mitp

Erich Gamma

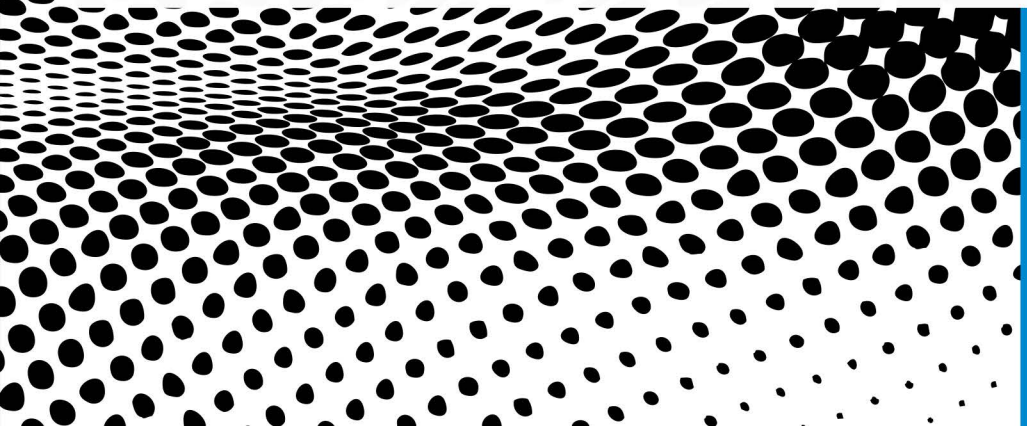
Richard Helm

Ralph Johnson

John Vlissides

Design Patterns

Entwurfsmuster als Elemente wiederverwendbarer objektorientierter Software



Design-Patterns-Katalog

Erzeugungsmuster (Creational Patterns)

Abstract Factory (*Abstrakte Fabrik*, siehe Abschnitt 3.1)

Bereitstellung einer Schnittstelle zum Erzeugen verwandter oder voneinander abhängiger Objektfamilien ohne die Benennung ihrer konkreten Klassen.

Builder (*Erbauer*, siehe Abschnitt 3.2)

Getrennte Handhabung der Erzeugungs- und Darstellungsmechanismen komplexer Objekte zwecks Generierung verschiedener Repräsentationen in einem einzigen Erzeugungsprozess.

Factory Method (*Fabrikmethode*, siehe Abschnitt 3.3)

Definition einer Schnittstelle zur Objekterzeugung, wobei die Bestimmung der zu instanziierenden Klasse den Unterklassen überlassen bleibt. Das Design Pattern *Factory Method (Fabrikmethode)* gestattet einer Klasse, die Instanziierung an Unterklassen zu delegieren.

Prototype (*Prototyp*, siehe Abschnitt 3.4)

Spezifikation der zu erzeugenden Objekttypen mittels einer prototypischen Instanz und Erzeugung neuer Objekte durch Kopieren dieses Prototyps.

Singleton (*Singleton*, siehe Abschnitt 3.5)

Sicherstellung der Existenz nur einer einzigen Klasseninstanz sowie Bereitstellung eines globalen Zugriffspunkts für diese Instanz.

Strukturmuster (Structural Patterns)

Adapter (*Adapter*, siehe Abschnitt 4.1)

Anpassung der Schnittstelle einer Klasse an ein anderes von den Clients erwartetes Interface. Das Design Pattern *Adapter (Adapter)* ermöglicht die Zusammenarbeit von Klassen, die ansonsten aufgrund der Inkompatibilität ihrer Schnittstellen nicht dazu in der Lage wären.

Bridge (*Brücke*, siehe Abschnitt 4.2)

Entkopplung einer Abstraktion von ihrer Implementierung, so dass beide unabhängig voneinander variiert werden können.

***Composite* (*Kompositum*, siehe Abschnitt 4.3)**

Komposition von Objekten in Baumstrukturen zur Abbildung von Teil-Ganzes-Hierarchien. Das Design Pattern *Composite (Kompositum)* gestattet den Clients einen einheitlichen Umgang sowohl mit individuellen Objekten als auch mit Objektkompositionen.

***Decorator* (*Dekorierer*, siehe Abschnitt 4.4)**

Dynamische Erweiterung der Funktionalität eines Objekts. `Decorator`-Objekte stellen hinsichtlich der Ergänzung einer Klasse um weitere Zuständigkeiten eine flexible Alternative zur Unterklassenbildung dar.

***Facade* (*Fassade*, siehe Abschnitt 4.5)**

Bereitstellung einer einheitlichen Schnittstelle zu einem Schnittstellensatz in einem Subsystem. Das Design Pattern *Facade (Fassade)* definiert eine Schnittstelle höherer Ebene, die die Nutzung des Subsystems vereinfacht.

***Flyweight* (*Fliegengewicht*, siehe Abschnitt 4.6)**

Gemeinsame Nutzung feingranularer Objekte, um sie auch in großer Anzahl effizient nutzen zu können.

***Proxy* (*Proxy*, siehe Abschnitt 4.7)**

Bereitstellung eines vorgelagerten Stellvertreterobjekts bzw. eines Platzhalters zwecks Zugriffssteuerung eines Objekts.

Verhaltensmuster (Behavioral Patterns)

***Chain of Responsibility* (*Zuständigkeitskette*, siehe Abschnitt 5.1)**

Unterbindung der Kopplung eines Request-Auslösers mit seinem Empfänger, indem mehr als ein Objekt in die Lage versetzt wird, den Request zu bearbeiten. Die empfangenden Objekte werden miteinander verkettet und der Request wird dann so lange entlang dieser Kette weitergeleitet, bis er von einem Objekt angenommen und abgearbeitet wird.

