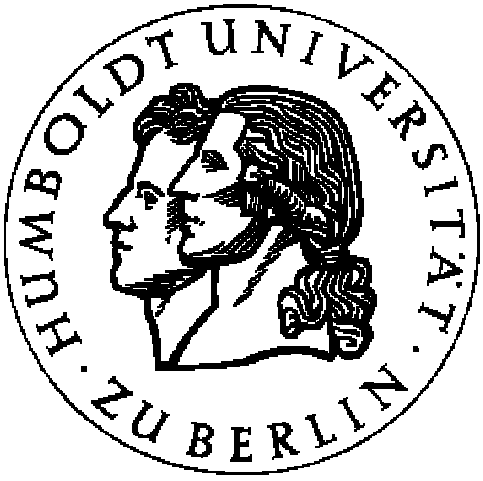


Hausarbeit

zum Hauptseminar „Fachdidaktik Informatik“

Thomas Baumfeld Matrikelnr. 160386
Uta Jannasch Matrikelnr. 158381

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Unsere Vorgabe war es, einen Lehrplan für das zweite Schulhalbjahr Informatik in der Klassenstufe 11 zu erstellen, welcher einen Einstieg in die objektorientierte Programmierung (OOP) bietet. Wie sich der Einstieg gestalten sollte, lag in unserem Ermessen.

Wir haben uns für einen Einstieg in die objektorientierte Programmierung über die objektorientierte Modellierung mit ihren Teilkomponenten objektorientierte Analyse, objektorientierter Entwurf und objektorientierte Programmierung entschieden. Der von uns benutzte Modellierungsstandard ist die Unified modelling language (UML), die benutzte Software für sowohl die objektorientierte Modellierung als auch die anschließende Programmierung ist Together 6.0. Die verwendete Programmiersprache ist Java in der Version 1.3.

Dies ist der erste Kontakt der Schüler zur Programmierung. Unsere Aufgabe ist es daher, die Grundlagen der Programmierung zu vermitteln und dabei speziell objektorientierte Vorgehensweisen zu beachten. Viele der Grundlagen, die wir bei den Schülern schaffen müssen, sind jedoch nicht explizit objektorientiert, für einen Einstieg in die Programmierung aber Voraussetzung.

Die Seminararbeit ist in zwei Teile untergliedert. Der erste Teil beschreibt den Ablauf des geplanten Schulhalbjahres. Er beginnt mit der Begründung, warum aus einer Vielzahl von möglichen Einstiegen die Modellierung als Einstieg in die OOP ausgewählt wurde. Die Frage nach den Lernvoraussetzungen bei den Schülern wird ebenso wie die nach den Lernzielen, die für das Schulhalbjahr gesteckt wurden, beantwortet. In der begründeten Stoffauswahl geht es um die Begründung, warum bestimmte Themen unterrichtet werden sollen, andere jedoch nicht. Der geplante Ablauf des Schulhalbjahres wird schließlich in sowohl Text- als auch Tabellenform vorgestellt.

Der zweite Teil der Seminararbeit stellt exemplarisch zwei Unterrichtsentwürfe vor, wobei einer dem Teilgebiet Objektorientierter Entwurf entnommen wurde und der andere der objektorientierten Programmierung zuzuordnen ist.

Im Anhang schließlich findet sich ein Fallbeispiel, welches in Grundzügen modelliert und programmiert wurde, um dem Leser eine Idee davon zu geben, was für das betrachtete Schulhalbjahr geplant wurde und machbar sein sollte.

2 Objektorientierte Modellierung als Einstieg in die objektorientierte Programmierung

Die objektorientierte Programmierung ist Schwerpunkt in den Klassenstufen 12 und 13. Die Grundlagen sollen jedoch schon in der 11. Klasse vermittelt werden, der Informatikunterricht in der 12. und 13. Klasse soll darauf aufbauend diese Grundlagenkenntnisse vertiefen.

Ein Einstieg in die objektorientierte Programmierung muss nicht zwingend über die objektorientierte Modellierung erfolgen. Man hätte als Hinführung zur objektorientierten Programmierung auch die imperative, funktionale oder deklarative Programmierung wählen oder den Einstieg spielerischer gestalten können, über Kara den Käfer beispielsweise.

Der von uns gewählte Einstieg erschien uns jedoch zielgerichteter und damit leichter, als zum Beispiel ein späterer Umstieg von anderen Programmierparadigmen. Hinzu kommt, dass die objektorientierte Programmierung oftmals als ein Bestandteil der objektorientierten Modellierung betrachtet wird. So unterteilen viele Lehrbücher zur UML die objektorientierte Modellierung in die drei Bestandteile Analyse, Entwurf und Programmierung. Diese Dreiteilung wollen wir bei der Gestaltung des Schulhalbjahres beibehalten.

Diese Begrifflichkeiten finden sich aber auch an anderer Stelle wieder: wird nämlich ein Softwareunternehmen beauftragt, eine Software für einen Kunden zu erstellen, dann ist jedem klar, dass der eigentlichen Realisierung der Software eine Phase der Planung vorausgehen muss. Die Planung ist ein elementarer Bestandteil der Softwareentwicklung, fast alle Konzepte zur Entwicklung von Anwendungssystemen unterscheiden die folgenden vier Phasen: Analyse, Entwurf (beides zusammen ergibt die Phase der Planung), Realisierung (die Phase der Programmierung) und Einführung. Es liegt auf der Hand, dass demzufolge die objektorientierte Analyse und der objektorientierte Entwurf Voraussetzung sind, um sinnvoll objektorientiert programmieren zu können.

Was in der professionellen Softwareentwicklung gilt, soll im verkleinerten Maßstab in unserem Informatikunterricht auch Gültigkeit besitzen. Es kann nicht Sinn eines Informatikunterrichts sein, einfach „drauf los zu programmieren“, dies ist für uns die wohl wichtigste Begründung für unseren Einstieg in die objektorientierte Programmierung über die objektorientierte Modellierung.

Dabei beschäftigt sich die Phase der Analyse mit der Aufgabe, die an einem Fallbeispiel beteiligten Akteure, ihre Handlungen (Methoden) und Eigenschaften zu klassifizieren. In der Phase des Entwurfs sollen die in der Analyse gefundenen Akteure mit ihren Methoden und Eigenschaften in verschiedenen Sichten modelliert werden. In der UML bieten sich dazu Klassen-, Zustands- und Sequenzdiagramme an. Vorgreifend auf die objektorientierte Programmierung bietet der Entwurf den Vorteil, dass er die Schüler in die Lage versetzt, den groben Aufbau eines Programms, die Zusammenhänge zwischen den Teilen eines Programms und die Wichtigkeit eines sinnvollen Aufbaus von Programmen schon im Voraus zu erkennen bzw. zu ermitteln.

Ein weiterer Vorteil des objektorientierten Entwurfs ist, dass man ohne ein tieferes Verständnis für die Syntax einer Programmiersprache auskommt. Mehr noch, Analyse und Entwurf legen die Grundlagen einer objektorientierten Programmsyntax und -semantik, indem beispielsweise grundlegende Begriffe der OOP, wie Objekt, Klasse, Methoden oder Variablen bildhaft und leicht verständlich eingeführt werden oder in Together Klassendiagramme in Programmcode überführt werden können.

Das direkte Programmieren des Quellcodes erfolgt dann im Teilgebiet objektorientierte Programmierung. Da bis zu diesem Punkt bei den Schülern ein allgemeines Verständnis über den Aufbau von Programmen geschaffen wurde, kann man sich nun primär auf die Probleme

der Codeimplementation konzentrieren. Schwerpunktmäßig stehen die Syntax und die Semantik von Java im Vordergrund.

