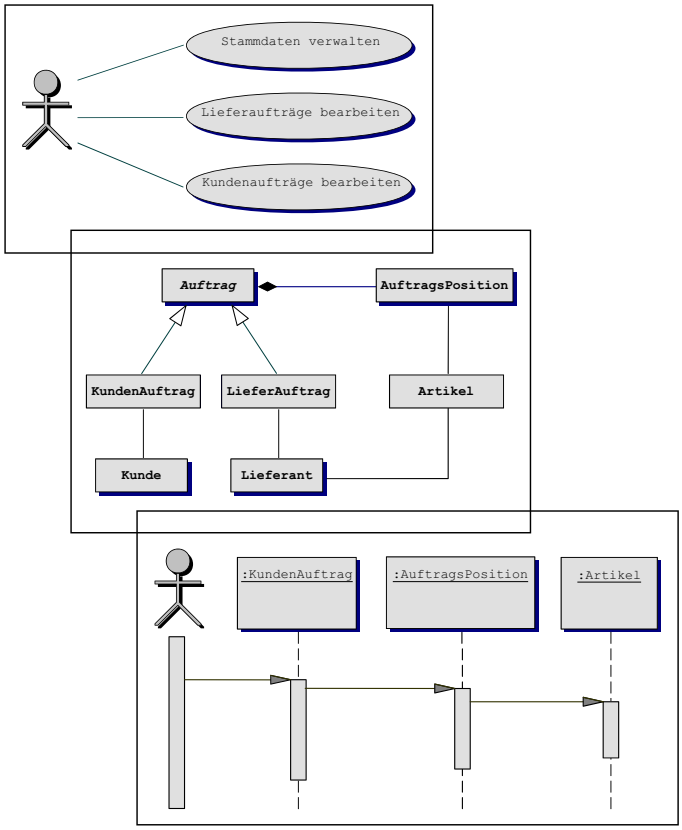
FernUniversität in Hagen



Objektorientierte Systemanalyse

Kurseinheit 1:

Grundlagen des Software Engineering

Objektorientierte Softwareentwi

LESEPROBE
zum Kurs 818
Objektorientierte Systemanalyse

Autor:
Dr. Andreas Bortfeldt

I. Inhaltverzeichnis

Kurseinheit 1: Objektorientierte Systemanalyse

Kurseinheit 2: Objektorientierte Systemanalyse

II. Leseprobe

1. Auszug aus Kurseinheit 1, Kapitel 2

2. Objektorientierte Softwareentwicklung im Überblick

2.1 Grundlegende Konzepte der Objektorientierung

grundlegende objektorientierte Konzepte und ...

Bei einer objektorientierten Softwareentwicklung erfolgen Analyse und Entwurf auf der Basis einheitlicher Konzepte. Die Implementierung wird mittels objektorientierter Programmiersprachen durchgeführt, die wiederum die objektorientierten Konzepte weitgehend direkt unterstützen.

Der objektorientierte Entwicklungsansatz bietet eine reichhaltige Auswahl von Konzepten an. Ihrer detaillierten Einführung wird eine Betrachtung des objektorientierten Ansatzes aus der Vogelperspektive vorangestellt. Die in Abb. 2.1 gezeigten grundlegenden Konzepte und ihre Beziehungen werden anschließend schrittweise erläutert.

... ihre Beziehungen

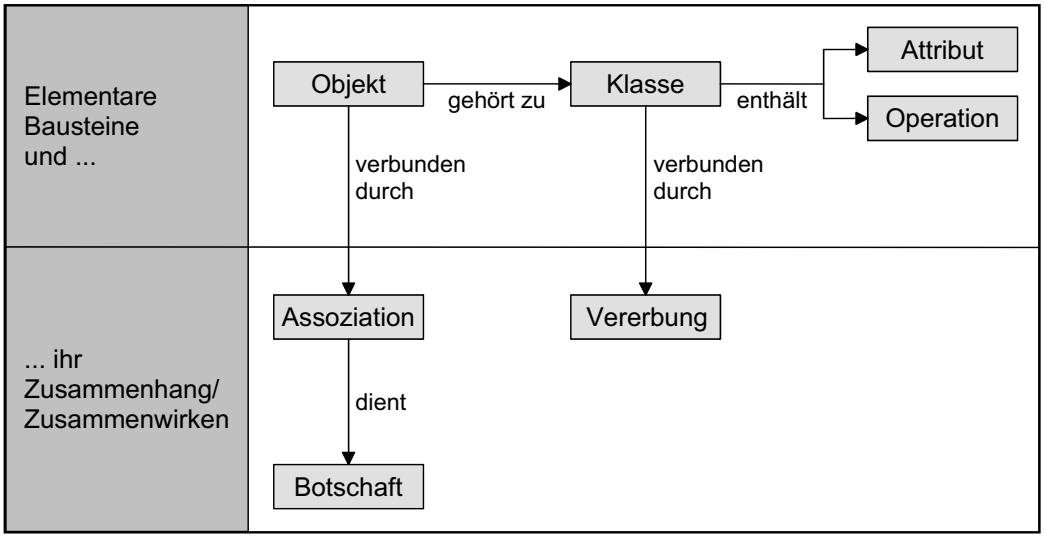


Abb. 2.1. Grundlegende objektorientierte Konzepte und ihre Beziehungen.

