Peter **FORBRIG**

OBJEKTORIENTIERTE SOFTWAREENTWICKLUNG MIT UML

4. Auflage

Zusatzmaterial unter
plus.hanser-fachbuch.de

HANSER

Forbrig/Dittmar

Objektorientierte Softwareentwicklung mit UML

Ihr Plus – digitale Zusatzinhalte!

Auf unserem Download-Portal finden Sie zu diesem Titel kostenloses Zusatzmaterial.

Geben Sie auf **plus.hanser-fachbuch.de** einfach diesen Code ein:

plus-Dt4Ug-39brm

Bleiben Sie auf dem Laufenden!

Hanser Newsletter informieren Sie regelmäßig über neue Bücher und Termine aus den verschiedenen Bereichen der Technik. Profitieren Sie auch von Gewinnspielen und exklusiven Leseproben. Gleich anmelden unter

www.hanser-fachbuch.de/newsletter

■ Lehrbücher zur Informatik

Begründet von
Prof. Dr. Michael Lutz und *Prof. Dr. Christian Märtin*
weitergeführt von
Prof. Dr. Christian Märtin
Hochschule Augsburg Fachbereich Informatik

■ Zu dieser Buchreihe

Die Werke dieser Reihe bieten einen gezielten Einstieg in grundlegende oder besonders gefragte Themenbereiche der Informatik und benachbarter Disziplinen. Alle Autoren verfügen über langjährige Erfahrung in Lehre und Forschung zu den jeweils behandelten Themengebieten und gewährleisten Praxisnähe und Aktualität.

Die Bände der Reihe können vorlesungsbegleitend oder zum Selbststudium eingesetzt werden. Sie lassen sich teilweise modular kombinieren. Wegen ihrer Kompaktheit sind sie gut geeignet, bestehende Lehrveranstaltungen zu ergänzen und zu aktualisieren.

Die meisten Werke stellen Ergänzungsmaterialien wie Lernprogramme, Software-Werkzeuge, Online-Kapitel, Beispielaufgaben mit Lösungen und weitere aktuelle Inhalte auf eigenen Websites zur Verfügung.

Titel in dieser Reihe

- Peter Forbrig, Objektorientierte Softwareentwicklung mit UML
- Rainer Oechsle, Parallele und verteilte Anwendungen in Java
- Claudia Reuter, Requirements Engineering – klassisch, agil und hybrid [in Planung]
- Wolfgang Riggert/Ralf Lübben, Rechnernetze
- Georg Stark, Robotik mit MATLAB
- Rolf Socher, Theoretische Grundlagen der Informatik
- Carsten Vogt, Von Java zu C, 2. A. [in Planung]

Peter Forbrig
Anke Dittmar

Objektorientierte Softwareentwicklung mit UML

4., aktualisierte und erweiterte Auflage

HANSER

Prof. Dr.-Ing. habil. Peter Forbrig, Rostock
Dr.-Ing. Anke Dittmar, Rostock

Alle in diesem Werk enthaltenen Informationen, Verfahren und Darstellungen wurden nach bestem Wissen zusammengestellt und mit Sorgfalt getestet. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grund sind die im vorliegenden Werk enthaltenen Informationen mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autor*in und Verlag übernehmen infolgedessen keine juristische Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieser Informationen – oder Teilen davon – entsteht. Ebenso wenig übernehmen Autor*in und Verlag die Gewähr dafür, dass die beschriebenen Verfahren usw. frei von Schutzrechten Dritter sind. Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt also auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Die endgültige Entscheidung über die Eignung der Informationen für die vorgesehene Verwendung in einer bestimmten Anwendung liegt in der alleinigen Verantwortung des Nutzers.

Aus Gründen der besseren Lesbarkeit wird auf die gleichzeitige Verwendung der Sprachformen männlich, weiblich und divers (m/w/d) verzichtet. Sämtliche Personenbezeichnungen gelten gleichermaßen für alle Geschlechter.

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.

Dieses Werk ist urheberrechtlich geschützt.
Alle Rechte, auch die der Übersetzung, des Nachdruckes und der Vervielfältigung des Buches, oder Teilen daraus, vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren) – auch nicht für Zwecke der Unterrichtsgestaltung – mit Ausnahme der in den §§ 53, 54 URG genannten Sonderfälle –, reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

© 2024 Carl Hanser Verlag München, http://www.hanser-fachbuch.de
Lektorat: Brigitte Bauer-Schiewek
Copy editing: Jürgen Dubau, Freiburg/Elbe
Umschlagdesign: Marc Müller-Bremer, www.rebranding.de, München
Umschlagrealisation: Max Kostopoulos
Titelmotiv: © Thomas West
Satz: Eberl & Koesel Studio, Kempten
Druck und Bindung: Hubert & Co. GmbH & Co. KG BuchPartner, Göttingen
Printed in Germany

Print-ISBN 978-3-446-47951-7
E-Book-ISBN 978-3-446-48053-7
E-Pub-ISBN 978-3-446-48144-2

Inhalt

Vorwort

Sowohl die objektorientierte Programmierung als auch die objektorientierte Modellierung haben in den letzten Jahren beträchtlich an Bedeutung gewonnen. Zunächst sind die Vorteile des objektorientierten Ansatzes bei der Programmierung sichtbar geworden. Das zeigt sich auch an der Vielzahl von Programmiersprachen, die die entsprechenden Konzepte unterstützen. Mehr und mehr hat sich der Einfluss aber auch auf die frühen Phasen der Softwareentwicklung ausgedehnt. Darin ist eine Parallele zur Entwicklung des strukturierten Ansatzes zu sehen. Auch dort ging die Entwicklung von der Durchsetzung der Konstrukte zur strukturierten Programmierung in der Algorithmierung und in den Programmiersprachen aus. Später führte dies zum strukturierten Entwurf und zur strukturierten Analyse.

Mit der Unified Modeling Language (UML) hat sich eine Modellierungssprache für die objektorientierte Spezifikation herausgebildet, die große Akzeptanz in der Industrie findet. Damit ist die UML auf dem besten Weg zum Standard: Die Sprache ist nicht nur standardisiert, sondern wird in vielen Bereichen angewendet und ist Gegenstand einer großen Zahl von Werkzeugen. Viele Hersteller von CASE-Tools in diesem Bereich sind bemüht, den vollständigen Sprachumfang von UML zu unterstützen.

Aus diesen Gründen müssen auch Ausbildungseinrichtungen wie Universitäten und Hochschulen die Anwendung der Sprachelemente der UML in ihren Lehrprogrammen berücksichtigen. Dabei geht es aber nicht nur um die richtige Notation der Spezifikationen, sondern auch um die Vermittlung der zugrunde liegenden Konzepte und deren korrekte Anwendung für bestimmte Problemstellungen.

Der Inhalt des Buches basiert auf Erfahrungen von Lehrveranstaltungen zur Softwaretechnik, in denen objektorientierte Konzepte vermittelt wurden. Das Buch versucht an Hand von Beispielen, einen Einstieg in die objektorientierte Spezifikation mit UML zu ermöglichen. Es ist nicht darauf angelegt, alle Einzelheiten darzustellen, die mit der Definition von UML zusammenhängen, sondern hier sollen die wichtigsten Informationen geliefert werden, die einen Einstieg in die Projektarbeit erleichtern. Die Beispiele sind in der aktuellen Version von UML 2.5.1 formuliert.