Die objektorientierte Modellierung spannt somit den Bogen von einer allgemeinen Problemstellung bis hin zur Programmierung des Quellcodes. Dieser Weg ist dem menschlichen Vorstellungsvermögen nahe und verhindert, wie eingangs gesagt, blindes drauf los programmieren. Durch die Benutzung einer Software, die Modellierung (UML) und Programmierung (Java) in sich vereint, wird Kontinuität bei der Entwicklung von Programmen demonstriert, die einzelnen Phasen werden zusammengeführt.

3 Voraussetzungen, Gegebenheiten und Lernziele

3.1 Voraussetzungen und Gegebenheiten

Unter Voraussetzungen sind die Lernvoraussetzungen der Schüler zu verstehen, also die Frage nach dem Wissensstand der Schüler.

Da wir uns am Anfang des Programmierunterrichts befinden, hat das Gros der Schüler keine Programmierkenntnisse, diejenigen Schüler, die solche Kenntnisse besitzen, haben sie sich außerhalb der Schule angeeignet. Demzufolge haben die meisten Schüler keine Erfahrung mit der Objektorientierung.

Die gemeinsame Basis bei den Schülern sind relativ fundierte Anwendungskenntnisse, die Schüler können einen Computer bedienen und haben keine Scheu mehr im Umgang mit dem Computer.

Wünschenswerte Gegebenheiten wären ausreichende Computerarbeitsplätze für alle Schüler, so dass jeder an einem eigenen Computer arbeiten kann.

Das betrachtete Schulhalbjahr umfasst 17 Wochen, der Informatikunterricht ist für 3 Stunden à 45 min in der Woche angelegt, das macht insgesamt 51 Schulstunden, deren Verlauf geplant werden muss.

3.2 Lernziele

Die Schüler sollen nach dem Halbjahr in der Lage sein, ein gegebenes relativ einfaches Problem auf Akteure, deren Eigenschaften und Handlungen hin zu analysieren, diesen Sachverhalt in einem UML Klassendiagramm zu modellieren, daraus ein Programmgerüst zu generieren und nach Anleitung „fertig“ zu programmieren.

Die Schüler sollen im Teilgebiet Objektorientierte Analyse:

- Aus dem Sachverhalt die Akteure, Handlungen, Eigenschaften ableiten können,
- Wenn vorhanden, gemeinsame Oberklassen finden,
- Beziehungen zwischen Objekten erkennen,
- Begriffe der Objektorientierung verstehen.

Die Schüler sollen im Teilgebiet Objektorientierter Entwurf:

- Den Umgang mit Together lernen,
- Klassendiagramme in Together modellieren können,
- Diese Diagramme schrittweise um die einzelnen Bestandteile ergänzen,
- Beziehungen zwischen den Diagrammen modellieren,
- Kardinalitäten zuordnen,
- Den generierten Quellcode interpretieren.

Die Schüler sollen im Teilgebiet objektorientierte Programmierung:

- Den Algorithmusbegriff kennen lernen und verstehen,
- Den Javacompiler und Interpreter bedienen lernen,
- Anweisungen, Zuweisungen und Operatoren anwenden können,
- Kontrollstrukturen kennen lernen,

- Methodenköpfe, Methodeninhalte und die Übergabe von Parametern programmieren lernen,
- Arrays als Möglichkeit zur Speicherung von Daten kennen lernen,
- Objekte als Ausprägungen von Klassen anlegen und benutzen können,
- Den Umgang mit Konstruktoren lernen,
- Die Punktnotation als Methodenzugriff kennen lernen,
- Das Prinzip der Gültigkeit von Bezeichnern und der Datenkapselung verstehen,
- Programme auf Fehler hin untersuchen, diese Fehler beheben und in Syntax-, Semantik- und logische Fehler klassifizieren können.

4 Begründete Stoffauswahl

Da dieses Halbjahr nur eine Einführung in die objektorientierte Programmierung bieten soll, ist es verständlich, dass einige Teilgebiete der Objektorientierung nicht betrachtet werden können.

Die objektorientierte Analyse beschränkt sich auf die handelnden Akteure und ihre Eigenschaften und Handlungen, es soll nicht betrachtet werden, welche verschiedenen Arten von Klassen und daraus resultierend welche Schnittstellen (Ein/Ausgabedialoge) eingesetzt werden können. Dies zu behandeln wäre an dieser Stelle verfrüht.

Im Objektorientierten Entwurf sollen ausschließlich Klassendiagramme modelliert werden. Sequenz- und Zustandsdiagramme bieten sich zwar besonders dafür an, Handlungsabfolgen darzustellen, die Beschäftigung mit ihnen ist aber in diesem Rahmen nicht notwendig und zeitlich gesehen nicht machbar.

Die Objektorientierung ist gekennzeichnet durch die Begriffe Identität, Klassifikation, Vererbung und Polymorphie. Vererbung soll als Konzept auf die Phase der Analyse und Modellierung beschränkt werden und in der Programmierung erst im nächsten Schuljahr betrachtet werden. Polymorphie soll ebenso erst im folgenden Schuljahr behandelt werden.

In der Programmierung gibt es viele Teilbereiche, die in diesem Schuljahr nicht behandelt werden sollen, auch wenn sie in die Thematik passen würden. So würde es sich bei der Einführung von Arrays anbieten, Such- und Sortieralgorithmen vorzustellen, darauf soll jedoch verzichtet werden. Auch das große Gebiet der Rekursion soll nicht Inhalt dieses Halbjahres sein.

Listen sollen ebenso nicht behandelt werden, auch wenn man argumentieren könnte, dass sie sich besser als Arrays zur Speicherung von Daten eignen. Hier kann man das Fallbeispiel zu einem späteren Zeitpunkt dementsprechend abändern.

Die Möglichkeiten von Together werden nicht vollständig genutzt, beispielsweise soll auf die Arbeit mit dem Debugger verzichtet werden. Together ist viel mächtiger, als es ein Einsatz in der Schule verlangt. Hier müssten, wenn nötig, Anpassungen vorgenommen werden, um dieses Werkzeug schülerfreundlicher zu gestalten.

5 Ablauf des Schulhalbjahres

5.1 Geplanter Verlauf des Schulhalbjahres

Ziel dieses Lehrplans ist es, den Schülern die Grundlagen der objektorientierten Analyse, des objektorientierten Entwurfs und der objektorientierten Programmierung näher zu bringen.

Die einzelnen Schritte bei der Vorgehensweise, d.h. die zu behandelnden Themen sind der nachfolgenden Tabelle zu entnehmen. Ausgangspunkt ist ein Fallbeispiel zu einer Autovermietung. Dieses Fallbeispiel bietet den roten Faden, welcher sich durch den gesamten Lehrplan zieht. Anhand dieses Beispiels sollen die einzelnen Abschnitte des Lehrplans geübt werden, die Autovermietung im Verlauf des Schulhalbjahres schrittweise ergänzt werden. Zusätzlich zum Fallbeispiel sollen die einzelnen Themen des Schulhalbjahres an weiteren Beispielen geübt werden, da die Beschäftigung mit dem Fallbeispiel allein nicht ausreicht, um das vermittelte Wissen zu festigen. Die Vorgehensweise kann dabei variieren: entweder wird das Fallbeispiel zur Einführung in ein neues Thema genutzt und die Übungsaufgaben sind eigenständig, oder die Einführung findet anhand vom Fallbeispiel unabhängigen Beispielen statt und die Autovermietung wird im Laufe der eigenständigen Schülerübungen ergänzt. Eine Vorgabe dessen, wie dieses Fallbeispiel am Ende des Schulhalbjahres in Java aussehen kann, befindet sich im Anhang.[1]

Bevor der Unterricht mit der objektorientierten Analyse beginnt, sollen zwei Schulstunden dazu dienen, Organisatorisches zu klären, den Verlauf des Schulhalbjahres anzuzeigen und eine Einführung in die Materie der objektorientierten Modellierung zu geben: was sie ist, wozu man sie braucht und warum man nicht auf sie verzichten sollte.