(1) Objekte und Klassen, Attribute und Operationen

Objekte, Klassen

Im Zentrum des objektorientierten Ansatzes stehen Objekte und ihre Klassen. Sie bilden die elementaren Bausteine objektorientierter Modelle und Programme und werden jeweils durch Abstraktion aus den realen Objekten des interessierenden Anwendungsbereiches gewonnen.

Attribute, Operationen

Ein Objekt besitzt einerseits gewisse Daten, andererseits kann es bestimmte Operationen ausführen. Jedes Objekt gehört zu einer Klasse. Eine Klasse legt einen bestimmten Typ von Objekten fest. Sie enthält Attribute, die Bedeutung und Wertebereich der Daten der zugehörigen Objekte angeben. In der Klasse werden ferner die von den Objekten ausführbaren Operationen als Algorithmen definiert.

Klasse und zugehörige Objekte

Zu einer Klasse können beliebig viele Objekte existieren. Alle Objekte einer Klasse stimmen in ihren Attributen und den ausführbaren Operationen überein. Sie unterscheiden sich jedoch im allgemeinen hinsichtlich der Attributwerte. Dies wird in Abb. 2.2 verdeutlicht. Beide Objekte der Klasse Artikel besitzen die Attribute Bezeichnung und Preis sowie einheitliche Operationen, während die Attributwerte differieren.

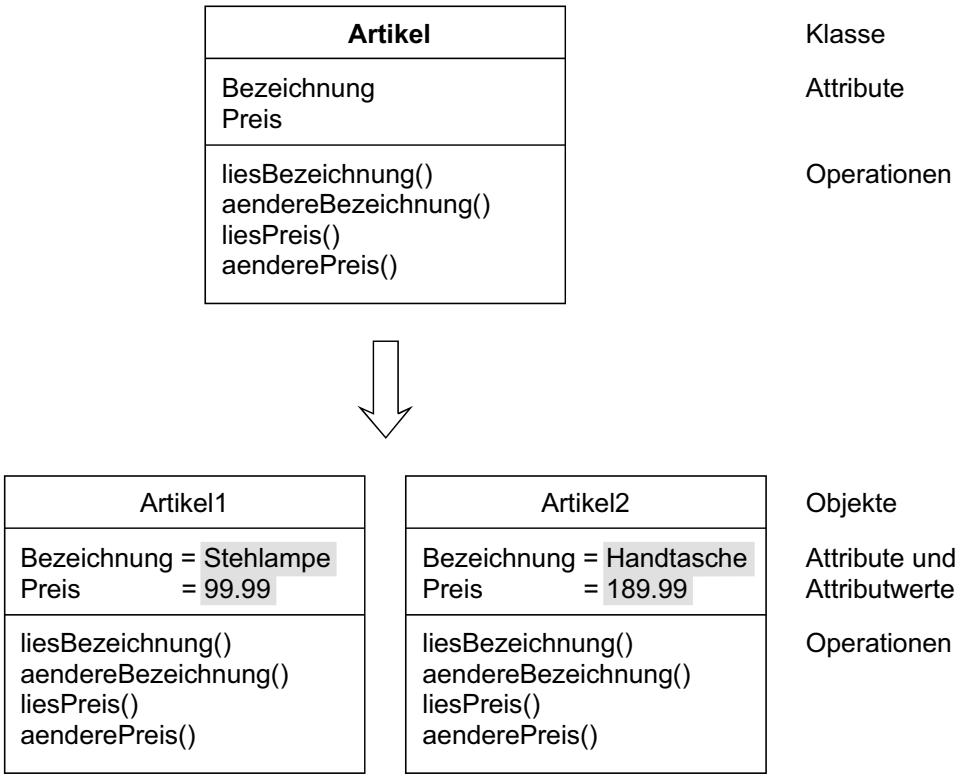


Abb. 2.2. Eine Klasse und zugehörige Objekte.

Kapselung

Charakteristisch für den objektorientierten Ansatz ist, daß Daten und Funktionen zu ihrer Verarbeitung nicht getrennt definiert, sondern vielmehr in Objekten und Klassen zu einer Einheit verschmolzen werden. Dies wird als Kapselung von Daten und Funktionen bezeichnet.

Infolge der Kapselung kann eine Funktion nur als Operation einer Klasse definiert und zugleich als Operation eines Objekts – also nicht unabhängig von einem Objekt – ausgeführt werden. Dementsprechend bezieht sich eine ausgeführte Operation auf ein bestimmtes Objekt. Der Aufruf einer Operation wird unter Angabe des Objekts notiert. So bewirkt etwa der Operationsaufruf

Artikel1.aenderePreis(49.99),

daß der Preis des Objekts Artikel1 auf 49.99 Euro geändert wird.