Besonders viel Aufmerksamkeit erfährt die Spezifikation dynamischer Zusammenhänge. Hierfür wurden Videos erarbeitet, die das Verständnis der Spezifikation mit endlichen Automaten in Form von Zustandsdiagrammen erleichtern sollen. Daneben werden auch die Möglichkeiten von Aktivitätsdiagrammen aufgezeigt und der Zusammenhang zu den besonders bei Banken und Versicherungen sehr beliebten Ereignis-Prozess-Ketten hergestellt.

Entwurfsmuster sind ein weiteres wichtiges Konzept. Durch die Wiederverwendung von Software auf einem völlig neuen Niveau sind sie ein Schlüssel für eine erfolgreiche Softwareentwicklung. Die Idee der Entwurfsmuster (engl. Design Patterns) wird vorgestellt und die Notationsmöglichkeiten in UML werden diskutiert. Auch Konsequenzen für die Programmierung in Java, Python, C# und Eiffel werden aufgezeigt. Die Quelltexte für jede Sprache sind auf der Webseite des Verlages verfügbar, sofern die Sprache dies ohne größeren Aufwand ermöglichte.

Auch der textuellen Spezifikation in OCL (Object Consraint Language) widmet sich ein Abschnitt. Bei OCL handelt es sich um eine textuelle Teilsprache von UML, die notwendig ist, wenn die grafischen Ausdrucksmittel nicht genügend Aussagekraft besitzen. Für die praktische Anwendung von UML in größeren Projekten hilft OCL, Mehrdeutigkeiten zu vermeiden.

Benutzerorientierte Entwicklungsansätze wurden im neuen Kapitel 4 aufgegriffen. Sie müssen in bestehende Vorgehensweisen des Softwareengineering, wie agile Softwareentwicklung, integriert werden, um die Perspektive der Benutzer stärker zu beachten.

Modelle der UML lassen sich gut mit Skizzen, Stories oder dem aus der Softwareergonomie bekannten Konzept der Personas verbinden. Es wird beispielsweise gezeigt, wie durch die Nutzung der Ideen von Personas die Spezifikation von Anwendungsfalldiagrammen ausdrucksstärker gestaltet werden kann.

Von den Fähigkeiten der Beteiligten und den konkreten Projektbedingungen hängt es ab, in welchem Maße beispielsweise Anwendungsfallmodelle, Klassendiagramme, Zustandsübergangsautomaten, Aufgabenmodelle, Szenarien, Personas, Skizzen oder Prototypen sinnvoll genutzt werden können.

Für die in der agilen Softwareentwicklung gebräuchlichen User Stories wird die Idee der Story-Splitting-Patterns diskutiert, die ein Pendant zu den Design Patterns der „Gang of Four“ darstellen.

Das Buch ist folgendermaßen aufgebaut:

- **Kapitel 1** gibt eine Einführung in die wichtigsten Grundbegriffe der Objektorientierung.
- **Kapitel 2** stellt die Sprache UML vor und ergänzt die verschiedenen Diagramme stets mit einer Reihe von Anwendungsbeispielen.
- **Kapitel 3** beschäftigt sich mit den Problemen der Softwarespezifikation bezogen auf den gesamten Lebenszyklus und stellt unterstützende Techniken zur Ermittlung von Anforderungen vor. Außerdem werden Entwurfsmuster und Modelltransformationen sowie deren Werkzeugunterstützung etwas genauer betrachtet.
- **Kapitel 4** wurde zusammen mit Frau Dr. Anke Dittmar geschrieben. Es widmet sich der kollaborativen Analyse und dem kollaborativen Design. Heutzutage müssen komplexe interaktive Systeme in interdisziplinären Teams gestaltet werden. Dazu ist die Einbeziehung der Expertise aller Beteiligten notwendig. Speziell die späteren Benutzer des Systems sollte man frühzeitig mit in die Entwicklung einbeziehen.

Mit den diskutierten Beispielen und Modellen hoffen wir, den Lesern Möglichkeiten aufgezeigt zu haben, wie Modelle der UML genutzt und mit weiteren Sichten kombiniert werden können.

Dieses Buch ermöglicht in der aktualisierten Fassung einen optimalen Einstieg in die Softwareentwicklung auf der Basis von UML. Und der Autor hofft, dass das Buch nicht nur Anfängern beim Einstieg in das Thema begleitet, sondern auch Fortgeschrittene neue Erkenntnisse gewinnen. Konstruktive Hinweise zur Verbesserung der Darstellung sind herzlich willkommen (*peter.forbrig@uni-rostock.de*).

Beim Hanser Fachbuchverlag möchte ich mich ganz herzlich für die Unterstützung bedanken. Mein besonderer Dank geht an Brigitte Bauer-Schiewek, Kristin Rothe, Irene Weilhart und Jürgen Dubau.

Rostock, im Januar 2024

Zusatzmaterial zum Buch

Zu diesem Buch stehen Ihnen weitere Inhalte digital zur Verfügung:

- Neben Korrekturen zum Manuskript befinden sich dort Interessante Links zu UML Systemunterlagen, zu im Buch diskutierten Werkzeugen wie ArgoUML, USE, CTTE und zur Unterstützung von Patterns, zu Lehrmaterialien zu Entwurfsmustern und zu Videos mit animierten Zustandsautomaten.
- Als Downloads finden Sie alle Abbildungen des Buches, die Quelltexte der Programme aus dem Buch sowie die entsprechenden Programme in Java, C#, Python und Eiffel, sowie Lösungen zu den im Buch formulierten Aufgaben. Außerdem gibt es dort Links zum Herunterladen von Programmierumgebungen, eigenen Werkzeugen und einem Word-Template zur textuellen Beschreibung von Anwendungsfällen.

Gehen Sie dazu einfach auf

plus.hanser-fachbuch.de

und geben Sie dort diesen Code ein:

`plus-Dt4Ug-39brm`

1 Grundbegriffe der objektorientierten Softwareentwicklung

1.1 Einführung

Mit der Objektorientierung ist in den letzten Jahren ein Paradigmenwechsel in der Softwareentwicklung eingetreten, der sich von der Implementation über den Entwurf bis zur Analyse in die sehr frühen Phasen durchgesetzt hat. Die zunächst gültige Trennung von Daten und Funktionen wurde überwunden. David Parnas [1.6] propagierte Anfang der 70er Jahre die Nutzung von Datenkapseln, bei denen der Zugriff auf die Daten nur über eine Menge bereitgestellter Funktionen, die sogenannte Schnittstelle, ermöglicht wurde. Es hat eine ganze Weile gedauert, bis sich diese Idee in der Praxis der Softwareentwicklung durchgesetzt hat. Um eine Vielzahl von derartigen Datenkapseln schnell erzeugen zu können, folgte später die Idee der Programmierung von abstrakten Datentypen. Auch hier trat der Erfolg nicht sofort ein. Erst die Einordnung dieser Datentypen in eine Hierarchie, die über Vererbungsmechanismen verfügt, führte zu einem durchgreifenden Erfolg. Dieser Ansatz wurde als *objektorientiert* charakterisiert. Er ist eng mit den Begriffen von *Klasse* und *Objekt* verbunden.