***Command* (*Befehl*, siehe Abschnitt 5.2)**

Kapselung eines Requests als Objekt, um so die Parametrisierung von Clients mit verschiedenen Requests, Warteschlangen- oder Logging-Operationen sowie das Rückgängigmachen von Operationen zu ermöglichen.

***Interpreter* (*Interpreter*, siehe Abschnitt 5.3)**

Definition einer Repräsentation für die Grammatik einer gegebenen Sprache und Bereitstellung eines Interpreters, der sie nutzt, um in dieser Sprache verfasste Sätze zu interpretieren.

***Iterator* (*Iterator*, siehe Abschnitt 5.4)**

Bereitstellung eines sequenziellen Zugriffs auf die Elemente eines aggregierten Objekts, ohne dessen zugrunde liegende Struktur offenzulegen.

***Mediator* (*Vermittler*, siehe Abschnitt 5.5)**

Definition eines Objekts, das die Interaktionsweise eines Objektsatzes in sich kapselt. Das Design Pattern *Mediator (Vermittler)* begünstigt lose Kopplungen, indem es die explizite Referenzierung der Objekte untereinander unterbindet und so eine individuelle Steuerung ihrer Interaktionen ermöglicht.

***Memento* (*Memento*, siehe Abschnitt 5.6)**

Erfassung und Externalisierung des internen Zustands eines Objekts, ohne dessen Kapselung zu beeinträchtigen, so dass es später wieder in diesen Zustand zurückversetzt werden kann.

***Observer* (*Beobachter*, siehe Abschnitt 5.7)**

Definition einer 1-zu-n-Abhängigkeit zwischen Objekten, damit im Fall einer Zustandsänderung eines Objekts alle davon abhängigen Objekte entsprechend benachrichtigt und automatisch aktualisiert werden.

***State* (*Zustand*, siehe Abschnitt 5.8)**

Anpassung der Verhaltensweise eines Objekts im Fall einer internen Zustandsänderung, so dass es den Anschein hat, als hätte es seine Klasse gewechselt.

***Strategy* (*Strategie*, siehe Abschnitt 5.9)**

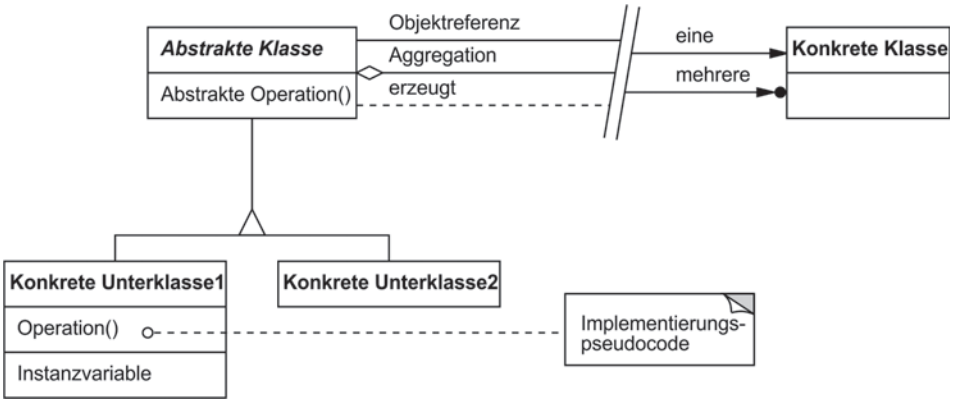
Definition einer Familie von einzeln gekapselten, austauschbaren Algorithmen. Das Design Pattern *Strategy (Strategie)* ermöglicht eine variable und von den Clients unabhängige Nutzung des Algorithmus.

***Template Method* (*Schablonenmethode*, siehe Abschnitt 5.10)**

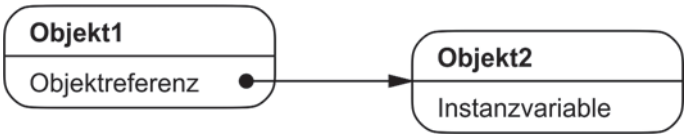
Definition der Grundstruktur eines Algorithmus in einer Operation sowie Delegation einiger Ablaufschritte an Unterklassen. Das Design Pattern *Template Method (Schablonenmethode)* ermöglicht den Unterklassen, bestimmte Schritte eines Algorithmus zu überschreiben, ohne dessen grundlegende Struktur zu verändern.