Geheimnisprinzip

Datenkapselung

Über die Kapselung von Daten und Funktionen hinausgehend realisieren die Konzepte Klasse und Objekt das Geheimnisprinzip (vgl. Kap. 2.4.1). So sind die in einer Klasse hinterlegten Algorithmen der Operationen für andere Klassen nicht sichtbar. Diese wissen z.B. nichts über eine evtl. Zerlegung der Algorithmen in Teilalgorithmen. Ferner – und darauf kommt es vor allem an – sind die Daten eines Objekts für andere Objekte nicht sichtbar. Der Zugriff auf die Daten eines Objekts kann daher nur über seine Operationen erfolgen. Anstelle von Kapselung und Geheimnisprinzip wird auch zusammenfassend von Datenkapselung gesprochen. Die Abb. 2.3 veranschaulicht die Datenkapselung.

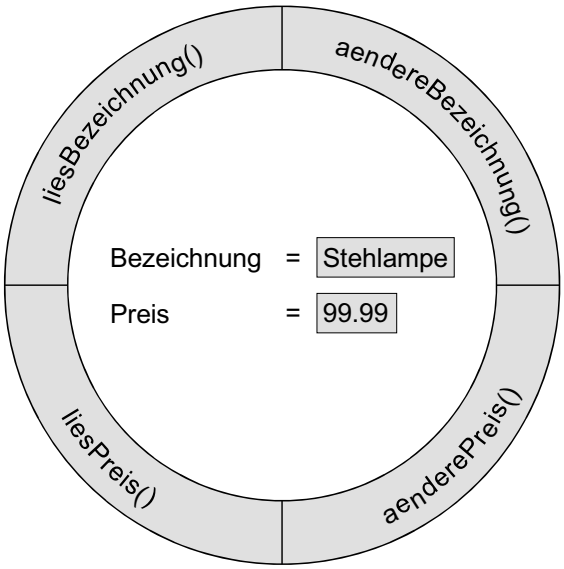


Abb. 2.3. Datenkapselung am Beispiel der Klasse Artikel.

Ergänzend sei bemerkt, daß in objektorientierten Programmiersprachen der äußere Zugriff auf Attributwerte von Objekten ggf. gesondert gesperrt werden muß. In der Systemanalyse wird jedoch üblicherweise – so auch hier – von der Einhaltung des Geheimnisprinzips bzw. der Datenkapselung ausgegangen.

Objektfabrik

Da zu einer Klasse beliebig viele Objekte angelegt werden können, werden Klassen auch als „Objektfabriken“ charakterisiert. Jedes Objekt „kennt“ seine Klasse und kann daher auf die in der Klasse definierten Operationen zugreifen.

(2) Botschaften und Assoziationen

In objektorientierten Modellen und Programmen wird die gesamte Funktionalität einer Anwendung einschließlich aller Anwendungsdaten auf verschiedene Klassen und ihre Objekte verteilt. Die Daten der Objekte sind gegen einen äußeren Zugriff geschützt. Die Objekte können zwar Operationen ausführen, tun dies aber nicht von selbst.

Botschaft

Die erforderliche Kooperation und Kommunikation von Objekten erfolgt über Botschaften. Mittels einer Botschaft übermittelt ein Sender-Objekt einem Empfänger-Objekt eine Aufforderung, eine bestimmte Operation auszuführen. Im Ergebnis werden Daten des Empfänger-Objekts modifiziert und/oder dem Sender-Objekt übergeben.

Beispiel zur Interaktion mittels Botschaften

Beispiel 2.1

Betrachtet sei neben der in Abb. 2.2 eingeführten Klasse Artikel eine Klasse Lieferant, die eine Operation druckeArtikelliste() besitzen soll. Die Operation gibt alle von einem Lieferanten gelieferten Artikel aus. Um die Daten der relevanten Artikel zu erhalten, werden an jedes korrespondierende Artikel-Objekt die Botschaften liesBezeichnung() und liesPreis() gesendet. Dies führt zum Aufruf der gleichnamigen Operationen. Diese senden die gewünschten Daten an das betreffenden Lieferanten-Objekt zurück. Dort können sie nun innerhalb der Operation druckeArtikelliste() gemeinsam ausgegeben werden.

Benutzung/ Ausführung einer Anwendung

Bei der hier vorwiegend interessierenden dialogorientierten Verarbeitung soll eine bestimmte Benutzung einer Anwendung zu einem vom Benutzer gewünschten, in sich abgeschlossenen Ergebnis – wie etwa einer Liste aller Artikel eines Lieferanten – führen. Wird die Anwendung objektorientiert erstellt, so stellt sich eine bestimmte Benutzung der Anwendung grob wie folgt dar:

- Der Benutzer initiiert über die Benutzungsoberfläche die gewünschte Verarbeitung, gibt benötigte Daten ein und stößt hierbei ein- oder mehrmals Operationen von Objekten an.
- Eine von der Benutzungsoberfläche angestoßene Operation führt im allgemeinen zu einer Kette weiterer Operationen. Alle Operationen werden über Botschaften an gewisse Objekte angestoßen. Ein- und Ausgabedaten sowie Zwischenergebnisse sind grundsätzlich Attributwerte verschiedener Objekte.
- Schließlich werden die Ausgabedaten über die Benutzungsoberfläche verfügbar gemacht.
- Operationen umfassen auch die Erzeugung und Vernichtung von Objekten. Dies trifft insbesondere auf die Initiierung der Verarbeitung zu.