Ein Objekt wird durch Eigenschaften und Fähigkeiten charakterisiert. Die Eigenschaften beschreiben den aktuellen Zustand des Objektes, und die Fähigkeiten stellen Tätigkeiten dar, die auf das Objekt angewendet werden können, um seine Eigenschaften zu verändern. So kann ein Füllfederhalter durch die Farbe der Tinte, mit der er gefüllt wurde, beschrieben werden. Diese Eigenschaft ist durch das *Entleeren* und nachfolgendes *Füllen* änderbar. Damit sind auch schon zwei Fähigkeiten genannt, die mithilfe des Füllfederhalters ausgeführt werden können. Die wichtigste Fähigkeit ist natürlich das *Schreiben*. Sie ist der Grund, warum man sich den Federhalter zulegt. Eine weitere Eigenschaft des Schreibgerätes beschreibt den Bereitschaftszustand. Es kann durch eine Schutzkappe *verschlossen* oder *unverschlossen* sein. Eine Veränderung des Bereitschaftszustandes erfolgt durch die Tätigkeiten *Öffnen* und *Schließen*, die weitere Fähigkeiten darstellen.

Bei der objektorientierten Analyse wird von den Objekten ausgegangen, die in der realen Welt existieren. Durch geeignete Abstraktion wird aus einem realen Objekt ein Objekt eines Modells. Dabei wird besonderes Augenmerk auf Charakteristika gelegt, die im Zusammenhang mit einer bestimmten Aufgabe von Interesse sind. Eine Modellierung ohne ein bestimmtes Ziel ist nicht möglich, weil die Anzahl der Charakteristika fast ins Unendliche steigt. Eigenschaften und Fähigkeiten werden durch *Attribute* und *Methoden* beschrieben.

Die Problematik der Modellierung der Charakteristika wird sicher am Beispiel deutlich. Gegenstand des Interesses sei eine Person. Da keine konkrete Zielvorgabe existiert, ist deren Modellierung praktisch nicht möglich. Ist die Haarfarbe von Interesse? Sind die Kinderkrankheiten wichtig? Welche Bedeutung haben Hobbys? Niemand kann das ohne präzisere Zusatzinformationen genau wissen. Für ein Programm zur Beratung von Farbvorschlägen für einen Friseur ist die momentane Haarfarbe der zu betreuenden Kundin von großer Bedeutung. Für die Diagnose von Erkrankungen sind die bereits durchlaufenen Kinderkrankheiten sicher wichtig. Ein System, das Literatur für einen Kunden empfehlen soll, möchte sicher auf die Hobbys der betreffenden Person zurückgreifen.

Hier sei eine Person im Kontext einer Universität exemplarisch modelliert. Zunächst wird sie als Felix identifiziert, der am 17.11.2007 geboren wurde und als Einkommen über ein Stipendium verfügt. Als besondere Fähigkeit fällt nur auf, dass er lernen und feiern kann. Diese Informationen können wie in Bild 1.1 dargestellt grafisch repräsentiert werden.

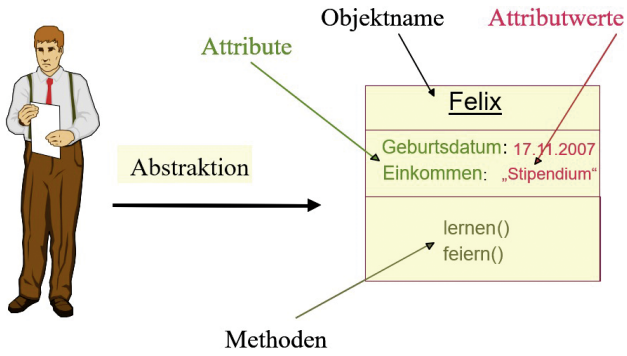


Bild 1.1 Modellierung eines Objektes

In der grafischen Repräsentation eines Objektes werden die Eigenschaften (`Geburtsdatum`, `Einkommen`) und die Fähigkeiten (`lernen`, `feiern`) getrennt dargestellt.

Fallen weitere Personen auf, die ähnliche Eigenschaften und Fähigkeiten besitzen, so sollten diese auch als Gruppe modellierbar sein. Dafür hat sich der schon in der Schule bekannte Begriff einer *Klasse*, der eine Reihe von Schülern beschreibt, die in etwa den gleichen Ausbildungsstand besitzen, eingebürgert. In diesem Zusammenhang betitelt der Begriff *Klasse* eine Sammlung von Objekten mit gleichen Charakteristika. Dabei handelt es sich um Objekte, die die gleichen Eigenschaften (Attribute) und Fähigkeiten (Methoden) besitzen.

Für obiges Beispiel weisen die Charakteristika von Felix darauf hin, dass die Klasse als Student bezeichnet werden kann. Dabei ist das Attribut entscheidend, welches das Einkommen charakterisiert. Studenten verfügen über ein Stipendium. Auch die beobachteten Fähigkeiten wie `lernen` und `feiern` stehen zu der Einschätzung nicht im Widerspruch.

Von dem Objekt Felix kann daher auf die Klasse Student abstrahiert werden. Das grafische Symbol für Objekte und Klassen ist gleich. Auch die Methoden werden identisch dargestellt. Nur an der Stelle des unterstrichenen Namens eines Objektes steht der Name einer Klasse (ohne Unterstrich). Die Darstellung der Namen von Attributen ist bei Objekten und

Klassen auch gleich. Nur haben Attribute von Objekten einen Wert, während das bei Klassen noch nicht vorliegt. Bei ihnen ist nur der Typ der Attribute bekannt (bei Programmiersprachen findet man Erweiterungen, die nicht dieser klassischen Sicht entsprechen).

Bild 1.2 zeigt die Darstellung einer Klasse, die mehrere Objekte repräsentiert.

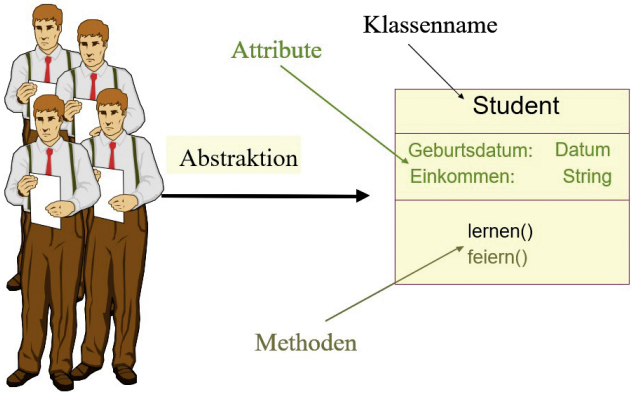


Bild 1.2 Modellierung einer Klasse

Eine recht anschauliche Abstraktion des Prozesses der Modellierung wurde von Heeg [1.4] gegeben, die hier aufgegriffen und etwas modifiziert in Bild 1.3 dargestellt wird.

Zunächst wird versucht, ein Problem der realen Welt aus einem subjektiven Blickwinkel zu modellieren. Der subjektive Blick des Modellierers wird durch das Auge symbolisiert. Er ist durch das Ziel der Modellierung beeinflusst. Dabei werden subjektiv Personen, Phänomene oder reale Dinge erfasst, die für das Modellierungsziel relevant sind. Deren Bezeichnungen werden in einem Glossar verwaltet. Dabei ist bereits auf Synonyme (unterschiedliche Bezeichnungen für gleiche Sachverhalte – z. B. Behälter, Ablage und Container) und Homonyme (gleiche Bezeichnungen für unterschiedliche Sachverhalte – z. B. Schloss) zu achten. In unbekannten Anwendungsbereichen ist das nicht immer ganz einfach.