***Visitor* (*Besucher*, siehe Abschnitt 5.11)**

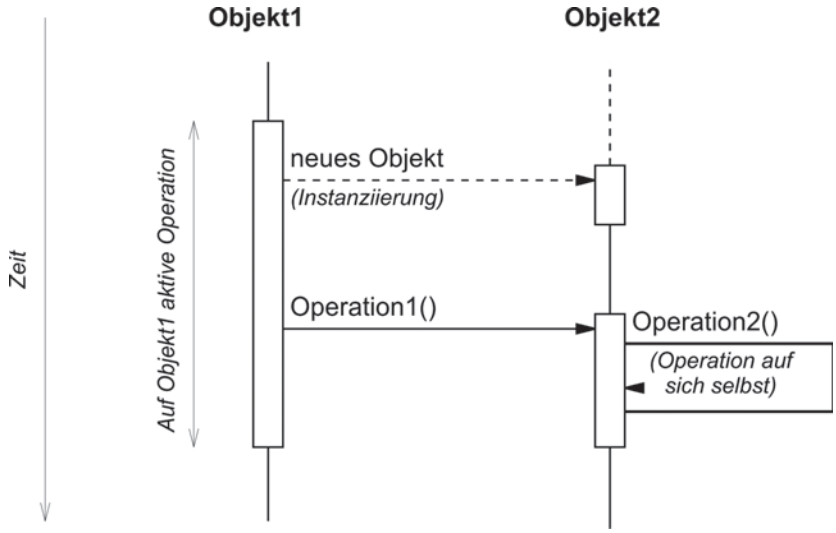
Darstellung einer auf die Elemente einer Objektstruktur auszuführenden Operation. Das Design Pattern *Visitor (Besucher)* ermöglicht die Definition einer neuen Operation, ohne die Klassen der von ihr bearbeiteten Elemente zu verändern.



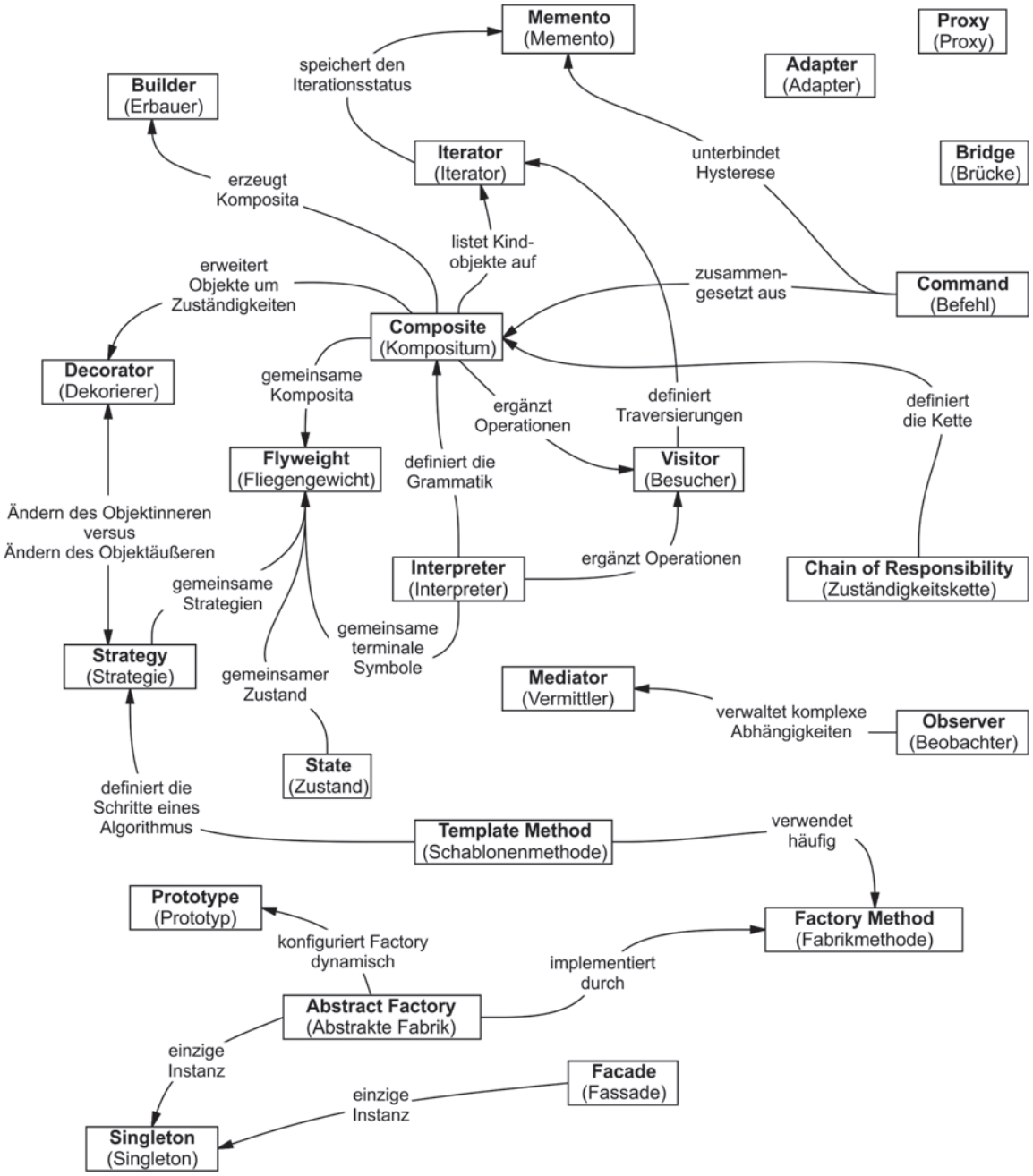
Notation für Klassendiagramme



Notation für Objektdiagramme



Notation für Interaktionsdiagramme



Beziehungen der Design Patterns

Design Patterns

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

Design Patterns

Entwurfsmuster als Elemente wiederverwendbarer objektorientierter Software

Übersetzung aus dem Amerikanischen
von Maren Feilen und Knut Lorenzen

mitp

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 978-3-8266-9903-0
1. Auflage 2015

www.mitp.de
E-Mail: mitp-verlag@sigloch.de
Telefon: +49 7953 / 7189 - 079
Telefax: +49 7953 / 7189 - 082

Authorized translation from the English language edition, entitled DESIGN PATTERNS: ELEMENTS OF REUSABLE OBJECT-ORIENTED SOFTWARE, 1st Edition, 0201633612 by GAMMA, ERICH; HELM, RICHARD, JOHNSON, RALPH; VLISSIDES, JOHN, published by Pearson Education, Inc, publishing as Addison-Wesley Professional, Copyright © 1995 by Addison-Wesley.

