Bernhard Lahres
Gregor Raýman

Objektorientierte Programmierung

Das umfassende Handbuch

- Objektorientierte Programmierung verständlich erklärt
- Von den Prinzipien über den Entwurf bis zur Umsetzung
- Praxisbeispiele in UML, Java, C#, C++, JavaScript, Ruby, Python und PHP

2., aktualisierte und erweiterte Auflage

Galileo Computing

Liebe Leserin, lieber Leser,

als Programmierer ist Ihnen wahrscheinlich nur allzu bewusst, wie schwer es ist, wirklich gute Software zu entwickeln. Sie muss nicht nur das tun, was sie tun soll, sondern sie muss zudem performant, benutzerfreundlich und leicht zu warten sein. In der Praxis hat sich gezeigt, dass die Prinzipien der Objektorientierung die Grundlage all dessen bilden können, wenn sie optimal angewendet werden.

Mit diesem Buch lernen Sie, wie Sie genau das erreichen. Anschaulich und verständlich erläutern Ihnen unsere Autoren Bernhard Lahres und Gregor Raýman alle Aspekte der objektorientierten Softwareentwicklung angefangen beim Entwurf, über die Modellierung von Objekten und Klassen, bis hin zur Integration objektorientierter Entwürfe in reale Systeme.

Anhand der zahlreichen Beispiele in UML sowie den gängigsten Sprachen wie Java, JavaScript, C#, C++, Ruby und Python wird es Ihnen leichtfallen, das Gelernte nicht nur wirklich zu verstehen, sondern vor allem auch selber erfolgreich umzusetzen. Um Sie dabei noch besser zu unterstützen, wurde die vorliegende zweite Auflage um ein Praxiskapitel ergänzt, in dem eine ganze Reihe der vorher im Buch erläuterten einzelnen OOP-Prinzipien im Rahmen eines komplexeren Anwendungsbeispiels in PHP durchgespielt werden. Als erfahrene Softwareentwickler kennen die beiden Autoren natürlich auch die potenziellen Stolpersteine, die sich bei der praktischen Umsetzung ergeben können, und können Ihnen daher wirksame Strategien aufzeigen, wie Sie diese umgehen können.

Dieses Buch wurde mit großer Sorgfalt geschrieben, geprüft und produziert. Sollte dennoch einmal etwas nicht so funktionieren, wie Sie es erwarten, freue ich mich, wenn Sie sich mit mir in Verbindung setzen. Ihre Kritik und konstruktiven Anregungen, aber natürlich auch Ihr Lob sind uns jederzeit herzlich willkommen!

Viel Spaß bei der Lektüre und viel Erfolg mit OOP wünscht Ihnen

Christine Siedle
Lektorat Galileo Computing

lektorat@galileo-press.de
www.galileocomputing.de
Galileo Press · Rheinwerkallee 4 · 53227 Bonn

Auf einen Blick

Meiner geliebten Ehefrau Janka

Gregor

Für Pia Susanna

Bernhard

Impressum

Dieses E-Book ist ein Verlagsprodukt, an dem viele mitgewirkt haben, insbesondere:

Lektorat Christine Siedle, Stephan Matteschеck
Korrektorat Marlis Appel, Troisdorf
Herstellung E-Book Katrin Müller
Covergestaltung Barbara Thoben, Köln
Coverbild Barbara Thoben, Köln
Satz E-Book III-satz, Husby

Bibliografische Information der Deutschen Nationalbibliothek:
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

ISBN 978-3-8362-3214-2

2., aktualisierte und erweiterte Auflage 2009, 1., korrigierter Nachdruck 2011

Inhalt

In diesem Kapitel stellen die Autoren sich und dieses Buch vor. Sie erklären, was Objektorientierung im Bereich der Softwareentwicklung bedeutet. Und am Ende des Kapitels geraten sie auch schon in die erste Diskussion miteinander.

1 Einleitung

1.1 Was ist Objektorientierung?

Objektorientierung ist eine mittlerweile bewährte Methode, um die Komplexität von Softwaresystemen in den Griff zu bekommen. Die Entwicklung der Objektorientierung als Konzept der Softwaretechnik ist sehr eng mit der Entwicklung von objektorientierten Programmiersprachen verbunden.

Von Alan Kay, dem Erfinder der Sprache Smalltalk, die als die erste konsequent objektorientierte Programmiersprache gilt, wird die folgende Geschichte erzählt:

Bei einer Präsentation bei Apple Mitte der 80er-Jahre hielt ein Mitarbeiter einen Vortrag über die erste Version der neu entwickelten Programmiersprache Oberon. Diese sollte der nächste große Schritt in der Welt der objektorientierten Sprachen sein.

Da meldete sich Alan Kay zu Wort und hakte nach:

»Diese Sprache unterstützt also keine Vererbung?«
»Das ist korrekt.«
»Und sie unterstützt keine Polymorphie?«
»Das ist korrekt.«
»Und sie unterstützt auch keine Datenkapselung?«
»Das ist ebenfalls korrekt.«
»Dann scheint mir das keine objektorientierte Sprache zu sein.«

Der Vortragende meinte darauf: »Nun, wer kann schon genau sagen, was nun objektorientiert ist und was nicht.«

Woraufhin Alan Kay zurückgab: »Ich kann das. Ich bin Alan Kay, und ich habe den Begriff geprägt.«

Der geschilderte Dialog enthält genau die drei Grundelemente, die als Basis von objektorientierten Programmiersprachen gelten:

- Unterstützung von Vererbungsmechanismen
- Unterstützung von Datenkapselung
- Unterstützung von Polymorphie

Auf alle drei Punkte werden wir eingehen. Objektorientierung geht allerdings mittlerweile weit über diese Grunddefinition hinaus. Die genannten Punkte bilden lediglich den kleinsten gemeinsamen Nenner.

