskontrolle auf ein Objekt, um dieses vor absichtlichem oder versehentlichem Missbrauch zu schützen. Eine Klasse für den Datenbankzugriff z.B. kann alle Elemente des Zugriffs implementieren (Anlegen, Anzeigen, Ändern und Löschen des Datensatzes). Diese Implementation erfolgt nur einmal und umfassend. Die Berechtigung der unterschiedlichen Benutzertypen (wie ***Administrator*** und Besucher) werden

[10] sogenannte Thumbnails
[11] RMI = Remote Method Invocation – Entfernter Methodenaufruf
[12] TCP/IP = Transmission Control Protocol / Internet Protocol

durch eine Instanz vom Typ ***Adminstrator*** Proxy bzw. ***Visitor*** Proxy gesichert. Beide Proxy-Klassen besitzen das gleiche Interface (d.h. je eine Methode für Anlegen, Anzeigen, Ändern und Löschen eines Datensatzes). Aber nur die ***Administrator*** Proxy reicht alle Befehle an die eigentliche Implementierung durch. Die ***Visitor*** Proxy Klasse reicht lediglich den Befehl „Anzeigen“ weiter und hat hinter den anderen befehlen keinerlei Funktionalität.

10.3 Problemstellungen

Wie kann man kostenintensive Prozesse so gestalten, dass die Kosten nur bei Bedarf anfallen?
Wie kann eine technische notwendige Implementation wie z.B. ein Protokoll verborgen und austauschbar gehalten werden?
Wie kann man unterschiedliche Benutzerberechtigungen implementieren ohne den Code mehrfach schreiben zu müssen?

10.4 Lösung

Beim **Virtual Proxy Pattern** entscheidet ein Stellvertreterobjekt darüber, in welcher Form die Klasse nach außen repräsentiert. Im Standardfall ist dies eine vereinfachte Form. Erst auf Anforderung wird die volle Funktionalität bereitgestellt.
Beim **Remote Proxy Pattern** werden die Zugriffe an ein Stellvertreterobjekt weitergereicht, welches die technische Umsetzung des Fernzugriffs gekapselt hat.
Beim **Security Proxy Pattern** wird dem eigentlichen Objekt die Instanz eines Stellvertreterobjektes mit ggf. eingeschränkter Funktionalität vorangestellt.

10.5 Vorteile

Der Vorteil des **Virtual Proxy Patterns** ist offensichtlich und besteht in der Verteilung zeitintensiver Prozesse auf diverse Teilschritte, von denen ggf. einige gar nicht benötigt werden, wodurch insgesamt Zeit gespart wird. Die Platz sparende Variante belegt den Speicherplatz für große Objekte erst dann wenn deren Funktionalität wirklich benötigt wird.
Die Verwendung eines **Remote Proxys** bietet den Vorteil der Entkopplung von Applikationslogik und technischen Belangen (wie z.B. einem Kommunikationsprotokoll). Dieses kann leichter ausgetauscht werden, da sich die technischen Belange in einer oder wenigen Klassen kumulieren anstatt über die gesamte Applikation verteilt zu sein.
Das **Security Proxy** verhindert, dass zu konfigurierendes Verhalten sich in der eigentlichen Implementation niederschlägt. Dies hat den Vorteil, dass die Applikationslogik für alle Anwender (gleich welcher Berechtigungsstufe) identisch ist.

10.6 Nachteile

Die Verwendung des **Remote Proxy Patterns** ist nur sinnvoll, wenn es sich um eine begrenzte Anzahl an betroffenen Klassen handelt, ansonsten ist abzuwägen, ob nicht die Verwendung eines anderen Patterns sinnvoller ist.
Der einzige sonstige Nachteil ist der Umstand, dass man eine höhere Anzahl an Klassen in Kauf nehmen muss.

10.7 C++ Beispiel

Das folgende Beispiel zeigt einen einfachen **Security Proxy**. Sowohl ein Anwender mit ***Visitor*** Status als der ***Administrator*** erhalten Objekte mit identischer

Schnittstelle zum Front-End. Die Prüfung der rechte erfolgt im Proxy-Objekt, nicht in der Benutzungsoberfäche.

10.7.1 C++ Klassendiagramm

Abbildung 10-1 C++ Klassendiagramm Proxy

10.7.2 C++ Beispielprogramm

```
//=============================================================================
// C++ Proxy Designpattern - Proxy.cpp
//=============================================================================
#include "stdafx.h"

#include <iostream>
#include "PXDatalist.h"
#include "PXAdministratorProxy.h"
#include "PXVisitorProxy.h"

int _tmain(int argc, _TCHAR* argv[])
{
   // Testdaten erzeugen
   PXDatalist rData;
   rData.add(PXDatalistItem("Donald",   "Duck", 1));
   rData.add(PXDatalistItem("Dagobert", "Duck", 2));
   rData.add(PXDatalistItem("Micky",    "Maus", 3));
   rData.add(PXDatalistItem("Minnie",   "Maus", 4));

   // Visitor-Proxy für Datenzugriff erzeugen
```

```
   PXVisitorProxy rVisitor(&rData);
   cout << "--- VISITOR 1 ---" << endl;
   rVisitor.anzeigen();
   rVisitor.anlegen("Gustav", "Gans", 5);
   rVisitor.aendern(0, "Daisy", "Duck", 1);
   rVisitor.loeschen(2);
   cout << "\n--- VISITOR 2 ---" << endl;
   rVisitor.anzeigen();

   // Administrator-Proxy für Datenzugriff erzeugen
   PXAdministratorProxy rAdmin(&rData);
   cout << "\n--- ADMINISTRATOR 1 ---" << endl;
   rAdmin.anzeigen();
   rAdmin.anlegen("Gustav", "Gans", 5);
   rAdmin.aendern(0, "Daisy", "Duck", 1);
   rAdmin.loeschen(2);
   cout << "\n--- ADMINISTRATOR 2 ---" << endl;
   rAdmin.anzeigen();
      return 0;
}
```

```
//============================================================================
// C++ Proxy Designpattern - PXDatalistItem.h
//============================================================================
#ifndef _PXDATALISTITEM_H_
#define _PXDATALISTITEM_H_

#include <string>

using namespace std;

class PXDatalistItem
{
   public:
      PXDatalistItem(string pVorname, string pNachname, int pKundenNr);

      int    getKundenNr(void);
      string getNachname(void);
      string getVorname(void);
      void   setKundenNr(int pKundenNr);
      void   setNachname(string pNachname);
      void   setVorname(string pVorname);

   private:
      string sVorname;
      string sNachname;
      int    nKundenNr;
};

#endif
```

```
//============================================================================
// C++ Proxy Designpattern - PXDatalistItem.cpp
//============================================================================
#include "stdafx.h"
#include "PXDatalistItem.h"

using namespace std;

//----------------------------------------------------------------------
PXDatalistItem::PXDatalistItem(string pVorname, string pNachname,
```

```
                                int pKundenNr)
{
   setVorname(pVorname);
   setNachname(pNachname);
   setKundenNr(pKundenNr);
}

//---------------------------------------------------------------------------
int PXDatalistItem::getKundenNr(void)
{
   return nKundenNr;
}

//---------------------------------------------------------------------------
string PXDatalistItem::getNachname(void)
{
   return sNachname;
}

//---------------------------------------------------------------------------
string PXDatalistItem::getVorname(void)
{
   return sVorname;
}

//---------------------------------------------------------------------------
void PXDatalistItem::setKundenNr(int pKundenNr)
{
   nKundenNr = pKundenNr;
}

//---------------------------------------------------------------------------
void PXDatalistItem::setNachname(string pNachname)
{
   sNachname = pNachname;
}

//---------------------------------------------------------------------------
void PXDatalistItem::setVorname(string pVorname)
{
   sVorname = pVorname;
}
```

```
//===========================================================================
// C++ Proxy Designpattern - PXDatalist.h
//===========================================================================
#ifndef _PXDATALIST_H_
#define _PXDATALIST_H_

#include <vector>
#include "PXDatalistItem.h"

using namespace std;

class PXDatalist
{
   public:
      PXDatalist(void);

      void add    (PXDatalistItem pItem);
      void remove (int pIdx);
      PXDatalistItem &get (int pIdx);
```

```
      size_t size ();

   private:
      vector<PXDatalistItem> vList;
};

#endif
```

```
//============================================================================
// C++ Proxy Designpattern - PXDatalist.cpp
//============================================================================
#include "stdafx.h"
#include "PXDatalist.h"
#include <assert.h>
using namespace std;

//----------------------------------------------------------------------------
PXDatalist::PXDatalist(void)
{
}

//----------------------------------------------------------------------------
void PXDatalist::add (PXDatalistItem pItem)
{
   vList.push_back(pItem);
}

//----------------------------------------------------------------------------
void PXDatalist::remove (int pIdx)
{
   unsigned int u = (unsigned)pIdx;
   if (u >= 0 && u < vList.size())
   {
      vector<PXDatalistItem>::iterator iList = vList.begin()+u;
      vList.erase(iList);
   }
}

//----------------------------------------------------------------------------
PXDatalistItem &PXDatalist::get (int pIdx)
{
   unsigned int u = (unsigned)pIdx;
   if (u >= 0 && u < vList.size())
   {
      return vList[u];
   }
   assert(0);
}

//----------------------------------------------------------------------------
size_t PXDatalist::size ()
{
   return vList.size();
}
```

```
//============================================================================
// C++ Proxy Designpattern - IPXDatalistProxy.h
//============================================================================
#ifndef _IPXDATALISTPROXY_H_
#define _IPXDATALISTPROXY_H_
```

```
#include <string>

using namespace std;

class IPXDatalistProxy
{
   public:
      virtual void anlegen  (string pVorname, string pNachname,
                             int pKdNr) = NULL;
      virtual void aendern  (int pIdx, string pVorname, string pNachname,
                             int pKdNr) = NULL;
      virtual void loeschen (int pIdx) = NULL;
      virtual void anzeigen (int pIdx) = NULL;
      virtual void anzeigen (void) = NULL;
};

#endif
```

```
//============================================================================
// C++ Proxy Designpattern - PXVisitorProxy.h
//============================================================================
#ifndef _PXVISITORPROXY_H_
#define _PXVISITORPROXY_H_

#include <string>
#include "IPXDatalistProxy.h"
#include "PXDatalist.h"

using namespace std;

class PXVisitorProxy : public IPXDatalistProxy
{
   public:
      PXVisitorProxy(PXDatalist *pList);

      void anlegen  (string pVorname, string pNachname, int pKdNr);
      void aendern  (int pIdx, string pVorname, string pNachname,
                     int pKdNr);
      void loeschen (int pIdx);
      void anzeigen (int pIdx);
      void anzeigen (void);

   private:
         PXDatalist *rList;
};

#endif
```

```
//============================================================================
// C++ Proxy Designpattern - PXVisitorProxy.cpp
//============================================================================
#include "stdafx.h"
#include "PXVisitorProxy.h"
#include <iostream>

using namespace std;

//----------------------------------------------------------------------------
PXVisitorProxy::PXVisitorProxy(PXDatalist *pList)
{
   rList = pList;
```

```
}

//------------------------------------------------------------------------
void PXVisitorProxy::anlegen (string pVorname, string pNachname, int pKdNr)
{
   cout << "Sie haben keine Berechtigung, um Daten anzulegen" << endl;
}

//------------------------------------------------------------------------
void PXVisitorProxy::aendern (int pIdx, string pVorname, string pNachname,
                              int pKdNr)
{
   cout << "Sie haben keine Berechtigung, um Daten zu ändern" << endl;
}

//------------------------------------------------------------------------
void PXVisitorProxy::loeschen (int pIdx)
{
   cout << "Sie haben keine Berechtigung, um Daten zu löschen" << endl;
}

//------------------------------------------------------------------------
void PXVisitorProxy::anzeigen (int pIdx)
{
   cout << rList->get(pIdx).getNachname() <<  ", "
        << rList->get(pIdx).getVorname() << endl;
}

//------------------------------------------------------------------------
void PXVisitorProxy::anzeigen (void)
{
   for (unsigned int i=0; i<rList->size(); i++)
   {
      anzeigen(i);
   }
}
```

```
//========================================================================
// C++ Proxy Designpattern - PXAdministratorProxy.h
//========================================================================
#ifndef _PXADMINISTRATORPROXY_H_
#define _PXADMINISTRATORPROXY_H_

#include <string>
#include "IPXDatalistProxy.h"
#include "PXDatalist.h"

using namespace std;

class PXAdministratorProxy : public IPXDatalistProxy
{
   public:
      PXAdministratorProxy(PXDatalist *pList);

      void anlegen  (string pVorname, string pNachname, int pKdNr);
      void aendern  (int pIdx, string pVorname, string pNachname,
                     int pKdNr);
      void loeschen (int pIdx);
      void anzeigen (int pIdx);
      void anzeigen (void);

   private:
```

```
         PXDatalist *rList;
};

#endif
```

```
//============================================================================
// C++ Proxy Designpattern - PXAdministratorProxy.cpp
//============================================================================
#include "stdafx.h"
#include "PXAdministratorProxy.h"
#include <iostream>

using namespace std;

//----------------------------------------------------------------------------
PXAdministratorProxy::PXAdministratorProxy(PXDatalist *pList)
{
   rList = pList;
}

//----------------------------------------------------------------------------
void PXAdministratorProxy::anlegen (string pVorname, string pNachname,
                                    int pKdNr)
{
   PXDatalistItem rTemp(pVorname, pNachname, pKdNr);
   rList->add(rTemp);
}

//----------------------------------------------------------------------------
void PXAdministratorProxy::aendern (int pIdx, string pVorname,
                                    string pNachname, int pKdNr)
{
   rList->get(pIdx).setVorname(pVorname);
   rList->get(pIdx).setNachname(pNachname);
   rList->get(pIdx).setKundenNr(pKdNr);
}

//----------------------------------------------------------------------------
void PXAdministratorProxy::loeschen (int pIdx)
{
   rList->remove(pIdx);
}

//----------------------------------------------------------------------------
void PXAdministratorProxy::anzeigen (int pIdx)
{
   cout << rList->get(pIdx).getNachname() << ", "
        << rList->get(pIdx).getVorname()
        << " (" << rList->get(pIdx).getKundenNr() << ")" << endl;
}

//----------------------------------------------------------------------------
void PXAdministratorProxy::anzeigen (void)
{
   for (unsigned int i=0; i<rList->size(); i++)
   {
      anzeigen(i);
   }
}
```

10.8 Java Beispiel

Das folgende Beispiel zeigt einen einfachen **Security Proxy**. Sowohl ein Anwender mit ***Visitor*** Status als der Administrator erhalten Objekte mit identischer Schnittstelle zum Front-End. Die Prüfung der rechte erfolgt im Proxy-Objekt, nicht in der Benutzungsoberfäche.

10.8.1 Java Klassendiagramm

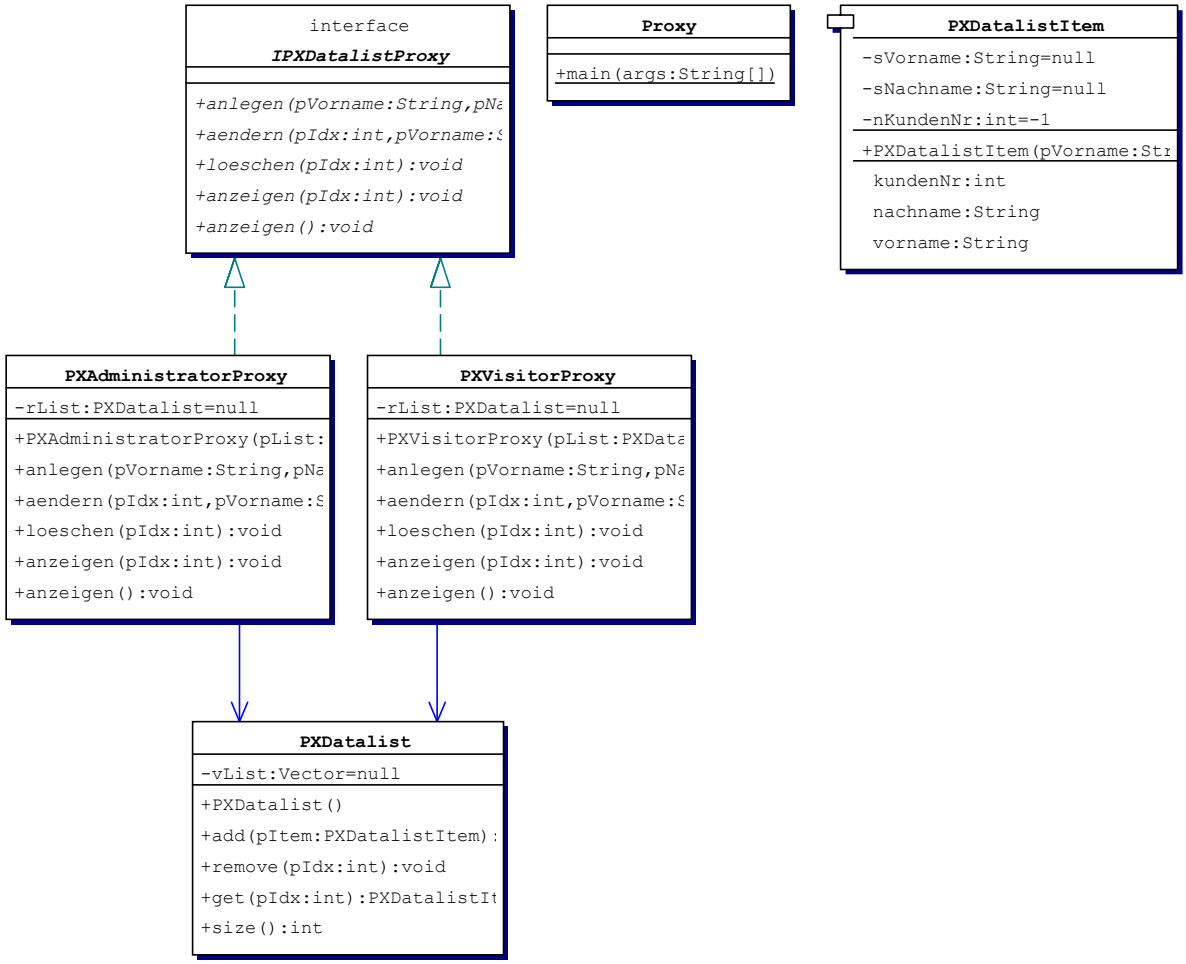

Abbildung 10-2 Java Klassendiagramm Proxy

10.8.2 Java Beispielprogramm

```
//======================================================================
/** Java Proxy Design Pattern - Testprogramm
 */
//======================================================================
public class Proxy
{
   public static void main(String[] args)
   {
      // Testdaten erzeugen
      PXDatalist rData = new PXDatalist();
      rData.add(new PXDatalistItem("Donald",   "Duck", 1));
      rData.add(new PXDatalistItem("Dagobert", "Duck", 2));
      rData.add(new PXDatalistItem("Micky",    "Maus", 3));
      rData.add(new PXDatalistItem("Minnie",   "Maus", 4));
```

```
      // Visitor-Proxy für Datenzugriff erzeugen
      PXVisitorProxy rVisitor = new PXVisitorProxy(rData);
      System.out.println("--- VISITOR 1 ---");
      rVisitor.anzeigen();
      rVisitor.anlegen("Gustav", "Gans", 5);
      rVisitor.aendern(0, "Daisy", "Duck", 1);
      rVisitor.loeschen(2);
      System.out.println("\n--- VISITOR 2 ---");
      rVisitor.anzeigen();

      // Administrator-Proxy für Datenzugriff erzeugen
      PXAdministratorProxy rAdmin = new PXAdministratorProxy(rData);
      System.out.println("\n--- ADMINSITRATOR 1 ---");
      rAdmin.anzeigen();
      rAdmin.anlegen("Gustav", "Gans", 5);
      rAdmin.aendern(0, "Daisy", "Duck", 1);
      rAdmin.loeschen(2);
      System.out.println("\n--- ADMINSITRATOR 2 ---");
      rAdmin.anzeigen();
   }
}
```

```
//============================================================================
/** Java Proxy Design Pattern - Interface für Proxy
 */
//============================================================================
public interface IPXDatalistProxy
{
   public void anlegen (String pVorname, String pNachname, int pKdNr);
   public void aendern (int pIdx, String pVorname, String pNachname,
                        int pKdNr);
   public void loeschen (int pIdx);
   public void anzeigen (int pIdx);
   public void anzeigen ();
}
```

```
//============================================================================
/** Java Proxy Design Pattern - einzelner Eintrag der Datenliste
 */
//============================================================================
public class PXDatalistItem
{
   private String sVorname  = null;
   private String sNachname = null;
   private int    nKundenNr = -1;

   //-------------------------------------------------------------------------
   public PXDatalistItem(String pVorname, String pNachname, int pKundenNr)
   {
      super();
      setVorname(pVorname);
      setNachname(pNachname);
      setKundenNr(pKundenNr);
   }

   //-------------------------------------------------------------------------
   public int getKundenNr()
   {
      return nKundenNr;
   }
```

```
   //-------------------------------------------------------------------------
   public String getNachname()
   {
      return sNachname;
   }

   //-------------------------------------------------------------------------
   public String getVorname()
   {
      return sVorname;
   }

   //-------------------------------------------------------------------------
   public void setKundenNr(int pKundenNr)
   {
      nKundenNr = pKundenNr;
   }

   //-------------------------------------------------------------------------
   public void setNachname(String pNachname)
   {
      sNachname = pNachname;
   }

   //-------------------------------------------------------------------------
   public void setVorname(String pVorname)
   {
      sVorname = pVorname;
   }
}
```

```
//=============================================================================
/** Java Proxy Design Pattern - Datenliste
 */
//=============================================================================
import java.util.Vector;

public class PXDatalist
{
   private Vector vList = null;

   //-------------------------------------------------------------------------
   public PXDatalist()
   {
      super();
      vList = new Vector();
   }

   //-------------------------------------------------------------------------
   public void add (PXDatalistItem pItem)
   {
      vList.add(pItem);
   }

   //-------------------------------------------------------------------------
   public void remove (int pIdx)
   {
      if (pIdx >= 0 && pIdx < vList.size())
      {
         vList.remove(pIdx);
      }
```

```
   }

   //-------------------------------------------------------------------------
   public PXDatalistItem get (int pIdx)
   {
      PXDatalistItem rRC = null;
      if (pIdx >= 0 && pIdx < vList.size())
      {
         rRC = (PXDatalistItem)vList.elementAt(pIdx);
      }
      return rRC;
   }

   //-------------------------------------------------------------------------
   public int size ()
   {
      return vList.size();
   }
}
```

```
//=============================================================================
/** Java Proxy Design Pattern - Visitorproxy
 */
//=============================================================================
public class PXVisitorProxy implements IPXDatalistProxy
{
   private PXDatalist rList = null;

   //-------------------------------------------------------------------------
   public PXVisitorProxy(PXDatalist pList)
   {
      super();
      rList = pList;
   }

   //-------------------------------------------------------------------------
   public void anlegen (String pVorname, String pNachname, int pKdNr)
   {
      System.out.println
                     ("Sie haben keine Berechtigung, um Daten anzulegen");
   }

   //-------------------------------------------------------------------------
   public void aendern (int pIdx, String pVorname, String pNachname,
                        int pKdNr)
   {
      System.out.println
                     ("Sie haben keine Berechtigung, um Daten zu ändern");
   }

   //-------------------------------------------------------------------------
   public void loeschen (int pIdx)
   {
      System.out.println
                    ("Sie haben keine Berechtigung, um Daten zu löschen");
   }

   //-------------------------------------------------------------------------
   public void anzeigen (int pIdx)
   {
      PXDatalistItem rItem = rList.get(pIdx);
      if (rItem != null)
```

```
      {
         System.out.println(rItem.getNachname() + ", "
                            + rItem.getVorname());
      }
   }

   //-------------------------------------------------------------------------
   public void anzeigen ()
   {
      for (int i=0; i<rList.size(); i++)
      {
         anzeigen(i);
      }
   }
}
```

```
//=============================================================================
/** Java Proxy Design Pattern - Administratorproxy
 */
//=============================================================================
public class PXAdministratorProxy implements IPXDatalistProxy
{
   private PXDatalist rList = null;

   //-------------------------------------------------------------------------
   public PXAdministratorProxy(PXDatalist pList)
   {
      super();
      rList = pList;
   }

   //-------------------------------------------------------------------------
   public void anlegen (String pVorname, String pNachname, int pKdNr)
   {
      rList.add(new PXDatalistItem(pVorname, pNachname, pKdNr));
   }

   //-------------------------------------------------------------------------
   public void aendern (int pIdx, String pVorname, String pNachname,
                        int pKdNr)
   {
      PXDatalistItem rItem = rList.get(pIdx);
      if (rItem != null)
      {
         rItem.setVorname(pVorname);
         rItem.setNachname(pNachname);
         rItem.setKundenNr(pKdNr);
      }
   }

   //-------------------------------------------------------------------------
   public void loeschen (int pIdx)
   {
      rList.remove(pIdx);
   }

   //-------------------------------------------------------------------------
   public void anzeigen (int pIdx)
   {
      PXDatalistItem rItem = rList.get(pIdx);
      if (rItem != null)
      {
```

```
         System.out.println(rItem.getNachname() + ", " + rItem.getVorname()
                            + " (" + rItem.getKundenNr() + ")");
      }
   }

   //-------------------------------------------------------------------------
   public void anzeigen ()
   {
      for (int i=0; i<rList.size(); i++)
      {
         anzeigen(i);
      }
   }
}
```