Objektverbindungen

Um einem Empfänger-Objekt eine Botschaft zu übermitteln, muß dieses dem Sender-Objekt bekannt sein. Es muß also eine Verbindung zwischen beiden Objekten hergestellt werden.

Assoziationen:

...bestehen zwischen Objekten

... sind dauerhafte Verbindungen

Verbindungen zwischen Objekten werden überwiegend als Assoziationen modelliert und realisiert. Assoziationen bilden nach einer treffenden Formulierung von RUMBAUGH et al. (1993) den „Klebstoff“ eines objektorientierten Modells. Objektverbindungen können temporär für eine einmalige Botschaftsübermittlung hergestellt und anschließend „vergessen“ werden. Assoziationen stellen dagegen dauerhafte Verbindungen zwischen Objekten dar, die einmal eingerichtet jederzeit wieder genutzt werden können. Die Objekte einer jeden Klasse differieren nur hinsichtlich ihrer Attributwerte. Folglich repräsentieren Objektverbindungen im wesentlichen Beziehungen zwischen den Daten einer Anwendung.

In der Analyse werden Assoziationen oder auch temporäre Verbindungen als solche, d.h. außerhalb der Objekte modelliert. In Entwurf und Implementierung werden Objektverbindungen z.B. durch Zeiger innerhalb der Objekte realisiert. Das folgende Beispiel verdeutlicht das Konzept der Assoziation.

Beispiel zur Anwendung von Assoziationen

Beispiel 2.2

In dem zuvor betrachteten Beispiel 2.1 wird man zwischen den Objekten der Klassen Lieferant und Artikel eine Assoziation vorsehen. Dies bedeutet später für die ausführbare Anwendung, daß eine Verbindung von einem Lieferanten-Objekt zu einem Artikel-Objekt eingerichtet wird, sobald feststeht, daß der betreffende Artikel von dem entsprechenden Lieferanten geliefert wird. Dies kann etwa nach Eingaben an der Benutzungsoberfläche der Fall sein. Die ständigen Verbindungen können dann u.a. zu der im Beispiel 2.1 dargestellten Ausgabe der Artikelliste genutzt werden.

Betrachtet werden hier nur einseitige Verbindungen, wonach nur Lieferanten-Objekte ihre Artikel-Objekte kennen. Möglich ist auch der umgekehrte Fall sowie eine Kombination beider Varianten, d.h. bidirektionale Verbindungen.

Betont sei nochmals, daß Assoziationen als Objektverbindungen unmittelbar zwischen Objekten, nicht zwischen Klassen bestehen. Ebenso werden Botschaften im allgemeinen zwischen Objekten versandt.

(3) Vererbung

Vererbung besteht zwischen Klassen und ...

Wie Botschaften und Assoziationen leisten Vererbungsbeziehungen einen Beitrag zum Zusammenhalt und Zusammenwirken der elementaren Bausteine eines objektorientierten Modells oder Programms (vgl. Abb. 2.1). Doch bestehen Vererbungsbeziehungen direkt zwischen Klassen.

Eine Vererbungsbeziehung liegt oft zwischen einer übergeordneten und mehreren untergeordneten Klassen vor. Die übergeordnete Klasse faßt gemeinsame Komponenten (Attribute, Operationen) der untergeordneten Klassen zusammen. Jede untergeordnete Klasse erbt diese gemeinsamen Komponenten und ergänzt sie durch weitere Attribute und Operationen. Die gemeinsamen Komponenten müssen nur einmal, nämlich in der übergeordneten Klasse, definiert werden.

... unterstützt Generalisierung/ Spezialisierung

Das Vererbungskonzept unterstützt die Generalisierung (Verallgemeinerung) und Spezialisierung von Klassen. Es trägt wesentlich zu übersichtlichen und redundanzarmen Modellen und Programmen bei. Die Realisierung einer Anwendung wird insbesondere durch die Wiederverwendung bereits implementierter Klassen aus Klassenbibliotheken erleichtert, die dann durch benötigte untergeordnete Klassen ergänzt werden.

Für das objektorientierte Paradigma ist das Konzept der Vererbung konstitutiv. Methoden der Analyse oder des Entwurfs wie auch Programmiersprachen, die das Konzept der Vererbung nicht unterstützen, gelten nicht als objektorientiert.

Die Abb. 2.4 zeigt ein Beispiel einer Vererbungsbeziehung.