Objektorientierte Analyse: Die Analyse erhält ihre Berechtigung dadurch, dass Softwareentwickler normalerweise keine Experten auf dem Gebiet der zu programmierenden Anwendung sind und daher Informationen über das zu lösende Problem sammeln müssen.

Wie sammelt man aber diese Informationen? Wie findet man beispielsweise die Klassen bei einem Anwendungsfall? Dafür lassen sich keine algorithmischen Antworten geben, dies ist ein Problem, welches innerhalb der objektorientierten Analyse zu lösen ist. Dabei sieht der Weg so aus, dass die Schüler das gegebene Fallbeispiel auf die beteiligten Akteure hin analysieren, die so gefundenen Objekte zu Klassen bündeln, ihnen Eigenschaften und Methoden zuweisen und eventuell die Beziehungen zwischen ihnen definieren und, wenn vorhanden, gemeinsame Oberklassen festlegen.

An dieser Stelle bietet es sich außerdem an, die Begrifflichkeiten der Objektorientierung zu klären: zum einen die Begriffe Objekt und Klasse, aber vor allem die Begrifflichkeiten, die einen objektorientierten Ansatz ausmachen: Identität, Klassifikation und Vererbung. Das Konzept der Polymorphie soll an dieser Stelle noch nicht betrachtet werden. Mir diesem Wissen sollten die Schüler in der Lage sein, die im Fallbeispiel bestimmten Akteure (Objekte) zu Klassen zusammenzufassen und eventuell weitere übergeordnete Klassen zu bestimmen.

Beziehungen zwischen den Objekten könnten im Rahmen der Analyse auch schon angesprochen werden, nicht theoretisch, sondern über Beispiele: ein Kunde hat eine Adresse, ein Privatkunde ist auch immer ein Kunde, ein Mitarbeiter arbeitet für die Autovermietung etc.

In der objektorientierten Analyse, welche die Schüler zum gegebenen Fallbeispiel durchführen, suchen sie nur nach den Klassen, welche Objekte der realen Welt darstellen;

[1] Das Fallbeispiel im Anhang beschränkt sich auf die Kundenverwaltung einer Autovermietung. Die Verwaltung der Autos wäre ähnlich zu implementieren, wurde hier jedoch aus Zeitgründen weggelassen. Einige der Klassen werden den Schülern vorgegeben.

Klassen zur Verwaltung von Objekten werden den Schülern später bei der Programmierung zur Verfügung gestellt.

Objektorientierter Entwurf: Bevor die Schüler UML Diagramme modellieren können, müssen sie mit der Modellierungsumgebung Together vertraut gemacht werden. Dazu gehört, die Oberfläche kennen zu lernen, die einzelnen Ansichten vorgestellt zu bekommen und neue Projekte anzulegen.

Die in der objektorientierten Analyse identifizierten Klassen sollen in UML Klassendiagramme umgesetzt werden. Diese Klassendiagramme sollen schrittweise um die einzelnen Bestandteile ergänzt werden, dazu müssen verschiedene Konzepte vorgestellt werden. Um beispielsweise die Eigenschaften von Objekten der Klassen zu modellieren, muss das Variablenkonzept zusammen mit den dazugehörigen Datentypen an dieser Stelle eingeführt werden. Bei den Datentypen sollen nicht nur die primitiven Datentypen, sondern auch die abstrakten Datentypen betrachtet werden. Analog zur Modellierung der Eigenschaften von Objekten mit Hilfe des Variablenkonzepts sollen die Handlungen durch Methoden beschrieben werden. An dieser Stelle müssen aber nur die Methodenköpfe und ihre Rückgabewerte modelliert werden.

Die einzelnen Klassen existieren nicht unabhängig voneinander, sondern treten zueinander in Beziehungen. Der Modellierung dieser Beziehungen soll ebenso Zeit eingeräumt werden. Die Einführungsstunde zur Modellierung von Beziehungen innerhalb eines Klassendiagramms befindet sich als ausführlicher Unterrichtsentwurf unter Punkt 6 in dieser Seminararbeit. Im Anschluss an die Modellierung der Beziehungen soll der Begriff der Kardinalitäten geklärt werden.

Abschließend zu diesem Themenbereich und überleitend zur eigentlichen Programmierung empfiehlt es sich, den zu den modellierten Klassendiagrammen dazugehörigen Programmcode zu betrachten. Dieses Gerüst des Quellcodes soll von den Schülern auf die allgemeine Struktur von Programmen (z.B. Stellung der Variablen) hin betrachtet werden.

Objektorientierte Programmierung: Der Teilbereich objektorientierte Programmierung beansprucht mit 32 Unterrichtsstunden den weitaus größten Anteil am Halbjahr. Auch wenn an dieser Stelle grundlegende Konzepte schon geklärt wurden, sollen trotzdem immer wieder Rückschlüsse zur Modellierung gezogen werden, das empfiehlt sich beispielsweise bei der Programmierung von Beziehungen. So kann man Assoziationen auf der Modellierungsebene zeichnen, man kann sie in Together aber auch programmieren und sie dadurch automatisch in das Modell einfügen. Durch die Betrachtung der Modelle – und dadurch die Visualisierung des Programmcodes - soll der Zusammenhang zwischen Programm und Modell verdeutlicht werden.

In einer Theoriestunde zu Beginn des Teilbereichs soll den Schülern der Algorithmusbegriff näher gebracht werden. Bevor mit Java gearbeitet werden kann, müssen außerdem die Begriffe Compiler und Interpreter geklärt werden.

Eine detaillierte Aufzählung der Programmierinhalte, die in diesem Teilgebiet vermittelt werden sollen, ist in der anschließenden Tabelle zu finden. Der Aufbau der Inhalte ist logisch, beginnend bei Zuweisungen als grundlegendste Programmeinheiten, über Kontrollstrukturen, Methoden, Parameterübergabe sowie Arrays hin zur eigentlichen Objektorientierung, wie z.B. dem Anlegen und Zugriff auf Objekte oder dem Erzeugen von Konstruktoren. Das Thema Konstruktoren wird in dem zweiten Unterrichtsentwurf in dieser Arbeit, zu finden unter Punkt 7, thematisiert.

Um das Thema Datenkapselung, welches eine grundlegende Eigenschaft der Objektorientierung ist, unterrichten zu können, müssen die Gültigkeitsbereiche von Bezeichnern im Vorfeld geklärt werden.

Die Fehlererkennung ist ein weiterer wichtiger Bestandteil des Programmierens und muss von Anfang an Inhalt eines guten Programmierunterrichts sein. Die Fehlererkennung und Behebung wird nichts erst an dieser Stelle eingeführt, sondern schon beim Programmieren der ersten Programme angewandt. Hier an dieser Stelle soll sie aber explizit thematisiert werden, die einzelnen Fehler sollen in Syntax-, Semantik- und logische Fehler unterteilt werden. Übungen zur Fehlererkennung helfen den Schülern beim Lesen und Verstehen von Programmcode.