11. Factory Method

Alternativnamen: Fabrikmethode, virtueller Konstruktor, virtual Constructor
Mustergruppe: Erzeugungsmuster

11.1 Anmerkungen

Unter dem allgemeinen Stichwort Factory werden zwei leicht verschiedene Patterns geführt, die Factory Method und die Abstract Factory.

11.2 Verwendungszweck

Das Factory Method Pattern wird verwendet um spezialisierte Instanzen aus einer Hierarchie polymorpher Klassen zu erzeugen, ohne dass deren Constructoren bzw. verschiedenen Ausprägungen der verwendenden Applikation bekannt sein müssen.
Typische Anwendungen für das Factory Method Designpattern sind z.B.:

- Objektorientierte Graphikprogramme, bei denen Graphiken aus Primitiven (Punkte, Linien, Rechtecke, Kreise etc.) zusammengesetzt werden. Alle diese Elemente erben von einer gemeinsamen Basisklasse „Graphikelement“.

Wie man sieht eröffnen sich über das Factory Method Pattern ähnliche Möglichkeiten wie über das Bridge Pattern (⇨ Bridge). Nur handelt es sich hier nicht um zwei Hierarchien mit unterschiedlichen Ausprägungen, sondern alle Elemente, die von einer Factory Method hergestellt werden, werden allein über ihre polymorphe Schnittstelle verwendet. Das Applikationsprogramm selbst weiß nach der Anforderung gar nicht mehr um welchen Typ es sich bei einer Objektinstanz handelt. dies ist beim Bridge Pattern anders, wo es eine Hierarchie unterschiedlicher Objekte (mit unterschiedlichen Schnittstellen) gibt, bei denen lediglich die Implementation verborgen wird.
Da eine Factory Method Klasse beliebig viele Objekte der Hierarchie erzeugen kann ist es üblicherweise unnötig mehr als eine Factory Method Instanz zu erzeugen. Die Factory Method wird daher zumeist unter Verwendung des Singleton Patterns (⇨ Singleton) erzeugt.

11.3 Problemstellungen

Wie können Instanzen von spezialisierten Klassen erzeugt werden, ohne dass das Applikationsprogramm alle möglichen spezialisierten Klassen kennen muss?
Wie kann die Menge spezialisierter Ableitungen erweitert werden ohne dass auch das Applikationsprogramm angepasst werden muss?
Wie kann man Klassenbibliotheken mit gemeinsamer Schnittstelle bereitstellen, ohne deren Implementierungen öffentlich zu machen?

11.4 Lösung

Das Applikationsprogramm selbst erzeugt keine Instanzen und ist somit von der Kenntnis der Vererbungshierarchie und den Konstruktionsdetails entlastet. Stattdessen übernimmt eine Factory Method Klasse die Erstellung von Instanzen gewünschten Typs auf Anfrage. Die Anfrage besteht üblicherweise aus einem String, einer Kennnummer oder (bei Java) einem Klassennamen.

11.5 Vorteile

Die Hierarchie kann problemlos erweitert werden. Kenntnisse über die Hierarchiestruktur (z.B. die Namen der Packages in Java) oder der Parameter für den Aufruf des gewünschten Constructors sind in der Factory Method Klasse konzentriert und nicht über das gesamte Applikationsprogramm verstreut.
Das Factory Method Pattern kann bei problemlos dergestalt erweitert werden, dass die Factory Method Klasse eine Liste konstruierbarer Objekte bereitstellt, so dass das Applikationsprogramm sogar diese Kenntnisse nicht besitzen muss.

11.6 Nachteile

Für C++ besteht die Einschränkung, dass die Konstruktion durch eine Factory Method Klasse eine polymorphe Klassenhierarchie voraussetzt. In Java ist dies durch die allen Instanzen gemeinsame Basisklasse „***Object***“ immer gegeben, allerdings sollte Sinnvollerweise eine höhere Vererbungsebene als Basiselement gewählt werden, da an der Klasse ***Object*** keinerlei gemeinsame, polymorphe Schnittstelle bereitsteht.

11.7 C++ Beispiel

Das folgende Beispiel zeigt einen typischen Anwendungsfall. Für eine graphische Oberfläche werden Elemente auf Anfrage in einer Factory Method erzeugt. Die Aufforderung ein Objekt zu erzeugen wird über eine Zeichenkette erteilt, die zumeist einer Konfigurationsdatei entnommen wird.

11.7.1 C++ Klassendiagramm

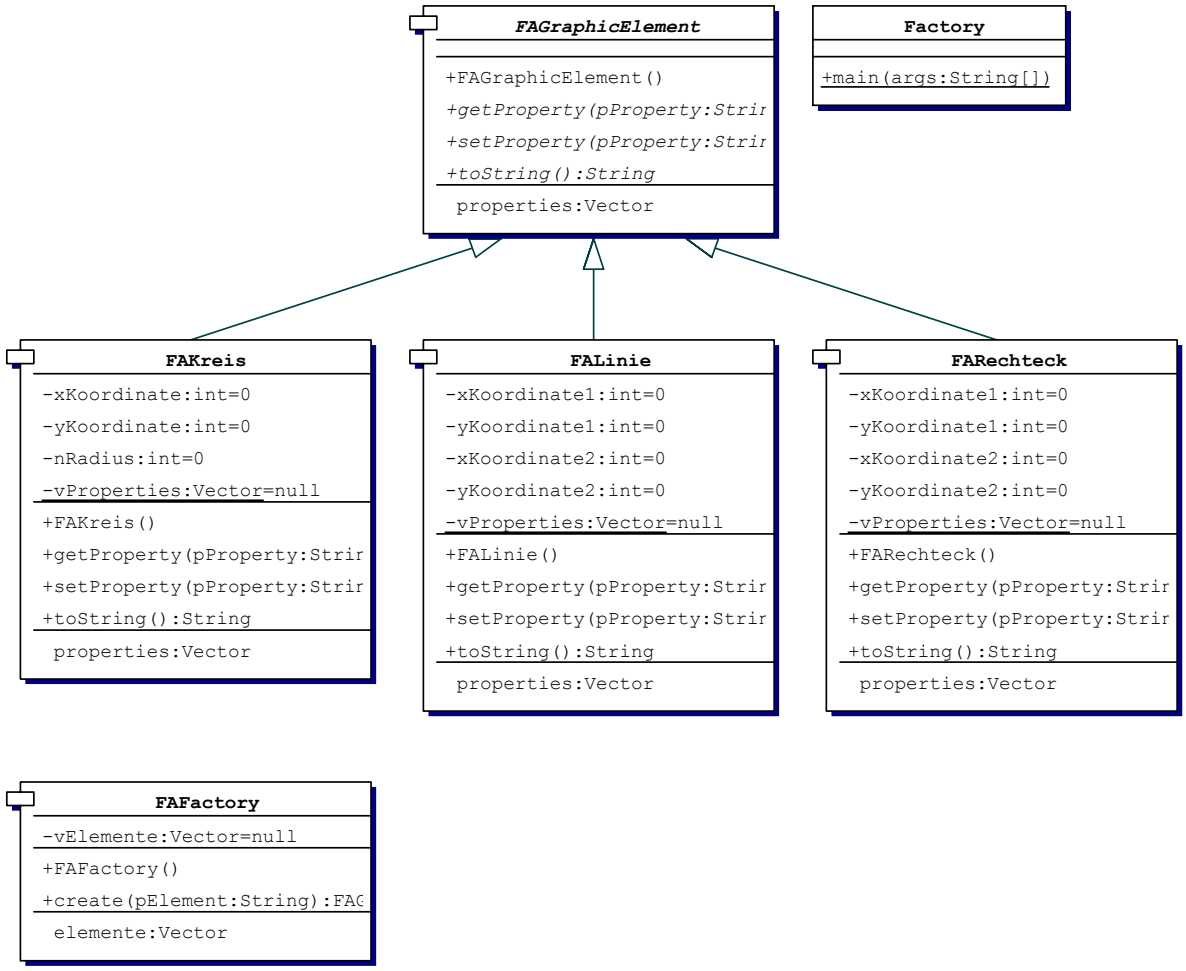

Abbildung 11-1 C++ Klassendiagramm Factory Method

11.7.2 C++ Beispielprogramm

```
//============================================================================
// C++ Factory Designpattern - Factory.cpp
//============================================================================
#include "stdafx.h"
#include "FAFactory.h"
#include "FAFactory.h"

#include <iostream>

int _tmain(int argc, _TCHAR* argv[])
{
   FAFactory                rFactory;
   vector<string>           vElemente = rFactory.getElemente();
   vector<FAGraphicElement*> vInstanzen;
```

```
   vector<string>          vProperties;
   string            sTemp;
   int               nTemp    = 111;
   FAGraphicElement* rElement = NULL;

   // Von jeder Art ein Element erstellen
   for (unsigned int i=0; i<vElemente.size(); i++)
   {
      sTemp = vElemente[i];
      vInstanzen.push_back(rFactory.create(sTemp));
   }

   // Inhalt des Instanzen-Vectors ausgeben
   cout << "\nAusgabe:\n" << endl;
   for (unsigned int i=0; i<vInstanzen.size(); i++)
   {
      cout << vInstanzen[i]->showContent() << endl;
   }

   // Erstes Element nehmen und die Properties generisch füllen
   rElement = vInstanzen[0];
   vProperties = rElement->getProperties();
   for (unsigned int i=0; i<vProperties.size(); i++)
   {
      char s[20] = "";
      sprintf(s, "%d", nTemp);
      rElement->setProperty(vProperties[i], s);
      nTemp = nTemp + 111;
   }

   // Inhalt des Instanzen-Vectors ausgeben
   cout << "\nAusgabe:\n" << endl;
   for (unsigned int i=0; i<vInstanzen.size(); i++)
   {
      cout << vInstanzen[i]->showContent() << endl;
   }

   // Instanzen freigeben
   cout << "\nFreigabe\n" << endl;
   for (unsigned int i=0; i<vInstanzen.size(); i++)
   {
      delete vInstanzen[i];
   }

   return 0;
}
```

```
//============================================================================
// C++ Factory Designpattern - FAFactory.h
//============================================================================
#ifndef _FAFACTORY_H_
#define _FAFACTORY_H_

#include <vector>
#include <string>
#include "FAGraphicElement.h"

using namespace std;

class FAFactory
{
   public:
```

```
      FAFactory(void);

      FAGraphicElement* create(string pElement);
      const vector<string> getElemente(void);

   private:
      vector<string> vElemente;
};

#endif
```

```
//============================================================================
// C++ Factory Designpattern - FAFactory.cpp
//============================================================================
#include "stdafx.h"
#include "FAFactory.h"
#include <iostream>

// Beim Erzeugen über dynamische Link-Libraries (DLLs)
// entfallen diese includes
#include "FAKreis.h"
#include "FARechteck.h"
#include "FALinie.h"

using namespace std;

FAFactory::FAFactory (void)
{
   // Dies würde man aus einer Konfigurationsdatei lesen
   vElemente.push_back ("Kreis");
   vElemente.push_back ("Rechteck");
   vElemente.push_back ("Linie");
}

// Das Erzeugen würde man über dynamische Link-Libraries (DLLs) machen
FAGraphicElement* FAFactory::create(string pElement)
{
   cout << "create: " << pElement << endl;
   FAGraphicElement* rRC = NULL;

   if (pElement == "Kreis")
   {
      rRC = new FAKreis();
   }
   else if (pElement == "Rechteck")
   {
      rRC = new FARechteck();
   }
   if (pElement == "Linie")
   {
      rRC = new FALinie();
   }
   return rRC;
}

//----------------------------------------------------------------------------
const vector<string> FAFactory::getElemente(void)
{
   return vElemente;
}
```

```
//============================================================================
// C++ Factory Designpattern - FAGraphicElement.h
//============================================================================
#ifndef _FAGRAPHICELEMENT_H_
#define _FAGRAPHICELEMENT_H_

#include <string>
#include <vector>

using namespace std;

class FAGraphicElement
{
   public:
      FAGraphicElement(void) {}

      virtual const vector<string> getProperties(void) = NULL;
      virtual string getProperty(string pProperty) = NULL;
      virtual void   setProperty(string pProperty, string pValue) = NULL;
      virtual string showContent(void) = NULL;
};

#endif
```

```
//============================================================================
// C++ Factory Designpattern - FALinie.h
//============================================================================
#ifndef _FALINIE_H_
#define _FALINIE_H_

#include <string>
#include <vector>
#include "FAGraphicElement.h"

using namespace std;

class FALinie : public FAGraphicElement
{
   public:
      FALinie(void);

      virtual const vector<string> getProperties(void);
      virtual string getProperty(string pProperty);
      virtual void setProperty(string pProperty, string pValue);
      virtual string showContent(void);

   private:
      int xKoordinate1;
      int yKoordinate1;
      int xKoordinate2;
      int yKoordinate2;
      static vector<string> vProperties;
};

#endif
```

```
//============================================================================
// C++ Factory Designpattern - FALinie.cpp
//============================================================================
#include "stdafx.h"
#include "FALinie.h"
```

```
using namespace std;

vector<string> FALinie::vProperties;

//--------------------------------------------------------------------------
FALinie::FALinie(void)
{
   if (vProperties.size() == 0)
   {
      vProperties.push_back ("xKoordinate1");
      vProperties.push_back ("yKoordinate1");
      vProperties.push_back ("xKoordinate2");
      vProperties.push_back ("yKoordinate2");
   }
   xKoordinate1 = 0;
   yKoordinate1 = 0;
   xKoordinate2 = 0;
   yKoordinate2 = 0;
}

//--------------------------------------------------------------------------
const vector<string> FALinie::getProperties(void)
{
   return vProperties;
}

//--------------------------------------------------------------------------
string FALinie::getProperty(string pProperty)
{
   string sRC;
   if (pProperty == "xKoordinate1")
   {
      sRC = xKoordinate1;
   }
   else if (pProperty == "yKoordinate1")
   {
      sRC = yKoordinate1;
   }
   if (pProperty == "xKoordinate2")
   {
      sRC = xKoordinate2;
   }
   else if (pProperty == "yKoordinate2")
   {
      sRC = yKoordinate2;
   }
   return sRC;
}

//--------------------------------------------------------------------------
void FALinie::setProperty(string pProperty, string pValue)
{
   if (pProperty == "xKoordinate1")
   {
      xKoordinate1 = atoi(pValue.c_str());
   }
   else if (pProperty == "yKoordinate1")
   {
      yKoordinate1 = atoi(pValue.c_str());
   }
   if (pProperty == "xKoordinate2")
```

```
   {
      xKoordinate2 = atoi(pValue.c_str());
   }
   else if (pProperty == "yKoordinate2")
   {
      yKoordinate2 = atoi(pValue.c_str());
   }
}

//---------------------------------------------------------------------
string FALinie::showContent(void)
{
   char sb[1000] = "";
   sprintf(sb, "Linie: %d, %d, %d, %d",
           xKoordinate1, yKoordinate1, xKoordinate2, yKoordinate2);
   return (string)sb;
}
```

```
//=====================================================================
// C++ Factory Designpattern - FARechteck.h
//=====================================================================
#ifndef _FARECHTECK_H_
#define _FARECHTECK_H_

#include <string>
#include <vector>
#include "FAGraphicElement.h"

using namespace std;

class FARechteck : public FAGraphicElement
{
   public:
      FARechteck(void);

      virtual const vector<string> getProperties(void);
      virtual string getProperty(string pProperty);
      virtual void setProperty(string pProperty, string pValue);
      virtual string showContent(void);

   private:
      int xKoordinate1;
      int yKoordinate1;
      int xKoordinate2;
      int yKoordinate2;
      static vector<string> vProperties;
};

#endif
```

```
//=====================================================================
// C++ Factory Designpattern - FARechteck.cpp
//=====================================================================
#include "stdafx.h"
#include "FARechteck.h"

using namespace std;

vector<string> FARechteck::vProperties;

//---------------------------------------------------------------------
```

```
FARechteck::FARechteck(void)
{
   if (vProperties.size() == 0)
   {
      vProperties.push_back ("xKoordinate1");
      vProperties.push_back ("yKoordinate1");
      vProperties.push_back ("xKoordinate2");
      vProperties.push_back ("yKoordinate2");
   }

   xKoordinate1 = 0;
   yKoordinate1 = 0;
   xKoordinate2 = 0;
   yKoordinate2 = 0;
}

//----------------------------------------------------------------------
const vector<string> FARechteck::getProperties(void)
{
   return vProperties;
}

//----------------------------------------------------------------------
string FARechteck::getProperty(string pProperty)
{
   string sRC;
   if (pProperty == "xKoordinate1")
   {
      sRC = xKoordinate1;
   }
   else if (pProperty == "yKoordinate1")
   {
      sRC = yKoordinate1;
   }
   if (pProperty == "xKoordinate2")
   {
      sRC = xKoordinate2;
   }
   else if (pProperty == "yKoordinate2")
   {
      sRC = yKoordinate2;
   }
   return sRC;
}

//----------------------------------------------------------------------
void FARechteck::setProperty(string pProperty, string pValue)
{
   if (pProperty == "xKoordinate1")
   {
      xKoordinate1 = atoi(pValue.c_str());
   }
   else if (pProperty == "yKoordinate1")
   {
      yKoordinate1 = atoi(pValue.c_str());
   }
   if (pProperty == "xKoordinate2")
   {
      xKoordinate2 = atoi(pValue.c_str());
   }
   else if (pProperty == "yKoordinate2")
   {
```

```
      yKoordinate2 = atoi(pValue.c_str());
   }
}

//-----------------------------------------------------------------------
string FARechteck::showContent(void)
{
   char sb[1000] = "";
   sprintf(sb, "Rechteck: %d, %d, %d, %d",
           xKoordinate1, yKoordinate1, xKoordinate2, yKoordinate2);
   return (string)sb;
}
```

```
//=======================================================================
// C++ Factory Designpattern - FAKreis.h
//=======================================================================
#ifndef _FAKREIS_H_
#define _FAKREIS_H_

#include <string>
#include <vector>
#include "FAGraphicElement.h"

using namespace std;

class FAKreis : public FAGraphicElement
{
   public:
      FAKreis(void);

      virtual const vector<string> getProperties(void);
      virtual string getProperty(string pProperty);
      virtual void setProperty(string pProperty, string pValue);
      virtual string showContent(void);

   private:
      int xKoordinate;
      int yKoordinate;
      int nRadius;
      static vector<string> vProperties;
};

#endif
```

```
//=======================================================================
// C++ Factory Designpattern - FAKreis.cpp
//=======================================================================
#include "stdafx.h"
#include "FAKreis.h"

using namespace std;

vector<string> FAKreis::vProperties;

//-----------------------------------------------------------------------
FAKreis::FAKreis(void)
{
   if (vProperties.size() == 0)
   {
      vProperties.push_back ("xKoordinate");
      vProperties.push_back ("yKoordinate");
```

```
      vProperties.push_back ("Radius");
   }
   xKoordinate = 0;
   yKoordinate = 0;
   nRadius     = 0;
}

//------------------------------------------------------------------------
const vector<string> FAKreis::getProperties()
{
   return vProperties;
}

//------------------------------------------------------------------------
string FAKreis::getProperty(string pProperty)
{
   string sRC;
   if (pProperty == "xKoordinate")
   {
      sRC = xKoordinate;
   }
   else if (pProperty == "yKoordinate")
   {
      sRC = yKoordinate;
   }
   if (pProperty == "Radius")
   {
      sRC = nRadius;
   }
   return sRC;
}

//------------------------------------------------------------------------
void FAKreis::setProperty(string pProperty, string pValue)
{
   if (pProperty == "xKoordinate")
   {
      xKoordinate = atoi(pValue.c_str());
   }
   else if (pProperty == "yKoordinate")
   {
      yKoordinate = atoi(pValue.c_str());
   }
   if (pProperty == "Radius")
   {
      nRadius = atoi(pValue.c_str());
   }
}

//------------------------------------------------------------------------
string FAKreis::showContent(void)
{
   char sb[1000] = "";
   sprintf(sb, "Kreis: %d, %d, %d",
           xKoordinate, yKoordinate, nRadius);
   return (string)sb;
}
```

11.7.3 Anmerkungen zum C++ Beispiel

Lädt man die in der Factory Method verfügbaren Objekttypen nicht über eine Konfiguration, sondern sammelt diese über einen generischen Mechanismus aus bereitgestellten Dynamischen Link Libraries (DLL), so erhält man den klassischen „Plugin“ Mechanismus.
Um Slicing-Phänomene zu vermeiden, ist die Verwendung von Pointern im Vector zwingend gegeben. Es ist darauf zu achten, dass der Speicherplatz auch wieder freigegeben wird.