Beispiel einer Vererbung

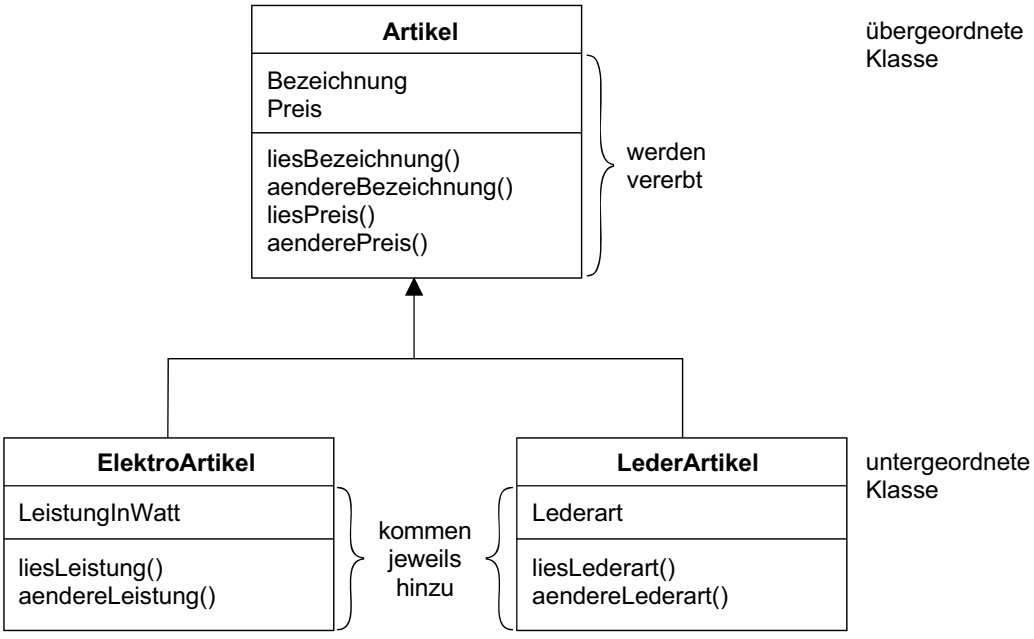


Abb. 2.4. Beispiel einer Vererbungsbeziehung.

Kern der Objektorientierung

Unter Beachtung der bisherigen Darlegung kann schließlich der Kern der Objektorientierung in folgender „Gleichung“ von COAD und YOURDON (1994) zusammengefaßt werden:

„Objektorientierte Methode	=	Klassen und Objekte + Vererbung + Kommunikation mit Nachrichten.“

Anstelle des Begriffs Nachricht wird hier ausschließlich das Synonym Botschaft benutzt.

2. Auszug aus Kurseinheit 2, Kapitel 4

4.1.2 Klasse

Begriff der Klasse

Jedes Objekt einer Anwendung gehört zu einer Klasse. Eine Klasse definiert einen gewissen Objekttyp, mit anderen Worten einen „Bauplan“ für gleichartige Objekte. Die Objekte einer Klasse werden auch als ihre Exemplare oder Instanzen bezeichnet. Eine Klasse wird unmittelbar durch folgende Komponenten festgelegt:

1. Attribute und
2. Operationen.

Eine Klasse besitzt ferner meist strukturelle Beziehungen zu anderen Klassen, nämlich Assoziationen und Vererbungsbeziehungen. Jedoch erscheinen diese Beziehungen in einem OOA-Modell als separate Modellelemente und nicht unmittelbar als Klassenkomponenten.

Eine Klasse definiert zum einen den Typ ihrer Objekte. Sie hat ferner im allgemeinen die Fähigkeit, Objekte der Klasse zu generieren. Klassen werden daher auch als Objektfabriken charakterisiert.

UML-Notation

Eine Klasse wird in der UML durch ihren Namen, ihre Attribute und ihre Operationen notiert. Im einzelnen gilt:

- Eine Klasse wird bei vollständiger Darstellung als ein Rechteck mit drei Feldern präsentiert. In dem oberen Namensfeld erscheint fettgedruckt, zentriert und mit einem Großbuchstaben beginnend der Klassenname. Im mittleren Attributefeld wird eine Liste der Attribute der Klasse aufgeführt. Das untere Operationenfeld enthält eine Liste der Operationen der Klasse.
- Als Klassenname ist stets ein Substantiv im Singular zu wählen. Es kann durch ein vorangestelltes Adjektiv ergänzt werden. Der Klassenname sollte innerhalb der Anwendung eindeutig sein (vgl. aber Kap. 4.1.7). Nur erwähnt sei, daß das Namensfeld durch ein Stereotyp (vgl. Kap. 3.2) und zusätzliche Merkmale zur Gruppierung bzw. Kommentierung der Klasse ergänzt werden kann.
- Attribute und Operationen können über ihre Benennung hinaus näher beschrieben werden. Hiermit beschäftigen sich die beiden folgenden Abschnitte. Die Attribute einer Klasse besitzen – anders als ihre Objekte und dem Typcharakter der Klasse gemäß – im allgemeinen keine Werte.
- Auch reduzierte Notationen einer Klasse sind möglich, bei der auf das Attributefeld und/oder das Operationenfeld verzichtet wird. Reduzierte Darstellungen kommen u.a. bei noch unvollständigen Informationen in Betracht.