1.2 Hallo liebe Zielgruppe

Die Frage »Für wen schreiben wir dieses Buch?« haben wir uns beim Schreiben selbst immer wieder gestellt.

Die Antwort darauf hängt eng mit der Frage zusammen, warum wir eigentlich dieses Buch geschrieben haben. Nun, neben dem ganz offensichtlichen Grund, dass die Veröffentlichung die beiden Autoren reich und berühmt machen wird, was ja ein ganz angenehmer Nebeneffekt ist, gibt es noch einen ganz zentralen weiteren Grund: Wir wollten das Buch schreiben, das wir uns selbst an bestimmten Punkten unseres Ausbildungswegs und unserer Berufslaufbahn gewünscht hätten.

Wir haben uns beide von Anfang an im Umfeld der objektorientierten Softwareentwicklung bewegt. Allerdings unterschied sich das, was wir in der Praxis an Anforderungen vorfanden, dann doch stark von dem, was Universität und Lehrbücher vorbereitet hatten.

Theorie und Praxis

Aufgrund dieser Erfahrung hätten wir uns ein Buch gewünscht, das den Brückenschlag zwischen Theorie und Praxis bewerkstelligt. Nicht mal so sehr auf der Ebene von praktischen Programmen, sondern eher: Wie werden die theoretischen Ansätze (die wir Prinzipien nennen) denn nun umgesetzt? Was hat es mit dem Thema Garbage Collection auf sich? Was ist denn wirklich mit Model-View-Controller gemeint? Warum müssen wir uns denn überhaupt mit tiefen Kopien, flachen Kopien, Identität, Gleichheit beschäftigen?

Wir glauben, dass wir es ein Stück in diese Richtung geschafft haben und Ihnen eine interessante Verbindung aus Theorie und Praxis präsentieren.

Wer sollte unser Buch lesen?

Wir wollen Studenten der Informatik eine praktische, interessante und hoffentlich auch unterhaltsame Einführung und Vertiefung zum Thema

Objektorientierung bieten. Wir wollen Berufseinsteigern im Bereich Softwareentwicklung einen leichteren Start ermöglichen, indem wir Begriffe praktisch klären, mit denen sie regelmäßig konfrontiert werden. Wir wollen im Beruf aktiven Softwareentwicklern die Chance geben, einfach noch mal darüber nachzudenken, was sie eigentlich in der täglichen Arbeit so machen und ob nicht vielleicht noch die eine oder andere Verbesserung möglich ist.

Wir wenden uns also an Softwareentwickler in Ausbildung oder im aktiven Einsatz, an Menschen, die Programme schreiben. Natürlich würden wir uns auch freuen, wenn Projektleiter und Projektmanager unser Buch in die Hand nehmen. Geschrieben haben wir es allerdings als Softwareentwickler für andere Softwareentwickler.

Was wir voraussetzen

Als Voraussetzung gehen wir davon aus, dass unsere Leser grundsätzliche Erfahrung mit der Programmierung haben. Erfahrungen mit der Programmierung in objektorientierten Sprachen sind hilfreich, obwohl nicht unbedingt notwendig. Unser Buch wendet sich damit auch explizit an Menschen, die bereits mit Programmiersprachen arbeiten, die objektorientierte Mechanismen anbieten, wie dies zum Beispiel bei Java der Fall ist. Da aber nur durch die Verwendung von objektorientierten Sprachen noch lange keine objektorientierte Software entsteht, wollen wir mit unserem Buch genau diesen Schritt ermöglichen.

Wir werden zu den verwendeten Programmiersprachen keine detaillierte Einführung geben. Stattdessen findet sich im Anhang jeweils eine Kurzbeschreibung der jeweiligen Programmiersprache. Diese erläutert die grundlegenden Sprachkonstrukte mit Bezug zur objektorientierten Programmierung. Außerdem geben wir Verweise auf weitere Informationsquellen zur Programmiersprache, sofern möglich auch mit Hinweisen auf verfügbare freie Versionen von Compilern, Interpretern oder Entwicklungsumgebungen.

1.3 Was bietet dieses Buch (und was nicht)?

Objektorientierung ist eine Vorgehensweise, die den Fokus auf modulare Systeme legt. Dieses Modulprinzip legen wir in zweierlei Hinsicht auch diesem Buch zugrunde.

Zum einen sehen wir das Buch als einen Baustein, der beim Verständnis des Themas Objektorientierung eine zentrale Rolle spielt. Wir beschreiben, wie Sie Objektorientierung einsetzen können, um auf der Grund-

lage von zentralen Prinzipien beherrschbare und änderbare Software erstellen können. Wenn wir auf diesem Weg an Entwurfsmustern vorbeikommen, werden wir diese vorstellen und ihren Nutzen für den Entwurf oder die aktuelle Problemstellung erläutern.

Zum anderen bauen die Kapitel des Buchs modular aufeinander auf. Ausgehend von abstrakten Prinzipien gehen wir immer weiter in die konkrete Anwendung der Objektorientierung. Diesen modularen Aufbau wollen wir in einer Übersicht kurz vorstellen.

1.3.1 Bausteine des Buchs

Abbildung 1.1 stellt die einzelnen Kapitel als Bausteine in der Übersicht dar.