11.8 Java Beispiel

Das folgende Beispiel zeigt einen typischen Anwendungsfall. Für eine graphische Oberfläche werden Elemente auf Anfrage in einer Factory Method erzeugt. Die Aufforderung ein Objekt zu erzeugen wird über eine Zeichenkette erteilt, die zumeist einer Konfigurationsdatei entnommen wird.

11.8.1 Java Klassendiagramm

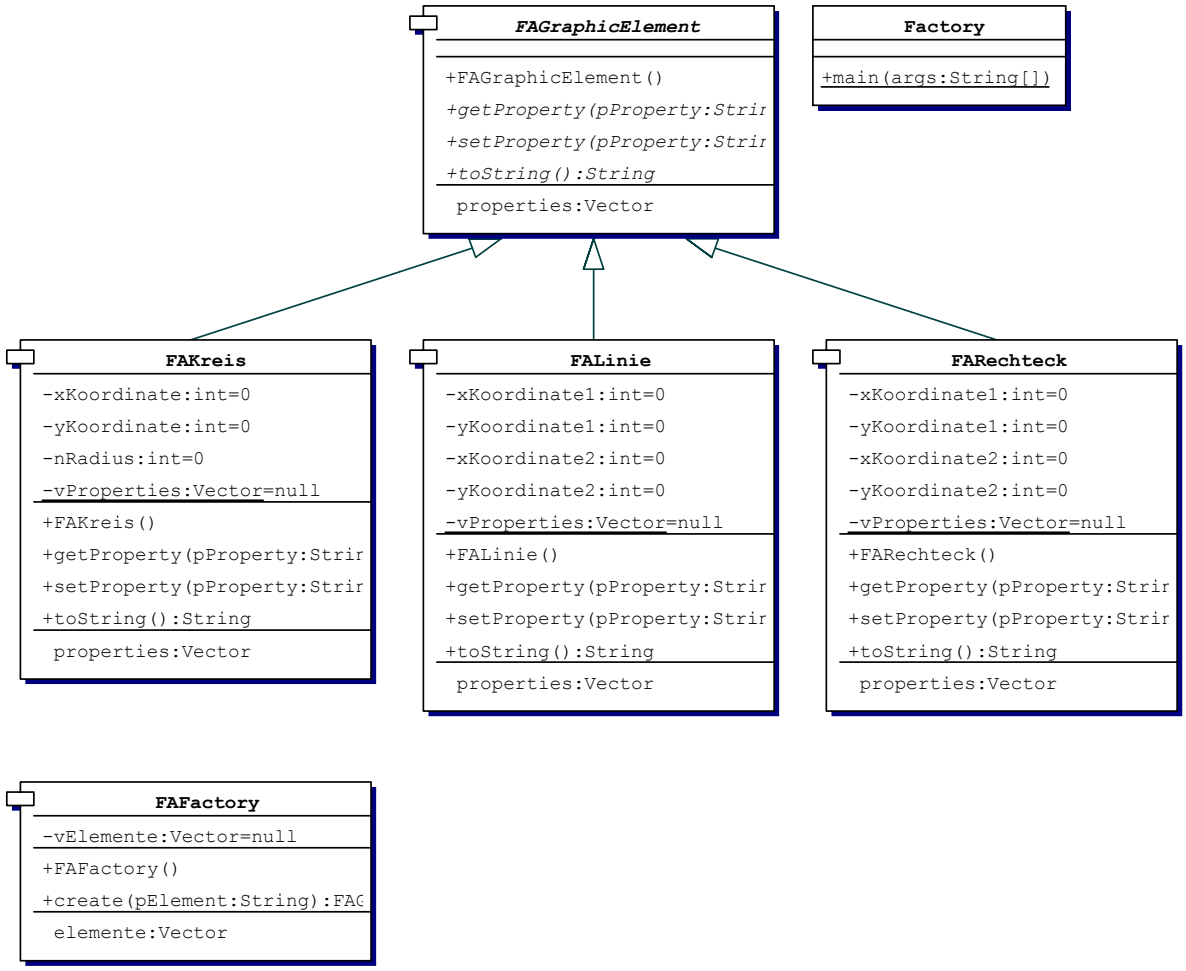

Abbildung 11-2 Java Klassendiagramm Factory Method

11.8.2 Java Beispielprogramm

```
//============================================================================
/** Java Factory Design Pattern - Testprogramm
 */
//============================================================================
import java.util.Vector;

public class Factory
{
   //------------------------------------------------------------------------
   // Hauptprogramm
   //------------------------------------------------------------------------
   public static void main(String[] args)
   {
      FAFactory        rFactory = new FAFactory();
```

```
      Vector           vElemente   = rFactory.getElemente();
      Vector           vInstanzen  = new Vector();
      Vector           vProperties = null;
      String           sTemp       = null;
      int              nTemp       = 111;
      FAGraphicElement rElement    = null;

      // Von jeder Art ein Element erstellen
      for (int i=0; i<vElemente.size(); i++)
      {
         sTemp = (String)vElemente.elementAt(i);
         vInstanzen.add(rFactory.create(sTemp));
      }

      // Inhalt des Instanzen-Vectors ausgeben
      System.out.println ("\nAusgabe:\n");
      for (int i=0; i<vInstanzen.size(); i++)
      {
         System.out.println((vInstanzen.elementAt(i).toString());
      }

      // Erstes Element nehmen und die Properties generisch füllen
      rElement = (FAGraphicElement)vInstanzen.elementAt(0);
      vProperties = rElement.getProperties();
      for (int i=0; i<vProperties.size(); i++)
      {
         sTemp = (String)vProperties.elementAt(i);
         rElement.setProperty(sTemp, Integer.toString(nTemp));
         nTemp = nTemp + 111;
      }

      // Inhalt des Instanzen-Vectors ausgeben
      System.out.println ("\nAusgabe:\n");
      for (int i=0; i<vInstanzen.size(); i++)
      {
         System.out.println((vInstanzen.elementAt(i).toString());
      }
   }
}
```

```
//============================================================================
/** Java Factory Design Pattern - Fabrik
 */
//============================================================================
import java.util.Vector;

public class FAFactory
{
   private Vector vElemente = null;

   public FAFactory ()
   {
      super();
      if (vElemente == null)
      {
         // Dies würde man aus einer Konfigurationsdatei lesen
         vElemente = new Vector();
         vElemente.add ("Kreis");
         vElemente.add ("Rechteck");
         vElemente.add ("Linie");
      }
   }
```

```
   // Das Konstruieren würde über Reflection geschehen
   public FAGraphicElement create(String pElement)
   {
      System.out.println ("create: " + pElement);
      FAGraphicElement rRC = null;
      if (pElement != null)
      {
         if (pElement.equals("Kreis"))
         {
            rRC = new FAKreis();
         }
         else if (pElement.equals("Rechteck"))
         {
            rRC = new FARechteck();
         }
         if (pElement.equals("Linie"))
         {
            rRC = new FALinie();
         }
      }
      return rRC;
   }

   //---------------------------------------------------------------------
   public Vector getElemente()
   {
      return vElemente;
   }
}
```

```
//============================================================================
/** Java Factory Design Pattern - abstrakte Basisklasse Graphikelement
 */
//============================================================================
import java.util.Vector;

public abstract class FAGraphicElement
{
   //---------------------------------------------------------------------
   public FAGraphicElement()
   {
      super();
   }

   public abstract Vector getProperties();
   public abstract String getProperty(String pProperty);
   public abstract void setProperty(String pProperty, String pValue);
   public abstract String toString();
}
```

```
//============================================================================
/** Java Factory Design Pattern - Linie
 */
//============================================================================
import java.util.Vector;

public class FALinie extends FAGraphicElement
{
   private int xKoordinate1 = 0;
   private int yKoordinate1 = 0;
```

```
   private int xKoordinate2 = 0;
   private int yKoordinate2 = 0;
   private static Vector vProperties = null;

   //-----------------------------------------------------------------------
   public FALinie()
   {
      super();
      if (vProperties == null)
      {
         vProperties = new Vector();
         vProperties.add ("xKoordinate1");
         vProperties.add ("yKoordinate1");
         vProperties.add ("xKoordinate2");
         vProperties.add ("yKoordinate2");
      }
   }

   //-----------------------------------------------------------------------
   public Vector getProperties()
   {
      return vProperties;
   }

   //-----------------------------------------------------------------------
   public String getProperty(String pProperty)
   {
      String sRC = null;
      if (pProperty != null)
      {
         if (pProperty.equals("xKoordinate1"))
         {
            sRC = new String (Integer.toString(xKoordinate1));
         }
         else if (pProperty.equals("yKoordinate1"))
         {
            sRC = new String (Integer.toString(yKoordinate1));
         }
         if (pProperty.equals("xKoordinate2"))
         {
            sRC = new String (Integer.toString(xKoordinate2));
         }
         else if (pProperty.equals("yKoordinate2"))
         {
            sRC = new String (Integer.toString(yKoordinate2));
         }
      }
      return sRC;
   }

   //-----------------------------------------------------------------------
   public void setProperty(String pProperty, String pValue)
   {
      if ((pProperty != null) && (pValue != null))
      {
         if (pProperty.equals("xKoordinate1"))
         {
            xKoordinate1 = Integer.parseInt(pValue);
         }
         else if (pProperty.equals("yKoordinate1"))
         {
            yKoordinate1 = Integer.parseInt(pValue);
```

```
         }
         if (pProperty.equals("xKoordinate2"))
         {
            xKoordinate2 = Integer.parseInt(pValue);
         }
         else if (pProperty.equals("yKoordinate2"))
         {
            yKoordinate2 = Integer.parseInt(pValue);
         }
      }
   }

   //-------------------------------------------------------------------------
   public String toString()
   {
      StringBuffer sb = new StringBuffer();
      sb.append ("Linie: " + xKoordinate1 + ", " + yKoordinate1 + ", "
                           + xKoordinate2 + ", " + yKoordinate2);
      return sb.toString();
   }
}
```

```
//=============================================================================
/** Java Factory Design Pattern - Rechteck
 */
//=============================================================================
import java.util.Vector;

public class FARechteck extends FAGraphicElement
{
   private int xKoordinate1 = 0;
   private int yKoordinate1 = 0;
   private int xKoordinate2 = 0;
   private int yKoordinate2 = 0;
   private static Vector vProperties = null;

   //-------------------------------------------------------------------------
   public FARechteck()
   {
      super();
      if (vProperties == null)
      {
         vProperties = new Vector();
         vProperties.add ("xKoordinate1");
         vProperties.add ("yKoordinate1");
         vProperties.add ("xKoordinate2");
         vProperties.add ("yKoordinate2");
      }
   }

   //-------------------------------------------------------------------------
   public Vector getProperties()
   {
      return vProperties;
   }

   //-------------------------------------------------------------------------
   public String getProperty(String pProperty)
   {
      String sRC = null;
      if (pProperty != null)
      {
```

```
         if (pProperty.equals("xKoordinate1"))
         {
            sRC = new String (Integer.toString(xKoordinate1));
         }
         else if (pProperty.equals("yKoordinate1"))
         {
            sRC = new String (Integer.toString(yKoordinate1));
         }
         if (pProperty.equals("xKoordinate2"))
         {
            sRC = new String (Integer.toString(xKoordinate2));
         }
         else if (pProperty.equals("yKoordinate2"))
         {
            sRC = new String (Integer.toString(yKoordinate2));
         }
      }
      return sRC;
   }

   //-------------------------------------------------------------------------
   public void setProperty(String pProperty, String pValue)
   {
      if ((pProperty != null) && (pValue != null))
      {
         if (pProperty.equals("xKoordinate1"))
         {
            xKoordinate1 = Integer.parseInt(pValue);
         }
         else if (pProperty.equals("yKoordinate1"))
         {
            yKoordinate1 = Integer.parseInt(pValue);
         }
         if (pProperty.equals("xKoordinate2"))
         {
            xKoordinate2 = Integer.parseInt(pValue);
         }
         else if (pProperty.equals("yKoordinate2"))
         {
            yKoordinate2 = Integer.parseInt(pValue);
         }
      }
   }

   //-------------------------------------------------------------------------
   public String toString()
   {
      StringBuffer sb = new StringBuffer();
      sb.append ("Rechteck: " + xKoordinate1 + ", " + yKoordinate1 + ", "
                              + xKoordinate2 + ", " + yKoordinate2);
      return sb.toString();
   }
}
```

```
//===========================================================================
/** Java Factory Design Pattern - Kreis
 */
//===========================================================================
import java.util.Vector;

public class FAKreis extends FAGraphicElement
{
```

```
   private int xKoordinate = 0;
   private int yKoordinate = 0;
   private int nRadius     = 0;
   private static Vector vProperties = null;

   //------------------------------------------------------------------------
   public FAKreis()
   {
      super();
      if (vProperties == null)
      {
         vProperties = new Vector();
         vProperties.add ("xKoordinate");
         vProperties.add ("yKoordinate");
         vProperties.add ("Radius");
      }
   }

   //------------------------------------------------------------------------
   public Vector getProperties()
   {
      return vProperties;
   }

   //------------------------------------------------------------------------
   public String getProperty(String pProperty)
   {
      String sRC = null;
      if (pProperty != null)
      {
         if (pProperty.equals("xKoordinate"))
         {
            sRC = new String (Integer.toString(xKoordinate));
         }
         else if (pProperty.equals("yKoordinate"))
         {
            sRC = new String (Integer.toString(yKoordinate));
         }
         else if (pProperty.equals("Radius"))
         {
            sRC = new String (Integer.toString(nRadius));
         }
      }
      return sRC;
   }

   //------------------------------------------------------------------------
   public void setProperty(String pProperty, String pValue)
   {
      if ((pProperty != null) && (pValue != null))
      {
         if (pProperty.equals("xKoordinate"))
         {
            xKoordinate = Integer.parseInt(pValue);
         }
         else if (pProperty.equals("yKoordinate"))
         {
            yKoordinate = Integer.parseInt(pValue);
         }
         else if (pProperty.equals("Radius"))
         {
            nRadius = Integer.parseInt(pValue);
```

```
         }
      }
   }

   //------------------------------------------------------------------------
   public String toString()
   {
      StringBuffer sb = new StringBuffer();
      sb.append ("Kreis: " + xKoordinate + ", " + yKoordinate + ", rad "
                           + nRadius);
      return sb.toString();
   }
}
```

11.8.3 Anmerkungen zum Java Beispiel

Ein generisches Laden aus bereitgestellten Dynamischen Link Libraries (DLL) ist in Java nicht möglich, da es sich um eine reine Windows-Technik handelt. Diese ließe sich zwar über JNI[13] erzeugen, wäre aber auf anderen Plattformen nicht brauchbar.
Die hier dargestellten verschiedenen Graphikelemente würde man unter Java jeweils in einem eigenen Package halten und über ein Reflection Verfahren aufrufen.
Da Java mit Referenzen arbeitet, ist eine explizite Freigabe des Speicherplatzes nicht erforderlich.

[13] JNI = Java Native Interface

12. Composite

Alternativnamen: **Kompositum**
Mustergruppe: **Strukturmuster**

12.1 Anmerkungen

Das Composite Pattern basiert auf dem Ansatz die zu verwaltenden Instanzen mit in einer Baumstruktur zu speichern und ist in der OOP relativ häufig anzutreffen.

12.2 Verwendungszweck

Verwaltung von hierarchisch und rekursiv anzuordnenden Instanzen. Typische Fälle der Anwendung dieses Patterns sind z.B. Objektorientierte Zeichenprogramme, die ihre Graphiken aus Primitiven (Punkt, Linie, Rechteck etc.) zusammensetzen. Fast alle dieser Programme besitzen die Funktion mehrere Primitive zu einem gemeinsamen Objekt zusammenzustellen („verbinden“), welches anschließend als Einheit betrachtet wird.

12.3 Problemstellungen

Wie können rekursive Strukturen gespeichert werden, ohne dass bei Erweiterung oder Änderung eine Vielzahl von Klassen angetastet werden müssen?

12.4 Lösung

Die Lösung durch das Composite Pattern besteht aus einer Baumstruktur, bei der untergeordnetes Element (Blatt) wie übergeordnetes Element (Knoten) von der gleichen Basisklasse abgeleitet sind. Das Wurzelelement besteht dabei immer aus einem Knoten.
Jeder Knoten kann (je nach Implementierung) eine beliebige Anzahl an Blättern verwalten. Die Zusammenfassung von Teilobjekten zu einem verbundenen Objekt (Kompositum) wird durch das Subsummieren unter eine Knoteninstanz erreicht. Üblicherweise wird zum Performancegewinn eine doppelt verkettete Baumstruktur verwendet, in dem jedes auch sein übergeordnetes Element kennt.

12.5 Vorteile

Das Composite Pattern ist eine hervorragende Form der Speicherung von rekursiven Informationsstrukturen. Der Overhead für den allgemein rekursiven Aufbau ist sehr gering und dürfte im Allgemeinen sogar unter dem Verwaltungsaufwand einer speziell zugeschnittenen Lösung liegen.

12.6 Nachteile

Nur auf rekursiv definierte Strukturen anwendbar.
Gegebenenfalls entsteht ein geringer Overhead bei der Verwaltung.

12.7 C++ Beispiel

Das nachfolgende Beispiel zeigt den typischen Anwendungsfall, bei dem mehrere Objekte zu einer Gruppe zusammengefasst werden, die anschließend wie ein Objekt behandelt werden.

