

# 1 Einführung

## 1.1 Motivation

Das dritte Jahrtausend hat begonnen und damit auch ein neues Zeitalter in der Informationstechnologie. Rasante Entwicklungen sind zu erwarten, da mit dem neuen Jahrtausend auch die Ära nach dem Y2K-Problem begonnen hat. Durch dieses waren viele Kapazitäten gebunden, die sich mit einem zwar einfachen, aber aufwendigen Erweitern von Datumsformaten auseinandersetzen mussten. Wie wenig kreativ die Behebung des Y2K-Problems war, zeigt beispielsweise folgender, wenn auch nicht ganz ernst gemeinter Brief eines SW-Entwicklungs-Chefs (um den Sinn wortgetreu wiederzugeben, bleibt er unübersetzt):

*Y2K-Sinnhaftigkeit*

### *Y2K-Problem*

*Dear Boss,*

*our staff has completed the 18 months of work on time and on budget. We have gone through every line of code in every program in every system. We have analysed all databases, all data files, including backups and historic archives and modified all data to reflect the change.*

*Y2K wörtlich genommen*

*We are proud to report that we have completed the »Y2K« date change mission, and have now implemented all changes to all programs and all data to reflect your new standards: Januark, Februark, March, April, Mak, June, Julk, August, September, October, November, December. As well as: Sundak, Mondak, Tuesdak, Wendsdak, Thursdak, Fridak, Saturdak*

*I trust that this is satisfactory, because to be honest, none of this »Y to K« problem has made any sense to me. But I understand it is a global problem, and our team is glad to help in any way possible.*

*And what does the year 2000 have to do with it? Speaking of which, what do you think we ought to do next year when the two digit year rolls over from 99 to 00? We'll await your direction.*

*Joan Duh*

*Snr Programmer*

Aber so wenig Fortschritt dieses Ändern der Programme war, so viel Kapazitäten band es, dies umso mehr, da in viele Programme noch zusätzlich eine neue Währung, der Euro, eingebaut werden musste. Dadurch war es für Programmierer – insbesondere für solche der alten Schule – einfach, einen geeigneten Arbeitsplatz zu finden. Da besonders alte Programme umgeschrieben werden mussten und diese Programme meist in COBOL programmiert waren, gab es für COBOL-Programmierer wieder gute Arbeitsmöglichkeiten, obwohl noch vor einigen Jahren aufgrund des Durchbruchs neuerer komfortabler Programmiersprachen die Lage eher schlecht aussah. Aber nicht nur bestehende COBOL-Programmierer wurden durch das Y2K-Problem wieder eingesetzt, sondern sogar neu aufgenommene Programmierer wurden in COBOL geschult. Aber wie sagten schon Experten dazu: Vor dem 1. Januar werden die Softwareentwickler gut am Y2K-Problem verdienen, nach dem 1. Januar nur mehr die Rechtsanwälte und diese sogar sehr gut.

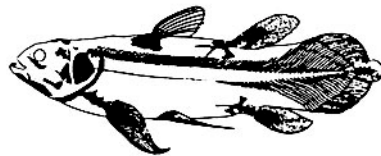
*Berufsaussicht*

In dieser neuen Ära nach dem Y2K-Problem kann nun die volle Konzentration auf Fortschritt und Weiterentwicklung gelegt werden. Natürlich werden damit auch in der Informationstechnologie noch genügend Arbeitskräfte und damit auch Programmierer gesucht, es wird aber vermehrt Kreativität und Leistung und nicht stumpfes »Dahinprogrammieren« gefordert werden. Darüber hinaus sind besonders neue Technologien, wie Internet oder WAP, im Vormarsch. Dadurch wird es für jetzige COBOL-Programmierer nötig sein, sich weiterzuentwickeln. Eine Studie von JOBSTATS über den Arbeitsmarkt vom September 2000 (<http://jobstats.jobet.com.au>) zeigt etwa, dass nur mehr bei 1,41% aller offenen Programmierstellen die Sprache COBOL gewünscht wird. Dies ist deutlich weniger als etwa bei Java (15,27%) oder C++ (13,91%), obwohl drei Millionen COBOL-Programmierer weltweit vorhanden wären.