5.2 Übersicht über das Schulhalbjahr

Thema	**Inhalte**	**Anmerkungen zu den Inhalten**	**Zeit (Stunden)**
Einführung	Verlauf des Halbjahres, Organisatorisches		1
Objektorientierte Modellierung:			
	Was ist ein Modell Ziele, Vorteile, warum modelliert man, Anforderungen an Modelle Objektorientierte Analyse, Objektorientierter Entwurf, Objektorientierte Programmierung	Als Erklärung, warum man modelliert: z.B. Bau einer Hundehütte: keine Planung nötig, Bau eines Hauses: Grundriss, Lageplan, verschiedene Außenansichten, Werkpläne für die Handwerker	1
Objektorientierte Analyse:			
Analyse des Fallbeispiels	Akteure aus Fallbeispiel identifizieren Handlungen (= Methoden) und Eigenschaften (= Variablen) aus Fallbeispiel identifizieren Beziehungen zwischen den Akteuren identifizieren	Fallbeispiel Autovermietung Weitere Übungsbeispiele	3
Begriffe der Objektorientierung	Was sind Klassen, was Objekte Identität, Klassifikation, Vererbung		2
Objektorientierter Entwurf:			
Einführung in Together	Oberfläche, Ansichten, Projekte anlegen		1
Modellierung von UML Klassendiagrammen in Together	Klassen Variablenkonzept Datentypen (primitive, abstrakte)	UML Klassendiagramme einführen Klassen aus Fallbeispiel anlegen	10

	abstrakte) Methodenköpfe, Rückgabewerte Beziehungen zwischen Klassen Kardinalitäten	Mit Variablen, Datentypen und Methodendeklarationen füllen Aggregationen, Assoziationen, Generalisierung, Komposition	
	Quellcode des modellierten Gerüsts anschauen	Entspricht dem Forward Engineering	1
Objektorientierte Programmierung:			
	Algorithmusbegriff Compiler /Interpreter		1
	Anweisungen: Operatoren Zuweisungen if – then – else Anweisungen, switch - Anweisungen, Schleifen		11
	Methoden, Parameterübergabe		2
	Arrays		3
	Erzeugung und Zugriff auf Objekte Konstruktoren		2
	Methoden fremder Klassen nutzen	Punktnotation	2
	Gültigkeitsbereich von Bezeichnern Datenkapselung Fehlererkennung	Public, private, jedoch nicht protected Syntax-, Semantik-, logische Fehler	11
Summe:			**51**

6 Unterrichtsentwurf 1: Beziehungen zwischen Klassen[2]

6.1 Stellung der Stunde im Unterricht und Schülervoraussetzungen

Die betrachtete Stunde ist Inhalt des Abschnitts Objektorientierter Entwurf.

Die allgemeine Einführung in das Schulhalbjahr und der Abschnitt objektorientierte Analyse sind zu diesem Zeitpunkt vollständig abgeschlossen, was bedeutet, dass die Schüler die am Fallbeispiel beteiligten Akteure herausgefunden haben, ihnen Attribute (Eigenschaften) zugeordnet haben und ihre Methoden (was tun die Objekte dieser Klassen?) bestimmt haben. Dazu ist es notwendig, dass die Schüler die Begriffe Klassen und Objekte verstanden haben.

Die Schüler haben im Rahmen des Objektorientierten Entwurfs bereits erste Gehversuche in Together unternommen und neue Projekte angelegt. Sie haben bereits Klassen angelegt und diese mit den entsprechenden Attributen und Methoden gefüllt. Die Schüler wissen bereits, dass Attribute in Together Variablen einer Programmiersprache entsprechen und diese Variablen Datentypen besitzen. Ihnen ist die Unterscheidung in primitive (ein Kunde enthält die Variable Name vom Typ String) und abstrakte Datentypen (ein Kunde hat eine Anschrift vom Typ Adresse mit allen Variablen und Methoden, die der Typ Adresse enthält) bekannt. In der Methodenvereinbarung haben sich die Schüler mit Rückgabewerten von Methoden beschäftigt. Sichtbarkeiten (public, private, protected) wurden bisher noch nicht behandelt. Der Konstruktor als spezielle Methode wurde an dieser Stelle auch noch nicht besprochen.

Mit diesem Vorwissen sind die Schüler nun in der Lage, Beziehungen zwischen Klassen zu bestimmen. Es gilt beispielsweise zu klären, welche Klassen andere Klassen benutzen oder welche Klassen von anderen Klassen erben. Das Vererbungskonzept soll an dieser Stelle besprochen werden, weitere Beziehungstypen, die zur Sprache kommen, sind die Assoziation und die Aggregation. Weiterführende Übungen zu Beziehungen, Übungen speziell zur Vererbung und das Thema Kardinalitäten von Assoziationen und Aggregationen sind die Inhalte der nächsten Stunden.

6.2 Sachanalyse

6.2.1 Vererbung

Vererbungsbeziehungen werden in Together Generalisierungen genannt. Eine Generalisierung ist eine Beziehung zwischen einer allgemeinen Klasse (der Oberklasse) und einer speziellen Klasse (der Unterklasse). Eine Generalisierungsbeziehung stellt „eine-Art-von“ Beziehung dar: so ist die Unterklasse Privatkunde „eine Art von“ Kunde, Kunde ist hierbei die Oberklasse. Gleiches gilt beispielsweise für die Beziehung zwischen Kunde (Oberklasse) und Geschäftskunde (Unterklasse). Die Oberklasse lässt sich durch die Unterklasse ersetzen, d.h., dass überall dort, wo die Oberklasse auftreten kann, auch die Unterklasse verwendet werden kann, umgekehrt aber nicht. Die Unterklasse erbt die Attribute und Methoden der Oberklasse, besitzt aber häufig zu denen der Oberklasse weitere Attribute und Methoden. Geerbte Attribute und Methoden werden in der Unterklasse nicht noch einmal aufgeführt. Besitzt aber eine Methode der Unterklasse dieselbe Signatur (dasselbe Aussehen) wie eine Methode der Oberklasse, wird die Methode der Oberklasse überschrieben, man spricht in diesem Fall von Polymorphie.

Eine Klasse kann null, eine oder mehrere Oberklassen haben. Hat die Klasse null Oberklassen, ist sie selbst eine Basisklasse. Hat sie eine Oberklasse, spricht man von Einfachvererbung. Mehrfachvererbung tritt dann auf, wenn eine Klasse mehr als eine Oberklasse besitzt. Dies ist in Java nur über Interfaces möglich.

[2] Ein vollständiger Unterrichtsentwurf enthält auch Angaben zu den Schülern sowie zum Lehrer bzw. zur Lehrerin. Da wir keinen Unterrichtsentwurf für eine konkrete Klasse erstellen, verzichten wir auf diese Angaben.

Grafisch werden Generalisierungen über durchgezogene Pfeile, die mit einer offenen Spitze auf die Oberklasse weisen, dargestellt:

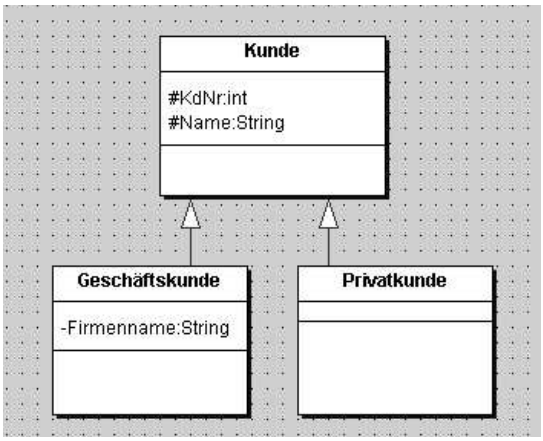


Abbildung 1: Generalisierungen in Together

6.2.2 Assoziationen

Assoziationen stellen Beziehungen zwischen Objekten einer Klasse dar, werden aber in der UML (und daher auch in Together) als Beziehungen zwischen den Klassen dieser Objekte modelliert. Eine Assoziation spezifiziert, dass ein Objekt einer Klasse mit einem Objekt einer anderen Klassen zusammenhängt. Die Art der Beziehung, die Objekte bei der Assoziation eingehen, kann man als „benutzt" Beziehung bezeichnen, grundsätzlich sind die Objekte voneinander unabhängig, treten aber in bestimmten Situationen miteinander in Beziehung.