12.7.1 C++ Klassendiagramm

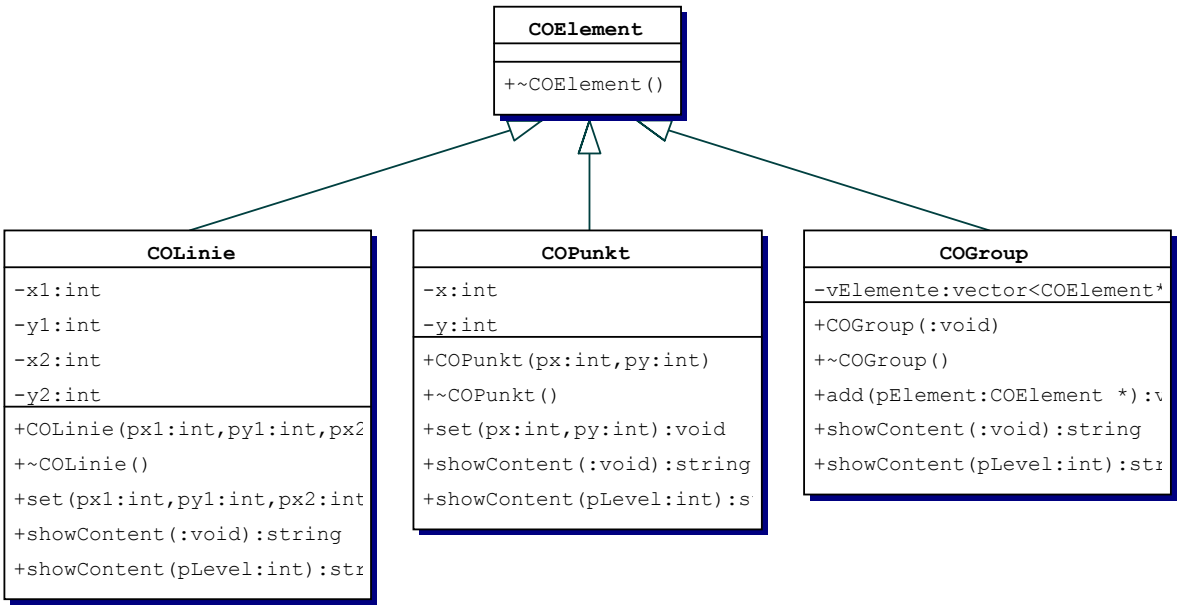

Abbildung 12-1 C++ Klassendiagramm Composite

12.7.2 C++ Beispielprogramm

```
//============================================================================
// C++ Composite Designpattern - Composite.cpp
//============================================================================
#include "stdafx.h"
#include <iostream>
#include "COGroup.h"
#include "COLinie.h"
#include "COPunkt.h"

int _tmain(int argc, _TCHAR* argv[])
{
   vector<COElement*>vElemente;

   vElemente.push_back(new COLinie(1,1,10,10));
   vElemente.push_back(new COPunkt(21,21));
   vElemente.push_back(new COPunkt(24,21));
   vElemente.push_back(new COLinie(31,31,10,10));

   COGroup *rG1 = new COGroup();
   rG1->add(new COLinie (4,4,44,44));
   rG1->add(new COPunkt (41,41));
   vElemente.push_back(rG1);

   COGroup *rG2 = new COGroup();
   rG2->add(new COLinie(5,5,55,55));
   rG2->add(new COPunkt(51,51));
```

```
   COGroup *rG3 = new COGroup();
   rG3->add(new COLinie(6,6,66,66));
   rG3->add(new COPunkt(61,61));
   rG3->add(rG2);

   vElemente.push_back(rG3);

   for (unsigned int i=0; i<vElemente.size(); i++)
   {
      COElement *rE = vElemente[i];
      cout << rE->showContent() << endl;
   }

   // alle Pointer freigeben
   for (unsigned int i=0; i<vElemente.size(); i++)
   {
      COElement *rE = vElemente[i];
      delete rE;
   }
   return 0;
}
```

```
//=========================================================================
// C++ Composite Designpattern - COElement.h
//=========================================================================
#ifndef _COELEMENT_H_
#define _COELEMENT_H_

#include <string>

using namespace std;

class COElement
{
   public:
      virtual ~COElement(){}

      virtual string showContent(void) = NULL;
      virtual string showContent(int pLevel) = NULL;
};

#endif
```

```
//=========================================================================
// C++ Composite Designpattern - COPunkt.h
//=========================================================================
#ifndef _COPUNKT_H_
#define _COPUNKT_H_

#include "COElement.h"
#include <vector>

using namespace std;

class COPunkt : public COElement
{
   public:
      COPunkt(int px, int py);
      virtual ~COPunkt();

      void set(int px, int py);
```

```
      virtual string showContent(void);
      virtual string showContent(int pLevel);

   private:
      int x;
      int y;
};

#endif
```

```
//============================================================================
// C++ Composite Designpattern - COPunkt.cpp
//============================================================================
#include "stdafx.h"
#include <iostream>
#include "COPunkt.h"

//----------------------------------------------------------------------------
COPunkt::COPunkt(int px, int py)
{
   set(px, py);
}

//----------------------------------------------------------------------------
COPunkt::~COPunkt()
{
   cout << "Destructor COPunkt" << " - "
        << x << ", " << y << endl;
}

//----------------------------------------------------------------------------
void COPunkt::set(int px, int py)
{
   x = px;
   y = py;
}

//----------------------------------------------------------------------------
string COPunkt::showContent(void)
{
   return showContent(0);
}

//----------------------------------------------------------------------------
string COPunkt::showContent(int pLevel)
{
   char s[1000] = "";
   for (int i=0; i<pLevel; i++)
   {
      strcat(s, "  ");
   }
   char t[100] = "";
   sprintf (t, "Punkt: %d, %d", x, y);
   strcat(s, t);
   return s;
}
```

```
//============================================================================
// C++ Composite Designpattern - COLinie.h
//============================================================================
#ifndef _COLINIE_H_
```

```
#define _COLINIE_H_

#include "COElement.h"
#include <vector>

using namespace std;

class COLinie : public COElement
{
   public:
      COLinie(int px1, int py1, int px2, int py2);
      virtual ~COLinie();

      void set(int px1, int py1, int px2, int py2);
      virtual string showContent(void);
      virtual string showContent(int pLevel);

   private:
      int x1;
      int y1;
      int x2;
      int y2;
};

#endif
```

```
//============================================================================
// C++ Composite Designpattern - COLinie.cpp
//============================================================================
#include "stdafx.h"
#include <iostream>
#include "COLinie.h"

//----------------------------------------------------------------------------
COLinie::COLinie(int px1, int py1, int px2, int py2)
{
   set(px1, py1, px2, py2);
}

//----------------------------------------------------------------------------
COLinie::~COLinie()
{
   cout << "Destructor COLinie" << " - "
        << x1 << ", " << y1 << ", "
        << x2 << ", " << y2 << endl;
}

//----------------------------------------------------------------------------
void COLinie::set(int px1, int py1, int px2, int py2)
{
   x1 = px1;
   y1 = py1;
   x2 = px2;
   y2 = py2;
}

//----------------------------------------------------------------------------
string COLinie::showContent(void)
{
   return showContent(0);
}
```

```
//-----------------------------------------------------------------------
string COLinie::showContent(int pLevel)
{
   char s[1000] = "";
   for (int i=0; i<pLevel; i++)
   {
      strcat(s, "  ");
   }
   char t[100] = "";
   sprintf (t, "Linie: %d, %d, %d, %d", x1, y1, x2, y2);
   strcat(s, t);
   return s;
}
```

```
//=======================================================================
// C++ Composite Designpattern - COGroup.h
//=======================================================================
#ifndef _COGROUP_H_
#define _COGROUP_H_

#include "COElement.h"
#include <vector>

using namespace std;

class COGroup : public COElement
{
   public:
      COGroup(void);
      virtual ~COGroup();

      void add(COElement *pElement);
      virtual string showContent(void);
      virtual string showContent(int pLevel);

   private:
      vector<COElement*> vElemente;

};

#endif
```

```
//=======================================================================
// C++ Composite Designpattern - COGroup.cpp
//=======================================================================
#include "stdafx.h"
#include <iostream>
#include "COGroup.h"

//-----------------------------------------------------------------------
COGroup::COGroup(void)
{
}

//-----------------------------------------------------------------------
COGroup::~COGroup()
{
   cout << "Destructor COGroup" << endl;
   for (unsigned int i=0; i<vElemente.size(); i++)
   {
      COElement *rE = vElemente[i];
```

```
      delete rE;
   }
   vElemente.clear();
}

//---------------------------------------------------------------------------
void COGroup::add(COElement *pElement)
{
   vElemente.push_back(pElement);
}

//---------------------------------------------------------------------------
string COGroup::showContent(void)
{
   return showContent(0);
}

//---------------------------------------------------------------------------
string COGroup::showContent(int pLevel)
{
   char s[10000] = "";
   for (int i=0; i<pLevel; i++)
   {
      strcat(s, "  ");
   }
   strcat(s, "Gruppe: \n");

   for (unsigned int i=0; i<vElemente.size(); i++)
   {
      strcat(s, (vElemente[i]->showContent(pLevel+1)).c_str());
      strcat(s, "\n");
   }
   return s;
}
```

12.7.3 Anmerkungen zum C++ Beispiel

Um Slicing-Phänomene zu vermeiden, ist die Verwendung von Pointern im Vector zwingend gegeben. Es ist darauf zu achten, dass der Speicherplatz im Composite Pattern auch hierarchisch wieder freigegeben wird.

12.8 Java Beispiel

Das nachfolgende Beispiel zeigt den typischen Anwendungsfall, bei dem mehrere Objekte zu einer Gruppe zusammengefasst werden, die anschließend wie ein Objekt behandelt werden.

12.8.1 Java Klassendiagramm

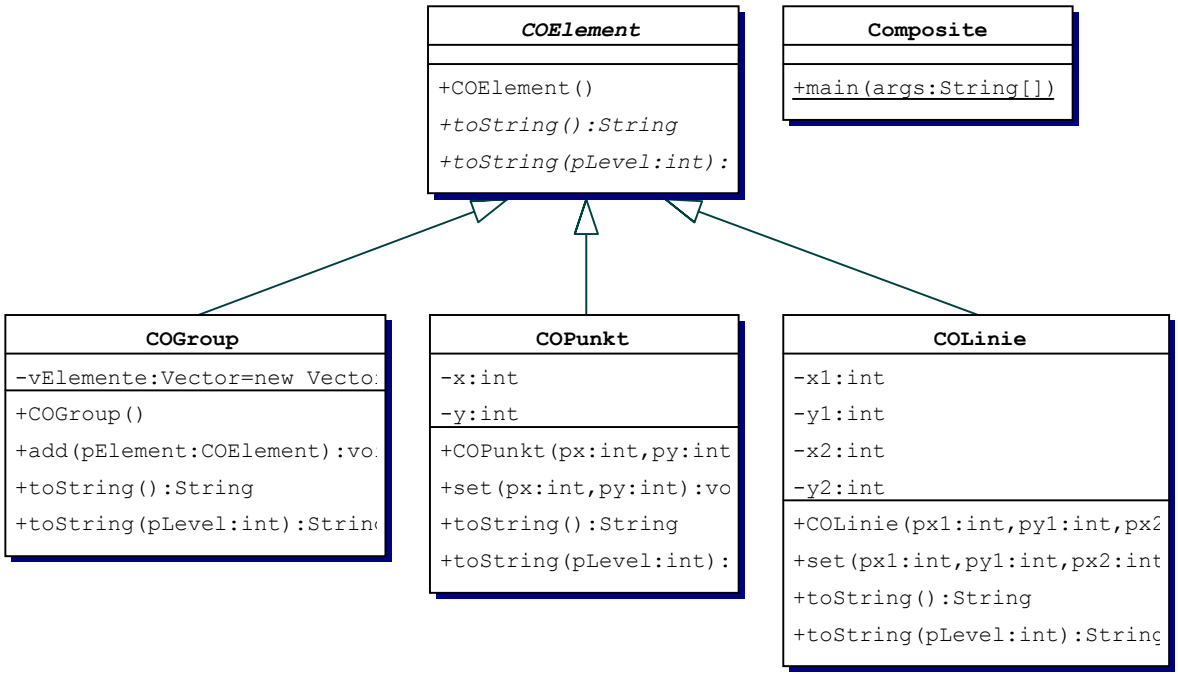

Abbildung 12-2 Java Klassendiagramm Composite

12.8.2 Java Beispielprogramm

```
//============================================================================
/** Java Composite Design Pattern - Testprogramm
 */
//============================================================================
import java.util.Vector;

public class Composite
{
   //----------------------------------------------------------------------
   // Hauptprogramm
   //----------------------------------------------------------------------
   public static void main(String[] args)
   {
      Vector vElemente = new Vector();

      vElemente.add(new COLinie(1,1,10,10));
      vElemente.add(new COPunkt(21,21));
      vElemente.add(new COPunkt(24,21));
      vElemente.add(new COLinie(31,31,10,10));

      COLinie rL1 = new COLinie(4,4,44,44);
      COPunkt rP1 = new COPunkt(41,41);
      COGroup rG1 = new COGroup();
      rG1.add(rL1);
      rG1.add(rP1);
      vElemente.add(rG1);

      COLinie rL2 = new COLinie(5,5,55,55);
      COPunkt rP2 = new COPunkt(51,51);
      COGroup rG2 = new COGroup();
      rG2.add(rL2);
      rG2.add(rP2);
```

```
      COLinie rL3 = new COLinie(6,6,66,66);
      COPunkt rP3 = new COPunkt(61,61);
      COGroup rG3 = new COGroup();
      rG3.add(rL3);
      rG3.add(rP3);
      rG3.add(rG2);

      vElemente.add(rG3);

      for (int i=0; i<vElemente.size(); i++)
      {
         System.out.println(
                    (vElemente.elementAt(i).toString());
      }
   }
}
```

```
//============================================================================
/** Java Composite Design Pattern - abstrakte Basisklasse Element
 */
//============================================================================
public abstract class COElement
{
   //------------------------------------------------------------------------
   public COElement()
   {
      super();
   }

   public abstract String toString();
   public abstract String toString(int pLevel);
}
```

```
//============================================================================
/** Java Composite Design Pattern - Punkt
 */
//============================================================================
public class COPunkt extends COElement
{
   private int x;
   private int y;

   //------------------------------------------------------------------------
   public COPunkt(int px, int py)
   {
      super();
      set(px, py);
   }

   //------------------------------------------------------------------------
   public void set(int px, int py)
   {
      x = px;
      y = py;
   }

   //------------------------------------------------------------------------
   public String toString()
   {
      return toString(0);
   }
```

```
   //-------------------------------------------------------------------
   public String toString(int pLevel)
   {
      StringBuffer sb = new StringBuffer();
      for (int i=0; i<pLevel; i++)
      {
         sb.append("  ");
      }
      sb.append ("Punkt: " + x + ", " + y);
      return sb.toString();
   }
}
```

```
//=====================================================================
/** Java Composite Design Pattern - Linie
 */
//=====================================================================
public class COLinie extends COElement
{
   private int x1;
   private int y1;
   private int x2;
   private int y2;

   public COLinie(int px1, int py1, int px2, int py2)
   {
      super();
      set(px1, py1, px2, py2);
   }

   //-------------------------------------------------------------------
   public void set(int px1, int py1, int px2, int py2)
   {
      x1 = px1;
      y1 = py1;
      x2 = px2;
      y2 = py2;
   }

   //-------------------------------------------------------------------
   public String toString()
   {
      return toString(0);
   }

   //-------------------------------------------------------------------
   public String toString(int pLevel)
   {
      StringBuffer sb = new StringBuffer();
      for (int i=0; i<pLevel; i++)
      {
         sb.append("  ");
      }
      sb.append ("Linie: " + x1 + ", " + y1 + ", " + x2 + ", " + y2);
      return sb.toString();
   }
}
```

```
//=====================================================================
/** Java Composite Design Pattern - Gruppe
```

```
 */
//============================================================================
import java.util.Vector;

public class COGroup extends COElement
{
   private Vector vElemente = new Vector();

   //------------------------------------------------------------------------
   public COGroup()
   {
      super();
   }

   //------------------------------------------------------------------------
   public void add(COElement pElement)
   {
      vElemente.add(pElement);
   }

   //------------------------------------------------------------------------
   public String toString()
   {
      return toString(0);
   }

   //------------------------------------------------------------------------
   public String toString(int pLevel)
   {
      StringBuffer sb = new StringBuffer(80);

      for (int i=0; i<pLevel; i++)
      {
         sb.append("  ");
      }
      sb.append("Gruppe: \n");

      for (int i=0; i<vElemente.size(); i++)
      {
         if (vElemente.elementAt(i) instanceof COGroup)
         {
            sb.append(
                 ((COElement)vElemente.elementAt(i)).toString(pLevel+1));
         }
         else
         {
            sb.append(
                 ((COElement)vElemente.elementAt(i)).toString(pLevel+1));
         }
         sb.append("\n");
      }
      return sb.toString();
   }
}
```

12.8.3 Anmerkungen zum Java Beispiel

Da Java mit Referenzen arbeitet, ist eine explizite Freigabe des Speicherplatzes nicht erforderlich.

13. Command

Alternativnamen: Befehl, Action, Transaction, Aktion, Transaktion
Mustergruppe: Verhaltensmuster

13.1 Anmerkungen

Das Command Pattern ist ein recht einfaches und häufig angewendetes Verhaltensmuster. Es ist üblicherweise Grundlage aller auf Fenstern basierten Benutzungsschnittstellen, selbst solcher, die in ihren Grundzügen gar nicht Objekt orientiert sind. So kann man das Messagesystem von Windows problemlos als Command Pattern betrachten.

13.2 Verwendungszweck

Auslösen einer Funktionalität in einem anderen (dem Auslöser vom Typ her unbekannten) Objekt ohne etwas über dessen tatsächliche Struktur oder Aufgaben wissen zu müssen. Typische Beispiele sind z.B. Menüs oder Buttons in einem Fenster.

13.3 Problemstellungen

Wie kann man einem Objekt etwas mitteilen oder eine Handlung auslösen, wenn man nicht weiß von welchem Typ das Objekt ist?
Wie kann man einen Verarbeitungsschritt rückgängig machen?
Wie kann man in einem Befehle mehrere Verarbeitungsschritte zusammenfassen?

13.4 Lösung

Das Command Pattern erzeugt von einer allgemeinen Befehlsklasse abgeleitete Befehlsobjekte, die an Objekte, welche eine Schnittstelle zum Befehlsempfang implementieren, weitergeleitet werden. Die Auslöser (Erzeuger des Befehlsobjektes) wissen üblicherweise nicht, in welchen Programmformen sie eingebettet sind. In einem MDI[14]-Programm wirken Menüauslösung oder Betätigung eines Buttons z.B. auf das aktuell aktivierte Element. D.h. der Befehl muss an eine übergeordnete Entscheidungsinstanz weitergeleitet werden, welche weiß, welches das aktuelle Dokument ist. Diese Entscheidungsinstanz wiederum gibt den Befehl an das Dokumentobjekt weiter, welches in Abhängigkeit seines eigenen Typs eine Aktion auslöst.
Da Zurücknehmen eines Arbeitsschrittes erfordert natürlich zu jeder Operation eine Umkehrung. Um die entsprechende Umkehrung auszulösen wird das die zuletzt ausgeführte (gegebenenfalls mehrere) Command Instanz gespeichert.
Um ein „Undo“ auszulösen wird diese Instanz an die Undo-Methode der Schnittstelle zum Befehlsempfang weitergeleitet (Es ist Aufgabe der Klasse sicherzustellen, das auf die vorherigen Parameter zurückgegriffen werden kann). Die Implementation eines funktionierenden „Undo“ und „Redo“ ist eine sehr komplexe Aufgabe, bei der sich (minimale) Fehler durch mehrfachen Aufruf von „Undo“ und „Redo“ zu erheblichen Abweichungen aufschaukeln können.
Das „stapeln“ mehrerer Befehle lässt sich durch Definition eines Makrobefehls realisieren. Zur Implementation eignet sich z.B. das Composite Pattern (⇨ Composite).

[14] MDI = Multi Document Interface – Ein Programm, mit dem mehrere Dokumente (= Datendateien) parallel bearbeitet werden können.

13.5 Vorteile

Objekte können miteinander kommunizieren (= sich gegenseitig aufrufen), ohne dass sie etwas über die spezialisierten Schnittstellen einer abgeleiteten klasse wissen müssen. Die Befehle können erweitert werden und neue Befehle entwickelt werden, ohne das die Kommunikationsstruktur (die Art und Weise, wie die Objekte mit einander agieren) verändert werden muss.

13.6 Nachteile

Es liegt in der Natur der Sache, dass nicht jedes Programm alle in Frage kommenden Befehle vollständig verarbeitet. Bei Fenstersystemen werden Command Instanzen, welche nicht von einem Befehlsempfänger verarbeitet werden üblicherweise einfach verworfen. Es kann daher nicht zwingend sichergestellt werden, dass das den Befehl erzeugende und das empfangende Objekt einander „verstehen“ (d.h. die empfangene Command Instanz auch wirklich eine Aktion auslöst).

13.7 C++ Beispiel

Das Beispiel zeigt die Anwendung des Command Patterns, wie man es in einer selbst geschriebenen Graphikanwendung nutzen würde.