*Alternativen:  
Der Quastenflosser  
hat auch überlebt*

Durch die großen Vorteile der objektorientierten Programmierung, die im nächsten Abschnitt behandelt werden, ist es auch verständlich, dass derartige Sprachen eher gewünscht sind. Daher ist es sehr wichtig, sich diese Technik anzueignen. Ein Vorhaben, das relativ einfach ist, der Leser ist keine 300 Seiten davon entfernt. Selbstverständlich wird es aber trotzdem eine gewisse Anstrengung benötigen und so mancher wird diese scheuen und sich fragen, ob es Alternativen gibt. Natürlich

gibt es die: Um einen Vergleich aus der Natur zu bringen, die Dinosaurier sind zwar ausgestorben, die Quastenflosser (eine Tierart, die immer hin 250 Millionen Jahre älter ist) leben aber immer noch. Somit werden wohl auch einige COBOL-Programmierer überleben, ohne sich umstellen zu müssen, wobei diese Alternative in Abbildung 1-1 zu sehen ist. Es bleiben dabei aber natürlich zwei Fragen:



**Abb. 1-1**

*Der Quastenflosser:  
eine Alternative für  
COBOL-Anwender?*

Erstens, gehört man wirklich zu den besten COBOL-Freaks, die überleben werden, und zweitens, ist das Leben als Quastenflosser wirklich so erstrebenswert? Alle jene, die sich auch nur bei einer der beiden Fragen nicht ganz sicher sind, ob sie mit einem klaren JA beantwortet werden kann, sollten weiterlesen. Im Rest dieses Kapitels werden die Vorteile der objektorientierten Programmierung erklärt. Außerdem wird darauf eingegangen, welche Vorbereitungen nötig sind, wie dieser Kurs aufgebaut ist, und für alle, die noch Zweifel haben, wird ein Beispiel aus der Praxis gebracht, das einerseits zeigt, dass ein Umstieg von COBOL auf objektorientierte Programmierung sinnvoll ist, und andererseits, dass das in diesem Buch gezeigte Konzept bereits erfolgreich angewandt wurde.

## 1.2 Vorteile der neuen Techniken

Die wichtigsten Vorteile der objektorientierten Programmierung sind:

- **Wiederverwendung:** Die objektorientierte Programmierung ermöglicht es, dass ein einmal geschriebenes Programmstück später, auch in ganz anderen Programmen, wiederverwendet werden kann. Durch diese Technik kann der Entwicklungsaufwand reduziert werden. Ein einfaches Beispiel wäre etwa eine Buchungsfunktionalität, die nur einmal geschrieben werden muss und von allen Programmen verwendet wird.
- **Erweiterbarkeit:** Bestehende Programmteile können aber nicht nur wiederverwendet werden, sie können durch die in Abschnitt 4.2 vorgestellte Vererbung auch erweitert werden, ohne dass die bestehenden Programmteile oder Programme, die diese bereits verwenden, davon irgendwie betroffen wären. Ein einfaches Beispiel dafür

*Vorteile von OOP*

wäre etwa, dass beim Programmieren einer österreichischen Lohnsoftware viele Teile aus einer bestehenden deutschen Lohnverrechnungssoftware wiederverwendet werden könnten.