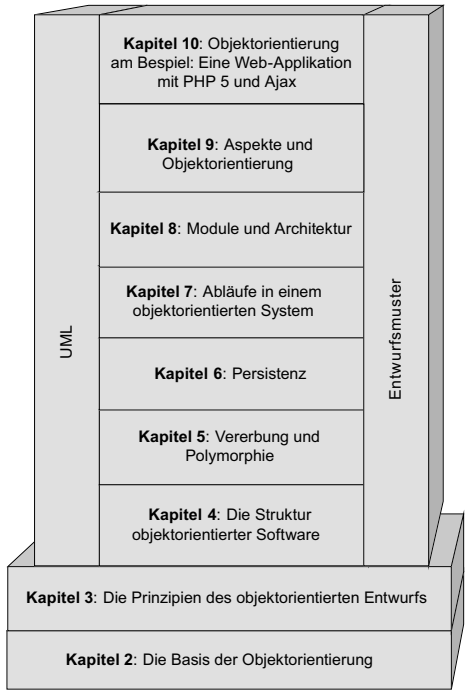

Abbildung 1.1 Modularer Aufbau der Kapitel

Kapitel 2: Basis

In Kapitel 2, »Die Basis der Objektorientierung«, stellen wir zunächst die grundlegenden Unterschiede der objektorientierten Vorgehensweise im Vergleich zur strukturierten Programmierung vor. Wir beschreiben dort auch im Überblick die Basismechanismen der Objektorientierung: Datenkapselung, Polymorphie und Vererbung.

Kapitel 3: Prinzipien

Danach schließt sich Kapitel 3, »Die Prinzipien des objektorientierten Entwurfs«, an. Das Kapitel stellt die grundlegenden Prinzipien vor, die für den objektorientierten Ansatz entscheidend sind. In Abschnitt 1.1 haben wir einen kleinen Dialog vorgestellt, in dem die grundlegenden Eigenschaften von objektorientierten Sprachen angesprochen werden. Genau wie die dort genannten Eigenschaften geben auch die Prinzipien der Objektorientierung einen Rahmen für eine bestimmte Art der Softwareentwicklung vor. Die Prinzipien sind zentral für das Vorgehen bei der Entwicklung von objektorientierter Software. Das Kapitel legt also die grundsätzliche Basis, ohne bereits detailliert auf Objekte, Klassen oder Ähnliches einzugehen.

Kapitel 4: Struktur

Anschließend erläutern wir in Kapitel 4, »Die Struktur objektorientierter Software«, das Konzept von Objekten, Klassen und der darauf aufbauenden Möglichkeiten. Wir erläutern das Konzept der Klassifizierung und betrachten die Rolle der Klassen als Module. Den verschiedenen Arten, in denen Objekte untereinander in Beziehung stehen können, ist ebenfalls ein Teilkapitel gewidmet.

Kapitel 5: Vererbung und Polymorphie

In Kapitel 5, »Vererbung und Polymorphie«, gehen wir darauf ein, wie die Vererbung der Spezifikation – im Zusammenspiel mit der Fähigkeit der Polymorphie – Programme flexibel und erweiterbar halten kann. Wir gehen dabei auf die verschiedenen Varianten der Vererbung im Detail ein und stellen deren Möglichkeiten und Probleme vor. An Beispielen erläutern wir dabei auch, wie die Möglichkeiten der Polymorphie am besten genutzt werden können.

Kapitel 6: Persistenz

Kapitel 6, »Persistenz«, beschreibt, wie Objekte in relationalen Datenbanken gespeichert werden können. In fast allen Anwendungen taucht die Notwendigkeit auf, dass Objekte persistent gespeichert werden müssen. Wir stellen die Abbildungsregeln für Vererbungsbeziehungen und andere Beziehungen zwischen Objekten und Klassen auf eine relationale Datenbank vor. Schließlich setzen wir diese Abbildungsregeln in Beziehung zu den verschiedenen Stufen der Normalisierung in einer relationalen Datenbank.

Kapitel 7: Abläufe

In Kapitel 7, »Abläufe in einem objektorientierten System«, beschreiben wir die Vorgänge innerhalb des Lebenszyklus von Objekten. Wir gehen detailliert auf die Erzeugung von Objekten ein und erläutern, wie Sie ein System erweiterbar halten, indem Sie möglichst flexible Methoden der Objekterzeugung einsetzen. Außerdem enthält das Kapitel Beschreibungen von Interaktionsszenarien, die sich häufig in objektorientierten Systemen finden. Der Abschluss des Objektlebenszyklus, der meist über den

Mechanismus der Garbage Collection stattfindet, wird ebenfalls in diesem Kapitel beschrieben.

Kapitel 8: Architektur

In Kapitel 8, »Module und Architektur«, stellen wir Beispiele dafür vor, wie objektorientierte Entwürfe in reale Systeme integriert werden. Wir stellen das Konzept von in der Praxis verwendeten Ansätzen wie Frameworks und Anwendungscontainern vor. Am Beispiel der Präsentationsschicht einer Anwendung erläutern wir, wie objektorientierte Verfahren die Interaktionen in einem System strukturieren können. Dazu stellen wir das Architekturmuster Model-View-Controller (MVC) und dessen Einsatzszenarien vor.

Kapitel 9: Aspekte

In Kapitel 9, »Aspekte und Objektorientierung«, stellen wir dar, wie sich eine Reihe von Einschränkungen der objektorientierten Vorgehensweise durch Aspektorientierung aufheben lässt. Wir erläutern, welche Beschränkungen der objektorientierten Vorgehensweise überhaupt zur Notwendigkeit einer aspektorientierten Sichtweise führen. Die Verwaltung von sogenannten übergreifenden Anliegen (engl. *Crosscutting Concerns*) ist mit den Mitteln der klassischen objektorientierten Programmierung nur sehr aufwändig zu realisieren. Bei Crosscutting Concerns handelt es sich um Anforderungen, die mit Mitteln der Objektorientierung nur klassen- oder komponentenübergreifend realisiert werden können.