13.7.1 C++ Klassendiagramm

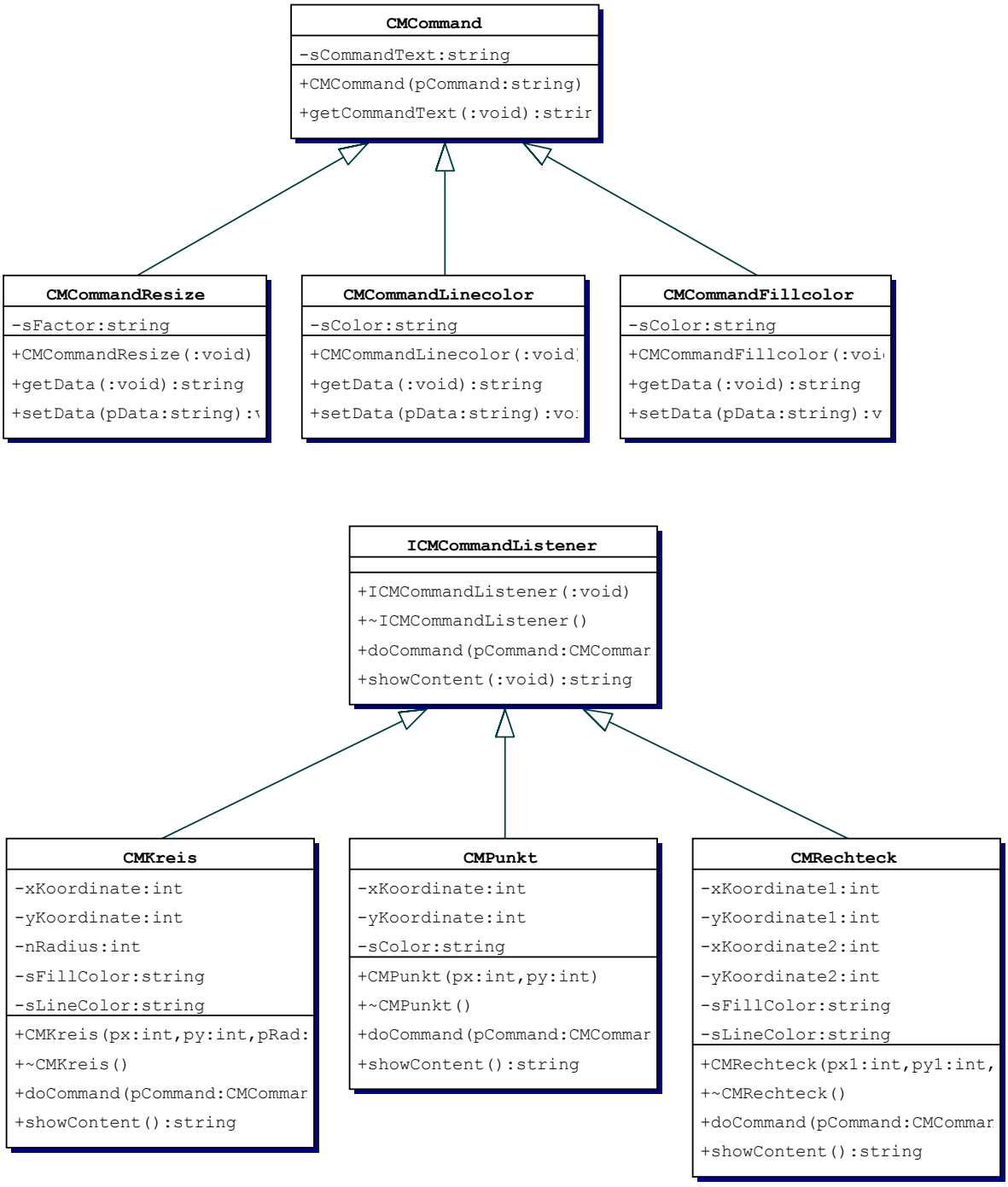

Abbildung 13-1 C++ Klassendiagramm Command

13.7.2 C++ Beispielprogramm

```
//========================================================================
// C++ Command Designpattern - Command.cpp
//========================================================================
```

```
#include "stdafx.h"
#include <vector>
#include <iostream>
#include "CMKreis.h"
#include "CMPunkt.h"
#include "CMRechteck.h"
#include "CMCommandResize.h"
#include "CMCommandFillcolor.h"
#include "CMCommandLinecolor.h"

using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
   vector<ICMCommandListener*> vCommandListener;
   ICMCommandListener* rListener = NULL;
   CMCommandResize     rResize;
   CMCommandFillcolor  rFill;
   CMCommandLinecolor  rLine;
   rResize.setData("0.5");
   rFill.setData("#FF0000");
   rLine.setData("#00FF00");

   CMKreis *rKreis1 = new CMKreis( 50,  50, 100);
   vCommandListener.push_back(rKreis1);
   CMKreis *rKreis2 = new CMKreis( 20,  20,  11);
   vCommandListener.push_back(rKreis2);

   CMPunkt *rPunkt1 = new CMPunkt(10, 10);
   vCommandListener.push_back(rPunkt1);
   CMPunkt *rPunkt2 = new CMPunkt(100, 100);
   vCommandListener.push_back(rPunkt2);

   CMRechteck *rRechteck1 = new CMRechteck( 25,  25, 125, 125);
   vCommandListener.push_back(rRechteck1);
   CMRechteck *rRechteck2 = new CMRechteck(200, 200, 300, 300);
   vCommandListener.push_back(rRechteck2);

   //------------------------------------------------------------------------
   // Ausgeben
   //------------------------------------------------------------------------
   for (unsigned int i=0; i<vCommandListener.size(); i++)
   {
      rListener = vCommandListener[i];
      cout << rListener->showContent() << endl;
   }
   cout << "----------------------" << endl;

   //------------------------------------------------------------------------
   // Größe ändern
   //------------------------------------------------------------------------
   for (unsigned int i=0; i<vCommandListener.size(); i++)
   {
      rListener = vCommandListener[i];
      rListener->doCommand(&rResize);
   }

   //------------------------------------------------------------------------
   // Ausgeben
   //------------------------------------------------------------------------
   for (unsigned int i=0; i<vCommandListener.size(); i++)
   {
```

```
      rListener = vCommandListener[i];
      cout << rListener->showContent() << endl;
   }
   cout << "----------------------" << endl;

   //-------------------------------------------------------------------
   // Farben ändern
   //-------------------------------------------------------------------
   for (unsigned int i=0; i<vCommandListener.size(); i++)
   {
      rListener = vCommandListener[i];
      rListener->doCommand(&rFill);
      rListener->doCommand(&rLine);
   }

   //-------------------------------------------------------------------
   // Ausgeben
   //-------------------------------------------------------------------
   for (unsigned int i=0; i<vCommandListener.size(); i++)
   {
      rListener = vCommandListener[i];
      cout << rListener->showContent() << endl;
   }
   cout << "----------------------" << endl;

   //-------------------------------------------------------------------
   // Freigeben
   //-------------------------------------------------------------------
   for (unsigned int i=0; i<vCommandListener.size(); i++)
   {
      delete vCommandListener[i];
   }
   return 0;
}
```

```
//=========================================================================
// C++ Command Designpattern - CMCommand.h
//=========================================================================
#ifndef _CMCOMMAND_H_
#define _CMCOMMAND_H_

#include <string>

using namespace std;

class CMCommand
{
   public:
      CMCommand(string pCommand);

      string getCommandText(void);
      virtual string getData(void) = NULL;
      virtual void   setData(string pData) = NULL;

   private:
      string sCommandText;
};

#endif
```

```
//=========================================================================
```

```
// C++ Command Designpattern - CMCommand.cpp
//============================================================================
#include "stdafx.h"
#include "CMCommand.h"

//----------------------------------------------------------------------------
CMCommand::CMCommand(string pCommand)
{
   sCommandText = pCommand;
}

//----------------------------------------------------------------------------
string CMCommand::getCommandText(void)
{
   return sCommandText;
}
```

```
//============================================================================
// C++ Command Designpattern - CMCommandResize.h
//============================================================================
#ifndef _CMCOMMANDRESIZE_H_
#define _CMCOMMANDRESIZE_H_

#include <string>
#include "CMCommand.h"

using namespace std;

class CMCommandResize : public CMCommand
{
   public:
      CMCommandResize(void);

      virtual string getData(void);
      virtual void   setData(string pData);

   private:
      string sFactor;
};

#endif
```

```
//============================================================================
// C++ Command Designpattern - CMCommandResize.cpp
//============================================================================
#include "stdafx.h"
#include "CMCommandResize.h"

//----------------------------------------------------------------------------
CMCommandResize::CMCommandResize(void) : CMCommand("Resize")
{
   sFactor = "1.0";
}

//----------------------------------------------------------------------------
string CMCommandResize::getData()
{
   return sFactor;
}

//----------------------------------------------------------------------------
```

```
void CMCommandResize::setData(string pData)
{
   sFactor = pData;
}
```

```
//===========================================================================
// C++ Command Designpattern - CMCommandLinecolor.h
//===========================================================================
#ifndef _CMCOMMANDLINECOLOR_H_
#define _CMCOMMANDLINECOLOR_H_

#include <string>
#include "CMCommand.h"

using namespace std;

class CMCommandLinecolor : public CMCommand
{
   public:
      CMCommandLinecolor(void);

      virtual string getData(void);
      virtual void   setData(string pData);

   private:
      string sColor;
};

#endif
```

```
//===========================================================================
// C++ Command Designpattern - CMCommandLinecolor.cpp
//===========================================================================
#include "stdafx.h"
#include "CMCommandLinecolor.h"

//---------------------------------------------------------------------------
CMCommandLinecolor::CMCommandLinecolor(void) : CMCommand("Linecolor")
{
   sColor="#000000";
}

//---------------------------------------------------------------------------
string CMCommandLinecolor::getData(void)
{
   return sColor;
}

//---------------------------------------------------------------------------
void CMCommandLinecolor::setData(string pColor)
{
   sColor = pColor;
}
```

```
//===========================================================================
// C++ Command Designpattern - CMCommandFillcolor.h
//===========================================================================
#ifndef _CMCOMMANDFILLCOLOR_H_
#define _CMCOMMANDFILLCOLOR_H_

#include <string>
```

```
#include "CMCommand.h"

using namespace std;

class CMCommandFillcolor : public CMCommand
{
   public:
      CMCommandFillcolor(void);

      virtual string getData(void);
      virtual void   setData(string pData);

   private:
      string sColor;
};

#endif
```

```
//============================================================================
// C++ Command Designpattern - CMCommandFillcolor.cpp
//============================================================================
#include "stdafx.h"
#include "CMCommandFillcolor.h"

//----------------------------------------------------------------------------
CMCommandFillcolor::CMCommandFillcolor(void) : CMCommand("Fillcolor")
{
   sColor="#000000";
}

//----------------------------------------------------------------------------
string CMCommandFillcolor::getData(void)
{
   return sColor;
}

//----------------------------------------------------------------------------
void CMCommandFillcolor::setData(string pColor)
{
   sColor = pColor;
}
```

```
//============================================================================
// C++ Command Designpattern - ICMCommandListener.h
//============================================================================
#ifndef _ICMCOMMANDLISTENER_H_
#define _ICMCOMMANDLISTENER_H_

#include <string>
#include "CMCommand.h"

using namespace std;

class ICMCommandListener
{
   public:
      ICMCommandListener(void);
      virtual ~ICMCommandListener();

      virtual void doCommand (CMCommand *pCommand);
      virtual string showContent(void);
```

```
};

#endif
```

```
//============================================================================
// C++ Command Designpattern - ICMCommandListener.cpp
//============================================================================
#include "stdafx.h"
#include <iostream>
#include "ICMCommandListener.h"

//----------------------------------------------------------------------------
ICMCommandListener::ICMCommandListener(void)
{
}

//----------------------------------------------------------------------------
ICMCommandListener::~ICMCommandListener()
{
}

//----------------------------------------------------------------------------
void ICMCommandListener::doCommand (CMCommand *pCommand)
{
}

//----------------------------------------------------------------------------
string ICMCommandListener::showContent(void)
{
   return "ICMCommandListener";
}
```

```
//============================================================================
// C++ Command Designpattern - CMPunkt.h
//============================================================================
#ifndef _CMPUNKT_H_
#define _CMPUNKT_H_

#include <string>
#include <vector>
#include "ICMCommandListener.h"

using namespace std;

class CMPunkt : public ICMCommandListener
{
   public:
      CMPunkt(int px, int py);
      virtual ~CMPunkt();

      virtual void   doCommand(CMCommand *pCommand);
      virtual string showContent();

   private:
      int xKoordinate;
      int yKoordinate;
      string sColor;
};

#endif
```

```
//============================================================================
// C++ Command Designpattern - CMPunkt.cpp
//============================================================================
#include "stdafx.h"
#include <iostream>
#include "CMPunkt.h"

//----------------------------------------------------------------------------
CMPunkt::CMPunkt(int px, int py)
{
   xKoordinate = px;
   yKoordinate = py;
   sColor.append ("#000000");
}

//----------------------------------------------------------------------------
CMPunkt::~CMPunkt()
{
   cout << "Destructor CMPunkt" << endl;
}

//----------------------------------------------------------------------------
void CMPunkt::doCommand (CMCommand *pCommand)
{
   // Punkt kann weder gefüllt noch skaliert werden
   if (pCommand->getCommandText() == "Linecolor")
   {
      sColor = pCommand->getData();
   }
}

//----------------------------------------------------------------------------
string CMPunkt::showContent(void)
{
   char s[1000] = "";
   sprintf (s, "Punkt: %d, %d - Linie: %s",
            xKoordinate, yKoordinate,
            sColor.c_str());
   string rc = s;
   return rc;
}
```

```
//============================================================================
// C++ Command Designpattern - CMRechteck.h
//============================================================================
#ifndef _CMRECHTECK_H_
#define _CMRECHTECK_H_

#include <string>
#include <vector>
#include "ICMCommandListener.h"

using namespace std;

class CMRechteck : public ICMCommandListener
{
   public:
      CMRechteck(int px1, int py1, int px2, int py2);
      virtual ~CMRechteck();

      virtual void   doCommand(CMCommand *pCommand);
      virtual string showContent();
```

```
   private:
      int xKoordinate1;
      int yKoordinate1;
      int xKoordinate2;
      int yKoordinate2;
      string sFillColor;
      string sLineColor;
};

#endif
```

```
//============================================================================
// C++ Command Designpattern - CMRechteck.cpp
//============================================================================
#include "stdafx.h"
#include <iostream>
#include "CMRechteck.h"

//----------------------------------------------------------------------------
CMRechteck::CMRechteck(int px1, int py1, int px2, int py2)
{
   xKoordinate1 = px1;
   yKoordinate1 = py1;
   xKoordinate2 = px2;
   yKoordinate2 = py2;
   sFillColor="#000000";
   sLineColor="#000000";
}

//----------------------------------------------------------------------------
CMRechteck::~CMRechteck()
{
   cout << "Destructor CMRechteck" << endl;
}

//----------------------------------------------------------------------------
void CMRechteck::doCommand (CMCommand *pCommand)
{
   if (pCommand->getCommandText() == "Fillcolor")
   {
      sFillColor = pCommand->getData();
   }
   else if (pCommand->getCommandText() == "Linecolor")
   {
      sLineColor = pCommand->getData();
   }
   else if (pCommand->getCommandText() == "Resize")
   {
      int nDiff = 0;
      string sVal;

      nDiff = xKoordinate2 - xKoordinate1;
      sVal = pCommand->getData();
      nDiff = (int)(nDiff * atof(sVal.c_str()));
      xKoordinate2 = xKoordinate1 + nDiff;

      nDiff = yKoordinate2 - yKoordinate1;
      sVal = pCommand->getData();
      nDiff = (int)(nDiff * atof(sVal.c_str()));
      yKoordinate2 = yKoordinate1 + nDiff;
   }
```

```
}

//------------------------------------------------------------------------
string CMRechteck::showContent(void)
{
   char s[1000] = "";
   sprintf (s, "Rechteck: %d, %d, %d, %d - Linie: %s - Fill: %s",
            xKoordinate1, yKoordinate1,
            xKoordinate2, yKoordinate2,
            sLineColor.c_str(), sFillColor.c_str());
   return s;
}
```

```
//============================================================================
// C++ Command Designpattern - CMKreis.h
//============================================================================
#ifndef _CMKREIS_H_
#define _CMKREIS_H_

#include <string>
#include <vector>
#include "ICMCommandListener.h"

using namespace std;

class CMKreis : public ICMCommandListener
{
   public:
      CMKreis(int px, int py, int pRad);
      virtual ~CMKreis();

      virtual void   doCommand(CMCommand *pCommand);
      virtual string showContent();

   private:
      int xKoordinate;
      int yKoordinate;
      int nRadius;
      string sFillColor;
      string sLineColor;
};

#endif
```

```
//============================================================================
// C++ Command Designpattern - CMKreis.cpp
//============================================================================
#include "stdafx.h"
#include <iostream>
#include "CMKreis.h"

//------------------------------------------------------------------------
CMKreis::CMKreis(int px, int py, int pRad)
{
   xKoordinate = px;
   yKoordinate = py;
   nRadius     = pRad;
   sFillColor="#000000";
   sLineColor="#000000";
}
```

```
//---------------------------------------------------------------------
CMKreis::~CMKreis()
{
   cout << "Destructor CMKreis" << endl;
}

//---------------------------------------------------------------------
void CMKreis::doCommand (CMCommand *pCommand)
{
   if (pCommand->getCommandText() == "Fillcolor")
   {
      sFillColor = pCommand->getData();
   }
   else if (pCommand->getCommandText() == "Linecolor")
   {
      sLineColor = pCommand->getData();
   }
   else if (pCommand->getCommandText() == "Resize")
   {
      string sVal;
      sVal = pCommand->getData();
      nRadius = (int)(nRadius * atof(sVal.c_str()));
   }
}

//---------------------------------------------------------------------
string CMKreis::showContent(void)
{
   char s[1000] = "";
   sprintf (s, "Kreis: %d, %d, %d - Linie: %s - Fill: %s",
            xKoordinate, yKoordinate, nRadius,
            sLineColor.c_str(), sFillColor.c_str());
   return s;
}
```

13.7.3 Anmerkungen zum C++ Beispiel

Um Slicing-Phänomene zu vermeiden, ist die Verwendung von Pointern im Vector zwingend gegeben. Es ist darauf zu achten, dass der Speicherplatz auch wieder freigegeben wird.

13.8 Java Beispiel

Das Beispiel zeigt die Anwendung des Command Patterns, wie man es in einer selbst geschriebenen Graphikanwendung nutzen würde.