Beispiel 4.3

Verschiedene Formen der Notation von Klassen zeigt anhand einer Klasse Konto die Abb. **4.**4.3:

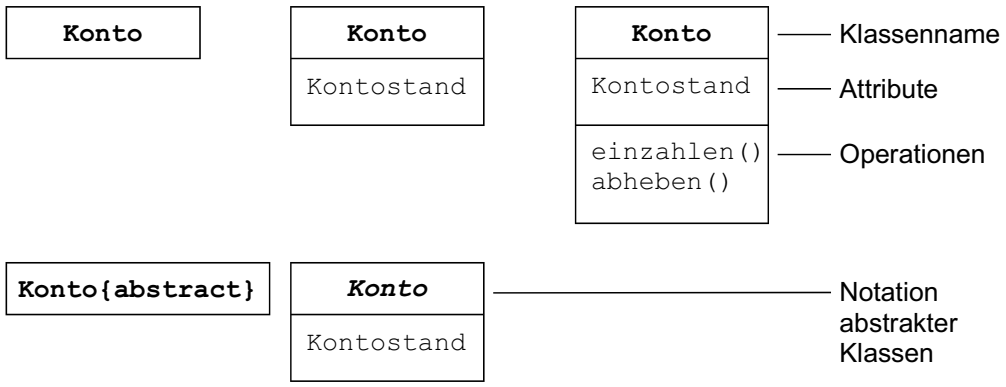


Abb. 4.3. Beispiele für die Notation von Klassen.

abstrakte Klassen

Neben Klassen, die Objekte erzeugen können, gibt es auch sogenannte abstrakte Klassen. Von einer abstrakten Klasse können keine Exemplare generiert werden. Eine abstrakte Klasse wird im Namensfeld durch das dem Klassennamen folgende Wort {abstract} (in geschweiften Klammern) oder durch Kursivschreibung des Klassennamens gekennzeichnet (vgl. Abb. 4.3). Abstrakte Klassen spielen bei Vererbungsbeziehungen eine Rolle und werden in diesem Zusammenhang näher betrachtet (vgl. Kap. 4.1.6). Ist eine Klasse nicht abstrakt, wird sie auch als konkrete Klasse bezeichnet.

Klassendiagramm, Klassenlexikon

In dem Klassendiagramm einer Anwendung werden ihre Klassen sowie die Assoziationen und Vererbungsbeziehungen zwischen den Klassen dargestellt. Da es alle statischen, d.h. zeitunabhängigen Aspekte der Anwendung modelliert, ist das Klassendiagramm der Anwendung das zentrale Dokument ihres statischen Modells. Zugleich dienen Klassendiagramme auch zur Visualisierung von Teilmengen der Klassen einer Anwendung sowie ihrer Beziehungen; für Beispiele sei auf folgende Abschnitte verwiesen. Details der Klassen einer Anwendung bzw. ihrer Komponenten werden in einem Klassenlexikon niedergelegt, worauf ebenfalls weiter unten eingegangen wird.

Klassen und Objekte

Die Beziehungen zwischen einer Klasse und ihren Objekten seien noch näher betrachtet. Die Attribute und Operationen eines Objekts werden innerhalb seiner Klasse definiert. Insbesondere werden die von den Objekten einer Klasse ausführbaren Operationen als Algorithmen bzw. als vollständige Funktionen in der Klasse abgelegt. Die innerhalb der Klasse als Algorithmen definierten Operationen werden in der UML als Methoden bezeichnet.

Methode

Jedes Objekt kennt seine Klasse. Dies bedeutet, daß es auf die in der Klasse abgelegten Informationen zu Attributen und Operationen zugreifen kann. Da alle Objekte einer Klasse hinsichtlich der ausführbaren Operationen übereinstimmen, müssen die zugehörigen Methoden nur einmal – nämlich innerhalb der Klasse – vorhanden sein. Soll ein Objekt eine gewisse Operation ausführen, so kommt die zugehörige, in der Klasse hinterlegte Methode zur Anwendung. Für jedes Objekt muß allerdings ein separater Satz von Attributwerten vorgesehen werden, da diese für verschiedene Exemplare der Klasse im allgemeinen nicht übereinstimmen. In

der UML werden Objekte daher ohne Operationen, aber mit Attributen und Attributwerten notiert.

Objektverwaltung

Kennt ein Objekt seine Klasse, so kennt deshalb eine Klasse ihre – zu einem gewissen Zeitpunkt existierenden – Objekte noch nicht. Doch wird auch dies im Rahmen der Analyse vorausgesetzt. Jede Klasse führt also eine Liste ihrer Objekte, die stets aktualisiert wird, wenn ein Objekt erzeugt oder gelöscht wird. Dies wird als Objektverwaltung bezeichnet. Die in der OOA vorausgesetzte implizite Verwaltung aller Objekte einer Klasse durch ihre Klasse vereinfacht die Modellierung von Operationen, die mehrere oder alle Objekte einer Klasse betreffen (vgl. Kap. 4.1.4). Einfach deshalb, weil die Objektverwaltung selbst nicht modelliert werden muß. Die Abb. 4.4 verbildlicht die Beziehungen zwischen einer Klasse und ihren Objekten.

Objekte sind temporärer Natur. Sie werden im Ablauf einer Anwendung erzeugt, ihre Attributwerte werden verändert, und sie werden spätestens dann zerstört, wenn die Ausführung der Anwendung endet. Welche Objekte bei einer konkreten Ausführung erzeugt werden und welche Werte diese annehmen, hängt von den jeweiligen Eingabedaten ab. Klassen sind dagegen permanente Einheiten, die von einer speziellen Ausführung der Anwendung nicht tangiert werden. Daher sind auch die Klassen einer Anwendung, nicht ihre Objekte, der eigentliche Gegenstand der Modellierung.