- Klassenbibliotheken: Die Möglichkeit der Wiederverwendung führt so weit, dass die wichtigsten Programmteile in Bibliotheken zusammengefasst werden. Aus diesen kann dann der Programmierer die gewünschten Programmteile auswählen und entweder direkt verwenden oder diese noch durch Vererbung erweitern, ohne dass die bestehenden Programmteile verändert werden müssen.
- Einheitlichkeit der Programme: Durch das Verwenden von Klassenbibliotheken wird aber nicht nur der Entwicklungsaufwand verringert, sondern es werden auch die neu entwickelten Programme einheitlich gestaltet. Stellt beispielsweise die Klassenbibliothek Objekte zur Gestaltung einer grafischen Benutzeroberfläche zur Verfügung, so sehen natürlich die mit Hilfe dieser Objekte neu geschriebenen Programme ähnlich aus. Nicht nur was das Erscheinungsbild, sondern auch was den Quellcode betrifft.

#### *OO-Techniken*

Diese Vorteile wären wohl schon Grund genug, auf eine objektorientierte Programmiersprache umzusteigen, aber um diesen Umstieg möglichst effizient zu machen, ist es sinnvoll, sich noch einige weitere vorteilhafte Techniken anzueignen, die auch in diesem Buch behandelt werden. Beispielfhaft dafür sind folgende erwähnt:

- Objektorientierte Modellierung: Im Regelfall steht am Anfang eines Projektes immer eine sehr komplexe, schwer zu verstehende Situation. Der objektorientierte Entwurf mit seinem ganzheitlichen Ansatz, bei welchem statische Struktureigenschaften und dynamische Zustandsänderungen berücksichtigt werden, hilft, diese komplexe Situation in einem einfachen Modell darzustellen.
- Verwendung einer Datenbank: Datenbanken sind eine Erweiterung des Prinzips der indexsequenziellen Dateien. Sie haben den Vorteil, dass sie viele zusätzliche Funktionalitäten, wie etwa Recovery (Garantiestellung, dass auch im Falle eines Programmabsturzes der Inhalt der Datenbank konsistent bleibt) oder SQL (eine einfache Abfragesprache), anbieten. Zusätzlich ermöglichen sie auch, dass der Endbenutzer sich einfache Auswertungen selbst schreiben kann, wodurch eventuelle individuelle Anpassungen nicht mehr nötig sind.

### 1.3 Aufbau des Buches

Grundsätzlich sollte dies genügend Motivation sein, um voll durchzustarten, und es wäre durchaus auch denkbar, sofort mit dem nächsten Kapitel und damit mit dem eigentlichen Lernstoff weiterzumachen. An dieser Stelle wird aber noch kurz die Gliederung dieses Buches vorgestellt. Ganz wichtig ist dabei, dass hier versucht wird, die objektorientierte Programmierung zu trainieren und nicht eine bestimmte Entwicklungsumgebung. Diese sind zwar meist sehr mächtig und schon in kürzester Zeit können schöne Programme erzeugt werden (manchmal sogar ohne auch nur eine einzige Zeile Code zu schreiben), aber dies birgt für den Anfänger auch gewisse Gefahren. »Erzeugen« ist dabei in der Tat das bessere Wort als Programmieren, denn mit Programmieren hat dies nur wenig zu tun, im Wesentlichen sind nur einige Mausklicks notwendig. Dennoch existieren viele Bücher, die genau mit diesen Mausklicks beginnen. Das Problem dabei ist, dass die dadurch entstehenden Anwendungen zwar sehr schön sind, in der Praxis aber irgendwann (und meistens sehr bald) nicht mehr ausreichen werden. Es werden Veränderungen und Erweiterungen notwendig sein und an dieser Stelle reichen einige Mausklicks nicht mehr aus, sondern es muss auf das Basiswissen zurückgegriffen werden, da Code verändert und auch produziert werden muss. Perfektion ist dabei verlangt und diese muss auch in der objektorientierten Programmierung erst erarbeitet werden. Genau diese Perfektion wird in diesem Buch vermittelt. Dabei wird es vielleicht anfangs ein eher steiniger Weg, weil das erste Ergebnis nicht sofort ein tolles Windows-Programm ist, aber das Endergebnis – die erlernte Perfektion – wird sicherlich dafür entschädigen.