13.8.1 Java Klassendiagramm

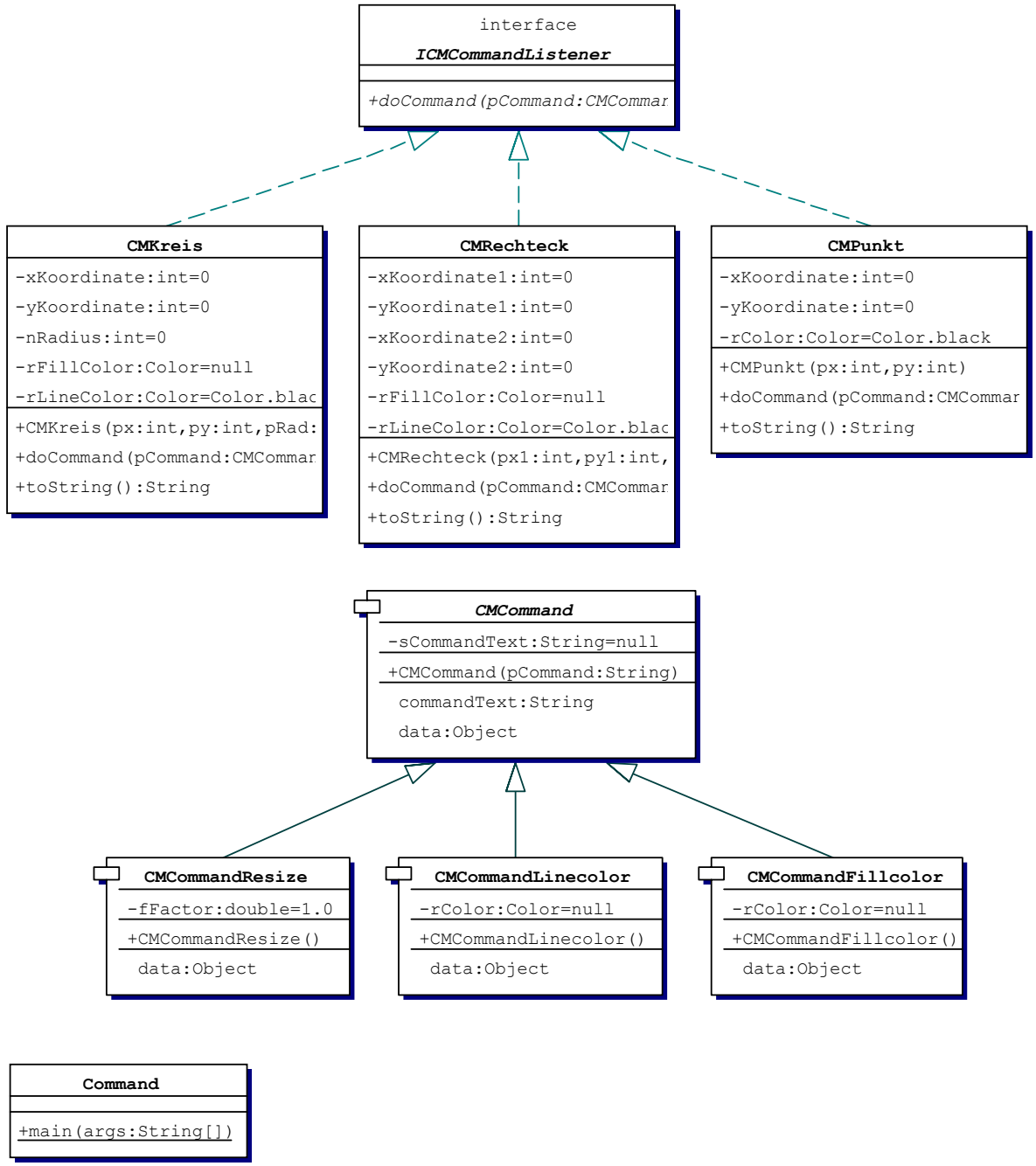

Abbildung 13-2 Java Klassendiagramm Command

13.8.2 Java Beispielprogramm

```
//======================================================================
/** Java Command Design Pattern - Testprogramm
 */
//======================================================================
import java.awt.Color;
```

```
import java.util.Vector;

public class Command
{
   //------------------------------------------------------------------------
   // Hauptprogramm
   //------------------------------------------------------------------------
   public static void main(String[] args)
   {
      Vector             vCommandListener = new Vector();
      ICMCommandListener rListener        = null;
      CMCommandResize    rResize          = new CMCommandResize();
      CMCommandFillcolor rFill            = new CMCommandFillcolor();
      CMCommandLinecolor rLine            = new CMCommandLinecolor();

      rResize.setData(new Double(0.5));
      rFill.setData(Color.red);
      rLine.setData(Color.blue);

      vCommandListener.add(new CMPunkt( 10,  10));
      vCommandListener.add(new CMPunkt(100, 100));
      vCommandListener.add(new CMKreis( 50,  50, 100));
      vCommandListener.add(new CMKreis( 20,  20,  11));
      vCommandListener.add(new CMRechteck( 25,  25, 125, 125));
      vCommandListener.add(new CMRechteck(200, 200, 300, 300));

      //---------------------------------------------------------------------
      // Ausgeben
      //---------------------------------------------------------------------
      for (int i=0; i<vCommandListener.size(); i++)
      {
         System.out.println(vCommandListener.elementAt(i).toString());
      }
      System.out.println("----------------------");

      //---------------------------------------------------------------------
      // Größe ändern
      //---------------------------------------------------------------------
      for (int i=0; i<vCommandListener.size(); i++)
      {
         if (vCommandListener.elementAt(i) instanceof ICMCommandListener)
         {
            rListener = (ICMCommandListener)vCommandListener.elementAt(i);
            rListener.doCommand(rResize);
         }
      }

      //---------------------------------------------------------------------
      // Ausgeben
      //---------------------------------------------------------------------
      for (int i=0; i<vCommandListener.size(); i++)
      {
         System.out.println(vCommandListener.elementAt(i).toString());
      }
      System.out.println("----------------------");

      //---------------------------------------------------------------------
      // Farben ändern
      //---------------------------------------------------------------------
      for (int i=0; i<vCommandListener.size(); i++)
      {
         if (vCommandListener.elementAt(i) instanceof ICMCommandListener)
```

```
         {
            rListener = (ICMCommandListener)vCommandListener.elementAt(i);
            rListener.doCommand(rFill);
            rListener.doCommand(rLine);
         }
      }

      //------------------------------------------------------------------------
      // Ausgeben
      //------------------------------------------------------------------------
      for (int i=0; i<vCommandListener.size(); i++)
      {
         System.out.println(vCommandListener.elementAt(i).toString());
      }
   }
}
```

```
//============================================================================
/** Java Command Design Pattern - abstrakte Basisklasse für Befehle
 */
//============================================================================
abstract public class CMCommand
{
   private String sCommandText = null;

   //------------------------------------------------------------------------
   public CMCommand(String pCommand)
   {
      super();
      sCommandText = pCommand;
   }

   //------------------------------------------------------------------------
   public String getCommandText()
   {
      return sCommandText;
   }

   //------------------------------------------------------------------------
   abstract public Object getData();

   //------------------------------------------------------------------------
   abstract public void setData(Object pData);
}
```

```
//============================================================================
/** Java Command Design Pattern - Befehl Resize
 */
//============================================================================
public class CMCommandResize extends CMCommand
{
   private double fFactor = 1.0;

   //------------------------------------------------------------------------
   public CMCommandResize()
   {
      super("Resize");
   }

   //------------------------------------------------------------------------
   public Object getData()
```

```
   {
      return new Double(fFactor);
   }

   //-------------------------------------------------------------------------
   public void setData(Object pData)
   {
      try
      {
         fFactor = ((Double)pData).doubleValue();
      }
      catch (ClassCastException e)
      {
         System.out.println(e);
      }
   }
}
```

```
//=============================================================================
/** Java Command Design Pattern - Befehl Füllfarbe
 */
//=============================================================================
import java.awt.Color;

public class CMCommandFillcolor extends CMCommand
{
   private Color rColor = null;

   //-------------------------------------------------------------------------
   public CMCommandFillcolor()
   {
      super("Fillcolor");
   }

   //-------------------------------------------------------------------------
   public Object getData()
   {
      return rColor;
   }

   //-------------------------------------------------------------------------
   public void setData(Object pColor)
   {
      try
      {
         rColor = (Color)pColor;
      }
      catch (ClassCastException e)
      {
         System.out.println(e);
      }
   }
}
```

```
//=============================================================================
/** Java Command Design Pattern - Befehl Linienfarbe
 */
//=============================================================================
import java.awt.Color;

public class CMCommandLinecolor extends CMCommand
```

```
{
   private Color rColor = null;

   //-------------------------------------------------------------------------
   public CMCommandLinecolor()
   {
      super("Linecolor");
   }

   //-------------------------------------------------------------------------
   public Object getData()
   {
      return rColor;
   }

   //-------------------------------------------------------------------------
   public void setData(Object pColor)
   {
      try
      {
         rColor = (Color)pColor;
      }
      catch (ClassCastException e)
      {
         System.out.println(e);
      }
   }
}
```

```
//=============================================================================
/** Java Command Design Pattern - Command-Interface
 */
//=============================================================================
public interface ICMCommandListener
{
   public void doCommand (CMCommand pCommand);
}
```

```
//=============================================================================
/** Java Command Design Pattern - Punkt
 */
//=============================================================================
import java.awt.Color;

public class CMPunkt implements ICMCommandListener
{
   private int xKoordinate = 0;
   private int yKoordinate = 0;
   private Color rColor   = Color.black;

   //-------------------------------------------------------------------------
   public CMPunkt(int px, int py)
   {
      super();
      xKoordinate = px;
      yKoordinate = py;
   }

   //-------------------------------------------------------------------------
   public void doCommand (CMCommand pCommand)
   {
```

```
      // Punkt kann weder gefüllt noch skaliert werden
      if (pCommand.getCommandText().equals("Linecolor"))
      {
         rColor = (Color)(pCommand.getData());
      }
   }

   //-------------------------------------------------------------------------
   public String toString()
   {
      StringBuffer sb = new StringBuffer();
      sb.append ("Punkt: " + xKoordinate + ", " + yKoordinate
                 + ", Color: " + rColor);
      return sb.toString();
   }
}
```

```
//=============================================================================
/** Java Command Design Pattern - Rechteck
 */
//=============================================================================
import java.awt.Color;

public class CMRechteck implements ICMCommandListener
{
   private int   xKoordinate1 = 0;
   private int   yKoordinate1 = 0;
   private int   xKoordinate2 = 0;
   private int   yKoordinate2 = 0;
   private Color rFillColor   = null;
   private Color rLineColor   = Color.black;

   //-------------------------------------------------------------------------
   public CMRechteck(int px1, int py1, int px2, int py2)
   {
      super();
      xKoordinate1 = px1;
      yKoordinate1 = py1;
      xKoordinate2 = px2;
      yKoordinate2 = py2;
   }

   //-------------------------------------------------------------------------
   public void doCommand (CMCommand pCommand)
   {
      if (pCommand.getCommandText().equals("Fillcolor"))
      {
         rFillColor = (Color)(pCommand.getData());
      }
      else if (pCommand.getCommandText().equals("Linecolor"))
      {
         rLineColor = (Color)(pCommand.getData());
      }
      else if (pCommand.getCommandText().equals("Resize"))
      {
         int nDiff = 0;
         nDiff = xKoordinate2 - xKoordinate1;
         nDiff = (int)(nDiff *
                       ((Double)(pCommand.getData())).doubleValue());
         xKoordinate2 = xKoordinate1 + nDiff;

         nDiff = yKoordinate2 - yKoordinate1;
```

```
         nDiff = (int)(nDiff *
                      ((Double)(pCommand.getData())).doubleValue());
         yKoordinate2 = yKoordinate1 + nDiff;
      }
   }

   //------------------------------------------------------------------------
   public String toString()
   {
      StringBuffer sb = new StringBuffer();
      sb.append ("Rechteck: " + xKoordinate1 + ", " + yKoordinate1 + ", "
                              + xKoordinate2 + ", " + yKoordinate2
                              + ", Line: "   + rLineColor
                              + ", Fill: " + rFillColor);
      return sb.toString();
   }
}
```

```
//============================================================================
/** Java Command Design Pattern - Kreis
 */
//============================================================================
import java.awt.Color;

public class CMKreis implements ICMCommandListener
{
   private int   xKoordinate = 0;
   private int   yKoordinate = 0;
   private int   nRadius     = 0;
   private Color rFillColor  = null;
   private Color rLineColor  = Color.black;

   //------------------------------------------------------------------------
   public CMKreis(int px, int py, int pRad)
   {
      super();
      xKoordinate = px;
      yKoordinate = py;
      nRadius     = pRad;
   }

   //------------------------------------------------------------------------
   public void doCommand (CMCommand pCommand)
   {
      if (pCommand.getCommandText().equals("Fillcolor"))
      {
         rFillColor = (Color)(pCommand.getData());
      }
      else if (pCommand.getCommandText().equals("Linecolor"))
      {
         rLineColor = (Color)(pCommand.getData());
      }
      else if (pCommand.getCommandText().equals("Resize"))
      {
         nRadius = (int)(nRadius *
                        ((Double)(pCommand.getData())).doubleValue());
      }
   }

   //------------------------------------------------------------------------
   public String toString()
   {
```

```
      StringBuffer sb = new StringBuffer();
      sb.append ("Kreis: " + xKoordinate + ", " + yKoordinate
                           + ", rad " + nRadius
                           + ", Line: " + rLineColor
                           + ", Fill: " + rFillColor);
      return sb.toString();
   }
}
```

13.8.3 Anmerkungen zum Java Beispiel

Da Java mit Referenzen arbeitet, ist eine explizite Freigabe des Speicherplatzes nicht erforderlich.

14. Flyweight

Alternativnamen: Fliegengewicht
Mustergruppe: Strukturmuster

14.1 Anmerkungen

Das Flyweight Pattern ist ein seltener anzutreffendes Strukturmuster, welches nur unter recht restriktiven Randbedingungen sinnvoll einsetzbar ist.

14.2 Verwendungszweck

das Flyweight Pattern dient der Reduktion des Speicherplatzbedarfs wenn große Mengen gleicher oder sehr ähnlicher Objekte erforderlich sind. Als typisches Beispiel für ein Flyweight Pattern werden üblicherweise Textverarbeitungen angesehen. Für sich betrachtet ist jeder Buchstabe eines Textes eine Objektinstanz mit einer ganzen Reihe von Attributen: Kursivschrift, Fettschrift, Schrifttyp, Schriftgröße, Vordergrund- und Hintergrundfarben, Unterstreichung etc. All diese Eigenschaften sollen für jeden Buchstaben des Dokumentes einzeln einstellbar sein. Gleichzeitig wird deutlich, dass sehr viele identische Instanzen erzeugt werden (man betrachte nur einmal die Häufigkeit des Buchstabens „e“ in diesem Abschnitt, der immer die gleichen Formatierungen aufweist). Das Flyweight Designpattern verringert diesen Aufwand auf eine Instanz pro unterschiedlichem Objekt.

14.3 Problemstellungen

Wie kann man Speicherplatz einsparen, wenn es um große Mengen von Objekten handelt, von denen sehr viele identisch sind?

14.4 Lösung

Statt eine vollständige Objektinstanz pro zu verwaltendem Objekt zu halten, wird lediglich eine Instanz pro unterschiedlichem Objekt angelegt und die Vielzahl der Instanzen stattdessen als Kette von „billigen“ Referenzen dargestellt.

14.5 Vorteile

Der Vorteil des Flyweight Musters ist eine erhebliche Speicherplatz und meist auch Zeitersparnis, da erheblich weniger „teure“ (d.h. voll ausgeprägte) Instanzen im Speicher liegen und auch weniger „teure“ Instanzen konstruiert werden müssen.

14.6 Nachteile

Die Verwaltung der Objektinstanzen ist deutlich schwieriger als bei einem einfachen, geradlinigen Ansatz. Bei letzterem kann jede Instanz einfach gelöscht werden, wenn sie nicht mehr benötigt wird. Bei einem Flyweight Designpattern kann eine Instanz nur dann gelöscht werden, wenn eine keine Referenz mehr auf diese Instanz gibt. Auch muss ständig geprüft werden, ob eine Instanz mit einer gewünschten Ausprägung bereits existiert oder neu anzulegen ist. Im Falle eines Textverarbeitungsprogramms ist das recht aufwendig, da sehr viele Formatierungen gleichzeitig verwendet werden können. man verwendet daher auch Strategien, die das Flyweight Pattern nicht auf den einzelnen Buchstaben herunterbrechen, sondern für eine bestimmte Formatierung immer einen kompletten Buchstabensatz anlegen.

14.7 C++ Beispiel

Das nachstehende Beispiel zeigt einen Platzhaltermechanismus, wie er z.B. in Textverarbeitungsprogrammen für Graphiken angeboten wird, um Speicherplatz zu sparen.

14.7.1 C++ Klassendiagramm

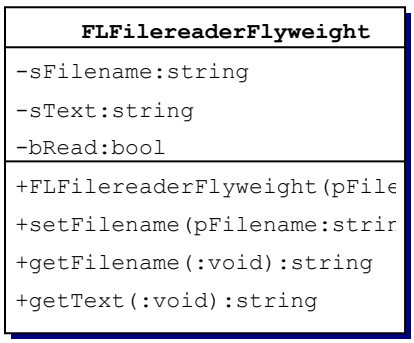

Abbildung 14-1 C++ Klassendiagramm Flyweight

14.7.2 C++ Beispielprogramm

```
//============================================================================
// C++ Flyweight Designpattern - Flyweight.cpp
//============================================================================
#include "stdafx.h"
#include "FLFilereaderFlyweight.h"

int _tmain(int argc, _TCHAR* argv[])
{
  FLFilereaderFlyweight rFly("c:\\test.txt");

  cout << rFly            << endl;
  cout << rFly.getText()  << endl;
  cout << rFly            << endl;
  cout << rFly.getText()  << endl;
  return 0;
}
```

```
//============================================================================
// C++ Flyweight Designpattern - FLFilereaderFlyweight.h
//============================================================================
#ifndef _FLFILEREADERFLYWEIGHT_H_
#define _FLFILEREADERFLYWEIGHT_H_

#include <iostream>
#include <string>

using namespace std;

class FLFilereaderFlyweight
{
  public:
     FLFilereaderFlyweight (string pFilename);

     void   setFilename (string pFilename);
     string getFilename (void);
     string getText     (void);
```

```
     friend ostream& operator<< (ostream& o,
                                 const FLFilereaderFlyweight& f);

  //-----------------------------------------
  private:
     string sFilename;
     string sText;
     bool   bRead;
};

#endif
```

```
//===========================================================================
// C++ Flyweight Designpattern - FLFilereaderFlyweight.cpp
//===========================================================================
#include "stdafx.h"
#include <fstream>
#include "FLFilereaderFlyweight.h"

using namespace std;

//---------------------------------------------------------------------------
FLFilereaderFlyweight::FLFilereaderFlyweight(string pFilename)
{
   setFilename(pFilename);
   bRead = false;
}

//---------------------------------------------------------------------------
string FLFilereaderFlyweight::getFilename (void)
{
   return sFilename;
}

//---------------------------------------------------------------------------
void FLFilereaderFlyweight::setFilename (string pFilename)
{
   if (sFilename != pFilename)
   {
      sFilename = pFilename;
      bRead = false;
   }
}

//---------------------------------------------------------------------------
string FLFilereaderFlyweight::getText (void)
{
   if (!bRead)
   {
      string sTemp;
      cout << "*** Text not in Memory - now Loading: " << sFilename
           << endl;
      ifstream rInput;
      rInput.open(sFilename.c_str());

      if (!rInput.fail())
      {
         while (!rInput.eof())
         {
            rInput >> sTemp;
            sText.append(sTemp);
            sText.append("\n");
```

```
         }
         rInput.close();
         bRead = true;
      }
      cout << sText << endl;
   }
   else
   {
      cout << "*** Text in Memory - no Loading needed: " << sFilename
           << endl;
   }
   return sText;
}

//-----------------------------------------------------------------------
ostream& operator<< (ostream& out, const FLFilereaderFlyweight& f)
{
   out << f.sFilename;
   if (!f.bRead)
   {
      out << " not read yet" << endl;
   }
   else
   {
      out << "Lenght: " + f.sText.length() << endl;
   }
   return out;
}
```

14.8 Java Beispiel

Das nachstehende Beispiel zeigt einen Platzhaltermechanismus, wie er z.B. in Textverarbeitungsprogrammen für Graphiken angeboten wird, um Speicherplatz zu sparen.

14.8.1 Java Klassendiagramm

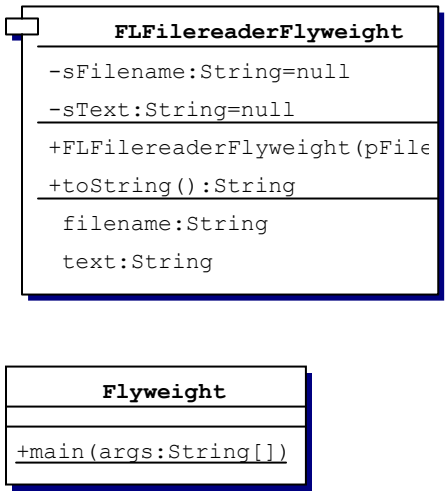

Abbildung 14-2 Java Klassendiagramm Flyweight

14.8.2 Java Beispielprogramm

```
//============================================================================
/** Java Flyweight Design Pattern - Testprogramm
 */
//============================================================================
public class Flyweight
{
   //------------------------------------------------------------------------
   // Hauptprogramm
   //------------------------------------------------------------------------
   public static void main(String[] args)
   {
      FLFilereaderFlyweight rFly = new
                                    FLFilereaderFlyweight("c:\\test.txt");

      System.out.println(rFly.toString());
      System.out.println(rFly.getText());
      System.out.println(rFly.toString());
      System.out.println(rFly.getText());
   }
}
```

```
//============================================================================
/** Java Flyweight Design Pattern - selbständig nachladende Klasse
 */
//============================================================================
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class FLFilereaderFlyweight
```

```
{
   private String sFilename = null;
   private String sText     = null;

   //-------------------------------------------------------------------------
   public FLFilereaderFlyweight(String pFilename)
   {
      super();
      setFilename(pFilename);
   }

   //-------------------------------------------------------------------------
   public String getFilename()
   {
      return sFilename;
   }

   //-------------------------------------------------------------------------
   public void setFilename(String pFilename)
   {
      if (!sFilename.equals(pFilename))
      {
         sFilename = pFilename;
         sText = null;
      }
   }

   //-------------------------------------------------------------------------
   public String getText()
   {
      if (sText == null)
      {
         System.out.println("*** Text not in Memory - now Loading: "
                            + sFilename);
         try
         {
            StringBuffer   sBuffer = new StringBuffer(4096);
            String         sLine   = null;
            FileReader     rFileIn = new FileReader(sFilename);
            BufferedReader rIn     = new BufferedReader(rFileIn);

            while ((sLine = rIn.readLine()) != null)
            {
               sBuffer.append(sLine);
               sBuffer.append("\n");
            }
            rIn.close();
            sText = sBuffer.toString();
         }
         catch (FileNotFoundException e)
         {
            System.out.println(e);
         }
         catch (IOException e)
         {
            System.out.println(e);
         }
      }
      else
      {
         System.out.println("*** Text in Memory - no Loading needed: "
                            + sFilename);
```

```
        }
        return sText;
    }

    public String toString()
    {
        if (sText == null)
        {
            return (sFilename + " not read yet");
        }
        return (sFilename + "Lenght: " + sText.length());
    }
}
```

15. Strategy

Alternativnamen: Strategie, Policy
Mustergruppe: Verhaltensmuster

15.1 Anmerkungen

Das Strategy Pattern ist ein noch relativ einfaches Verhaltensmuster, welches sich unter anderem sehr gut zur Konfiguration verwenden lässt. Da Strategy Klassen üblicherweise zustandslos (stateless) sind (d.h. keine eigenen Attribute besitzen sondern nur Methoden, statische Methoden oder statische Attribute) lassen sich die Instanzen häufig recht gut mit dem Flyweight Pattern (⇨ Flyweight) realisieren.

15.2 Verwendungszweck

Das Strategy Designpattern dient dazu Algorithmen in Objekte zu verpacken, so dass diese von verschiedenen Klassen genutzt werden können und dabei austauschbar bleiben. Typische Anwendungen für das Strategy Pattern sind z.B. Sortierungen und Formatierungen.

15.3 Problemstellungen

Wie kann man eine Familie austauschbarer Algorithmen in Objekten verpacken und von verschiedenen Klassen aus nutzen?

15.4 Lösung

Die Lösung ist relativ einfach und besteht aus einer Hierarchie von Instanzen, die verschiedene Algorithmen gleichen Zwecks (z.B. Sortierung oder Formatierung) gemäß einer gemeinsamen, abstrakten Schnittstelle realisieren.
Die Objekte, welche die Strategy Instanzen nutzen kennen und verwenden nur die gemeinsame Schnittstelle (Polymorphie), wissen also gar nicht welchen Algorithmus sie gerade verwenden.

15.5 Vorteile

Die verwendeten Algorithmen lassen sich bei Bedarf leicht durch andere ersetzen bzw. mit zusätzlichen Algorithmen ergänzen.

15.6 Nachteile

Die Wiederverwendung von Objekten die das Strategy Pattern implementieren erfordert eine strikt abstrakte Programmierung innerhalb der Applikation. Ist dies nicht gegeben, so sind die Algorithmen nur für eine Hierarchie nutzender Klassen (Clients) verwendbar und müssen für eine andere Hierarchie neu implementiert werden (Redundanz).

15.7 C++ Beispiel

Das nachstehende Beispiel zeigt unterschiedliche Sicherungsstrategien, in Form unterschiedlicher Dateitypen.