Eine Assoziation, die zwei Klassen miteinander verbindet, wird als binäre Assoziation bezeichnet.

Assoziationen sind standardmäßig bidirektional. Ihre Richtung kann jedoch eingeschränkt werden. Wenn der Zugriff nur in eine Richtung erfolgt, spricht man von einer gerichteten Assoziation.

Eine Assoziation kann einen Namen haben. Dieser soll dazu dienen, die Beschaffenheit der Beziehung zu beschreiben. Ein Name kann aus Gründen der Unzweideutigkeit eine Richtung besitzen. Außerdem können Klassen, die über eine Assoziation miteinander verbunden sind, Rollen besitzen. Rollen benennen die Bedeutungen, die Klassen in einer Assoziation spielen, explizit. Möchte man angeben, wie viele Objekte in einer Assoziation zusammenhängen, vergibt man Kardinalitäten (in der UML auch Multiplizitäten genannt). Die folgende Abbildung zeigt eine Assoziation, in der 1 bis viele Personen in der Rolle Arbeitnehmer für ein Unternehmen arbeiten, welches die Rolle Arbeitgeber besitzt.

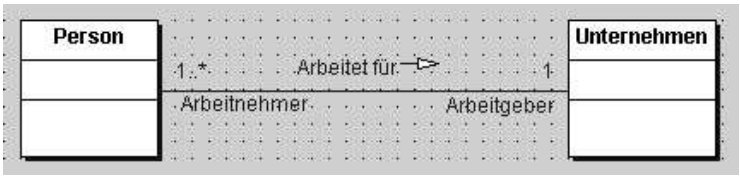


Abbildung 2: Assoziation mit Name, Rollen und Kardinalitäten

6.2.3 Aggregation

Eine Aggregation stellt eine besondere Assoziation dar, bei der die beteiligten Objekte nicht gleichberechtigt sind, wie bei der einfachen Assoziation, sondern bei der eine Ganzes / Teil Beziehung dargestellt wird. Diese Art der Beziehung kann man als „Teil von" Beziehung bezeichnen. Aggregationen sind immer gerichtet. Die Klasse, die das Teil Objekt

repräsentiert, ist die Komponente, die Klasse, die für das Ganze steht, das Aggregat. Man unterscheidet zwei Arten der Aggregation, die einfache Aggregation und die Komposition.

6.2.3.1 EINFACHE AGGREGATION

In einer einfachen Aggregation kann ein Teil zu mehreren Ganzen gehören. In einem Haus kann beispielsweise eine Wand Teil mehrerer Raum Objekte sein. Außerdem existiert das Teil unabhängig vom Ganzen. Die Darstellung entspricht einer Assoziation mit einer offenen Raute an dem Ende, an dem sich das Ganze befindet. Abbildung 3 stellt eine einfache Aggregation dar.

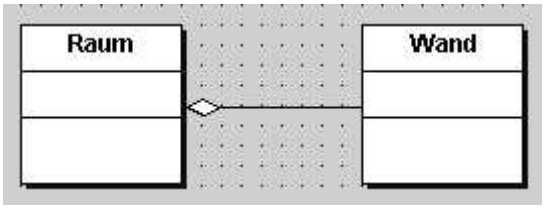


Abbildung 3: Eine einfache Aggregation

6.2.3.2 KOMPOSITION

In einer Komposition gehört ein Teil immer genau zu einem Ganzen. Eine Stuhllehne gehört genau zu einem Stuhl, nicht zu mehreren. Erzeugt man das Ganze (das Aggregat), erzeugt man auch das Teil (die Komponente), das gleiche gilt für das Löschen. Die Darstellung entspricht einer einfachen Aggregation mit dem Unterschied, dass die Raute ausgefüllt ist. Abbildung 4 verdeutlich diesen Sachverhalt.

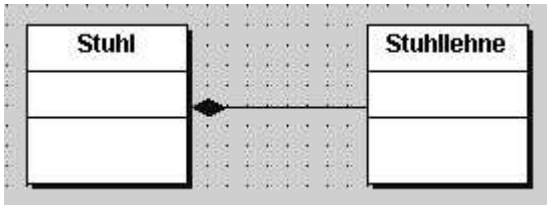


Abbildung 4: Eine Komposition

6.3 Begründete Stoffauswahl

Die in der Sachanalyse vorgestellten Beziehungstypen stellen die gebräuchlichsten Arten von Beziehungen dar. Sie sollen daher in der betrachteten Unterrichtsstunde behandelt werden. Weitere UML Beziehungstypen, wie Realisierungen, Einschränkungen von Assoziationen oder Abhängigkeiten sind zu speziell und gehören daher nicht in einen Unterricht für Programmieranfänger.

Bei der Betrachtung der Generalisierung soll die Möglichkeit der Mehrfachvererbung ausgespart werden. Eine Mehrfachvererbung ist in Java (indirekt) nur über Interfaces möglich, Interfaces werden aber in diesem Schulhalbjahr nicht behandelt.

Bei der Modellierung der Beziehungen bewegen sich die Schüler nur auf der Ebene des Together „Designer Pane“, der eigentlichen Modellieroberfläche. Wie sich die Modellierung der Beziehungen auf die Javasyntax auswirkt, soll an dieser Stelle noch nicht bzw. nur dann betrachtet werden, wenn die Schüler den entsprechenden Wunsch äußern. Zu diesem Zweck müsste man in Together auf die Ebene des „Editor Pane“wechseln.

Der Einstieg in die Modellierung von Beziehungen wird anhand allgemeiner Beispiele gegeben, mit diesem Wissen sollen sich die Schüler über die Beziehungen im Fallbeispiel Autovermietung Gedanken machen und dieses Fallbeispiel in einer der darauffolgenden Stunden dahingehend ergänzen.

6.4 Unterrichtsziele

Die Schüler sollen:

- Selbstständig erkennen, dass Objekte und Klassen miteinander in Beziehung treten und dass man diese Beziehungen auch modellieren muss (LZ 1),
- Beispiele für Beziehungen nennen können (LZ 2),
- Zwischen den Beziehungstypen Generalisierung, Assoziation, einfache Aggregation und Komposition unterscheiden können sowie Einsatzmöglichkeiten der Beziehungstypen erkennen (LZ3),
- Die Beziehungstypen Generalisierung, Assoziation, einfache Aggregation und Komposition in Together modellieren können (LZ 4),
- Beziehungen an der richtigen Stelle einsetzen können bzw. in der Lage sein, fehlerhaft gesetzte Beziehungen auszubessern (LZ 5),
- Aus Textaufgaben Klassen, Objekte, Attribute, Methoden und Beziehungen herausfinden und diese in ein UML Modell sowohl in schriftlicher Form als auch in Together übertragen können (LZ 6).

6.5 Weg- und Medienentscheidung

Phase I: Als Hinführung zum Thema dient das Fallbeispiel Autovermietung, welches auf aktuellem Stand ist, mit allen Klassen, Attributen und Methoden, die von den Schülern bisher erkannt und modelliert wurden. Anhand dieses Fallbeispiels sollen die Schüler erkennen, dass dieses Beispiel noch nicht vollständig ist, da Objekte bzw. Klassen miteinander in Beziehungen treten, diese aber noch nicht modelliert sind. Die Schüler sollen sich Gedanken darüber machen, in was für Beziehungen die Objekte und Klassen des Fallbeispiels miteinander treten können. Es ist wahrscheinlich, dass die Schüler bisher noch keine Vererbungen modelliert haben, da sie das Prinzip noch nicht kannten. Daher ist anzunehmen, dass sie diese Art der Beziehung nicht nennen werden. Diese Annahme ist aber rein spekulativ.