Deshalb zeigen wir in diesem Kapitel verschiedene Möglichkeiten auf, Lösungen dafür in objektorientierte Systeme zu integrieren. Wir erläutern den Weg zu den aspektorientierten Lösungen und stellen diese an praktischen Beispielen vor.

Kapitel 10: Objektorientierung am Beispiel einer Web-Applikation

Abgerundet wird unser Buch dann durch Kapitel 10, »Objektorientierung am Beispiel: Eine Web-Applikation mit PHP 5 und Ajax«. Dort greifen wir am Beispiel einer Web-Anwendung eine ganze Reihe von Konzepten auf, die in den vorhergehenden Kapiteln erläutert wurden. Im Kontext einer einfachen, aber vollständigen Web-Anwendung auf Basis von PHP 5 und Ajax stellen wir Schritt für Schritt vor, wie Geschäftslogik und Präsentationsschicht durch objektorientierte Vorgehensweisen klar voneinander entkoppelt werden. Dabei gehen wir auch darauf ein, durch welchen Aufbau ein Austausch von Teilen der Präsentationsschicht erleichtert wird.

Anhang: Programmiersprachen

Im Anhang werden wir die im Buch verwendeten Programmiersprachen jeweils mit einer Kurzreferenz vorstellen und Hinweise auf weitere Informationen geben.

1.3.2 Crosscutting Concerns: übergreifende Anliegen

Die beschriebenen Kapitel bauen in Form einer Modulstruktur aufeinander auf. Aber ähnlich wie bei der Strukturierung objektorientierter Software gibt es natürlich auch hier Themen, die übergreifend sind und sich nicht genau einem der Kapitel zuordnen lassen.

Im Folgenden gehen wir kurz darauf ein, welche Rolle diese weiteren Themengebiete spielen.

Unified Modeling Language

Die Unified Modeling Language (UML) hat sich mittlerweile als ein sinnvolles und weit verbreitetes Modellierungsmittel durchgesetzt. Wir werden die nach unserer Ansicht wichtigsten Darstellungselemente der UML anhand von Beispielen vorstellen. Wir werden diese dabei immer dann einführen, wenn die betreffende Modellierung für unser aktuell behandeltes Thema relevant wird. Wir verwenden die UML in der Regel auch zur Illustration von Strukturen und Abläufen.

Objektorientierte Analyse

Die objektorientierte Analyse betrachtet eine Domäne als System von kooperierenden Objekten. Obwohl objektorientierte Analysemethoden nicht unser zentrales Thema sind, werden wir diese immer dann heranziehen, wenn wir auf die Verbindung von Analyse und Design eingehen.

Entwurfsmuster

Entwurfsmuster (engl. *Design Patterns*) für den objektorientierten Entwurf wurden mit der Publikation *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software* als Begriff geprägt. Die Autoren Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides führen dabei eine standardisierte Beschreibungsform für wiederkehrende Vorgehensweisen beim Entwurf guter objektorientierter Software ein[1]. Aufgrund von Umständen, die mit der Sortierung unseres Alphabets zusammenhängen, wird die Publikation meist mit Erich Gamma assoziiert. Da das aber nicht korrekt ist, werden wir uns mit dem Namen [Entwurfsmuster 2004] darauf beziehen, da die aktuelle deutsche Ausgabe aus dem

1 Allerdings sind auch die Autoren des Entwurfsmuster-Buchs mittlerweile nicht mehr bei allen der vorgestellten Muster davon überzeugt, dass es sich wirklich um empfehlenswerte Vorgehensweisen handelt. So hat sich beispielsweise das Entwurfsmuster »Singleton« mittlerweile einen eher fragwürdigen Ruf erarbeitet.

Jahr 2004 stammt und eine Verwechslungsgefahr mit anderen Entwurfsmusterbüchern in diesem Jahr nicht besteht.

Grundsätzlich sind solche Entwurfsmuster unabhängig davon, ob wir eine objektorientierte Vorgehensweise gewählt haben, da sie nur eine Schablone für gleichartige Problemstellungen sind. Allerdings hat sich in der Praxis herausgestellt, dass bei Verwendung von objektorientierten Methoden diese Muster einfacher zu erkennen und besser zu beschreiben sind.

In der Folge ist eine ganze Reihe von unterschiedlichen Entwurfsmustern entstanden, die meisten mit Bezug auf objektorientierte Methoden. In der Kategorie Anti-Patterns werden häufige Fehler beim Systementwurf zusammengefasst.

Wir werden eine ganze Reihe von Entwurfsmustern verwenden. Allerdings werden wir diese dann vorstellen, wenn ein Muster ein bestimmtes Thema gut illustriert oder dafür zentral ist. In diesem Fall geben wir eine kurze Vorstellung des Musters und erläutern dessen Anwendung.

Deutsche und englische Begriffe

Wir werden außerdem weitgehend deutsche Begriffe verwenden. Wenn allerdings der englische Begriff der wesentlich gängigere ist, werden wir diesen verwenden. Dies gilt auch, wenn eine deutsche Übersetzung die Bedeutung verzerren würde oder zu umständlich wird. So werden wir zum Beispiel den Begriff *Multiple Dispatch* verwenden, weil die in Frage kommenden Übersetzungen *Multiple Verteilung* oder *Verteilung auf der Grundlage von mehreren Objekten* nicht wesentlich erhellender oder aber sehr umständlich sind.