Ist das Glossar ausreichend mit Informationen gefüllt, kann mit der Modellierung von Klassen begonnen werden. Zu den zunächst nur durch Begriffe charakterisierten Objekten werden Eigenschaften und Fähigkeiten ermittelt, die mit dem Modellierungsziel in Zusammenhang stehen.

Auf der Basis der modellierten Klassen findet die Entwicklung von Programmen statt, in denen Instanzen der Klassen erzeugt werden. Während der Laufzeit der Programme erfolgt die Erzeugung der entsprechenden Objekte einer Klasse, die durch Variablen repräsentiert werden. Diese Variablen ermöglichen die Beschreibung des Zugriffs auf die Objekte bereits während der Programmierung.

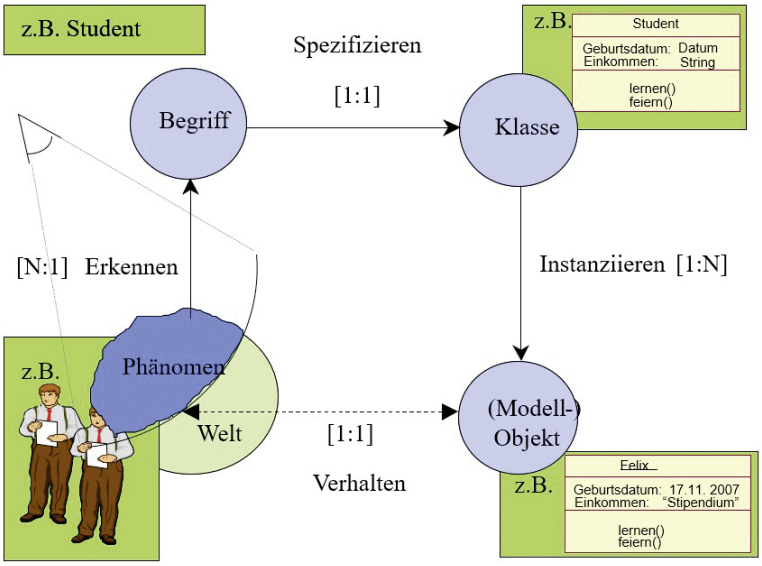


Bild 1.3 Objektorientierte Sichtweise der Softwareentwicklung

Im Allgemeinen entsprechen mehrere Objekte oder Phänomene der realen Welt einem Begriff, zu dem dann eineindeutig eine Klasse gleichen Namens existiert. Während der Laufzeit eines Programms kann aus einer Klasse eine Vielzahl von Objekten erzeugt werden. Ist die Modellierung korrekt gelungen, dann sollte das beobachtbare Verhalten der Objekte im Modellbereich dem beobachtbaren Verhalten in der realen Welt unter Einschränkung auf das Modellierungsziel exakt entsprechen. Es sollte jedem Modellobjekt eineindeutig ein Objekt aus der realen Welt zugeordnet werden können.

Genauso wie in der realen Welt kommunizieren Objekte in der Modellwelt über Botschaften.

An der Hochschule wird eventuell eine Lehrkraft einem Studenten die Botschaft `lernen` schicken, wenn sie der Meinung ist, dass diese Aufforderung notwendig ist. Bild 1.4 stellt diesen Sachverhalt grafisch dar. Ein Student wird entsprechend auf die Botschaft reagieren und seine Methode `lernen` aktivieren.

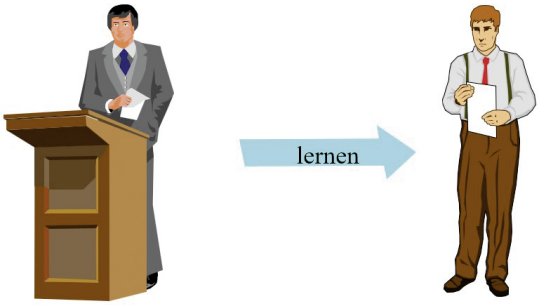


Bild 1.4 Übermittlung einer Botschaft in der Realität

Genauso sieht es auch in der Modellwelt aus (Bild 1.5). Trifft eine Botschaft bei einem Objekt ein, so wird überprüft, ob sie für das jeweilige Objekt von Bedeutung ist, ob eine Interpretation möglich ist.

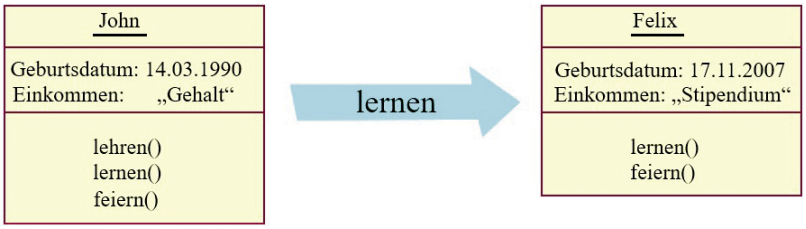


Bild 1.5 Kommunikation von Objekten in der Modellwelt

Mitunter treffen Botschaften ein, bei denen dem jeweiligen Adressaten nicht klar ist, wie er darauf reagieren soll. In der Modellwelt ist dies der Fall, wenn zu einer eintreffenden Nachricht keine gleichnamige Methode existiert.

In der Realität ist mitunter nicht so genau festzustellen, ob eine Botschaft verstanden wurde oder nicht. In der Modellwelt wird die Botschaft nicht verstanden, wenn zu einer eintreffenden Nachricht keine gleichnamige Methode existiert. In dem Falle reagiert ein Objekt nicht.

Ein Objekt reagiert nur auf eine Botschaft, wenn eine Methode gleichen Namens existiert.

Von den Programmiersprachen wird meistens gefordert, dass man nur Nachrichten an Objekte verschickt, auf die der Adressat auch reagieren kann. Es wird bei der Analyse von Programmen bereits überprüft, ob die Versendung von Nachrichten sinnvoll ist. Ein Compiler signalisiert meist schon einen Fehler. Das gilt für Sprachen wie Pascal, Java, C++ oder C#. Bei der Nutzung eines Interpreters kommt es erst zur Laufzeit der Programme zu Fehlermeldungen. Das kann man beispielsweise bei der Nutzung der Entwicklungsumgebung der Sprache Python beobachten. Es wird also nicht einfach ignoriert, dass die Nachricht nicht verstanden wird.

Programmiersprachen fordern, dass Objekte zu jeder eintreffenden Botschaft eine entsprechende Methode besitzen.

Es gibt damit einen Unterschied zwischen der Realität und den in Programmiersprachen spezifizierten Modellwelten.

Nicht alle Studenten haben im Detail das gleiche Verhalten. Man kann sie beispielsweise nach ihren Studiendisziplinen klassifizieren, die unterschiedliche Fähigkeiten fordern.

Die allgemeinen Eigenschaften und Fähigkeiten eines Studenten haben alle Studierenden, sie besitzen aber mitunter noch weitere Fähigkeiten wie Singen oder Tanzen. Was diese Fähigkeiten im Einzelnen sind, soll hier zunächst erst einmal nicht interessieren. Bild 1.6 stellt grafisch dar, dass es verschiedene spezielle Klassen gibt. Sie haben die Eigenschaften

einer allgemeinen Klasse Student. Man spricht auch davon, dass sie diese Charakteristika erben (Genaueres wird dazu später noch unter Definition 1.9 erläutert).