15.7.1 C++ Klassendiagramm

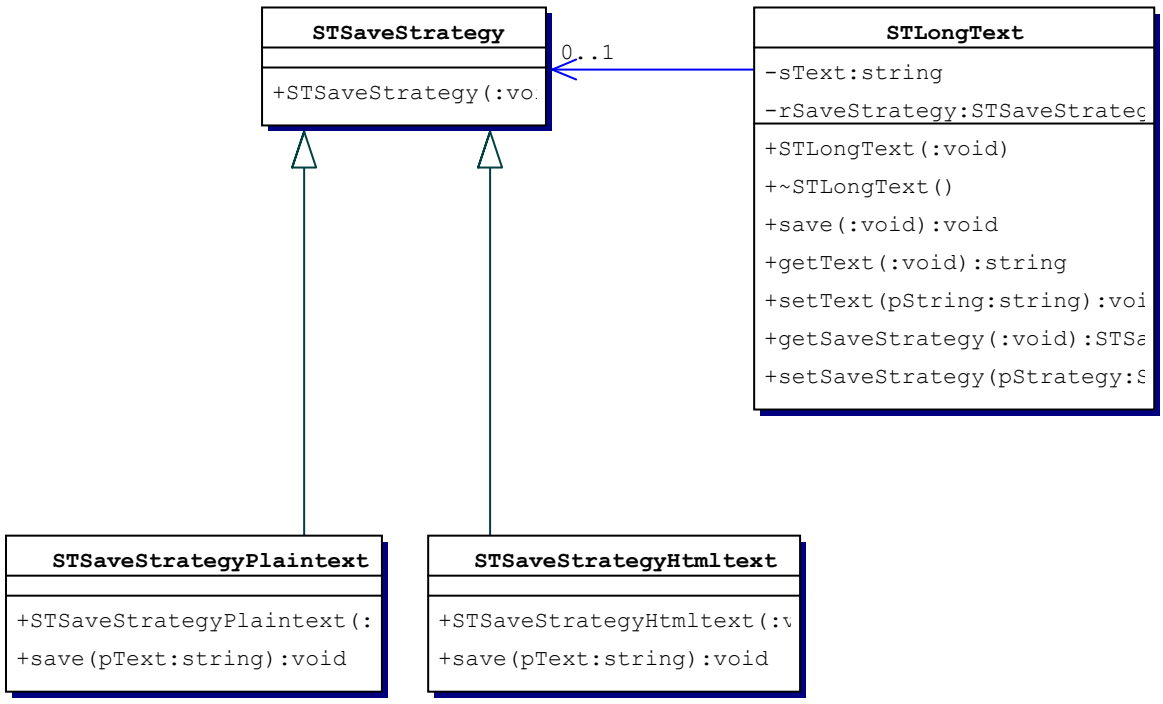

Abbildung 15-1 C++ Klassendiagramm Strategy

15.7.2 C++ Beispielprogramm

```
//=====================================================================
// C++ Strategy Designpattern - Strategy.cpp
//=====================================================================
#include "stdafx.h"
#include "STSaveStrategyPlaintext.h"
#include "STSaveStrategyHtmltext.h"
#include "STLongText.h"

using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
   STLongText              sText;

   sText.setText("Hallo, dies ist ein Test");
   sText.save();

   sText.setSaveStrategy(new STSaveStrategyPlaintext());
   sText.save();

   sText.setSaveStrategy(new STSaveStrategyHtmltext());
   sText.save();
   return 0;
}
```

```
//============================================================================
// C++ Strategy Designpattern - STLongText.h
//============================================================================
#ifndef _STLONGTEXT_H_
#define _STLONGTEXT_H_

#include <string>
#include "STSaveStrategy.h"

using namespace std;

class STLongText
{
   //-------------------------------------------------------------------------
   public:
      STLongText (void);
      ~STLongText();

      void            save            (void);
      string          getText         (void);
      void            setText         (string pString);
      STSaveStrategy* getSaveStrategy (void);
      void            setSaveStrategy (STSaveStrategy* pStrategy);

   //-------------------------------------------------------------------------
   private:
      string sText;
      STSaveStrategy* rSaveStrategy;
};

#endif
```

```
//============================================================================
// C++ Strategy Designpattern - STLongText.cpp
//============================================================================
#include "stdafx.h"
#include "STLongText.h"
#include <iostream>

using namespace std;

//----------------------------------------------------------------------------
STLongText::STLongText (void)
{
   rSaveStrategy = NULL;
}

//----------------------------------------------------------------------------
STLongText::~STLongText ()
{
   if (rSaveStrategy)
   {
      delete rSaveStrategy;
   }
}

//----------------------------------------------------------------------------
string STLongText::getText()
{
   return sText;
}
```

```
//-------------------------------------------------------------------------
void STLongText::setText(string pString)
{
   sText = pString;
}

//-------------------------------------------------------------------------
STSaveStrategy* STLongText::getSaveStrategy()
{
   return rSaveStrategy;
}

//-------------------------------------------------------------------------
void STLongText::setSaveStrategy(STSaveStrategy* pStrategy)
{
   if (rSaveStrategy)
   {
      delete rSaveStrategy;
   }
   rSaveStrategy = pStrategy;
}

//-------------------------------------------------------------------------
void STLongText::save()
{
   if (!rSaveStrategy)
   {
      cout << "Cannot save, no Save-Strategy chosen" << endl;
   }
   else
   {
      cout << "saving Strategy-Adress: " << rSaveStrategy << endl;
      rSaveStrategy->save(sText);
   }
}
```

```
//=========================================================================
// C++ Strategy Designpattern - STSaveStrategy.h
//=========================================================================
#ifndef _STSAVESTRATEGY_H_
#define _STSAVESTRATEGY_H_

#include <string>

using namespace std;

class STSaveStrategy
{
   //----------------------------------------------------------------------
   public:
      STSaveStrategy (void){};

      virtual void save (string pText) = NULL;
};

#endif
```

```
//=========================================================================
// C++ Strategy Designpattern - STSaveStrategyPlaintext.h
//=========================================================================
#ifndef _STSAVESTRATEGYPLAINTEXT_H_
```

```
#define _STSAVESTRATEGYPLAINTEXT_H_

#include "STSaveStrategy.h"
#include <string>

using namespace std;

class STSaveStrategyPlaintext : public STSaveStrategy
{
   //-------------------------------------------------------------------------
   public:
      STSaveStrategyPlaintext (void);

      virtual void save (string pText);
};

#endif
```

```
//=============================================================================
// C++ Strategy Designpattern - STSaveStrategyPlaintext.cpp
//=============================================================================
#include "stdafx.h"
#include <fstream>
#include "STSaveStrategyPlaintext.h"

using namespace std;

//-----------------------------------------------------------------------------
STSaveStrategyPlaintext::STSaveStrategyPlaintext (void)
{
}

//-----------------------------------------------------------------------------
void STSaveStrategyPlaintext::save (string pText)
{
   ofstream rOutput ("C:\\test.txt");
   rOutput << pText << endl;
   rOutput.close();
}
```

```
//=============================================================================
// C++ Strategy Designpattern - STSaveStrategyHtmltext.h
//=============================================================================
#ifndef _STSAVESTRATEGYHTMLTEXT_H_
#define _STSAVESTRATEGYHTMLTEXT_H_

#include "STSaveStrategy.h"
#include <string>

using namespace std;

class STSaveStrategyHtmltext : public STSaveStrategy
{
   //-------------------------------------------------------------------------
   public:
      STSaveStrategyHtmltext (void);

      virtual void save (string pText);
};

#endif
```

```
//============================================================================
// C++ Strategy Designpattern - STSaveStrategyHtmltext.cpp
//============================================================================
#include "stdafx.h"
#include <fstream>
#include "STSaveStrategyHtmltext.h"

using namespace std;

//----------------------------------------------------------------------------
STSaveStrategyHtmltext::STSaveStrategyHtmltext (void)
{
}

//----------------------------------------------------------------------------
void STSaveStrategyHtmltext::save (string pText)
{
   ofstream rOutput ("C:\\test.htm");

   rOutput << "<html>"     << endl;
   rOutput << "   <head>"  << endl;
   rOutput << "      <title>Strategy Test</title>" << endl;
   rOutput << "<meta http-equiv=\"Content-Type\" content=\"text/html; "
           << "charset=iso-8859-1\">" << endl;
   rOutput << "   </head>" << endl;
   rOutput << "   <body>"  << endl;
   rOutput << pText        << endl;
   rOutput << "   </body>" << endl;
   rOutput << "</html>"    << endl;

   rOutput.close();
}
```

15.8 Java Beispiel

Das nachstehende Beispiel zeigt unterschiedliche Sicherungsstrategien, in Form unterschiedlicher Dateitypen.

15.8.1 Java Klassendiagramm

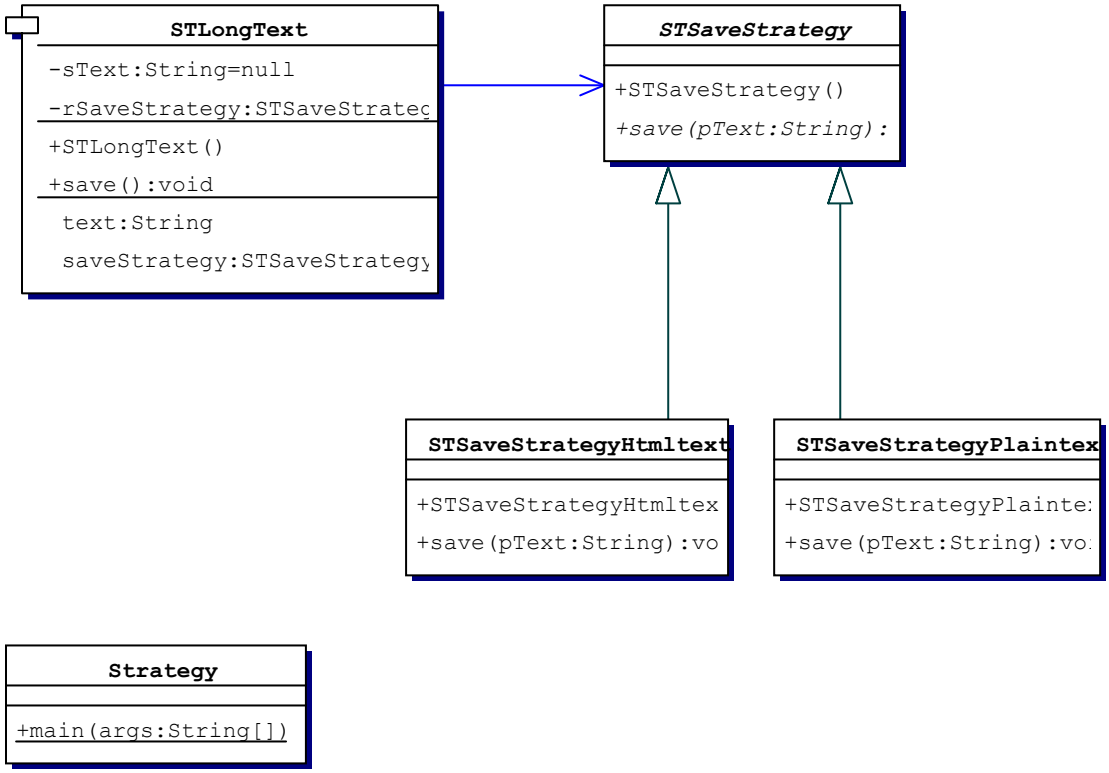

Abbildung 15-2 Java Klassendiagramm Strategy

15.8.2 Java Beispielprogramm

```
//=========================================================================
/** Java StrategyDesign Pattern - Testprogramm
 */
//=========================================================================
public class Strategy
{
   public static void main(String[] args)
   {
      STLongText sText = new STLongText();
      STSaveStrategyPlaintext rPlain = new STSaveStrategyPlaintext();
      STSaveStrategyHtmltext  rHtml  = new STSaveStrategyHtmltext();

      sText.setText("Hallo, dies ist ein Test");
      sText.save();

      sText.setSaveStrategy(rPlain);
      sText.save();

      sText.setSaveStrategy(rHtml);
      sText.save();
   }
}
```

```
//============================================================================
/** Java Singleton Design Pattern - Text, zu speicherndes Objekt
 */
//============================================================================
public class STLongText
{
   private String         sText         = null;
   private STSaveStrategy rSaveStrategy = null;

   //-------------------------------------------------------------------------
   public STLongText()
   {
      super();
   }

   //-------------------------------------------------------------------------
   public String getText()
   {
      return sText;
   }

   //-------------------------------------------------------------------------
   public void setText(String pString)
   {
      sText = pString;
   }

   //-------------------------------------------------------------------------
   public STSaveStrategy getSaveStrategy()
   {
      return rSaveStrategy;
   }

   //-------------------------------------------------------------------------
   public void setSaveStrategy(STSaveStrategy pStrategy)
   {
      rSaveStrategy = pStrategy;
   }

   //-------------------------------------------------------------------------
   public void save()
   {
      if (rSaveStrategy == null)
      {
         System.out.println ("Cannot save, no Save-Strategy chosen");
      }
      else
      {
         rSaveStrategy.save(sText);
      }
   }
}
```

```
//============================================================================
/** Java Singleton Design Pattern - Sicherungsstrategie,
 *  abstrakte Basisklasse
 */
//============================================================================
public abstract class STSaveStrategy
{
   //-------------------------------------------------------------------------
```

```
   public STSaveStrategy()
   {
      super();
   }

   //-------------------------------------------------------------------------
   public abstract void save(String pText);
}
```

```
//=============================================================================
/** Java Singleton Design Pattern - Sicherungsstrategie, sichern als
 *  einfacher ANSI-Text
 */
//=============================================================================
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class STSaveStrategyPlaintext extends STSaveStrategy
{
   //-------------------------------------------------------------------------
   public STSaveStrategyPlaintext()
   {
      super();
   }

   //-------------------------------------------------------------------------
   public void save(String pText)
   {
      try
      {
         FileWriter  rFileOut = new FileWriter("C:\\test.txt");
         PrintWriter rOut     = new PrintWriter(rFileOut);
         rOut.println(pText);
         rFileOut.close();
         System.out.println ("Saved as Plaintext");
      }
      catch (IOException e)
      {
         System.out.println(e);
      }
   }
}
```

```
//=============================================================================
/** Java Singleton Design Pattern - Sicherungsstrategie, sichern als HTML
 */
//=============================================================================
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class STSaveStrategyHtmltext extends STSaveStrategy
{
   //-------------------------------------------------------------------------
   public STSaveStrategyHtmltext()
   {
      super();
   }

   //-------------------------------------------------------------------------
```

```
   public void save(String pText)
   {
      try
      {
         FileWriter  rFileOut = new FileWriter("C:\\test.htm");
         PrintWriter rOut     = new PrintWriter(rFileOut);

         rOut.println("<html>");
         rOut.println("   <head>");
         rOut.println("      <title>Strategy Test</title>");
         rOut.println(
 "<meta http-equiv='Content-Type' content='text/html; charset=iso-8859-1'>"
         );
         rOut.println("   </head>");
         rOut.println("   <body>");
         rOut.println(pText);
         rOut.println(   "</body>");
         rOut.println("</html>");

         rFileOut.close();
         System.out.println ("Saved as HTML");
      }
      catch (IOException e)
      {
         System.out.println(e);
      }
   }
}
```

16. Builder

Alternativnamen: Erbauer
Mustergruppe: Erzeugungsmuster

16.1 Anmerkungen

Das Builder Pattern gehört zu den relativ einfachen aber vielseitigen Mustern.

16.2 Verwendungszweck

Das Builder Pattern wird verwendet um mit Hilfe polymorpher Klassen aus einem komplexen Objekt eine andere Darstellungsform des gleichen Objektes zu erzeugen, ohne das zum Erstellungszeitpunkt der Anwendung die Anzahl und Form des zu erstellenden Objektes festgelegt werden muss.
Typische Anwendungen für Builder Designpattern sind z.B.:

- Objektorientierte Konvertierungsprogramme, bei denen Graphiken von einem Format (z.B. JPEG) in ein anderes Format überführt werden (z.B. GIF oder Bitmap).
- Erweiterbare Sammlungen von Konvertierungsmodulen (Plugins), z.B. im Bereich Textverarbeitung, CAD, Graphik, 3D-Modeller etc.

16.3 Problemstellungen

Wie kann ein Programm ein Objekt von einer Repräsentationsform in eine andere Repräsentationsform überführt werden, ohne dass zum Zeitpunkt der Programmerstellung die Anzahl und der Aufbau der Quell- und/oder Ziel-Repräsentationen bekannt sein muss?
Wie kann ein Programm nachträglich um bestimmte Funktionalitäten, die einer bestimmten Schnittstellenspezifikation entsprechen, erweitert werden?

16.4 Lösung

Eine Instanz, die ein Objekt in einer bestimmten Repräsentationsform beinhaltet (z.B. eine Bitmap) wird mit einem Konvertierungsobjekt versehen, welches einer bestimmten Schnittstelle genügt, bzw. von einer (ggf. abstrakten) Basisklasse abgeleitet ist. Dieses Konvertierungsobjekt ist in der Lage mit Hilfe der in der Schnittstelle spezifizierten Methoden die vorhandene Repräsentation in eine Zielrepräsentation zu überführen.
Der Prozess mit Verlust behaftet sein kann (z.B. bei der Überführung eines HTML-Textes in einen ASCII-Text) ist er nicht zwangsläufig umkehrbar.

16.5 Vorteile

Die Anzahl der Konvertierungsmodule kann problemlos erweitert werden. Bei geschickter Wahl des Interfaces ist es nicht notwendig bereits zum Erstellungszeitpunkt des Hauptprogramms alle Zielrepräsentationen zu kennen.

16.6 Nachteile

Eine plugin-artige Erweiterung von Programmen ist in Java aber nur unter Nutzung der Reflection-API erreichbar. Diese gilt aber als grundsätzlich problematisch, da die Kapselung der Objekte (Data-Encapsulation) aufbricht. Reflection ist eigentlich nur für die Erstellung von Debuggern und RMI[15] entwickelt worden.

16.7 C++ Beispiel

Das nachstehende Beispiel zeigt das Builder Pattern anhand eines sehr einfachen Konvertierungsprogramms für Längeneinheiten.

16.7.1 C++ Klassendiagramm

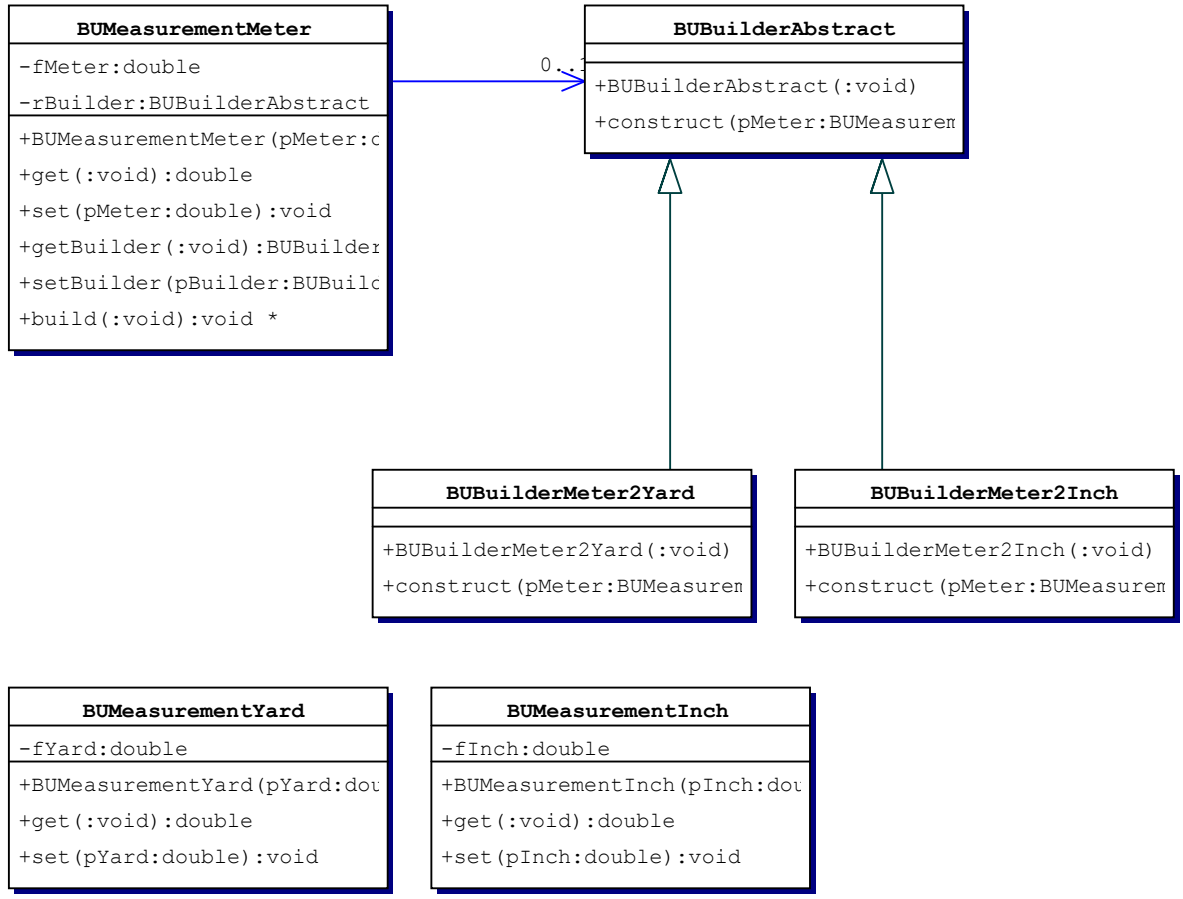

Abbildung 16-1 C++ Klassendiagramm Builder

16.7.2 C++ Beispielprogramm

```
//=====================================================================
// C++ Builder Designpattern - Builder.cpp
//=====================================================================
#include "stdafx.h"
#include <iostream>
#include "BUMeasurementMeter.h"
#include "BUMeasurementYard.h"
#include "BUMeasurementInch.h"
#include "BUBuilderMeter2Inch.h"
```

[15] RMI = Remote Messsage Invocation. Entfernter Fzunktionsaufruf einer Server-Methode durch einen Client.

```
#include "BUBuilderMeter2Yard.h"

using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
   BUMeasurementMeter rMeter(1.0);
   BUMeasurementYard  *rYard  = NULL;
   BUMeasurementInch  *rInch  = NULL;

   BUBuilderMeter2Inch rM2I;
   BUBuilderMeter2Yard rM2Y;

   rMeter.setBuilder(&rM2Y);
   rYard = (BUMeasurementYard *)rMeter.build();
   cout << "Yards: " << rYard->get() << endl;

   rMeter.setBuilder(&rM2I);
   rInch = (BUMeasurementInch *)rMeter.build();
   cout << "Inch: " << rInch->get() << endl;

   delete rYard;
   delete rInch;

   return 0;
}
```

```
//============================================================================
// C++ Builder Designpattern - BUMeasurementMeter.h
//============================================================================
#ifndef _BUMEASUREMENTMETER_H_
#define _BUMEASUREMENTMETER_H_

class BUBuilderAbstract;

class BUMeasurementMeter
{
   public:
      BUMeasurementMeter(double pMeter);

      double get(void);
      void   set(double pMeter);
      BUBuilderAbstract *getBuilder(void);
      void   setBuilder(BUBuilderAbstract *pBuilder);
      void  *build(void);

   private:
      double fMeter;
      BUBuilderAbstract *rBuilder;
};

#endif
```

```
//============================================================================
// C++ Builder Designpattern - BUMeasurementMeter.cpp
//============================================================================
#include "stdafx.h"
#include "BUMeasurementMeter.h"
#include "BUBuilderAbstract.h"

//----------------------------------------------------------------------------
```

```
BUMeasurementMeter::BUMeasurementMeter (double pMeter)
{
   rBuilder = NULL;
   set(pMeter);
}

//---------------------------------------------------------------------------
double BUMeasurementMeter::get(void)
{
   return fMeter;
}

//---------------------------------------------------------------------------
void BUMeasurementMeter::set(double pMeter)
{
   fMeter = pMeter;
}

//---------------------------------------------------------------------------
BUBuilderAbstract *BUMeasurementMeter::getBuilder(void)
{
   return rBuilder;
}

//---------------------------------------------------------------------------
void BUMeasurementMeter::setBuilder(BUBuilderAbstract *pBuilder)
{
   rBuilder = pBuilder;
}

//---------------------------------------------------------------------------
void *BUMeasurementMeter::build(void)
{
   void *rRC = NULL;
   if (rBuilder != NULL)
   {
      rRC = rBuilder->construct(this);
   }
   return rRC;
}
```

```
//===========================================================================
// C++ Builder Designpattern - BUMeasurementYard.h
//===========================================================================
#ifndef _BUMEASUREMENTYARD_H_
#define _BUMEASUREMENTYARD_H_

class BUMeasurementYard
{
   public:
      BUMeasurementYard(double pYard);

      double get(void);
      void   set(double pYard);

   private:
      double fYard;
};

#endif
```

```
//============================================================================
// C++ Builder Designpattern - BUMeasurementYard.cpp
//============================================================================
#include "stdafx.h"
#include "BUMeasurementYard.h"

//---------------------------------------------------------------------------
BUMeasurementYard::BUMeasurementYard(double pYard)
{
   set(pYard);
}

//---------------------------------------------------------------------------
double BUMeasurementYard::get()
{
   return fYard;
}

//---------------------------------------------------------------------------
void BUMeasurementYard::set(double pYard)
{
   fYard = pYard;
}
```

```
//============================================================================
// C++ Builder Designpattern - BUMeasurementInch.h
//============================================================================
#ifndef _BUMEASUREMENTINCH_H_
#define _BUMEASUREMENTINCH_H_

class BUMeasurementInch
{
   public:
      BUMeasurementInch(double pInch);

      double get(void);
      void   set(double pInch);

   private:
      double fInch;
};

#endif
```

```
//============================================================================
// C++ Builder Designpattern - BUMeasurementInch.cpp
//============================================================================
#include "stdafx.h"
#include "BUMeasurementInch.h"

//---------------------------------------------------------------------------
BUMeasurementInch::BUMeasurementInch(double pInch)
{
   set(pInch);
}

//---------------------------------------------------------------------------
double BUMeasurementInch::get(void)
{
   return fInch;
}
```

```
//---------------------------------------------------------------------
void BUMeasurementInch::set(double pInch)
{
   fInch = pInch;
}
```

```
//=====================================================================
// C++ Builder Designpattern - BUBuilderAbstract.h
//=====================================================================
#ifndef _BUMBUILDERABSTRACT_H_
#define _BUMBUILDERABSTRACT_H_

class BUMeasurementMeter;

class BUBuilderAbstract
{
   public:
      BUBuilderAbstract(void);
      virtual void *construct(BUMeasurementMeter *pMeter);
};

#endif
```

```
//=====================================================================
// C++ Builder Designpattern - BUBuilderAbstract.cpp
//=====================================================================
#include "stdafx.h"
#include "BUBuilderAbstract.h"

//---------------------------------------------------------------------
BUBuilderAbstract::BUBuilderAbstract(void)
{
}

//---------------------------------------------------------------------
 void *BUBuilderAbstract::construct(BUMeasurementMeter *pMeter)
{
   return NULL;
}
```

```
//=====================================================================
// C++ Builder Designpattern - BUBuilderMeter2Inch.h
//=====================================================================
#ifndef _BUMBUILDERMETER2INCH_H_
#define _BUMBUILDERMETER2INCH_H_

#include "BUMeasurementMeter.h"
#include "BUBuilderAbstract.h"

class BUBuilderMeter2Inch : public BUBuilderAbstract
{
   public:
      BUBuilderMeter2Inch(void);
      virtual void *construct(BUMeasurementMeter *pMeter);
};

#endif
```

```
//=====================================================================
```

```
// C++ Builder Designpattern - BUBuilderMeter2Inch.cpp
//=============================================================================
#include "stdafx.h"
#include "BUBuilderMeter2Inch.h"
#include "BUMeasurementInch.h"
#include "BUMeasurementMeter.h"

//-----------------------------------------------------------------------
BUBuilderMeter2Inch::BUBuilderMeter2Inch(void)
{
}

//-----------------------------------------------------------------------
void *BUBuilderMeter2Inch::construct(BUMeasurementMeter *pMeter)
{
   BUMeasurementInch *rInch =
     new BUMeasurementInch(pMeter->get() * 100 / 2.54);
   return rInch;
}
```

```
//=============================================================================
// C++ Builder Designpattern - BUBuilderMeter2Yard.h
//=============================================================================
#ifndef _BUMBUILDERMETER2YARD_H_
#define _BUMBUILDERMETER2YARD_H_

#include "BUMeasurementMeter.h"
#include "BUBuilderAbstract.h"

class BUBuilderMeter2Yard : public BUBuilderAbstract
{
   public:
      BUBuilderMeter2Yard(void);
      virtual void *construct(BUMeasurementMeter *pMeter);
};

#endif
```

```
//=============================================================================
// C++ Builder Designpattern - BUBuilderMeter2Yard.cpp
//=============================================================================
#include "stdafx.h"
#include "BUBuilderMeter2Yard.h"
#include "BUMeasurementYard.h"
#include "BUMeasurementMeter.h"

//-----------------------------------------------------------------------
BUBuilderMeter2Yard::BUBuilderMeter2Yard(void)
{
}

//-----------------------------------------------------------------------
void *BUBuilderMeter2Yard::construct(BUMeasurementMeter *pMeter)
{
   BUMeasurementYard *rYard = new BUMeasurementYard(pMeter->get() / 0.9);
   return rYard;
}
```