Der Computer der Schüler kommt in dieser Phase noch nicht zum Einsatz, das zu betrachtende Fallbeispiel wird mittels Beamer an die Wand projiziert. Die genutzte Lehrform ist der fragend-entwickelnde Unterricht.

Phase II: In Phase II werden die Beziehungstypen Generalisierung, Assoziation, einfache Aggregation und Komposition vom Lehrer vorgestellt. Die gewählte Lehrform ist ein Lehrervortrag, da dieser sich insbesondere dann gut eignet, wenn man Inhalte zügig vermitteln möchte. Aber auch in dieser Phase sollen die Schüler v.a. zu Beispielen zu den vorgestellten Beziehungen befragt werden, mittels der Lehrform Unterrichtsgespräch. Da die Schüler später in der Lage sein müssen, nicht nur am Computer, sondern auch auf dem Papier zu modellieren, werden die Klassen und ihre Beziehungen an die Tafel gezeichnet.

Phase III: In dieser Phase kommt der Computer der Schüler zum ersten Mal zum Einsatz. Sie sollen nun die Beispiele, die ihnen zu den verschiedenen Beziehungstypen an der Tafel vorgegeben wurden, in Together modellieren. Die Schüler arbeiten in dieser Phase selbstständig.

Phase IV: ist eine weitere Phase der selbstständigen Schülerarbeit, die Schüler erhalten ein Arbeitsblatt (Arbeitsblatt 1), welches fehlerhaft gesetzte Beziehungen enthält. Ihre Aufgabe ist es, diese Fehler auf dem Papier zu korrigieren. Dieses können sie alleine oder zu zweit tun. Anschließend werden die Lösungen besprochen.

Phase V: In dieser Phase bearbeiten die Schüler das Arbeitsblatt 2, welches eine Textaufgabe enthält. Hier müssen sie ihr Wissen nicht nur dieser Stunde, sondern auch der vergangenen Stunden einsetzen, da Klassen, Attribute, Methoden und Beziehungen modelliert werden sollen. Die Lösung soll von den Kindern in Together übertragen werden. Es ist relativ unwahrscheinlich, dass die Aufgabe in dieser Stunde fertiggestellt wird, eher ist zu erwarten, dass Phase V in dieser Stunde gar nicht begonnen wird. Die Schüler können daher in der nächsten Stunde an dem Arbeitsblatt weiterarbeiten. Die Lösungen werden in der nächsten Stunde besprochen.

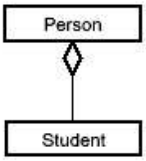
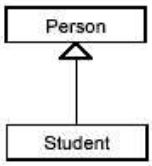
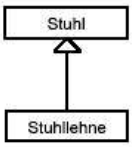
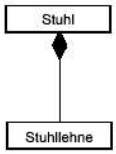
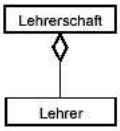
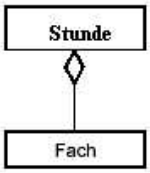
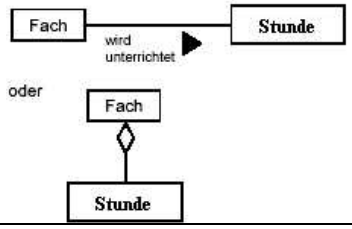
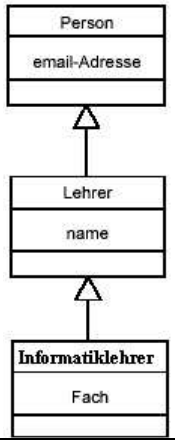
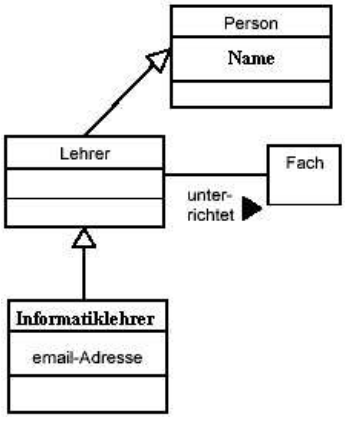
6.6 Geplanter Unterrichtsverlauf

Phase / Dauer	**Inhalt**	**Lernziele**	**Medien**	**Lehrform**	**Sozialform**
Phase I, 5 min	Untersuchung des Fallbeispiels hinsichtlich Beziehungen	LZ 1, 2	Computer des Lehrers, Beamer	Fragend-entwickelnder Unterricht	Frontalunterricht
Phase II, 13 min	Definition und Beispiele für Generalisierung, Assoziation, einfache Aggregation, Komposition	LZ 3	Tafel	Lehrervortrag, Gespräch	Frontalunterricht
Phase III, 10 min	Übertragen der Beispiele aus Phase II in Together	LZ 4	Computer der Schüler	Selbsttätige Schülerarbeit	Einzelarbeit der Schüler
Phase IV, 12 min	Bearbeitung des ersten Arbeitsblatts: Fehlererkennung	LZ 5	Arbeitsblatt 1	Selbsttätige Schülerarbeit	Einzel- oder Partnerarbeit der Schüler
Phase V, 5 min	Schüler **beginnen** mit der Bearbeitung des zweiten Arbeitsblatts: Textaufgabe	LZ 6	Arbeitsblatt 2, Computer der Schüler	Selbsttätige Schülerarbeit	Einzel- oder Partnerarbeit der Schüler

6.7 Anhang

6.7.1 Arbeitsblatt 1 (inklusive Lösungen)[3]

Sind die nachfolgenden Klassendiagramme richtig oder falsch? Beurteilen Sie und korrigieren Sie die falschen Diagramme.

Klassendiagramm	Richtig oder falsch?	Korrigiertes Klassendiagramm
	Falsch! Ein Student ist nicht Bestandteil einer Person, sondern eine spezielle Person.	
	Falsch! Eine Stuhllehne ist kein spezieller Stuhl, sondern ein fester Bestandteil eines Stuhls.	
	Richtig!	
	Falsch! Eine (Unterrichts)stunde besteht nicht aus Fächern, eher umgekehrt: ein Fach wird in einer Stunde unterrichtet.	
	Falsch! Die Attribute sind auf den falschen Spezialisierungsebenen, nicht jede Person hat eine email-Adresse, dafür aber einen Namen. Jeder Lehrer unterrichtet mindestens ein Fach, Informatiklehrer haben (zumindest in diesem Beispiel) eine email-Adresse.	

[3] Angelehnt an Niklaus Mannhart, OO Softwareentwicklung, zu finden unter: http://www.inf.ethz.ch/personal/mannhart/teaching/hsr/loesung_uebung_uml.pdf

6.7.2 Arbeitsblatt 2 (inklusive Lösungen)[4]

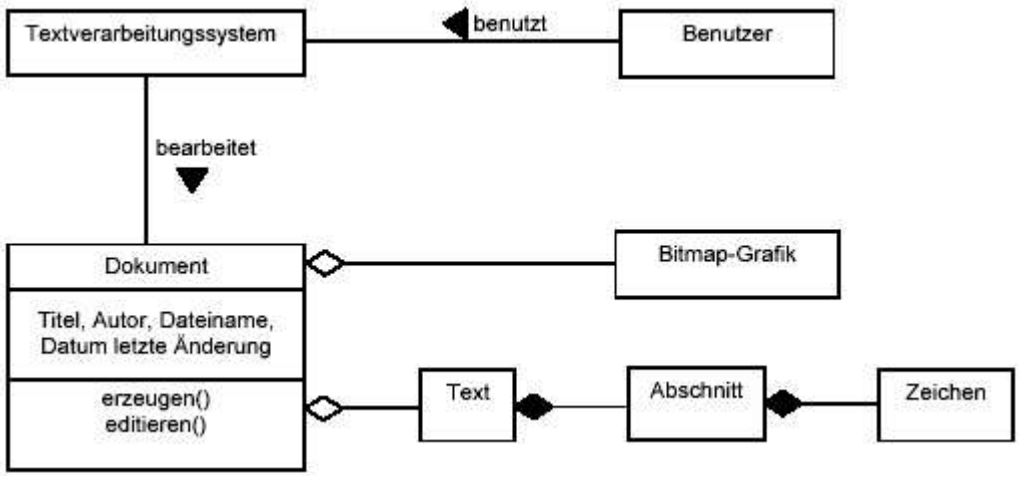
Gegeben seien folgende Anforderungen an ein Textverarbeitungssystem.