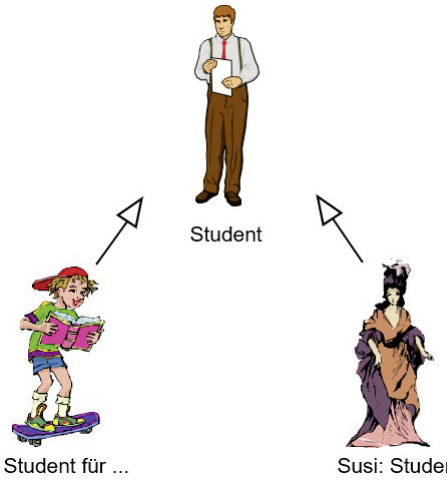


Bild 1.6
Weiterleitung (Vererbung) von Eigenschaften und Fähigkeiten

Die beiden etwas exotischer anmutenden Klassen von `Student für ...` haben die gleichen Attribute und Methoden wie die Klasse `Student`. Alles, was ein Student kann, das können sie auch. Sie haben sogar noch zusätzliche Eigenschaften und/oder Fertigkeiten. Alle sind also in der Lage, auf die Botschaft `feiern` zu reagieren. Das bedeutet nun aber nicht unbedingt die Ausführung des gleichen Algorithmus. Jede Gruppe (Klasse) von Studenten hat ihre eigene Methodenimplementierung, ihren eigenen Algorithmus, ihre eigene Idee zu feiern.

Die Auslösung verschiedener Algorithmen durch die gleiche Botschaft wird auch als Polymorphismus (Definition 1.12) bezeichnet. Bild 1.7 versucht eine Visualisierung dieses Sachverhaltes.

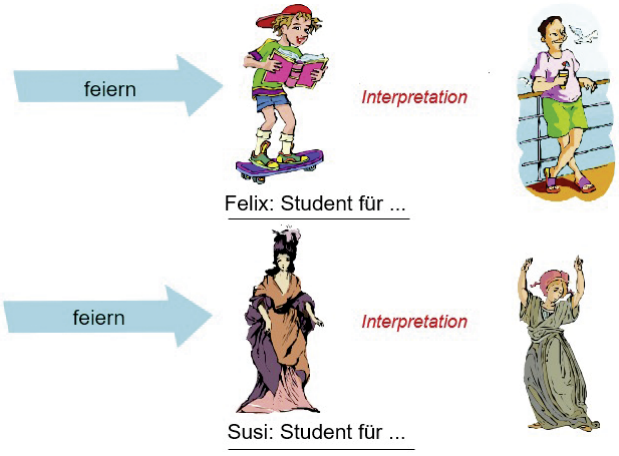


Bild 1.7 Unterschiedliche Reaktion auf Botschaften – Polymorphismus

1.2 Konzepte und Notationen

Dieser Abschnitt beschäftigt sich mit einigen grundlegenden Konzepten und deren Notation. Späteren Kapiteln ist die Erläuterung weiterer Details vorbehalten. Hier soll zunächst ein Grundverständnis von Basismodell, statischem Modell, dynamischem Modell und dem Modell der Systemnutzung gegeben werden.

1.2.1 Basismodell

Zum Verständnis des Basismodells der objektorientierten Softwareentwicklung gehört die richtige Vorstellung von den Begriffen Objekt, Attribut, Methode und Klasse. Im vorigen Abschnitt wurden diese Ausdrücke bereits intuitiv benutzt. Hier sollen sie teils unter Verwendung von Zitaten etwas ausführlicher diskutiert werden. In der Literatur gibt es zahlreiche Nuancen der Definition des Begriffs Objekt. Wir wollen auf die Definition von Balzert [1.1] zurückgreifen.

Definition 1.1: Objekt

Ein Objekt ist allgemein ein Gegenstand des Interesses, insbesondere einer Beobachtung, Untersuchung oder Messung. Objekte können Dinge und Begriffe sein.

In der objektorientierten Softwareentwicklung besitzt ein Objekt bestimmte Eigenschaften und reagiert mit einem definierten Verhalten auf seine Umgebung. Außerdem besitzt jedes Objekt eine Identität, die es von allen anderen Objekten unterscheidet.

Die Eigenschaften eines Objektes werden durch dessen Attributwerte ausgedrückt, sein Verhalten durch eine Menge von Methoden.

Das Verhalten eines Objektes kann mit anderen Worten auch als die Menge seiner Fähigkeiten bezeichnet werden. Es ist in der Lage, bestimmte Aktivitäten auszuführen. Für das Eingangsbeispiel (Bild 1.1) ist der Student *Felix* ein Objekt, dessen Verhalten durch die Tätigkeiten des *Lernens* und des *Feierns* bestimmt ist. Außerdem besitzt es die Attribute *Geburtsdatum* und *Einkommen*, die seine Eigenschaften charakterisieren und deren Attributwerte „17. 11. 2007“ und „Stipendium“ sind.

Ein Objekt gehört zu einer Klasse gleichartiger Objekte, die gleiche Attribute und Methoden besitzen.

Definition 1.2: Klasse

Eine Klasse beschreibt eine Sammlung von Objekten mit gleichen Eigenschaften (Attributen), gemeinsamer Funktionalität (Methoden), gemeinsamen Beziehungen zu anderen Objekten und gemeinsamer Semantik.

Der Klassenname ist nach Coad [1.2] stets ein Substantiv im Singular, das durch ein Adjektiv ergänzt werden kann.

Die Eigenschaften eines Objektes, die durch Werte charakterisierbar sind, werden in Klassen in Form von Attributen modelliert.

Definition 1.3: Attribut

Die Attribute beschreiben die Daten bzw. Eigenschaften einer Klasse. Alle Objekte einer Klasse besitzen dieselben Attribute, jedoch unterschiedliche Attributwerte. Das bedeutet für die Implementation, dass jedes Objekt Speicherplatz für alle seine Attribute erhalten muss.

Der Attributname muss im Kontext eines Objektes eindeutig sein. Im Allgemeinen wird ein Substantiv dafür verwendet.

Eine Methode ist eine Tätigkeit, die ein Objekt ausführen kann. Sie wird durch ein Verb oder durch ein Substantiv mit Verb bezeichnet und muss im Kontext eines Objektes eindeutig sein. Das Verhalten eines Objektes wird durch eine Menge von Methoden charakterisiert.

Beispielsweise hat der Student *Felix* die Fähigkeiten (Methoden) zu *feiern* und zu *lernen*. Die Definition der Algorithmen, die mit den Methoden verbunden sind, erfolgt durch die jeweiligen Klassen. Im Falle von *Felix* ist das die Klasse *Student*.

Definition 1.4: Methode

Eine Methode ist ein Algorithmus, der einem Objekt zugeordnet ist und von diesem abgearbeitet werden kann.

Objekte kommunizieren untereinander über Nachrichten, die Botschaften genannt werden. Wird an ein Objekt eine Botschaft geschickt, so führt dies zum Aufruf der entsprechenden Methode. Im Eingangsbeispiel hat die Lehrkraft *John* dem Studenten *Felix* die Botschaft *lernen* gesandt.

Definition 1.5: Botschaft

Eine Botschaft ist eine Nachricht, die den Aufruf einer Methode gleichen Namens zur Folge hat.

Auf die Botschaft (bitte) *lernen* führt ein freundlicher Felix in der Realität die entsprechenden Aktivitäten durch. In der Modellwelt würde ein Objekt *Felix* mit der Ausführung der gleichnamigen Methode reagieren.

Nach absolviertem Lernen sendet Felix eine Vollzugsmeldung an seinen Lehrer.