Wir werden die englische Entsprechung bei der ersten Erwähnung meist mit aufführen. So werden wir zum Beispiel das Prinzip *Offen für Erweiterung, geschlossen für Änderung* bei der ersten Erwähnung auch als *Open Closed Principle* vorstellen.

1.3.3 Die Rolle von Programmiersprachen

Die meisten der von uns diskutierten Prinzipien der Objektorientierung finden sich in den objektorientierten Programmiersprachen wieder, entweder als direkte Sprachkonstrukte oder als Möglichkeiten bei der Programmierung.

Allerdings: Die Unterstützung variiert sehr stark zwischen den einzelnen objektorientierten Sprachen. Bestimmte Prinzipien werden nur von wenigen Sprachen unterstützt, andere in unterschiedlichem Ausmaß.

Wir haben deshalb eine überschaubare Anzahl von Programmiersprachen ausgewählt, um diese jeweils dann als Beispiel heranzuziehen, wenn ein bestimmtes Prinzip besonders gut (oder vielleicht auch besonders schlecht) unterstützt wird.

Objektorientierte Softwareentwicklung lässt sich nicht beschreiben, ohne auf die Entwicklung der objektorientierten Programmiersprachen einzugehen. Durch ihre jeweils spezifischen Möglichkeiten sind Programmiersprachen selbst sehr gute Beispiele dafür, wie Konzepte der Objektorientierung in der Praxis umgesetzt werden.

Alan Perlis hatte mit seiner Aussage völlig Recht, als er schrieb: *Eine Programmiersprache, die nicht die Art beeinflusst, in der du über das Programmieren nachdenkst, ist es nicht wert, dass man sie kennt.*[2]

Deshalb werden Sie in den folgenden Kapiteln auch einiges über die Besonderheiten und speziellen Möglichkeiten von mehreren objektorientierten Sprachen erfahren. Alleine schon durch die Betrachtung der Unterschiede zwischen einzelnen Sprachen lassen sich Konzepte gut illustrieren. Jede der Sprachen hat ihre eigenen Stärken, jede macht bestimmte Sachen einfach. Jede hat ihre eigenen Idiome und Muster.

Wenn man mehrere Programmiersprachen kennt, lernt man oft neue Vorgehensweisen. Auch wenn man in der Praxis die meiste Zeit nur mit einer Programmiersprache arbeitet, lohnt es sich, auch einen Blick auf andere zu werfen. Vielleicht nur um die ausgetretenen Pfade zu verlassen und zu erkennen, dass manche Aufgaben, die man für schwierig hielt, sich in Wirklichkeit ganz einfach lösen lassen.

Hier also die Liste der Programmiersprachen, die sich in den Code-Beispielen wiederfinden:

- **Java**
  Java ist natürlich dabei, weil es mittlerweile eine sehr verbreitete Sprache ist, die einen Fokus auf Objektorientierung setzt.

2 Das Originalzitat in Englisch lautet: »A language that doesn't affect the way you think about programming, is not worth knowing.« und ist enthalten in einem Artikel von Alan Perlis in den *SIGPLAN Notices Vol. 17, No. 9, September 1982*.

- **C++**
  Auch C++ hat immer noch einen hohen Verbreitungsgrad und unterstützt die Basisprinzipien der Objektorientierung zu großen Teilen.
- **JavaScript**
  JavaScript wird hauptsächlich als Beispiel für eine objektorientierte Sprache, die mit Prototypen arbeitet, angeführt.
- **Ruby**
  Ruby ist eine Skriptsprache, die wegen ihrer einfachen und intuitiven Handhabung immer beliebter wird. Ruby hat einen sehr starken Fokus auf Objektorientierung. Außerdem sind Klassenerweiterungen (Mixins) möglich.
- **C#**
  Die von Microsoft entwickelte Sprache aus der .NET-Familie fasst eine ganze Reihe von Konzepten der Objektorientierung gut zusammen.
- **Python**
  Python ist eine interaktive objektorientierte Skriptsprache, die nach der britischen Komikertruppe Monty Python benannt ist. Python betrachtet alle im Programm vorhandenen Daten als Objekte.

Wir werden Beispiele in den verschiedenen Sprachen immer dann einbringen, wenn sich eine Sprache gerade besonders gut zur Illustration eignet. Außerdem werden wir zur Erläuterung der aspektorientierten Erweiterungen zur Objektorientierung hauptsächlich AspectJ heranziehen.

1.4 Warum überhaupt Objektorientierung?

Wir wollen hier zunächst einmal eine Aussage vorausschicken, die trivial anmuten mag: Es ist nicht einfach, gute Software zu entwickeln. Speziell die Komplexität von mittleren und großen Softwaresystemen stellt oft ein großes Problem dar.

Objektorientierte Softwareentwicklung ist nicht die einzige Methode, um diese Komplexität in den Griff zu bekommen, aber sie hat sich in verschiedenen Anwendungskontexten bewährt. Außerdem liefert sie mittlerweile ein Instrumentarium für eine ganze Reihe von Problemfeldern.

Objektorientierte Vorgehensweisen ergänzen sich außerdem gut mit einigen anderen Ansätzen in der Softwareentwicklung. Deshalb ist das

letzte Kapitel auch dem Thema Aspektorientierung gewidmet, da dieser Ansatz einige der Defizite der objektorientierten Vorgehensweise ausbügeln kann.