Wenn die Welt von COBOL und der objektorientierten Programmierung verglichen werden, dann fallen einige grundsätzliche Unterschiede auf, die in Tabelle 1-1 dargestellt werden.

COBOL	Objektorientierte Programmierung
Begriffe (OCCURS, PERFORM, ...)	Begriffe (Records, Arrays, ...)
Statische Datentypen	Dynamische Datentypen
Typen	Klassen
(Index-)sequenzielle Dateien	Datenbanken
Terminallösungen	Grafische Benutzeroberflächen

*Perfektion ist gesucht!  
Dieses Buch hilft sie zu  
erlernen.*

**Tab. 1-1**  
*Unterschiede  
COBOL – OOP*

Diese Unterschiede zwischen COBOL und OOP dienen als Grundlage für den folgenden OOP-Kurs, denn jedem ist ein Kapitel gewidmet. Aufbauend auf diese grundlegenden Unterschiede zwischen COBOL

und OOP ist der Lernstoff in diesem Buch auf fünf Blöcke (und insgesamt 24 Lernabschnitte) aufgeteilt.

*Erfahrung des Autors:  
Übung macht den  
Meister*

Jeder dieser Blöcke besteht aus einzelnen Lernabschnitten, die im Buch aber auch am Bildschirm durchgearbeitet werden können, da sich auf der beiliegenden CD Powerpoint-Lektionen befinden, die mit denen im Buch identisch sind. Jeder dieser Abschnitte wird dabei mit Übungen (meistens Programmieraufgaben) abgerundet, wobei es sich empfiehlt, diese durchzuarbeiten, da Programmieren nicht etwas ist, das auswendig oder aber alleine durch stundenlange Vorlesungen oder Lesen von Büchern erlernt werden kann (auch wenn das natürlich hilft). Programmieren wird vorwiegend durch Übung erlernt. Als der Autor in der Universität das Programmieren erlernte, gab es jede Woche eine 90-minütige Vorlesung, zu der dann daheim (oder damals eigentlich noch am Großrechner der Universität) eine Übung ausgearbeitet werden musste, die ca. zehn Stunden in Anspruch nahm. Dies ist eine sehr effiziente Lernmethode, die deutlich macht, dass Programmieren nur am Computer erlernt werden kann. Daher sind auch die Lernabschnitte in diesem Buch so aufgebaut, dass sie beim Durcharbeiten jeweils ca. 60 Minuten in Anspruch nehmen und mit Übungen abgeschlossen werden, die ein Vielfaches dieser Zeit benötigen. Dabei ist es nicht notwendig, alle Übungen durchzuarbeiten, und es ist auch erlaubt, einen (wenn auch kurzen) Blick auf die Musterlösungen zu werfen (diese befinden sich vollständig auf der CD und auch – sofern es der Umfang zuließ – in Anhang B dieses Buches).

*Aufbau der  
Lernabschnitte*

Jeder Lernabschnitt ist demnach nach folgendem Muster aufgebaut:

- Kurzübersicht
- Lernstoff
- Übungen
- Musterlösungen zu den Übungen (aus didaktischen Gründen im Anhang und auf der CD enthalten)

*Überspringen von  
Lernabschnitten*

Grundsätzlich ist dieses Buch als Lehrbuch gedacht, das vom Anfang bis zum Ende durchgearbeitet werden kann. Selbstverständlich können aber auch nur einzelne Kapitel durchgearbeitet oder aber – je nach Vorwissen – Kapitel übersprungen werden. Falls ein Lernabschnitt übersprungen werden soll, so wäre es vorteilhaft, vorher sicherzugehen, ob dieses Überspringen auch gerechtfertigt ist. Dies kann durch Betrachten der Übungen entschieden werden.