Lektorat: Sabine Schulz
Sprachkorrektorat: Maren Feilen, Knut Lorenzen
Satz: III-satz, Husby, www.drei-satz.de
Coverbild: © art_of_sun @ Fotolia.com

Für Karin
– E.G.

Für Sylvie
– R.H.

Für Faith
– R.J.

Für Dru Ann und Matthew
Josua 24:15b
– J.V.

Inhaltsverzeichnis

Vorwort

Dieses Buch wird Ihnen keine Einführung in die objektorientierte Programmierung oder das Anwendungsdesign als solches bieten – denn immerhin gibt es bereits zahlreiche sehr gute Nachschlagewerke, die diese Themen ausführlich behandeln. Die Inhalte dieses Buches setzen vielmehr voraus, dass Sie mindestens eine objektorientierte Programmiersprache relativ gut beherrschen und über grundlegende Erfahrungen im objektorientierten Design verfügen, so dass Sie nicht zum nächstbesten Fachlexikon greifen müssen, wenn Begriffe wie »Typen«, »Polymorphie« oder »Schnittstellenvererbung« statt »Implementierungsvererbung« zur Sprache kommen.

Andererseits handelt es sich bei diesem Buch aber auch nicht um eine technische Abhandlung für fortgeschrittene Entwickler. Es befasst sich mit **Design Patterns**, sprich Entwurfsmustern, die einfache und elegante Lösungen für spezifische Problemstellungen im objektorientierten Softwaredesign anbieten. Design Patterns repräsentieren im Grunde genommen Problemlösungen, die sich in der Praxis als sinnvoll und hilfreich erwiesen haben. Sie beschreiben Designs, die man normalerweise nicht ohne Weiteres in dieser Art entwickelt. Die Patterns basieren auf zahllosen Design- und Programmrevisionen, die Softwareentwickler auf der Suche nach besseren Wiederverwendungsmöglichkeiten und größerer Flexibilität im Laufe der Zeit erarbeitet haben – und sie stellen diese Lösungen in einer konzentrierten und einfach anzuwendenden Form zur Verfügung.

Die in dem hier vorgestellten Katalog enthaltenen Patterns erfordern weder außergewöhnliche programmiersprachliche Funktionen noch irgendwelche raffinierten Programmiertricks, mit denen man Freunde und Vorgesetzte beeindrucken kann. Sie können allesamt in den standardmäßigen objektorientierten Programmiersprachen implementiert werden, sind allerdings mit etwas mehr Aufwand verbunden als Ad-hoc-Lösungen – dieser Mehraufwand wird jedoch immer mit einer deutlich besseren Wiederverwendbarkeit und höherer Flexibilität belohnt.

Wer die Arbeitsweise der Design Patterns erst einmal verstanden hat und diesbezüglich zu einem »Aha!«-Erlebnis gelangt ist, wird fortan völlig anders über das objektorientierte Design denken. Die Patterns vermitteln Ihnen Einsichten, die es Ihnen ermöglichen, Ihre Designs flexibler, modularer, wiederverwendbarer und verständlicher zu gestalten – und genau das zählt schließlich zu den Hauptgründen, sich der objektorientierten Programmierung zuzuwenden, richtig?

An dieser Stelle sei aber auch gleich angemerkt: Machen Sie sich keine Gedanken, wenn Sie den Inhalt dieses Buches nicht schon bei der ersten Lektüre vollumfänglich verstehen. Selbst wir Autoren haben nicht immer alle Zusammenhänge dessen, was wir da zu Papier gebracht haben, sofort zu 100% verstanden! Dieses Buch ist kein »Schmöker«, den man einmal liest und dann ins Bücherregal stellt – es ist im wahrsten Sinne des Wortes ein Nachschlagewerk, von dem wir hoffen, dass Sie es zukünftig immer wieder aufschlagen werden, um neue Erkenntnisse und Anregungen für Ihre eigenen Designs zu gewinnen.

Die Fertigstellung dieses Buches hat sich recht lange hingezogen. Es hat sozusagen vier Länder »bereist«, die Hochzeiten von drei seiner Autoren und sogar die Geburten von zwei (nicht miteinander verwandten) Sprösslingen erlebt. Und es waren sehr viele Menschen an der Entstehung dieses Werkes beteiligt. Besonderer Dank gebührt in diesem Zusammenhang Bruce Anderson, Kent Beck und André Weinand für ihre Inspiration und ihre hilfreichen Ratschläge. Unser Dank richtet sich auch an all jene, die unsere Manuskriptentwürfe gelesen und kritisch kommentiert und bewertet haben: Roger Bielefeld, Grady Booch, Tom Cargill, Mashall Cline, Ralph Hyre, Brian Kernighan, Thomas Laliberty, Mark Lorenz, Arthur Riel, Doug Schmidt, Clovis Tondo, Steve Vinoski und Rebecca Wirfs-Brock. Weiterhin danken wir dem Team von Addison-Wesley für seine Hilfe und Geduld: Kate Habib, Tiffany Moore, Lisa Raffaele, Pradeepa Siva und John Wait. Ein ganz besonderer Dank geht an Carl Kessler, Danny Sabbah und Mark Wegman von IBM Research für ihre unermüdliche Unterstützung für dieses Projekt.