In [1.1] wird zur Verdeutlichung der Beziehung zwischen Klasse und Objekten die Metapher des Stempels herangezogen. Die Metapher stimmt nicht vollständig, gibt aber einen ganz anschaulichen Eindruck. Die Besonderheit dieses Stempels ist, dass nicht nur eine

vollständige Kopie der Attribute und Methoden erzeugt wird, sondern dass die Attribute mit Attributwerten versehen werden können.

Der Stempel entspricht einer Klasse, die unterschiedliche Exemplare von Objekten erzeugen kann. Die Stempelabdrücke symbolisieren die Objekte. Das Problem bei der Metapher besteht darin, dass die Abdrücke teilweise andere Informationen besitzen als der Stempel. Ein Attributwert für das Geburtsdatum ist in der Klasse Student noch nicht enthalten. Dort steht nur der Typ dieses Attributes. Die Objekte *Felix*, *Antje* und *Heike* haben jeweils aber einen solchen Attributwert.

Trotz dieses kleinen Fehlers ist in Bild 1.8 die Beziehung zwischen Klasse und Objekt gut sichtbar und verdeutlicht, dass Objekte ihre Eigenschaften nicht direkt durch Vererbung erhalten. Sie profitieren nur indirekt davon. Um sich diesen Sachverhalt ständig vor Augen zu führen, ist die Stempelmetapher sehr gut geeignet.

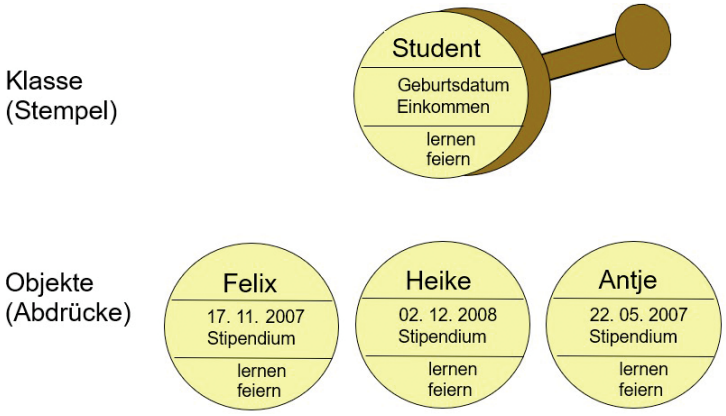


Bild 1.8 Stempelmetapher für Klassen

Eine Klasse dient in der objektorientierten Welt zur Erzeugung von Objekten. Sie prägt sie im gewissen Sinne. In der Umgangssprache wird der Klassenbegriff mehr zur Gruppierung von Objekten genutzt.

Die erzeugten Objekte nennt man auch Instanzen. Der Prozess ihrer Generierung wird dementsprechend als Instanziierung bezeichnet. Damit sind zunächst die wichtigsten Begriffe des Basismodells des objektorientierten Paradigmas dargestellt worden.

Die folgenden Abschnitte befassen sich mit weiteren grundsätzlichen Konzepten, die in Form spezieller Modelle ausgedrückt werden. Dabei handelt es sich um das statische Modell, das dynamische Modell und das Modell der Systemnutzung.

Das statische Modell beschreibt Beziehungen zwischen Modellelementen, die strukturelle Zusammenhänge aufweisen. Das Verhalten der einzelnen Modellelemente wird durch ein dynamisches Modell beschrieben. Der Aspekt der Systemnutzung ist im Fokus des letzten Modells, das auch den entsprechenden Namen trägt.

Bild 1.9 gibt eine grafische Veranschaulichung, wie diese Modelle zusammenhängen. Es zeigt auch, wie sich geschichtlich die Sicht auf objektorientierte Softwarespezifikationen erweitert hat.

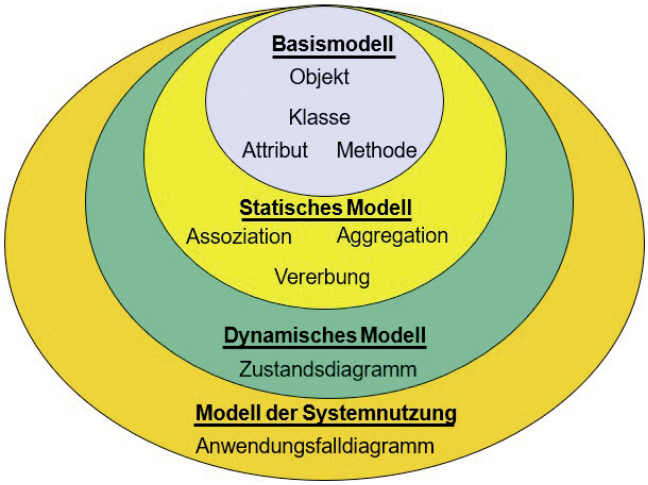


Bild 1.9 Entwicklung des objektorientierten Modellverständnisses

Das Basismodell entspricht der Idee der Datenkapselung und der Idee der Programmierung mithilfe von abstrakten Datentypen. Dieser Gedanke wurde von der Implementation bis in die frühen Phasen der Softwareentwicklung verbreitet. Dazu kam die Idee der Nutzung von Assoziation und Vererbung, die aus dem Bereich der Datenmodellierung in die allgemeine Softwareentwicklung Einzug hielten. Danach wurde der Aspekt der Spezifikation der Dynamik von Objekten genauer untersucht.

Dabei erhielt das Konzept der endlichen Automaten ein neues Anwendungsfeld. Ursprünglich zur Spezifikation mathematischer Systeme eingeführt, wurde es bereits bei der Modellierung von Dialogabläufen und bei der Spezifikation von Reihenfolgen bei der „Strukturierten Analyse“ genutzt. Das Konzept hielt speziell durch die Arbeiten von Harel [1.3] verstärkt Einzug in die Modellierung allgemeiner dynamischer Systeme. Später wurde es zur Spezifikation der Dynamik von Klassen herangezogen.

Jacobson hat es durch seine Arbeiten verstanden, den Aspekt der Systemnutzung mehr in den Vordergrund zu stellen und als Ausgangsbasis für die Systemmodellierung zu nutzen. Seine Vorgehensweise integriert einige Ideen aus der aufgabenorientierten Softwareentwicklung, die auch als strukturierter Ansatz bezeichnet werden, mit der objektorientierten Modellierung.

Nicht nur die Objekte sind für die Softwareentwicklung von Interesse, sondern die Aufgaben, die mithilfe eines Systems zu erfüllen sind, stehen im Mittelpunkt. Die von Jacobson entwickelte, auf Anwendungsfälle aufbauende Methodik erfreut sich großer Beliebtheit. Angelehnt an die Struktur der Modelle in Bild 1.9 werden auch die folgenden einführenden Abschnitte zu Methoden und Konzepten gegliedert. Nach dem Basismodell folgen nun noch das statische Modell, das dynamische Modell und das Modell der Systemnutzung.

Aufgaben

1. Erläutern Sie, warum die Stempelmetapher nicht vollständig das Zusammenspiel zwischen Klasse und Objekt widerspiegelt.
2. Gibt es Objekte, die zu keiner Klasse gehören?
3. Welcher Zusammenhang besteht zwischen einer Botschaft und einer Methode?
4. Was versteht man unter der Instanz einer Klasse?