1.4.1 Gute Software: Was ist das eigentlich?

Je nachdem, wen wir fragen, wird die Antwort auf die Frage »Was macht gute Software für Sie aus?« unterschiedlich ausfallen.

Fragen wir doch zunächst den *Anwender* von Software. Das sind wir ja praktisch alle. Hier werden wir häufig die folgenden Anforderungen hören:

- Software soll das machen, was ich von ihr erwarte: Software muss korrekt sein.
- Software soll es mir möglichst einfach machen, meine Aufgabe zu erledigen: Software muss benutzerfreundlich sein.
- Ich möchte meine Arbeit möglichst schnell und effizient erledigen, ich möchte keine Wartezeiten haben: Software muss effizient und performant sein.

Fragen wir nun denjenigen, der für Software und Hardware *bezahlt*, kommen weitere Anforderungen hinzu:

- Ich möchte keine neue Hardware kaufen müssen, wenn ich die Software einsetze: Software muss effizient mit Ressourcen umgehen.
- Ich möchte, dass die Software möglichst günstig erstellt werden kann.
- Mein Schulungsaufwand sollte gering sein.

Fragen wir den *Projektmanager*, der die Entwicklung der Software vorantreiben soll, dann kommt noch mehr zutage:

- Meine Auftraggeber kommen oft mit neuen Ideen. Es darf nicht aufwändig sein, diese neuen Ideen auch später noch umzusetzen: Software muss sich anpassen lassen.
- Ich habe feste Zieltermine, das Management sitzt mir im Nacken. Deshalb muss ich diese Software in möglichst kurzer Zeit fertigstellen.
- Das Programm soll über Jahre hinweg verwendet werden. Ich muss Änderungen auch dann noch mit überschaubarem Aufwand umsetzen können.

Trotz der unterschiedlichen Sichtweisen kristallisieren sich hier einige zentrale Kriterien für gute Software heraus[3]:

- **Korrektheit**
  Software soll genau das tun, was von ihr erwartet wird.
- **Benutzerfreundlichkeit**
  Software soll einfach und intuitiv zu benutzen sein.
- **Effizienz**
  Software soll mit wenigen Ressourcen auskommen und gute Antwortzeiten für Anwender haben.
- **Wartbarkeit**
  Software soll mit wenig Aufwand erweiterbar und änderbar sein.

Ein Hauptfeind dieser Kriterien ist die Komplexität, die sich in der Regel bei Softwaresystemen mit zunehmender Größe aufbaut.

1.4.2 Die Rolle von Prinzipien

Wenn wir Sie motivieren wollen, dass Sie objektorientierte Methoden einsetzen sollten, müssen wir drei verschiedene Arten von Fragen stellen:

1. Welche Probleme wollen Sie eigentlich angehen? Häufig wird die Aufgabenstellung sein, die bereits genannten Kriterien wie Korrektheit, Benutzerfreundlichkeit, Effizienz und Wartbarkeit einzuhalten.
2. Welche abstrakten Prinzipien helfen Ihnen dabei? Beispiele sind hier Kapselung von Information oder klare Zuordnung von Veranwortlichkeiten zu Modulen.
3. Wie können Sie diese Prinzipien in einem Softwaresystem sinnvoll umsetzen?

Da Objektorientierung eben nur eine Methode ist, um diese Ziele umzusetzen, ist es beim Entwurf von Systemen immer wichtig, dass Sie sich bewusst sind, warum Sie einen bestimmten Entwurf gewählt haben.

Am Ende kommt es darauf an, den ursprünglichen Zielen möglichst nahe zu kommen. Sie werden dabei oft Kompromisse finden müssen. Die mit

3 Neben den aufgelisteten Kriterien gibt es noch weitere allgemeine Anforderungen an Software. Es ist von der jeweiligen Anwendung abhängig, wie zentral die jeweilige Anforderung ist. So ist für Software zur Steuerung eines Atomkraftwerks Korrektheit und Fehlertoleranz wichtiger als der effiziente Umgang mit Ressourcen.

der objektorientierten Vorgehensweise verbundenen Prinzipien sind dabei Richtlinien, diese müssen aber oft gegeneinander abgewogen werden.

Deshalb beginnen wir in Kapitel 3, »Die Prinzipien des objektorientierten Entwurfs«, auch mit der Vorstellung dieser grundlegenden Prinzipien, die der objektorientierten Softwareentwicklung zugrunde liegen. Zum überwiegenden Teil können diese völlig unabhängig von Begriffen wie Klasse oder Objekt vorgestellt werden. Erst nach der Einführung der Prinzipien werden wir also erklären, wie diese mittels Klassen und Objekten umgesetzt werden können.

1.4.3 Viele mögliche Lösungen für ein Problem

Auch wenn Sie sich einmal entschieden haben, nach dem objektorientierten Paradigma vorzugehen, werden für eine Problemstellung häufig verschiedene Lösungen möglich sein.

Obwohl sich die beiden Autoren grundsätzlich darüber einig sind, dass objektorientierte Techniken zu sinnvollen Problemlösungen führen, ergeben sich hinsichtlich der in einem konkreten Fall zu wählenden Lösung doch häufig Differenzen.

Diskussionen können hilfreich sein.

In so einem Fall werden wir die resultierende Diskussion einfach ganz offen vor unserer Leserschaft austragen und diese dabei mit Beispielen aus unseren praktischen Erfahrungen mit Objekttechnologie unterfüttern. Das wird ein ganz schön lebhaftes Buch werden, das können wir an dieser Stelle schon versprechen.