Selbstverständlich danken wir auch all den Menschen, die uns per Internet und auf anderen Wegen ihre Erfahrungen mit und Meinungen zu den verschiedenen Patterns mitgeteilt haben, uns ermutigt haben und uns wissen ließen, dass unsere Arbeit die Mühe wert war. Stellvertretend für all diese Menschen bedanken wir uns an dieser Stelle bei Jon Avotins, Steve Berczuk, Julian Berdych, Matthias Bohlen, John Brant, Allan Clarke, Paul Chisholm, Jens Coldewey, Dave Collins, Jim Coplien, Don Dwiggins, Gabriele Elia, Doug Felt, Brian Foote, Denis Fortin, Ward Harold, Hermann Hueni, Nayeem Islam, Bikramjit Kalra, Paul Keefer, Thomas Kofler, Doug Lea, Dan LaLiberte, James Long, Ann Louise Luu, Pundi Madhavan, Brian Marick, Robert Martin, Dave McComb, Carl McConnell, Christine Mingins, Hanspeter Mössenböck, Eric Newton, Marianne Ozkan, Roxsan Payette, Larry Podmolik, George Radin, Sita Ramakrishnan, Russ Ramirez, Alexander Ran, Dirk Riehle, Bryan Rosenburg, Aamod Sane, Duri Schmidt, Robert Seidl, Xin Shu und Bill Walker.

Wir betrachten den in diesem Buch vorgestellten Design-Pattern-Katalog keineswegs als vollständig und unveränderlich – er repräsentiert vielmehr eine Momentaufnahme unserer Überlegungen und Erwägungen in Bezug auf gutes Softwaredesign. Wir freuen uns über Meinungen und Kommentare jeder Art, sei es Kritik oder Lob zu unseren Beispielen, Hinweise auf Referenzen und Praxisbei-

spiele, die wir übersehen haben, oder Vorschläge für weitere Patterns, die wir ebenfalls in den Katalog hätten aufnehmen sollen. Ihre diesbezüglichen Anfragen erreichen uns per E-Mail an `design-patterns@cs.uiuc.edu`. Die in diesem Buch verwendeten Codebeispiele stehen unter `www.mitp.de/9700` zum Download zur Verfügung oder können alternativ per E-Mail mit dem Inhalt »send design pattern source« an `design-patterns-sourceics.uiuc.edu` bei uns angefordert werden.

E.G., Mountain View, California

R.H., Montreal, Quebec

R.J., Urbana, Illinois

J.V., Hawthorne, New York

August 1994

Geleitwort von Grady Booch

Alle gut strukturierten objektorientierten Architekturen basieren auf Mustern, auf Englisch »Patterns« genannt. Ich persönlich bemesse die Qualität eines objektorientierten Systems im Prinzip daran, wie viel Sorgfalt die Entwickler der grundsätzlich möglichen Zusammenarbeit der einzelnen Objekte gewidmet haben. Konzentriert man sich bei der Systementwicklung vorranging auf diese Mechanismen, dann gelangt man letztendlich zu Architekturen, die kompakter, schlichter und erheblich besser zu verstehen sind, als wenn diese Muster außer Acht gelassen werden.

Die Bedeutung von Patterns für die Erstellung komplexer Systeme wurde in anderen Disziplinen schon längst erkannt. Allen voran der Architekturtheoretiker Christopher Alexander und seine Mitarbeiter gehörten zu den Ersten, die die Etablierung und Instrumentalisierung einer »Pattern Language« (dt. »Muster-Sprache«) für die Konstruktion von Gebäuden und Städten vorschlugen. Seine Ideen und die Beiträge weiterer seiner Mitstreiter haben inzwischen auch in der objektorientierten Software Community Fuß gefasst. Kurzum: Das Konzept der Design Patterns verkörpert in der Programmentwicklung das Schlüsselelement, um sich die Erfahrungen und das Wissen anderer talentierter (Software-)Architekten zunutze zu machen.

Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides erläutern in diesem Buch die Grundsätze der Design Patterns und stellen einen Katalog solcher Muster vor. Damit leistet dieses Nachschlagewerk zwei wichtige Beiträge: Zum einen veranschaulicht es nachdrücklich die Rolle, die Patterns bei der Entwicklung komplexer Systeme einnehmen können. Und zum anderen dient es als äußerst praktische Referenz für eine Sammlung wohldurchdachter Patterns, die praktizierende Entwickler bei der Gestaltung ihrer eigenen spezifischen Anwendungen nutzen können.

Ich bin stolz darauf, dass mir die Ehre zuteilwurde, mit einigen der Autoren dieses Buches unmittelbar an ihren architektonischen Designbemühungen zusammenarbeiten zu dürfen. Ich habe viel von ihnen gelernt und bin überzeugt, dass es Ihnen bei der Lektüre dieses Buches ebenso ergehen wird.