1.2.2 Statisches Modell

Assoziationen beschreiben Beziehungen zwischen Objekten. Sie werden zwischen Klassen formuliert, beziehen sich aber auf die Instanzen dieser Klassen.

Die Beziehung zwischen den Objekten wird als Verknüpfung bezeichnet. Sie ist die Instanz einer Assoziation zwischen den Klassen der Objekte.

So gibt es eine Assoziation *besucht*, die beschreibt, dass Studenten an Lehrveranstaltungen teilnehmen. Nun hört aber nicht allgemein ein Student eine allgemeine Vorlesung, sondern *Felix*, *Paul*, *Susi* und *Peter* hören sie als Instanzen (Objekte hört sich etwas unschön an) der Klasse *Student*. Sie besuchen eine konkrete Lehrveranstaltung, vielleicht die Vorlesung Softwaretechnik am Montag im Hauptgebäude der Universität. Diese Vorlesung ist ein Objekt der Klasse *Lehrveranstaltung*.

Diese konkrete Beziehung zwischen den Objekten kann abstrakter als Assoziation zwischen den Klassen *Student* und *Lehrveranstaltung* (Bild 1.10) modelliert werden.

Definition 1.6: Assoziation

Eine Assoziation beschreibt eine Sammlung von Verknüpfungen mit einer gemeinsamen Struktur und Semantik.

Assoziationen beschreiben mithilfe von Klassen mögliche Verknüpfungen der den Klassen zugeordneten Objekte. Sie können wie folgt grafisch modelliert werden.

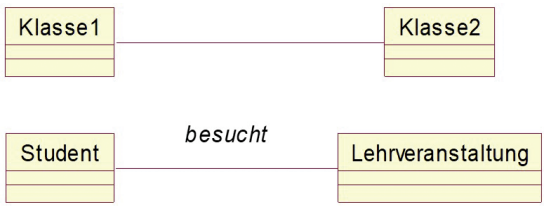


Bild 1.10
Grafische Darstellung einer Assoziation

Welche Funktion ein Objekt in einer Assoziation erfüllt, ist durch eine Rolle beschreibbar. Der Rollenname wird dabei an den Enden einer Assoziationsverbindung notiert. Bei einer binären Assoziation sind es damit immer zwei Angaben. Daneben sind Informationen zu

Kardinalitäten (bei UML wird das mit Multiplizität bezeichnet) möglich. Sie geben an, wie viele Objekte der Klassen in der späteren Verknüpfung einbezogen sind.

Die verschiedenen Arten der Multiplizitätsangaben (bei der Datenmodellierung spricht man von Kardinalitätsangaben) werden in Kapitel 2 noch genauer betrachtet. Zunächst seien nur die Multiplizitäten erwähnt, die ausdrücken, dass genau ein Objekt betroffen ist (1), dass kein Objekt oder viele Objekte (0..n) bzw. dass ein Objekt oder viele Objekte (1..*) auftreten können. Die Notationen „*“ und „n“ haben hier die gleiche Bedeutung. Die Rolle, die Objekte einer Klasse spielen, und die Multiplizität, in der sie in Beziehung zu den Objekten der anderen Klassen stehen, werden an dem Ende der Assoziation notiert, an dem die Klasse selbst steht.

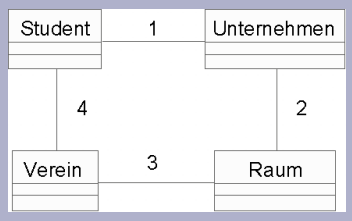
Es sei darauf hingewiesen, dass es auch Notationen gibt, bei denen andere Festlegungen getroffen werden. Beim Lesen von Spezifikationen ist es daher wichtig, darauf zu achten, welche Festlegungen gelten.

Bild 1.11
Beispiel einer Assoziation mit Rollen und Kardinalitäten

In der Assoziation *liest* in Bild 1.11 spielt ein Student die Rolle des *Lesers* und ein Buch die Rolle des *Lehrbuches*. Außerdem wird ausgedrückt, dass ein Student ein Lehrbuch oder viele Lehrbücher liest. Umgekehrt wird ein Buch von keinem oder mehreren Studenten gelesen.

Aufgaben

Folgendes Diagramm sei gegeben



5. Wie könnten die Bezeichnungen für die Assoziationen lauten?
   a) 1="arbeitet für", 2="besitzt", 3="nutzt", 4="ist Mitglied"
   b) 1="befreundet mit", 2="nutzt", 3="besitzt", 4="befreundet mit"
   c) 1="arbeitet für", 2="nutzt", 3="besitzt", 4="arbeitet für"
   d) 1="arbeitet für", 2="nutzt", 3="besitzt", 4="schraubt"

6. Welche Kardinalitäten würden Sie der Assoziation 1 zuordnen?
   a) auf beiden Seiten 1
   b) links 1 rechts 0..*
   c) auf beiden Seiten 0..*
   d) links 0..* und rechts 1
7. Modellieren Sie auch die Kardinalitäten für die anderen Assoziationen.
8. Modellieren Sie Ihre Beziehungen zu Wohnung, Auto, Bus, Taxi und Restaurant in Form von Assoziationen. Nutzen Sie Kardinalitäten und Rollen, wenn das die Lesbarkeit erleichtert.

Definition 1.7: Aggregation

Eine Aggregation bezeichnet eine „Teil-Ganzes"- oder „Ist Teil von"-Verknüpfung.

Eine spezielle Form der Assoziation ist die Aggregation. Sie wird zur Modellierung genutzt, wenn die Beziehung zwischen den Objekten unterschiedlicher Klassen sehr eng ist und mit einer „Ist Teil von"-Beziehung charakterisiert werden kann. Diese Beziehung schließt die Mitgliedschaft in einer Gruppe ein. Ein Mitglied ist in diesem Sinne also ein Teil eines Vereins. Um diese noch etwas lockere „Ist Teil von"-Beziehung von existenziellen Beziehungen zu unterscheiden, in denen das Teil ohne das Ganze gar nicht existieren kann, werden unterschiedliche Notationen für beide Varianten genutzt. Die etwas losere Beziehung wird durch ein nicht gefülltes Drachenviereck charakterisiert, während die enge Beziehung durch ein ausgefülltes Drachenviereck wie in Bild 1.12 repräsentiert wird. Sie wird auch als Komposition bezeichnet.

Definition 1.8: Komposition

Eine Aggregation mit sehr starker Bindung wird als Komposition bezeichnet.

Eine Komposition ist eine stärkere Beziehung als eine Aggregation, und diese ist wiederum stärker als eine Assoziation.

Als Beispiel sei erneut ein Buch herangezogen. Seine Kapitel sind ohne das gesamte Buch im Allgemeinen nicht existent, deshalb ist eine sehr enge Kopplung in Form einer Komposition gerechtfertigt.

Die Verbindung zwischen einem Verein und seinen Mitgliedern ist noch so eng, dass der Verein ohne Mitglieder in der Existenz gefährdet ist und etwas mehr als eine einfache Assoziation vorliegt. Eine Aggregation ist damit eine passende Modellierung.

Werden die Verbindungen zwischen den Objekten noch loser wie bei Vortragender und Zuhörer, dann sollte man eine Assoziation modellieren. Hier handelt es sich wirklich um eine Beziehung, die zwar wichtig sein kann, beide Seiten aber nicht in der Existenz bedroht.