## 1.4 Auswahl einer Programmierumgebung

Eine der ersten Entscheidungen zu Beginn eines objektorientierten Projektes ist die, welche Programmiersprache gewählt werden sollte. Dabei gibt es verschiedenste Möglichkeiten, da beginnend mit der ersten objektorientierten Programmiersprache SIMULA eine Vielzahl derartiger Sprachen entwickelt wurde. Insbesondere im deutschsprachigen Raum haben sich dabei drei durchgesetzt:

*Sprachen:  
Delphi, C++, Java –  
welche ist die beste?*

- Delphi: Bei dieser Sprache handelt es sich um eine objektorientierte Erweiterung von Pascal [JeWi74], einer sowohl auf Universitäten als auch in der Praxis weit verbreiteten Sprache. Delphi ist schön strukturiert und nach Meinung des Autors für einen ehemaligen COBOL-Programmierer relativ leicht zu erlernen. Sie ist vor allem für kaufmännische Anwendungen gedacht.
- C++: Eine objektorientierte Erweiterung der Programmiersprache C, die vor allem für systemnahes Programmieren gedacht ist. Die Sprache ist nicht ganz einfach zu erlernen und verführt etwas zu unstrukturierter Programmierung. Nichtsdestotrotz ist sie die zurzeit am meisten verwendete OOP-Sprache.
- Java: Diese OOP-Sprache ist C++ sehr ähnlich und vor allem für die Programmierung von Internetseiten gedacht. Durch den Internetboom hat diese Sprache die größten Wachstumsraten. Problematisch ist dabei, dass Programme, die mit Java programmiert sind, dazu neigen, langsam zu sein. Dies trifft insbesondere bei rechenintensiven Aufgaben zu, da der Code nicht kompiliert, sondern interpretiert wird. Im Internet ist Geschwindigkeit in diesem Bereich eben nicht so wichtig, da der Engpass sowieso durch die Leitungskapazität bestimmt ist und sicherlich nicht durch die Programmablaufgeschwindigkeit.

Die Auswahl der Sprache wäre bereits schwer genug, aber gerade in der objektorientierten Programmierung reicht es nicht aus, sich auf eine Sprache festzulegen, sondern es ist zusätzlich nötig, eine geeignete Programmierumgebung auszuwählen. Diese enthält neben dem Compiler noch andere Werkzeuge wie einen Debugger oder ein Projektverwaltungswerkzeug. Wesentlich ist aber, dass zu dieser Umgebung auch eine Klassenbibliothek gehört. Wie bereits angeschnitten und in Kapitel 4.5 noch ausführlich erklärt, ist diese oftmals sogar wichtiger als die Programmiersprache.

Die Auswahl einer geeigneten Entwicklungsumgebung ist daher eine wichtige, aber auch eine komplexe Angelegenheit. Wie bereits erwähnt, beginnt im Regelfall jedes größere Projekt mit dieser Ent-

*Kriterien für die Auswahl*

scheidungsfrage. Sinnvoll ist es hier, einen Anforderungskatalog aufzustellen und die möglichen Kandidaten entsprechend zu bewerten. Kriterien in einem derartigen Katalog könnten vollständige Unterstützung der objektorientierten Technik, Internetfunktionalität, Anbindung an Datenbanksysteme, bereits vorhandene Erfahrung, Ansprechpartner oder aber auch Kosten sein.

*Projektleiter  
entscheidet oft*

Der COBOL-Programmierer, der nun in ein konkretes objektorientiertes Projekt einsteigt, muss meist mit der von der Projektleitung getroffenen Entscheidung leben. Daher haben wir versucht dieses Buch von Programmierumgebungen her möglichst unabhängig zu schreiben. Dies ist auch für jene, die nur einmal in die Objektorientierung reinschnuppern wollen, sinnvoll, da sie sich wohl kaum im Vorhinein auf eine einzige Programmiersprache oder gar Entwicklungsumgebung festlegen wollen.