Diskussion: Was gibt's denn hier zu diskutieren?

Gregor: *Findest du das denn eine gute Idee, einfach vor unseren Leserinnen und Lesern zu diskutieren? Bestimmt erwarten sie doch, dass wir uns auf unserem Fachgebiet einig sind und auch klare Lösungen vorstellen. Ich weiß natürlich, dass wir uns wirklich nicht immer einig sind, aber wir könnten doch zumindest so tun.*

Bernhard: *Nun, wenn wir ständig diskutieren würden, wäre das sicherlich etwas irritierend. Und wir sind uns ja auch größtenteils einig, in vielen Fällen gibt es einfach auch generell anerkannte Vorgehensweisen. Aber spannend wird das Ganze erst an den Punkten, an denen das Vorgehen eben nicht mehr völlig klar ist und wir uns für eine von mehreren möglichen Vorgehensweisen entscheiden müssen.*

Gregor: *Stimmt schon, hin und wieder so eine kleine Diskussion zwischendurch war ja auch beim Schreiben des Buchs durchaus nützlich. Dann lass uns jetzt aber loslegen. Die nächste Diskussion kommt bestimmt bald.*

Icons Um Ihnen die Orientierung im Buch zu erleichtern, haben wir zwei Icons verwendet:

[»] Hier finden Sie hilfreiche Definitionen, die die wesentlichen Aspekte des Themas zusammenfassen.

[zB] Konkrete Beispiele erleichtern Ihnen das Verstehen.

Weiterführende Informationen finden Sie auf der Webseite zum Buch unter *www.objektorientierte-programmierung.de*.

In diesem Kapitel werfen wir vorab bereits einen Blick auf die technischen Möglichkeiten, die uns die Objektorientierung bietet. Und wir stellen die Basiskonzepte Datenkapselung, Vererbung und Polymorphie kurz vor, ohne bereits ins Detail zu gehen.

2 Die Basis der Objektorientierung

Vor der Frage, wie objektorientierte Verfahren am besten eingesetzt werden, drängt sich die Frage auf, warum Sie denn solche Verfahren einsetzen sollten.

Die Objektorientierung hat sich seit Anfang der Neunzigerjahre des letzten Jahrhunderts als Standardmethode der Softwareentwicklung etabliert. Aber nur weil etwas mittlerweile als Standard gilt, muss es noch lange nicht nützlich sein. Das alleine reicht als Motivation für objektorientierte Verfahren nicht aus.

Objektorientierung löst einige Probleme, ...

Die Techniken der objektorientierten Softwareentwicklung unterstützen uns dabei, Software *einfacher erweiterbar*, *besser testbar* und *besser wartbar* zu machen.

... aber nicht alle.

Allerdings dürfen Sie sich von der Objektorientierung nicht Antworten auf alle Probleme und Aufgabenstellungen der Softwareentwicklung erwarten. Die Erwartungen an die Möglichkeiten dieser Vorgehensweise werden in vielen Projekten zu hoch gesteckt.

Zum einen führt die Nutzung objektorientierter Basismechanismen und objektorientierter Programmiersprachen nicht automatisch zu guten Programmen. Zum anderen adressiert die Objektorientierung einige Problemstellungen gar nicht oder bietet nur unvollständige Lösungsstrategien dafür. Bei der Vorstellung des *Prinzips der Trennung von Anliegen* im nächsten Kapitel werden Sie zum Beispiel sehen, dass die Objektorientierung dieses Prinzip nur unvollständig unterstützt.

Grundelemente der Objektorientierung

Die Objektorientierung bietet aber einen soliden Werkzeugkasten an, der es uns erlaubt, die Zielsetzungen der Entwicklung von Software anzugehen. Die Basiswerkzeuge in diesem Werkzeugkasten sind die drei Grundelemente objektorientierter Software:

- Datenkapselung
- Polymorphie
- Vererbung

Wir geben im Folgenden einen kurzen Überblick über die drei Basistechniken. Dabei werden wir auf Begriffe vorgreifen müssen, die erst später im Buch eingeführt werden. Sie sollen aber hier bereits einen ersten Eindruck davon erhalten, welche Möglichkeiten die Objektorientierung bietet. Die Details und formalen Definitionen werden wir Ihnen im weiteren Verlauf des Buchs nachreichen, versprochen.

Um Ihnen die Vorteile der objektorientierten Programmierung verdeutlichen zu können, beginnen wir aber zunächst mit einer Kurzzusammenfassung der Verfahren der strukturierten Programmierung, die ja der Vorläufer der objektorientierten Vorgehensweise ist.

2.1 Die strukturierte Programmierung als Vorläufer der Objektorientierung

Die objektorientierte Softwareentwicklung baut auf den Verfahren der strukturierten Programmierung auf. Um die Motivation für die Verwendung von objektorientierten Methoden zu verstehen, gehen wir einen Schritt zurück und werfen einen Blick auf die Mechanismen der strukturierten Programmierung und auch auf deren Grenzen.

Programmiersprachen, die dem Paradigma[1] des strukturierten Programmierens folgen, sind zum Beispiel PASCAL oder C.

Die Struktur von Programmen und Daten

Der Inhalt des benutzten Computerspeichers kann für die meisten Programme in zwei Kategorien unterteilt werden. Einerseits enthält der Speicher *Daten*, die bearbeitet werden, andererseits enthält er *Instruktionen*, die bestimmen, was das Programm macht.[2]

1 Als Paradigma wird in der Erkenntnistheorie ein System von wissenschaftlichen Leitlinien bezeichnet, das die Art von Fragen und die Methoden zu deren Beantwortung eingrenzt und leitet. Im Bereich der Programmierung bezieht sich der Begriff auf eine bestimmte Sichtweise, welche die Abbildung zwischen Wirklichkeit und Programm bestimmt.