16.7.3 Anmerkungen zum C++ Beispiel

Wie anhand des Beispiels ersichtlich, ist es nicht notwendig, dass die verschiedenen Repräsentationsformen von einer gemeinsamen Basisklasse abgeleitet werden, allerdings muss man sich hier des nicht typsicheren Weges über einen ***void** ** Pointer bedienen.

16.8 Java Beispiel

Das nachstehende Beispiel zeigt das Builder Pattern anhand eines sehr einfachen Konvertierungsprogramms für Längeneinheiten.

16.8.1 Java Klassendiagramm

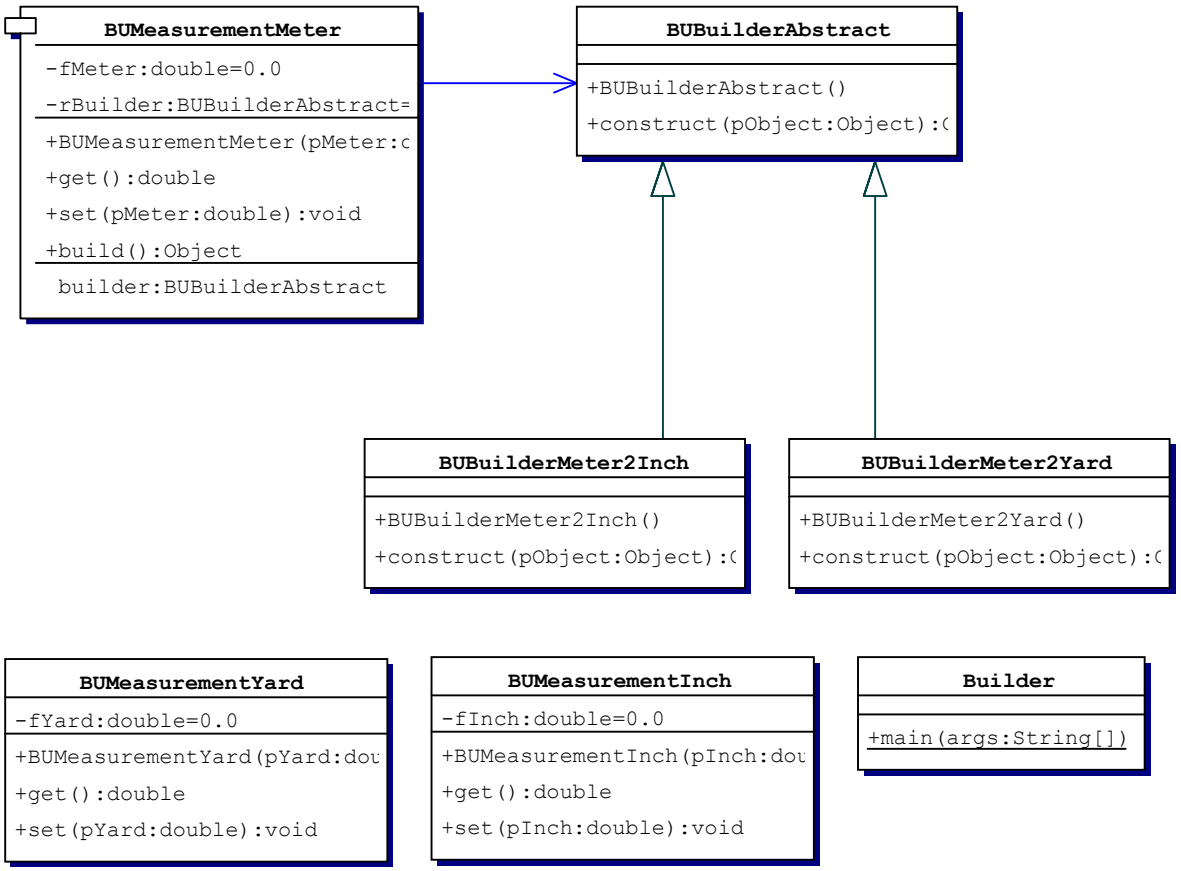

Abbildung 16-2 Java Klassendiagramm Builder

16.8.2 Java Beispielprogramm

```
//======================================================================
/** Java Builder Design Pattern - Testprogramm
 *  Created on 11.12.2003
 */
//======================================================================
public class Builder
{
   //--------------------------------------------------------------------
   // Hauptprogramm
   //--------------------------------------------------------------------
   public static void main(String[] args)
   {
      BUMeasurementMeter rMeter = new BUMeasurementMeter(1.0);
      BUMeasurementYard  rYard  = null;
      BUMeasurementInch  rInch  = null;

      BUBuilderMeter2Inch rM2I  = new BUBuilderMeter2Inch();
      BUBuilderMeter2Yard rM2Y  = new BUBuilderMeter2Yard();

      rMeter.setBuilder(rM2Y);
```

```
      rYard = (BUMeasurementYard)rMeter.build();
      System.out.println("Yards: " + rYard.get());

      rMeter.setBuilder(rM2I);
      rInch = (BUMeasurementInch)rMeter.build();

      System.out.println("Inch: " + rInch.get());
   }
}
```

```
//============================================================================
/** Java Builder Design Pattern - Maßeinheit, enthält Angaben in Inch/Zoll
 *  Created on 11.12.2003
 */
//============================================================================
public class BUMeasurementInch
{
   private double fInch = 0.0;

   //----------------------------------------------------------------------
   public BUMeasurementInch(double pInch)
   {
      super();
      set(pInch);
   }

   //----------------------------------------------------------------------
   public double get()
   {
      return fInch;
   }

   //----------------------------------------------------------------------
   public void set(double pInch)
   {
      fInch = pInch;
   }
}
```

```
//============================================================================
/** Java Builder Design Pattern - Maßeinheit, enthält Angaben in Yards
 *  Created on 11.12.2003
 */
//============================================================================
public class BUMeasurementYard
{
   private double fYard = 0.0;

   //----------------------------------------------------------------------
   public BUMeasurementYard(double pYard)
   {
      super();
      set(pYard);
   }

   //----------------------------------------------------------------------
   public double get()
   {
      return fYard;
   }
```

```
   //-------------------------------------------------------------------------
   public void set(double pYard)
   {
      fYard = pYard;
   }
}
```

```
//=============================================================================
/** Java Builder Design Pattern - Maßeinheit, enthält Angaben in Metern
 *  Created on 11.12.2003
 */
//=============================================================================
public class BUMeasurementMeter
{
   private double            fMeter   = 0.0;
   private BUBuilderAbstract rBuilder = null;

   //-------------------------------------------------------------------------
   public BUMeasurementMeter (double pMeter)
   {
      super();
      set(pMeter);
   }

   //-------------------------------------------------------------------------
   public double get()
   {
      return fMeter;
   }

   //-------------------------------------------------------------------------
   public void set(double pMeter)
   {
      fMeter = pMeter;
   }

   //-------------------------------------------------------------------------
   public BUBuilderAbstract getBuilder()
   {
      return rBuilder;
   }

   //-------------------------------------------------------------------------
   public void setBuilder(BUBuilderAbstract pBuilder)
   {
      rBuilder = pBuilder;
   }

   //-------------------------------------------------------------------------
   public Object build()
   {
      Object rRC = null;
      if (rBuilder != null)
      {
         rRC = rBuilder.construct(this);
      }
      return rRC;
   }
}
```

```
//=============================================================================
```

```
/** Java Builder Design Pattern - anbstrakte Basisklasse
 *  Konvertierungsobjekt
 *  Created on 11.12.2003
 */
//============================================================================
public class BUBuilderAbstract
{
   //------------------------------------------------------------------------
   public BUBuilderAbstract()
   {
      super();
   }

   //------------------------------------------------------------------------
   public Object construct(Object pObject)
   {
      return null;
   }
}
```

```
//============================================================================
/** Java Builder Design Pattern - Konvertierungsobjekt, Meter->Inch
 *  Created on 11.12.2003
 */
//============================================================================
public class BUBuilderMeter2Inch extends BUBuilderAbstract
{
   //------------------------------------------------------------------------
   public BUBuilderMeter2Inch()
   {
      super();
   }

   //------------------------------------------------------------------------
   public Object construct(Object pObject)
   {
      BUMeasurementInch rInch = null;
      if (pObject instanceof BUMeasurementMeter)
      {
         BUMeasurementMeter rMeter = (BUMeasurementMeter)pObject;
         rInch = new BUMeasurementInch(rMeter.get() * 100 / 2.54);
      }
      return rInch;
   }
}
```

```
//============================================================================
/** Java Builder Design Pattern - Konvertierungsobjekt, Meter->Yard
 *  Created on 11.12.2003
 */
//============================================================================
public class BUBuilderMeter2Yard extends BUBuilderAbstract
{
   //------------------------------------------------------------------------
   public BUBuilderMeter2Yard()
   {
      super();
   }

   //------------------------------------------------------------------------
   public Object construct(Object pObject)
```

```
   {
      BUMeasurementYard rYard = null;
      if (pObject instanceof BUMeasurementMeter)
      {
         BUMeasurementMeter rMeter = (BUMeasurementMeter)pObject;
         rYard = new BUMeasurementYard(rMeter.get() / 0.9);
      }
      return rYard;
   }
}
```

16.8.3 Anmerkungen zum Java Beispiel

Wie anhand des Beispiels ersichtlich, ist es nicht notwendig, dass die verschiedenen Repräsentationsformen von einer gemeinsamen Basisklasse abgeleitet werden.

17. Abstract Factory

Alternativnamen: Abstrakte Fabrik, Kit
Mustergruppe: Erzeugungsmuster

17.1 Anmerkungen

Unter dem allgemeinen Stichwort Factory werden zwei leicht verschiedene Patterns geführt, die Factory Method und die Abstract Factory.

17.2 Verwendungszweck

Das Abstract Factory Pattern kapselt die Entscheidung, welche konkrete Factory Methode im Einzelfall verwendet werden soll.
Typische Anwendungen für das Abstract Factory Designpattern sind z.B.:

- Die Bereitstellung von Dialog-Elementen in aktuellen Betriebssystemen. Je nach Auswahl des Anwenders werden dabei die Oberflächen-Elemente unterschiedlich dargestellt. In Windows ist dieses unter dem Begriff „Themes“ oder „Skins“ gängige Praxis.
- Die Klassen zum Look and Feel von Swing (Java), bei denen je nach Anforderung und eingestelltem Look and Feel unterschiedliche Objekte gleicher Funktionalität erzeugt werden. Alle diese Elemente erben von einer gemeinsamen Basisklasse „Component“. Jede Look and Feel Factory kann dabei die gleichen Elemente erzeugen (Buttons, Checkboxen usw.) – auch die Funktionalität ist grundsätzlich gleich, allein die Bedienung und das Aussehen unterscheiden sich.

Die Ähnlichkeit zum Factory Method Pattern ist deutlich, nur dass die von der Fabrik erzeugten und verwalteten Instanzen jeweils ganze Fabriken darstellen.
Da eine Abstract Factory Klasse beliebig viele Fabriken erzeugen und bereitstellen kann ist es üblicherweise unnötig mehr als eine Abstract Factory Instanz zu erzeugen. Die Abstract Factory wird daher zumeist unter Verwendung des Singleton Patterns (⇨ Singleton) erzeugt.

17.3 Problemstellungen

Wie kann man mit wenig Aufwand ganze Hierarchien oder Gruppen von Objekttypen austauschen, ohne dass das Programm diese kennen muss.

17.4 Lösung

Objekte, welche das Factory Method Pattern realisieren werden in einer weiteren Factory erzeugt und verwaltet. Letzteres ist kein Problem, da die erzeigten Fabriken üblicherweise Singletons sind.

17.5 Vorteile

Sie genauen Eigenschaften der fabrizierten Objekte (Fabriken) werden erst durch die konkrete Fabrik festgelegt, während die abstrakte Fabrik die davon unabhängige Erstellung des Programms erlaubt.

17.6 Nachteile

Ein erhöhter Overhead bei der Verwaltung durch die zusätzliche Abstraktionsebene. Mögliche Designunterschiede müssen vielfach in Form von Berechnungen und Parametrisierung vorausgedacht werden.

17.7 Beispiele

Die Realisierung einer Abstrakten Fabrik ist identisch mit der Programmierung eines Factory Method Patterns.

18. Anhänge

18.1 Patternkatalog

Patternnamen	Mustertyp	Verzeichnet unter	Kapitel
Abstract Factory	Erzeugungsmuster		Kap. 17
Abstrakte Fabrik	Erzeugungsmuster	→ Abstract Factory	Kap. 17
Action	Verhaltensmuster	→ Command	Kap. 13
Adapter	Strukturmuster		Kap. 5
Aktion	Verhaltensmuster	→ Command	Kap. 13
Ambassador	Strukturmuster	→ Proxy	Kap. 10
Befehl	Verhaltensmuster	→ Command	Kap. 13
Body	Strukturmuster	→ Bridge	Kap. 9
Botschafter	Strukturmuster	→ Proxy	Kap. 10
Bridge	Strukturmuster		Kap. 9
Brücke	Strukturmuster	→ Bridge	Kap. 9
Builder	Erzeugungsmuster		Kap. 16
Command	Verhaltensmuster		Kap. 13
Composite	Strukturmuster		Kap. 12
Decorator	Strukturmuster		Kap. 6
Dekorierer	Strukturmuster	→ Decorator	Kap. 6
Erbauer	Erzeugungsmuster	→ Builder	Kap. 16
Fabrikmethode	Erzeugungsmuster	→ Factory Method	Kap. 11
Façade	Strukturmuster		Kap. 7
Factory Method	Erzeugungsmuster		Kap. 11
Fassade	Strukturmuster	→ Façade	Kap. 7
Fliegengewicht	Strukturmuster	→ Flyweight	Kap. 14
Flyweight	Strukturmuster		Kap. 14
Handle	Strukturmuster	→ Bridge	Kap. 9
Kit	Erzeugungsmuster	→ Abstract Factory	Kap. 17
Kompositum	Strukturmuster	→ Composite	Kap. 12
Platzhalter	Strukturmuster	→ Proxy	Kap. 10
Policy	Verhaltensmuster	→ Strategy	Kap. 15
Prototyp	Erzeugungsmuster	→ Prototype	Kap. 12
Prototype	Erzeugungsmuster		Kap. 12
Proxy	Strukturmuster		Kap. 10
Schablonenmethode	Verhaltensmuster	→ Template Method	Kap. 3
Singleton	Erzeugungsmuster		Kap. 4
Stellvertreter	Strukturmuster	→ Proxy	Kap. 10
Strategie	Verhaltensmuster	→ Strategy	Kap. 15
Strategy	Verhaltensmuster		Kap. 15
Surrogat	Strukturmuster	→ Proxy	Kap. 10
Template Method	Verhaltensmuster		Kap. 3
Transaction	Verhaltensmuster	→ Command	Kap. 13
Transaktion	Verhaltensmuster	→ Command	Kap. 13
Umwickler	Strukturmuster	→ Adapter	Kap. 5
Umwickler (gebunden)	Strukturmuster	→ Decorator	Kap. 6
Virtual Constructor	Erzeugungsmuster	→ Factory Method	Kap. 11
Virtueller Konstruktor	Erzeugungsmuster	→ Factory Method	Kap. 11
Wrapper	Strukturmuster	→ Adapter	Kap. 5
Wrapper (bounded)	Strukturmuster	→ Decorator	Kap. 6

18.2 Pattern Kurzbeschreibung

Abstract Factory **(Erzeugung)**

Erzeugung von Objekten einer bestimmten Hierarchie auf Anfrage, so dass die Applikation die Details der Implementation abgeleiteter Klassen nicht kennen muss.

Adapter **(Struktur)**

Wird verwendet, um das Interface einer Klasse auf ein anderes Interface abzubilden Die ursprüngliche Schnittstelle des Objektes wird üblicherweise vollständig verborgen.

Bridge **(Struktur)**

Wird verwendet, um das Interface zum Model konstant zu halten, während die Darstellung variiert. Die notwendige Unterscheidung erfolgt in der Controlschicht.

Builder **(Erzeugung)**

Verbindet den In halt mehrerer Einzelobjekte teilweise oder insgesamt zu einem neuen Objekt.

Command **(Verhalten)**

Kapselt einen Befehl als Objekt, welches vom Empfänger interpretiert und ggf. ignoriert wird. Dadurch wird die Kommunikation der Applikation mit Darstellungsinhalten stark vereinfacht, da die Applikation weniger Kenntnisse über die dargestellten Objekte benötigt.

Composite **(Struktur)**

Sammlung von Objekten, bei denen jedes entweder ein Primitiv oder wiederum ein Composite ist (Baumstruktur).

Decorator **(Struktur)**

Wird verwendet um die Schnittstelle eines Objektes um neue Methoden zu erweitern. Die ursprüngliche Schnittstelle des Objektes bleibt vollständig erhalten.

Facade **(Struktur)**

Wird verwendet, um die Komplexität eines unterliegenden Subsystems zu verbergen. Dies geschieht in Form einer vereinfachten Schnittstelle.

Factory **(Erzeugung)**

Wählt unter mehreren ähnlichen Instanzen eine bestimmte aus. Ist dieses nicht vorhanden, so wird es erzeugt, ist es vorhanden, wird diese zurückgegeben. (Beispiel: Font-Factory)

Flyweight **(Struktur)**

Dient zur Verringerung der Anzahl kleiner, statischer Teilobjekte, indem diese aus dem eigentlichen Objekt ausgelagert und mehrfach verwendet werden.

Prototype **(Erzeugung)**

Erzeugung einer neuen Instanz nach einem Vorbild. Dadurch muss der Typ des Objektes am Ort der Erzeugung nicht bekannt sein. Geschieht durch Kopieren des Vorbildobjektes. Wird auch verwendet, wenn die Neukonstruktion eines Objektes teurer ist, als das Kopieren.

Proxy **(Struktur)**

(Virtual) Wird verwendet um ein in Hinsicht auf Platz/Zeit sehr „teures“ Objekt erst dann zu instanziieren, wenn es wirklich benötigt wird.
(Security) Bezeichnet eine Zwischenschicht, die Berechtigungen prüft und die Aktionen des Anwenders ggf. blockiert. Dadurch bleiben sowohl View, Control wie auch Model konstant und die Berechtigung zu bestimmten Handlungen wird aus der eigentlichen Applikation extrahiert.
(Remote) Verbirgt das technische Protokoll einer Client/Server-Anwendung, durch Repräsentation der Serverfunktionalität als Stellvertreterobjekt.

Singleton **(Erzeugung)**

Stellt sicher, dass von einem bestimmten Typ nur eine oder eine limitierte Anzahl von Instanzen existieren kann, auf welche dann global zugegriffen wird.

Strategy **(Verhalten)**

Definiert eine Familie ähnlicher Algorithmen und kapselt diese in Objekten. Einem verwendenden Objekt werden die Algorithmen als ausgewählte Instanz zugewiesen.

Template Method **(Verhalten)**

Spaltet verwandte Algorithmen in einzelne Teile auf, um die allgemeingültigen Anteile nur einmal zu implementieren, bzw. ein Gerüst für die Erstellung eigener Spezialisierungen zu liefern.

18.3 Abbildungsverzeichnis