Grady Booch

Chief Scientist, Rational Software Corporation

Einleitung

Dieses Buch gliedert sich im Wesentlichen in zwei Teile. Der erste Teil umfasst die Kapitel 1 und 2 und beschreibt, was Design Patterns sind und wie sie Ihnen bei der Entwicklung objektorientierter Software von Nutzen sein können. Zudem wird ihre praktische Anwendung anhand einer ausführlichen Fallstudie demonstriert. Im zweiten Teil des Buches (die Kapitel 3, 4 und 5) wird dann der eigentliche Katalog der einzelnen Design Patterns präsentiert.

Dieser Katalog macht den größten Teil des Buches aus. Die zugehörigen Kapitel 3, 4 und 5 unterteilen die Design Patterns in drei Kategorien: *Erzeugungsmuster (Creational Patterns)*, *Strukturmuster (Structural Patterns)* und *Verhaltensmuster (Behavorial Patterns)*.

Sie können den Katalog auf verschiedene Art und Weise nutzen: Entweder lesen Sie ihn von Anfang bis Ende durch oder wechseln von Pattern zu Pattern. Oder aber Sie konzentrieren sich jeweils auf ein Kapitel, um zu ergründen, inwiefern sich die Patterns ein und derselben Kategorie voneinander unterscheiden.

Die in den einzelnen Kapiteln angegebenen Querverweise zwischen den Patterns bilden einen roten Faden durch den Katalog. Auf diese Weise können Sie sich bequem einen Überblick darüber verschaffen, in welcher Beziehung die Patterns zueinander stehen, wie sie kombiniert werden können und welche Design Patterns gut zusammenarbeiten. Die verwandtschaftlichen Beziehungen der einzelnen Design Patterns sind in Abbildung 1.2 in einer Übersicht dargestellt.

Wenn Sie möchten, können Sie beim Lesen des Katalogs aber auch problemorientiert vorgehen. So könnten Sie z. B. gleich mit dem Abschnitt 1.6 beginnen, um sich über diverse allgemeine Probleme hinsichtlich des Designs wiederverwendbarer objektorientierter Software zu informieren, und anschließend die Ausführungen zu den Design Patterns lesen, die für die jeweiligen Problemstellungen geeignet sind. Oder Sie arbeiten zuerst den gesamten Katalog durch und gehen danach problemorientiert vor, um die passenden Patterns in Ihren Projekten anzuwenden.

Wenn Sie auf dem Gebiet der objektorientierten Entwicklung noch nicht so viel Erfahrung haben, empfiehlt es sich, mit den einfachsten und geläufigsten Patterns anzufangen:

- *Abstract Factory* (*Abstrakte Fabrik*, siehe Abschnitt 3.1)
- *Adapter* (*Adapter*, siehe Abschnitt 4.1)

- *Composite* (*Kompositum*, siehe Abschnitt 4.3)
- *Decorator* (*Dekorierer*, siehe Abschnitt 4.4)
- *Factory Method* (*Fabrikmethode*, siehe Abschnitt 3.3)
- *Observer* (*Beobachter*, siehe Abschnitt 5.7)
- *Strategy* (*Strategie*, siehe Abschnitt 5.9)
- *Template Method* (*Schablonenmethode*, siehe Abschnitt 5.10)

Kaum ein objektorientiertes System kommt ohne wenigstens ein paar Patterns aus – große Systeme nutzen sogar nahezu alle der hier genannten Exemplare. Diese Auswahl wird Ihnen helfen, die Design Patterns im Besonderen und das Konzept eines guten objektorientierten Designs im Allgemeinen wirklich zu verstehen.

Kapitel 1

Einführung

Das Entwickeln bzw. Entwerfen objektorientierter Software ist zweifellos kein leichtes Unterfangen – und das Designen *wiederverwendbarer* objektorientierter Software gestaltet sich sogar noch anspruchsvoller und komplexer: Neben der Bestimmung der relevanten Objekte und deren Abstrahierung zu Klassen in geeigneter Granularität müssen außerdem passende Schnittstellen und Vererbungshierarchien definiert sowie die zentralen Beziehungen zwischen den einzelnen Klassen bestimmt werden. Darüber hinaus sollte sich das Softwaredesign natürlich einerseits speziell an den jeweiligen Erfordernissen orientieren, andererseits aber gleichermaßen eine hinreichende Universalität aufweisen, um auch für zukünftige Problemstellungen und Anforderungen gewappnet zu sein. Und Designrevisionen sollten nach Möglichkeit ebenfalls vermieden oder zumindest auf ein Minimum reduziert werden.

Erfahrene objektorientierte Programmierer sind sich im Allgemeinen darüber im Klaren, dass es sehr schwierig, wenn nicht gar unmöglich ist, ein wiederverwendbares, flexibles Design gleich von Anfang an »richtig« hinzubekommen. Aus diesem Grund greifen sie häufig mehrfach auf ihre früheren Entwürfe zurück und versuchen zunächst, weitere Modifikationen zu implementieren, bevor sie ein Softwaredesign endgültig als »abgeschlossen« betrachten.

Das Erlernen und Verinnerlichen der maßgeblichen Aspekte, die ein gutes objektorientiertes Design ausmachen, erfordert viel Zeit und Geduld. Folglich fällt es erfahrenen objektorientierten Programmierern in der Regel leichter, gute Designs zu entwerfen, als weniger geübten Entwicklern, die sich nicht selten von den zahlreichen Designoptionen überfordert fühlen und daher oft doch lieber die nicht objektorientierten Techniken anwenden, die sie früher schon eingesetzt haben. Erfahrenen Programmierern ist also offenbar spezielles Wissen zu eigen, über das unerfahrene Entwicklern (noch) nicht verfügen. Doch worum genau handelt es sich dabei?