Es ist nicht immer ganz einfach, die richtige Art der Beziehung festzulegen. Im Zweifelsfalle sollte man eine Assoziation modellieren. Für die weitere Softwareentwicklung hat das nicht so einen entscheidenden Einfluss. Unterschiede bestehen zwar bei den entsprechen-

den Erzeugungsmethoden. Die exakte Modellierung dient hauptsächlich zum besseren Verständnis der Zusammenhänge.

Welche Auswirkung die Modellierung auf den Quelltext haben kann, das wird in Kapitel 2 am Beispiel eines Würfelspiels erläutert.

In Bild 1.12 sind die gerade erwähnte Aggregation und ihr Spezialfall, die Komposition, grafisch dargestellt.

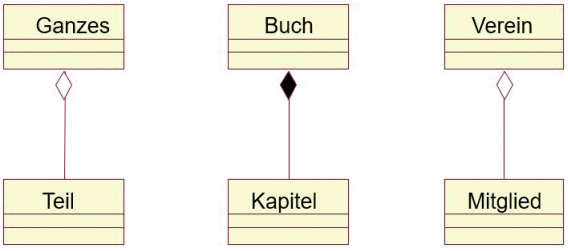


Bild 1.12
Darstellung der Aggregation und der Komposition

Zwischen *Buch* und *Kapitel* sowie zwischen *Verein* und *Mitglied* liegt eine Aggregation vor. Bei *Buch* und *Kapitel* ist es sogar eine Komposition.

Eine Aggregation ist transitiv (Wenn A Teil von B und B Teil von C, dann ist auch A Teil von C) und antisymmetrisch (Wenn A Teil von B, dann ist nicht B Teil von A). Sie liegt nach Rumbaugh [1.7] vor, wenn die folgenden Fragen mit „Ja“ beantwortet werden können:

a) Ist die Beschreibung „Teil von“ zutreffend?

b) Werden manche Operationen auf das „Ganze“ automatisch auch auf die „Teile“ angewandt?

c) Pflanzen sich manche Attribute vom „Ganzen“ auf alle oder einige „Teile“ fort?

d) Ist die Verbindung durch eine Asymmetrie gekennzeichnet, bei der die „Teile“ dem „Ganzen“ untergeordnet sind?

Aufgaben

9. Gegeben seien die Klassen `Haus` und `Zimmer`. Welche Art der Assoziation würden Sie zwischen beiden modellieren?
   a) Eine einfache Assoziation „hat Beziehung zu“
   b) Eine Aggregation „besteht aus“
   c) Eine Komposition „besteht aus“
10. Gegeben seien die Klassen `Haus` und `Mieter`. Welche Art der Assoziation würden Sie hier modellieren?
   a) Eine einfache Assoziation „hat“
   b) Eine Aggregation „setzt sich zusammen aus“
   c) Eine Komposition „besteht aus“
11. Gegeben seien die Klassen `Mieterverein` und `Mieter`. Wie sieht hier die Modellierung aus?
   a) Eine einfache Assoziation „besitzt“
   b) Eine Aggregation „setzt sich zusammen aus“
   c) Eine Komposition „besteht aus“

Eine ganz wichtige Beziehung zwischen Klassen bei der objektorientierten Softwarespezifikation ist die Vererbungsbeziehung. Dabei werden Attribute und Methoden einer Oberklasse an eine Unterklasse weitergeleitet.

Man spricht von einer Spezialisierung der Oberklasse in eine Unterklasse oder von einer Generalisierung der Unterklasse in eine Oberklasse. Entsprechend kann man die Vererbungsrelation auch als „Spezialisieren“ oder „Generalisieren“ betrachten.

Definition 1.9: Vererbung

Klassen werden in eine Hierarchie eingeordnet, die eine Weiterleitung von Informationen von oben nach unten ermöglicht. Eine Unterklasse verfügt dann über die Eigenschaften und das Verhalten der Oberklasse. Sie „erbt“ diese Charakteristika. Entsprechend wird die Hierarchie auch Vererbungsstruktur oder Klassenhierarchie genannt.

In einer Vererbungshierarchie gilt das Substitutionsprinzip. Objekte der Unterklasse können in einem Programm stets Objekte der Oberklasse vertreten.

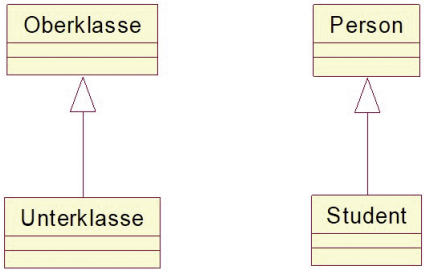


Bild 1.13
Grafische Darstellung der Vererbung

Als Beispiel erbt die Klasse `Student` alle Charakteristika von der Klasse `Person`. Damit sind alle Attribute und Methoden der Klasse `Person` auch Attribute und Methoden der Klasse `Student`. Ein Student hat alle Charakteristika einer Person, aber zusätzliche Eigenschaften und/oder Methoden.

Im Gegensatz zur Assoziation und Aggregation, die zwar zwischen Klassen modelliert werden, aber eine Verknüpfung zwischen Objekten beschreiben, ist die Vererbung nur eine Relation zwischen Klassen.

Vererbung findet nur zwischen Klassen statt!

Aufgaben

12. Kann ein Objekt Attribute und Methoden erben?
13. Welche Dinge erbt die Klasse `Informatikstudent`, wenn sie als Unterklasse von `Student` modelliert wird?
    a) Nur die Attribute
    b) Nur die Methoden
    c) Alle Attribute und Methoden
    d) Nur die Methode `feiern`
    e) Nur die Methode `lernen`

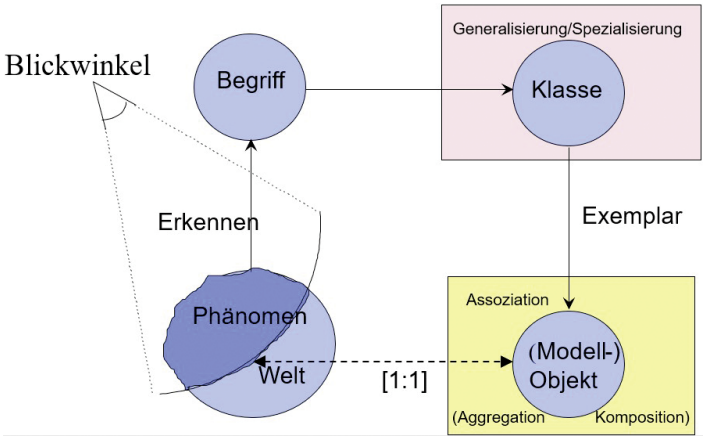


Bild 1.14 Darstellung des Anwendungsfeldes von Vererbung und Assoziation

Definition 1.10: Abstrakte Klasse

Eine Klasse, zu der keine Objekte existieren können, bezeichnet man als abstrakte Klasse.

Definition 1.11: Konkrete Klasse

Wenn zu einer Klasse Objekte erzeugt werden können, bezeichnet man sie als konkrete Klasse.

Abstrakte Klassen werden aus konzeptionellen Gründen benötigt, um Informationen gut zu strukturieren. Sie werden nie als Schablonen oder Stempel zur Erzeugung von Objekten benutzt.

Aus dem täglichen Leben sind Gattungsbegriffe für derartige Gruppierungen bekannt. Säugetiere sind unterteilt in Pflanzenfresser und Fleischfresser. Ein Fleischfresser ist beispielsweise ein Tiger. Im objektorientierten Sinne entsprechen Säugetier, Pflanzenfresser und