Natürlich ist dies ein schwieriges Unterfangen, insbesondere, wenn Programmbeispiele gezeigt werden. Diese müssen in einer konkreten Programmiersprache angegeben werden. Wenn immer dies notwendig war, wurde die jeweilige Sprache in der Marginalspalte angeführt. Wir haben uns dabei entschieden, sämtliche Beispiele in den Programmiersprachen Delphi und Java zu präsentieren, C++ wurde weggelassen, weil diese Sprache erstens Java nicht unähnlich ist und zweitens durch Java vermehrt abgelöst wird.

## 1.5 Objektorientiertes COBOL

Eine objektorientierte Programmiersprache wurde in obiger Auflistung vernachlässigt: objektorientiertes COBOL. Dies soll nun in diesem Teilkapitel nachgeholt werden. OO-COBOL ist eine Erweiterung von COBOL um objektorientierte Funktionalitäten, ähnlich wie C++ eine objektorientierte Erweiterung der Sprache C ist. Damit stellt sich natürlich in gewisser Weise die Frage, warum dieses Buch nicht auch auf diese Sprache näher eingeht und sie nicht als Grundlage zum Erlernen der objektorientierten Programmierung nimmt. Dies hat mehrere Gründe:

*Beurteilung von  
OO-COBOL*

- Jetzige COBOL-Programmierer stoßen nach ihrer Umschulung auf objektorientierte Programmierung meist zu neuen Projektteams dazu. Gerade aber in neuen Projekten wird OO-COBOL eher nicht eingesetzt.
- Beim Erlernen der objektorientierten Programmierung geht es nicht nur um das Erlernen von neuen Konzepten, sondern auch und insbesondere um deren Anwendung. Das Arbeiten mit OO-

COBOL würde aber eventuell dazu verführen, so wie bisher weiterzuarbeiten und nur die bereits bekannten Teile zu verwenden. Daher ist es sinnvoll, diese Techniken anhand einer neuen Sprache zu erlernen und sie von Anfang an zu verwenden.

- Auch wenn bestehende COBOL-Pakete von traditioneller Programmierung auf objektorientierte Programmierung umgestellt werden, macht es nach Meinung des Autors Sinn, nicht OO-COBOL, sondern eine andere objektorientierte Programmiersprache zu verwenden. Wenn nur OO-COBOL statt COBOL verwendet wird, besteht die Gefahr, dass sich gar nichts ändert, wenn niemand die Programmierer dazu zwingt, diese Konzepte zu verwenden.

## 1.6 Schreibweise und Konventionen

Schreibweisen wie zum Beispiel »der Anwender« werden in einer allgemeinen und geschlechtsneutralen Bedeutung verwendet.

Damit nicht Stichwörter, die einer großen Anzahl der Leser bekannt sein werden, immer wieder erklärt werden müssen, befindet sich im Anhang des Buches ein ausführliches Glossar.

Programmbeispiele werden ebenfalls durch Vermerke am Rand gekennzeichnet, wobei der Vermerk immer angibt, um welche Sprache (meistens Delphi oder Java) es sich handelt. Programmfragmente, etwa Schlüsselwörter, die im Text vorkommen, sind durch einen eigenen Formatierungsstil, `program`, gekennzeichnet.

*Quellcode*

## 1.7 Die beigelegte CD

Um den Lerneffekt zu verbessern, liegt dem Buch eine CD bei, auf der sich sämtliche Lernabschnitte als Powerpoint-Präsentationen und sämtliche Musterlösungen zu den Übungen in kompilierbarem Zustand befinden. Für diejenigen unter der Leserschaft, die noch keine Programmierumgebung besitzen, befinden sich zusätzlich Testversionen von Borland Delphi und Borland JBuilder auf der CD, damit die Übungen auf jeden Fall ausprobiert werden können.

Die CD hat dabei folgenden Aufbau: Im Hauptverzeichnis befinden sich drei Ordner:

*CD-Inhalt*

- Powerpoint: Lernabschnitte
- Delphi: Musterlösungen zu den Übungen in der Programmiersprache Delphi
- Java: Musterlösungen zu den Übungen in der Programmiersprache Java

Unterhalb dieser Ordner befindet sich eine Struktur, die immer gleichartig aufgebaut ist. Jedes Kapitel ist wieder ein eigener Ordner, jeder Abschnitt wiederum ein Unterordner.