Nun, versierten Softwaredesignern ist beispielsweise bewusst, dass sie *nicht* ständig versuchen müssen, für jedes Problem eine völlig neue Lösung zu entwickeln. Stattdessen machen sie sich Lösungsmöglichkeiten zunutze, die sich nach ihrer persönlichen Erfahrung bereits bewährt haben – mit anderen Worten: Haben sie erst einmal gute Lösungsansätze gefunden, dann »recyceln« sie diese und wenden sie immer wieder an. Dieser Erfahrungsvorsprung zeichnet die Experten im Bereich des Softwaredesigns aus – und erklärt auch, warum viele objektorientierte

Systeme häufig wiederkehrende Klassenmuster und kommunizierende Objekte enthalten. Solche Muster (engl. *Patterns*) beheben bestimmte Designprobleme und gestalten objektorientierte Softwareentwürfe nicht nur flexibler und eleganter, sondern gestatten letztlich überhaupt erst ihre Wiederverwendbarkeit. Sie ermöglichen den Entwicklern, neue Designs auf gelungenen Entwürfen aufzusetzen und darauf aufzubauen. Kurz gesagt: Softwaredesigner, die mit Patterns vertraut sind, können diese unmittelbar auf die jeweils vorliegenden Problemstellungen anwenden, ohne erst »das Rad neu erfinden« zu müssen.

Ein vergleichbares Prinzip findet auch in anderen kreativen Prozessen Anwendung. So erarbeiten beispielsweise Roman- und Bühnenautoren die Handlungen ihrer Geschichten und Theaterstücke nur höchst selten von Grund auf – vielmehr bedienen sie sich in den meisten Fällen ebenfalls eines bewährten »Musters« bzw. Leitmotivs wie etwa »tragische Heldenfigur« (Macbeth, Hamlet etc.) oder »romantische Erzählung« (zahllose Liebesromane). In ähnlicher Weise setzen auch die objektorientierten Programmierer Entwurfsmuster (engl. *Design Patterns*) wie etwa »Zustände werden durch Objekte repräsentiert« oder »Objekte werden zwecks einfacher Ergänzung/Entfernung von Eigenschaften dekoriert« ein. Und steht das Muster erst einmal fest, ergeben sich viele weitere Designentscheidungen ganz automatisch.

Als Entwickler wissen wir alle den Wert der praktischen Designerfahrung zu schätzen. Haben Sie beim Betrachten eines Softwaredesigns nicht auch schon mal ein *Déjà-vu*-Erlebnis gehabt – das Gefühl, ein bestimmtes Problem in der Vergangenheit bereits bewältigt zu haben, ohne dass Sie sich daran erinnern könnten, wie genau oder in welchem Zusammenhang? Wenn Ihnen die exakte Problemstellung bzw. der seinerzeit eingeschlagene Lösungsweg wieder einfallen würde, könnten Sie auf diese Erfahrung zurückgreifen, statt ganz von vorn anfangen zu müssen. Leider werden die im Laufe eines objektorientierten Designprozesses gewonnenen Erkenntnisse in der Regel jedoch nicht so umfassend protokolliert, dass auch Dritte davon profitieren könnten.

Deshalb lautet die Zielsetzung dieses Buches, besagten Erkenntnisgewinn anhand von sogenannten **Design Patterns** (auch **Entwurfsmuster** oder kurz **Patterns** genannt) zu dokumentieren, damit er für jedermann zugänglich und effizient nutzbar wird. Zu diesem Zweck haben wir die wichtigsten Patterns in Katalogform zusammengefasst, wobei jedes Muster systematisch je ein bedeutsames wiederkehrendes Design benennt, beschreibt und evaluiert.

Design Patterns erleichtern nicht nur die Wiederverwendung erfolgreicher Softwaredesigns und -architekturen, sondern bieten Entwicklern zudem auch einen besseren, ungehinderten Zugang zu bewährten Techniken. Außerdem dienen sie als Entscheidungshilfe bei der Wahl möglicher Designalternativen zwecks Gewährleistung der Wiederverwendbarkeit eines Systems und verhindern gleichzeitig Alternativen, die einer Wiederverwendung entgegenstehen. Im Übrigen können Patterns ebenfalls dazu beitragen, die Dokumentation und Wartung bereits existenter Systeme zu verbessern, indem sie explizite Spezifikationen für

die Klassen- und Objektinteraktionen sowie deren Intention bereitstellen. Unterm Strich heißt das: Design Patterns unterstützen Programmierer dabei, ihre Softwaredesigns schneller »richtig« hinzubekommen.

Keins der in diesem Buch beschriebenen Patterns ist vollkommen neu oder unerprobt. Im Gegenteil: Es werden ausschließlich solche Design Patterns verwendet, die schon mehrfach in verschiedenen Systemen eingesetzt wurden. Die meisten von ihnen wurden allerdings noch nie zuvor dokumentiert, sondern entstammen entweder dem gemeinschaftlichen Fundus der objektorientierten Community oder erfolgreichen objektorientierten Systemen – also zwei Bereichen, die sich unerfahrenen Programmierern nicht so leicht erschließen. Aber auch wenn die vorgestellten Patterns nicht brandneu sind, werden sie doch in einer neuartigen, leicht verständlichen Art und Weise vorgestellt: in Form eines Design-Pattern-Katalogs, der einer einheitlichen Präsentationskonvention folgt.