- Das Textverarbeitungssystem erlaubt es Herrn Müller und anderen Benutzern, Dokumente anzulegen und zu editieren.
- Ein Dokument kann Text und Bitmap-Grafiken enthalten. Ein Text besteht aus Abschnitten, jeder Abschnitt aus Zeichen.
- Ein Dokument enthält Informationen, wie seinen Titel, seinen Autor, seinen Dateinamen sowie das Datum der letzten Änderung.

Aufgaben:

1. Welche Objekte / Klassen lassen sich im obigen Beispiel identifizieren?
2. Welche Beziehungen bestehen zwischen den Klassen? Zeichnen Sie das entsprechende Klassendiagramm in der UML Notation.
3. Welche Attribute und Methoden hat die Klasse Dokumente? Ergänzen Sie das gezeichnete Klassendiagramm entsprechend.

Klassen: Textverarbeitungssystem, Benutzer, Dokument, Text, Bitmap-Grafik, Abschnitt, Zeichen



[4] Angelehnt an Niklaus Mannhart, OO Softwareentwicklung, zu finden unter: http://www.inf.ethz.ch/personal/mannhart/teaching/hsr/loesung_uebung_uml.pdf

7 Unterrichtsentwurf 2: Konstruktoren

7.1 Stellung der Stunde im Unterricht und Schülervoraussetzung

Die Stunde ist Teil des Bereiches objektorientierte Programmierung.

Die Schüler haben die Teile der objektorientierten Analyse und des objektorientierten Entwurfes bereits absolviert und beschäftigen sich nun primär mit der Problematik der Programmierung. Die Schüler haben ein allgemeines Verständnis vom Ablauf von Programmen und kennen theoretisch die Zusammenhänge von Programmteilen eines objektorientierten Programms. Insbesondere wissen sie um den Ablauf für das Anlegen eines Objekts. Sie können mit den Begriffe Klassen, Objekt, Methode arbeiten und kennen das Variablenkonzept. Erste Erfahrungen mit der Syntax von Java wurden gemacht. Hier sind die Schüler in der Lage, Kontrollstrukturen in Programme zu implementieren und Methoden- sowie Klassen grob zu schreiben (Variablen, Methoden die eine Klasse besitzt). Das Arbeiten in Together 6.0 sind die Schüler gewöhnt.

Auf dieser Basis ist es möglich, den Schülern die Konstruktoren als spezielle Methoden vorzustellen. Durch das Wissen um die Möglichkeit und Notwendigkeit in objektorientierten Programmen Objekte zu erzeugen, sollte der Sinn eines Konstruktors schnell erarbeitet werden. Die speziellen syntaktischen Probleme werden gemeinsam erarbeitet, durch das Üben an einem Beispiel sollte das eigenständige Programmieren von Konstruktoren nicht allzu viele Schwierigkeiten bereiten. Das Aufrufen von Methoden aus einer anderen Klasse wird ebenfalls in dieser Stunde geklärt.

In den folgenden Stunden wird dann die Problematik der Konstruktoren gefestigt. Insbesondere wird der Selbstverweis this behandelt. Weitere Sachverhalte werden speziell von der Seite der Syntax her betrachtet.

7.2 2. Sachanalyse

7.2.1 Anlegen von Objekten

Um Objekte einer Klasse zu erzeugen, nutzt man spezielle Methoden. Diese Methoden heißen Konstruktoren. Je nach den Anforderungen an ein Objekt kann eine Klasse mehrere Konstruktoren bereithalten (Überladen von Konstruktoren).

7.2.2 Der Konstruktor

Im Sinne einer klaren Programmstrukturierung ist es zweckmäßig, die Anweisungen zur Bildung des Anfangszustandes eines Objekts in einer eigenen Methode zusammenzufassen. Java bietet hierfür eine spezielle Methode, den Konstruktor.

Im Wesentlichen wird durch den Aufruf der Konstruktormethode ein neues Objekt einer Klasse angelegt. Gleichzeitig werden somit die Membervariablen und die Methoden des Objekts, wie in der Klasse vorgegeben, erstellt.

Der Konstruktor hat stets den gleichen Namen wie die Klasse, der er angehört. Es ist möglich und je nach Anforderung sinnvoll, bestimmte Parameter an einen Konstruktor zu übergeben. In Java hat jede Klasse automatisch einen Konstruktor. Ein Konstruktor ist stets für alle Klassen sichtbar (public). Sind mehrere Konstruktoren für eine Klasse implementiert, so unterscheiden diese sich in ihren Parametern (Überladen von Konstruktoren).

7.2.3 Besonderheiten des Konstruktors als Methode

Der Konstruktor hat stets den gleichen Namen wie die Klasse. Ein Konstruktor liefert als Ergebnis stets einen Verweis auf das neu erzeugte Objekt an die Aufrufstelle. Da der Rückgabetyp nicht frei wählbar ist, wird er nicht geschrieben.

Wenn eine Klasse mehrere Konstruktoren enthält, ist es nicht erlaubt, dass zwei Konstruktoren in der Anzahl und den Datentypen ihrer Parameter übereinstimmen.

7.2.4 Allgemeiner Aufbau eines Konstruktors ohne Parameter

```
class Tier {
                    Int beine;
                    String laut;
                    //Konstruktor der Klasse Tier
                    public Tier ()
                    {
                            //Initialisieren der Variablen
                            beine = 0;
                            laut = '';
                    }
                    …
        }
```

7.2.5 Initialisieren eines Objekts sowie Aufruf des Konstruktors

Das Initialisieren eines Objekts erfolgt in zwei Schritten. Zuerst wird eine ‚Variable' (genauer ein ADT) für das Objekt an sich angelegt. Dann wird der Konstruktor der Klasse aufgerufen, deren Typ das Objekt haben soll.

```
zu erzeugendes Objekt:   ich benötige ein Objekt der Klasse Tier
→                        Tier    tier1;  // ADT tier1 vom Typ Tier
erzeugen des Objekts:    Aufrufen des Konstruktors der Klasse Tier
                         tier1 = new Tier()
```

Es besteht auch die Möglichkeit beide Schritte zusammenzufassen.

```
Bsp.:                    Tier tier1 = new Tier();
```

In diesem Falle würde der ADT tier1 angelegt und gleichzeitig ein Objekt der Klasse Tier erzeugt werden.

7.3 Begründete Stoffauswahl

Den Schülern muss die Syntax der Konstruktormethode vermittelt werden. Folgend müssen die Schüler wissen, wie man einen Konstruktor aufruft. Die Schüler müssen wissen, dass der Aufruf eines Konstruktors, welcher eine Methode einer anderen Klasse ist, sich vom sonstigen Aufruf klassenfremder Methoden unterscheidet

Es wird sich nur auf den Defaultkonstruktor bezogen, welcher sämtliche Membervariablen der typgebenden Klasse anlegt. Alle Membervariablen werden als ‚leere' Variablen angelegt, eine Wertzuweisung erfolgt erst später.