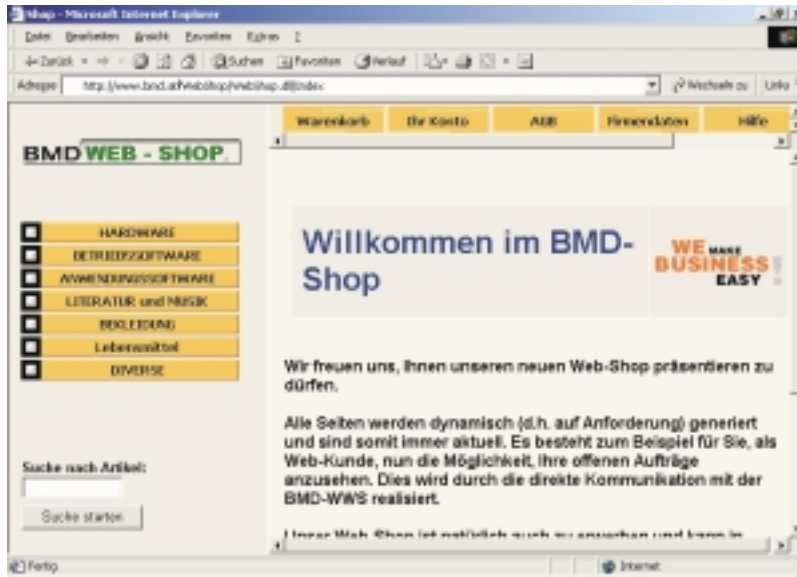
Falls eine der beiden Demoversionen installiert werden soll, so muss nur die Datei `borland.html` im Hauptverzeichnis der CD geöffnet werden. Die weitere Vorgangsweise ist dort genauestens beschrieben.

## 1.8 Ein Beispiel aus der Praxis

*BMD* Sollte es nach den bereits vorgestellten Vorteilen der objektorientierten Programmierung noch einer weiteren Motivation bedürfen, um mit Schwung und Elan in den OOP-Kurs zu gehen, sei hier noch ein kurzes Beispiel aus der Praxis angebracht. Die in dem Kurs vorgestellte Methode des Erlernens der objektorientierten Programmierung wurde bei der österreichischen Firma BMD Systemhaus GmbH erfolgreich angewandt, um über 40 COBOL-Programmierer (mit teilweise an die 30 Jahre COBOL-Erfahrung) in die Welt der objektorientierten Programmierung einzuführen. Bei dieser Firma, deren SW-Entwicklungsabteilung vom Autor dieses Buches seit 1997 geleitet wird, handelt es sich um den größten österreichischen Produzenten von Rechnungswesensoftware mit mehr als 10.000 Kunden.

*Erfahrung aus der Praxis:  
OOP zahlt sich aus*

Im Rahmen dieses Projektes [Kna99] wurde und wird die gesamte Softwareentwicklung von COBOL auf objektorientierte Programmierung umgestellt, wobei es sich dabei um immerhin mehr als 10 Millionen Lines of Code handelt. Kapitel 7 geht im Rahmen der Tipps für objektorientierte Projekte noch näher auf die Erfahrungen des BMD-Projektes ein. Um dem Leser noch einen Motivationsschub zu geben, sei an dieser Stelle aber bereits erwähnt, dass durch die neuen objektorientierten Techniken innerhalb kürzester Zeit (jeweils nur einiger weniger Monate) einige neue Pakete, wie ein Dokumentenarchiv und ein Webshop (siehe Abbildung 1-2), entwickelt werden konnten.



**Abb. 1-2**  
Der neue BMD-Webshop  
(Kernentwicklungszeit:  
2 Monate)