Aufgrund des naturgemäß begrenzten Platzangebots in diesem Buch repräsentieren die hier erläuterten Muster lediglich einen Bruchteil des Pattern-Fundus, der echten Programmierexperten zur Verfügung steht. So werden beispielsweise keine Design Patterns für die nebenläufige, die verteilte oder die Echtzeitprogrammierung oder für spezifische Anwendungsdomänen verwendet. Ebenso wenig werden Sie in diesem Buch Informationen zur Entwicklung von Benutzeroberflächen, zum Schreiben von Gerätetreibern oder zum Einsatz objektorientierter Datenbanken vorfinden. Für all diese Aufgabenbereiche existieren wiederum spezielle Design Patterns – die es jedoch sicherlich ebenfalls wert wären, eigens katalogisiert zu werden.

1.1 Was ist ein Design Pattern?

Der US-amerikanische Architekturtheoretiker Christopher Alexander erklärt in seinem Buch *»Eine Muster-Sprache«* [Löcker Verlag, Wien, 1995, Seite *x*]: »Jedes Muster beschreibt zunächst ein in unserer Umwelt immer wieder auftretendes Problem, beschreibt dann den Kern der Lösung dieses Problems, und zwar so daß man diese Lösung millionenfach anwenden kann, ohne sich je zu wiederholen.« Natürlich bezieht sich Alexander in diesem Fall auf Muster, die sich in Gebäuden und Stadtplanungsentwürfen wiederfinden, dennoch trifft seine Definition auch auf objektorientierte Design Patterns zu – mit dem Unterschied, dass die Lösungen hier nicht in Form von Wänden und Türen, sondern von Objekten und Schnittstellen ausgedrückt werden. Prinzipiell repräsentieren jedoch beide Musterspezies Problemlösungen für bestimmte Situationen in bestimmten Kontexten.

Ein Design Pattern umfasst im Wesentlichen vier maßgebliche Elemente:

1. Der kurze, meist aus ein oder zwei Wörtern bestehende **Pattern-Name** dient zur Referenzierung des jeweiligen Designproblems sowie der zugehörigen Lösungen und deren Auswirkungen. Durch die Benennung eines Design Patterns erweitern wir unser designbezogenes Vokabular in der Art, dass ein

Arbeiten auf einer höheren Abstraktionsebene möglich wird – denn dieses Vokabular erleichtert die Dokumentation des Softwaredesigns und vor allem auch die Kommunikation mit Kollegen und Dritten, z. B. zur Erörterung der Vor- und Nachteile neuer Ideen und Vorschläge, in erheblichem Maße. Wirklich geeignete Namen zu finden, ist allerdings keine leichte Aufgabe – das zeigt auch die Tatsache, dass die Vergabe passender Bezeichnungen (im Englischen wie im Deutschen) für die in diesem Buch vorgestellten Design Patterns bei der Entwicklung unseres Katalogs eine der größten Herausforderungen überhaupt darstellte.

2. Die **Problemstellung** beschreibt die Situation, in der das Design Pattern anzuwenden ist. Sie erläutert die jeweils vorliegende Problematik sowie deren Kontext. Dabei kann es sich um spezifische Designprobleme handeln, wie z. B. in welcher Form Algorithmen als Objekte abgebildet werden sollten, aber auch um für unflexible Designs symptomatische Klassen- oder Objektstrukturen. Mitunter werden im Rahmen der Problembeschreibung außerdem bestimmte Bedingungen aufgeführt, die erfüllt sein müssen, damit der Einsatz des Design Patterns überhaupt sinnvoll ist.
3. Die **Lösung** berücksichtigt neben den konkreten designbildenden Elementen auch deren Beziehungen zueinander, ihre Zuständigkeiten sowie ihre Interaktionen. Sie definiert allerdings kein bestimmtes Design oder eine konkrete Implementierung, denn ein Pattern entspricht eher einer Art Schablone, die in vielen verschiedenen Situationen anwendbar ist: Es skizziert eine abstrakte Beschreibung eines Designproblems und zeigt auf, wie dieses durch eine generelle Anordnung von Elementen (in unserem Fall Klassen und Objekten) bewältigt werden kann.
4. Die **Konsequenzen** bilden eine Übersicht der Auswirkungen und Kompromisse ab, die sich aus der Anwendung des Patterns ergeben. Leider bleiben sie, obwohl sie für die Bewertung möglicher Designalternativen und das Abwägen ihrer Vor- und Nachteile von entscheidender Bedeutung sind, in der Argumentation für eine Designentscheidung häufig unerwähnt.

   In der Softwareentwicklung stehen die Konsequenzen eines Pattern-Einsatzes oftmals in direktem Zusammenhang mit dem Speicherplatzbedarf und den Ausführungszeiten, sie können aber ebenso gut auch Sprach- und andere Implementierungsaspekte betreffen. Und da die Wiederverwendbarkeit in der objektorientierten Programmierung eine wichtige Rolle spielt, schließt dies selbstverständlich auch den Einfluss des infrage stehenden Design Patterns auf die Flexibilität, Erweiterbarkeit und Portabilität des Systems mit ein. Generell gilt also: Die ausdrückliche, sachliche Erwägung der zu erwartenden Konsequenzen ist für die lückenlose Evaluierung eines Patterns unverzichtbar.

Die Interpretation des tatsächlichen Nutzwertes eines Design Patterns hängt im Endeffekt immer auch von der Perspektive des Betrachters ab: Während ein Pat-