2 Die Daten eines Programms können Instruktionen eines anderen Programms sein. Zum Beispiel stellt der Java-Bytecode Instruktionen für ein Java-Programm dar, für die Java Virtual Machine (JVM) ist der Java-Bytecode jedoch eine Sammlung von Daten. Ein anderes Beispiel sind Compiler, deren Ausgabedaten Instruktionen für die kompilierten Programme sind.

Jetzt werden Sie auf einige Begriffe treffen, die Ihnen sehr wahrscheinlich bekannt sind, die aber unterschiedlich interpretiert werden können. Wir schicken deshalb einige kurze Definitionen vorweg.

Routine [«]

Ein abgegrenzter, separat aufrufbarer Bestandteil eines Programms. Eine Routine kann entweder Parameter haben oder ein Ergebnis zurückgeben. Eine Routine wird auch als Unterprogramm bezeichnet.

Routinen sind das Basiskonstrukt der strukturierten Programmierung. Indem ein Programm in Unterprogramme zerlegt wird, erhält es seine grundsätzliche Struktur.

Funktion

Eine Funktion ist eine Routine, die einen speziellen Wert zurückliefert, ihren Rückgabewert.

Prozedur

Eine Prozedur ist eine Routine, die keinen Rückgabewert hat. Eine Übergabe von Ergebnissen an einen Aufrufer kann trotzdem über die Werte der Parameter erfolgen.

Die Unterscheidung zwischen Funktion und Prozedur, die wir hier getroffen haben, bewegt sich auf der Ebene von Programmen. Mathematische Funktionen ließen sich sowohl über Prozeduren als auch Funktionen abbilden.

Strukturierung von Instruktionen und Daten

Ein Weg, der stets wachsenden Komplexität der erstellten Programme Herr zu werden, ist die Strukturierung der Instruktionen und der Daten. Statt einfach die Instruktionen als einen monolithischen Block mit Sprüngen zu implementieren, werden die Instruktionen in Strukturen wie Verzweigungen, Zyklen und Routinen unterteilt. In Abbildung 2.1 ist ein Ablaufdiagramm dargestellt, das die Berechnung von Primzahlen als einen solchen strukturierten Ablauf darstellt.

Außerdem werden Daten bei der strukturierten Programmierung nicht als ein homogener Speicherbereich betrachtet, man benutzt globale und lokale, statisch und dynamisch allozierte Variablen, deren Inhalte definierte Strukturen wie einfache Typen, Zeiger, Records, Arrays, Listen, Bäume oder Mengen haben.

Typen von Daten

In den strukturierten Programmiersprachen definieren wir Typen der Daten, und wir weisen sie den Variablen, die sie enthalten können, zu. Auch die Parameter der Routinen, also der Prozeduren und der Funktionen, haben definierte Typen, und wir können sie nur mit entsprechend

strukturierten Daten aufrufen. Eine Prozedur mit falsch strukturierten Parametern aufzurufen, ist ein Fehler, der im besten Falle von einem Compiler erkannt wird, im schlimmeren Falle zu einem Laufzeitfehler oder einem Fehlverhalten des Programms führt.

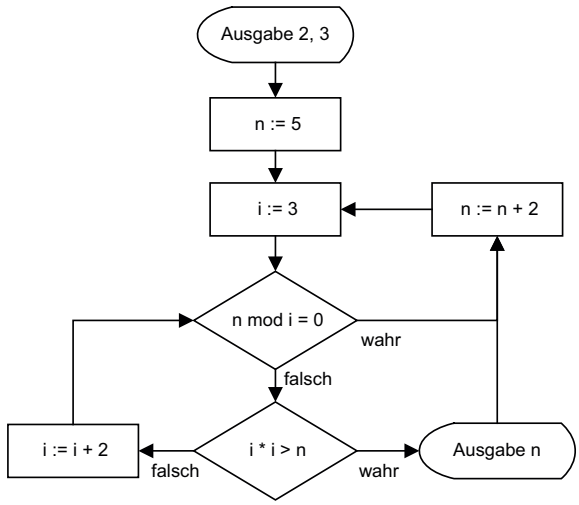

Abbildung 2.1 Ablaufdiagramm zur Berechnung von Primzahlen

Kontrolle und Verantwortung beim Programmierer

Der Programmierer hat die volle Kontrolle darüber, welche Routinen mit welchen Daten aufgerufen werden. Die Macht des Programmierers ist jedoch mit der Verantwortung verbunden, dafür zu sorgen, dass die richtigen Routinen mit den richtigen Daten aufgerufen werden.

In Abbildung 2.2 ist eine andere Variante der Darstellung für den Ablauf bei der Berechnung von Primzahlen dargestellt. Die in den sogenannten Nassi-Shneiderman-Diagrammen gewählte Darstellung bildet besser als ein Ablaufdiagramm die zur Verfügung stehenden Kontrollstrukturen ab.

Wir sehen: Das strukturierte Programmieren war ein großer Schritt in die Richtung der Beherrschung der Komplexität. Doch wie jede Vorgehensweise stößt auch diese bei bestimmten Problemen an ihre Grenzen. Eine Erweiterung der gewählten Vorgehensweise wird dadurch notwendig.

Die Objektorientierung stellt die Erweiterungen bereit. Im Folgenden lernen Sie die drei wichtigsten Erweiterungen in einem kurzen Überblick kennen.