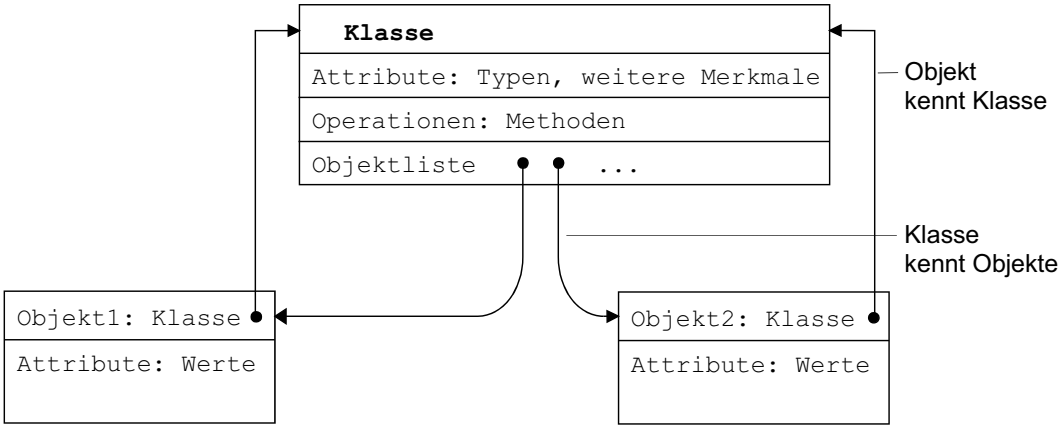


Abb. 4.4. Beziehungen zwischen einer Klasse und ihren Objekten (Objektverwaltung).

4.1.3 Attribut

Begriff des Attributs

Attribute legen die Daten der Objekte einer Klasse fest. Jedes Attribut besitzt einen Namen, einen Typ und optional weitere Merkmale, die im folgenden vorgestellt werden.

Als Attributname wird meist ein Substantiv, manchmal auch ein Adjektiv gewählt. Die UML sieht kleine Anfangsbuchstaben für Attributnamen vor. Hier wird jedoch für Substantive ein großer Anfangsbuchstabe gewählt. Attributnamen müssen innerhalb einer Klasse eindeutig sein. Außerhalb einer Klasse kann ein Attributname durch den vorangestellten und mit einem Punkt abgetrennten Klassennamen qualifiziert, d.h. erweitert werden, um die Eindeutigkeit zu wahren, z.B.: Artikel.Bezeichnung.

Attributtypen

Der Attributtyp beschreibt die möglichen Werte eines Attributs. Attributwerte können aus einem oder auch mehreren elementaren Werten, z.B. aus mehreren ganzen Zahlen bestehen. Attribute können also auch komplexe Datenobjekte darstellen. Die Definition von Attributtypen wird von der UML nicht festgelegt. Im Sinne einer einheitlichen Beschreibung von Attributtypen wird hier folgende Auswahl möglicher Typen von Attributen vereinbart:

- Standardtypen, die üblichen vordefinierten Datentypen von Programmiersprachen entsprechen, wobei hier Typnamen angelehnt an C++ verwendet werden.
- Aufzählungstypen, die eine endliche Menge verschiedener, als Namen angegebener Werte zusammenfassen. Ein Attribut eines Aufzählungstyps kann jeweils nur einen dieser Werte annehmen.
- Auch Klassen sind als Attributtypen zugelassen. Es sei z.B. eine Klasse Adresse mit entsprechenden Attributen wie Postleitzahl, Strasse usw. definiert worden. Dann kann in einer Klasse Person ein Attribut PersonenAdresse eingeführt werden und für dessen Typ die Klasse Adresse gewählt werden. Ein Objekt der Klasse Person besitzt in diesem Fall mit seinem Attribut PersonenAdresse ein Teilobjekt, das zur Klasse Adresse gehört. Record- bzw. Strukturtypen gelten in der OOA als spezielle Klassen ohne Operationen. Damit sind Recordtypen ebenfalls als Attributtypen zugelassen.
- Listentypen beschreiben Listen mit beliebig vielen Elementen eines einheitlichen Typs.

Die Tab. 4. enthält eine Aufstellung der hier verwendeten Attributtypen.

Typ	Erläuterung
Standardtypen	
Int	Beliebige ganze Zahl.
UInt	Beliebige nicht negative ganze Zahl.
Float	Beliebige reelle Zahl.
Boolean	Wahrheitswert, mögliche Werte: true (wahr), false (falsch).
Date	Datum.
Time	Zeit.
String, String(Länge)	String ohne bzw. mit spezifizierter maximaler Zeichenanzahl.
Weitere Typen	
Aufzählung	Aufzählungstyp. Seine möglichen Werte müssen in einer separaten Typdefinition festgelegt werden. Diese wird notiert gemäß: Typname = enum(Wert_1, Wert_2,..., Wert_n); z.B.: Ampelfarbe = enum(rot, gelb, gruen).
Klasse	Beliebige Klasse, die separat definiert wird. Recordtypen gelten ebenfalls als Klassen.
list of Typ	Listentyp, der durch die Angabe des Typs der Listenelemente als definiert gilt. *Typ* kann einer der zuvor aufgeführten Typen oder wiederum ein Listentyp sein.

Tab. 4.2. Zulässige Attributtypen für die Analyse.

Anfangswert

Einem Attribut kann ein Anfangswert zugeordnet werden. Dieser wird dem Attribut bei der Erzeugung eines Objekts automatisch zugewiesen.

Die bisher behandelten Merkmale von Attributen werden wie folgt notiert:

Attribut [:Typ] [=Anfangswert]

Dabei steht Attribut für den Attributnamen; optionale Bestandteile der Attributnotation sind in eckigen Klammern eingeschlossen.

weitere Attributmerkmale

Weitere optionale Merkmale eines Attributs werden anschließend in geschweiften Klammern notiert:

{Merkmal-1, Merkmal-2,...}

Als weitere optionale Merkmale von Attributen werden hier berücksichtigt:

Muß-/Kann-Attribut

- Muß-Attribute müssen stets einen Wert besitzen, während Kann-Attributen kein Wert zugeordnet sein muß. So wird etwa das Attribut Rechnungsdatum bei der Erzeugung eines Bestellungs-Objekts noch keinen Wert besitzen, während das Attribut Bestellnummer ein Muß-Attribut ist. Da Muß-Attribute überwiegen, werden nur Kann-Attribute durch die Merkmalsangabe {optional} besonders gekennzeichnet.

Schlüsselattribut

- Ein Attribut kann als Schlüsselattribut ausgezeichnet werden, wenn es – allein oder gemeinsam mit anderen Attributen (zusammengesetzter Schlüssel) – ein Objekt eindeutig identifiziert. Als Schlüsselattribut für Artikel kommt z.B. die Artikelnummer in Frage. Eine Kennzeichnung eines Schlüsselattributs wird durch die Merkmalsangabe {key} vorgenommen.

-

konstante Attribute

- Konstante Attribute dürfen ihren Wert nicht ändern, nachdem er einmal zugewiesen wurde. Dies gilt z.B. für die Artikelnummer eines Artikels. Die einmalige Wertzuweisung erfolgt meist bei der Erzeugung des Objekts mittels einer Konstruktoroperation (vgl. Kap. 4.1.4). Ein konstantes Attribut wird durch die Merkmalsangabe {readonly} notiert.
- Stellt ein Attribut eine quantitative Größe dar, ist oft die Angabe einer Einheit erforderlich. So wird etwa für das Attribut Verkaufspreis einer Klasse Artikel die Einheit Euro angegeben. Eine Einheit wird gemäß {Einheit} notiert.

Restriktionen

- Attribute können mit Restriktionen versehen werden. Diese beschreiben Bedingungen, denen Attributwerte genügen müssen. Restriktionen können ein oder mehrere Attribute einer Klasse einbeziehen. Die Restriktion {Verkaufspreis ≥ 10 Euro} fordert z.B., daß ein Attribut Verkaufspreis den Betrag von 10 Euro nicht unterschreitet. Die Restriktion {Rechnungsdatum ≥ Lieferdatum} verlangt, daß das Datum einer Rechnungserstellung nicht vor dem Lieferdatum liegt. Beide Attribute sollen derselben Klasse Bestellung angehören.

Klassenattribute

Schließlich seien zwei Sonderformen von Attributen betrachtet. Klassenattribute beschreiben Merkmale, die nicht einzelnen Objekten, sondern ihrer Klasse zukommen. Sie werden folglich nicht den Objekten, sondern der Klasse selbst zugeordnet. Daher besitzt ein Klassenattribut auch nur einen einzigen Wert für alle Objekte der Klasse und existiert unabhängig von diesen. Es kann bereits mit einem Wert initialisiert werden, wenn noch kein Objekt der Klasse existiert. Ein Klassenattribut wird durch die Unterstreichung des Attributnamens gekennzeichnet. In einer Klasse Artikel könnten als Klassenattribute etwa die Attribute Artikelanzahl und Rabatt eingeführt werden, wobei Artikelanzahl vor der Erzeugung von Artikel-Objekten mit dem Wert 0 zu initialisieren ist.

abgeleitetete Attribute

Einen weiteren Sonderfall stellen abgeleitete Attribute dar. Der Wert eines abgeleiteten Attributs läßt sich aus anderen Attributwerten derselben Klasse bestimmen. So kann etwa für eine Klasse Person das abgeleitete Attribut Alter aus dem Attribut Geburtsdatum berechnet werden. Ein unabhängiger Zugriff auf abgeleitete Attribute ist dagegen nicht möglich. Diese stellen daher zugleich konstante Attribute dar. Abgeleitete Attribute werden durch einen Schrägstrich vor dem Attributnamen notiert wie z.B. /Alter.

Beispiel 4.4

Die Attributbeschreibung einer Klasse Person könnte wie folgt aussehen:

Name: String
Vorname: String {optional}
PersonenNr: UInt {key, readonly, PersonenNr > 0}
Geburtsdatum: Date
/Alter: UInt
Familienstand: TFamilienstand = ledig;
    TFamilienstand = enum (ledig, verheiratet, geschieden, verwitwet)
LetzteAenderungFamilienstand: Date
    {optional, LetzteAenderungFamilienstand > Geburtsdatum}
KoerperGroesse: Float {m}
Hobbies: list of String
AnzahlPersonen: UInt
MaximaleKoerperGroesse: Float {m}