Der Aufruf des Konstruktors anderer Klassen wird nur als einfacher Aufruf eingeführt, da keine Werte übergeben werden. Durch den Defaultkonstruktor bedingt, werden keine Übergabewerte behandelt/ betrachtet.

Weiterhin ist nicht Inhalt der Stunde das Überladen von Konstruktoren, es wird nur im theoretischen Teil erwähnt.

7.4 Unterrichtsziele

- Die Schüler können einen Default Konstruktor programmieren. Sie arbeiten mit der richtigen Syntax (LZ 1).
- Die Schüler können einen Konstruktor aufrufen. Sie arbeiten mit der richtigen Syntax (LZ 2).
- Die Schüler können ein Objekt einer Klasse anlegen. Dazu vereinbaren sie ein ADT und initialisieren an anderer Stelle das Objekt (LZ 3).
- Die Schüler wissen, das der Konstruktor eine Methode ist. Sie wissen, dass der Konstruktor von fremden Klassen aufgerufen wird. Sie wissen in diesem Zusammenhang, das ‚reguläre' Methoden fremder Klassen anders aufgerufen werden (LZ 4).
- Die Schüler arbeiten im zweiten Stundenteil in Gruppen zusammen (Teamfähigkeit) (LZ 5, sozial).

7.5 Weg und Medienentscheidung

Phase I: Nach dem Organisatorischen erfolgt als Einstimmung auf die Stunde eine Wiederholung der Assoziation, als Grundlage für eine Klasse, die ein Objekt einer anderen Klasse benötigt. Hier wird insbesondere noch einmal an den theoretischen Ablauf des Erzeugens von Objekten erinnert. Dazu versuchen die Schüler sich vorzustellen, welche Schritte im Programm ablaufen, wenn ein Objekt erzeugt wird. Als Parallele dienen hier das benutzen/ anlegen einfacher Variablen.

Objekt der Klasse A braucht ein Objekt der Klasse B → A ruft Methode von B auf, die ein Objekt von B erzeugt → Methode erzeugt Objekt, welches nun zur Verfügung steht.

Dieser Teil erfolgt frontal im Gespräch. Die Schüler sitzen an ihren Plätzen. Der Algorithmus wird gemeinsam erarbeitet.

Phase II: Die Methode zur Erzeugung von Objekten erhält einen Namen: Konstruktor. Es wird noch einmal verdeutlicht, dass dies eine spezielle Methode ist. Der Konstruktor legt grundlegende Eigenschaften fest, die das Objekt später haben muss, ohne genaue Werte zu definieren.

Da je nach Anforderung bestimmte Werte von Eigenschaften schon bekannt sein können, wird kurz das Überladen von Konstruktoren erwähnt.

Die Phase erfolgt ebenfalls frontal. Der Inhalt wird vom Lehrer präsentiert. Die Schüler sitzen an ihren Plätzen.

Phase III: Es folgt die Erarbeitung der Syntax eines Konstruktors, sowie die Umsetzung des Aufrufs und des Erzeugen eines Objekts.

Zuerst wird die Syntax des Konstruktors geklärt. Es wird auf die Besonderheiten eingegangen. Hierzu zählen, dass der Name der gleiche wie der Name der Klasse ist und das der Konstruktor öffentlich deklariert wird (public). Da es sich um einen Defaultkonstruktor handelt, werden alle Membervariablen initialisiert. Alle Variablen werden auf Null gesetzt (Folie 1).

Im Anschluss schreiben die Schüler selbständig einen Konstruktor für die Klasse Tier (Übung 1).

Zum Aufruf wird auf den Stundenanfang zurückgegriffen und der Algorithmus jetzt ausprogrammiert. Es wird zuerst ein ADT der typgebenden Klasse angelegt. Im Anschluss wird der Konstruktor aufgerufen und somit ein Objekt erzeugt.

Phase III erfolgt im Gespräch, der Lehrer notiert die Ergebnisse an der Tafel. Die Konstruktorsyntax wird aus den allgemeinen Methoden hergeleitet. Das Anlegen der Variablen erfolgt analog dem Anlegen von Variablen in Klassen. Der Aufruf wird vom Unterrichtsanfang abgeleitet. Hier wird die spezielle Syntax vom Lehrer vorgegeben.

Die Schüler sitzen an ihren Plätzen.

Phase IV: Die Schüler sollen nun zu einem vorgegebenen Beispiel die Konstruktoren implementieren, sowie diese entsprechend aufrufen. Als Beispiel wird hier das Vermietungsprogramm benutzt. Die Schüler ergänzen im Programm den Konstruktor der Klasse Adresse und der Klasse Kunde sowie den Aufruf der Konstruktoren in der Klasse Vermietung (Methode neuKunde()).

Die Kontrolle der Beispiele erfolgt nach einer bestimmten Zeit. Eine Gruppe stellt ihr Ergebnis vor. Dieser Stundenteil erfolgt als Gruppenarbeit. Die Schüler arbeiten zu viert das Beispiel aus. Es wird am PC gearbeitet.

Ende der Stunde. ☺

7.6 Geplanter Unterrichtsverlauf

Zeit	**Stundenteil + didaktische Funktion**	**Unterrichtsstoff + Dosierung**	**Organisation**	**Methodische Hinweise**
5 min.	Begrüßung/ Organisatorisches	Anwesenheit Stundenziel	Frontal	
10 min	Phase I Phase II	Algorithmus erinnern (Anlegen/ Erzeugen) Theorie Konstruktor allgemein	Frontal / Gespräch	Folie 1
15 min	Phase III	Syntax Konstruktor Konstruktor schreiben Syntax Aufruf Aufruf schreiben	Frontal/ Gespräch	Folie 1 Übung 1 Folie 2 Folie 3
12 min	Phase IV	selbständiges Implementieren eines Konstruktors mit Aufruf Vergleich der Lösungen	Zu Zweit (Gruppe), PC Platz	Vermietungs-Beispiel
3 min	Schlussbemerkung, Thema nächste Stunde, Vorbereitung auf nächste Stunde		Frontal	

7.7 Anhang

7.7.1 Folie 1: allgemeine Syntax eines Konstruktors

Der Konstruktor

Der Konstruktor erzeugt Objekte einer Klasse. Dabei werden sowohl die Variablen als auch sämtliche Methoden, die in der Klasse vereinbart werden, für das Objekt angelegt.

- Der Konstruktor hat stets den gleichen Namen wie die Klasse.
- Ein Konstruktor liefert als Ergebnis stets einen Verweis auf das neu erzeugte Objekt an die Aufrufstelle.
- Da der Rückgabetyp nicht frei wählbar ist, wird er nicht geschrieben.
- Wenn eine Klasse mehrere Konstruktoren enthält, ist es nicht erlaubt, dass zwei Konstruktoren in der Anzahl und den Datentypen ihrer Parameter übereinstimmen.

Allgemeine Syntax:

```
public NameDerKlasse () {

        var1 = ... ;
        var2 = ... ;
...
}
```

7.7.2 *Folie 2: Aufruf des Konstruktors*

Aufruf des Konstruktors

Konstruktor der Klasse Quader:

```
public class Quader {
...
public Quader()
{
        px = 0;
        py = 0;
        pz = 0;         //linke untere Ecke

        pdx =0;
        pdy = 0;
        pdz =0;         //Länge zu einer zweiten Ecke
}
...
}
class QuaderDemo {

        public static void main(String argv []){

                Quader q;
                q = new Quader();

                q.setze(1,2,3,4,5,6);
                System.out.println( "Quaderpunkt 1:" + q.zeigePunkt1);
                System.out.println( "Volumen:" +q.zeigeVolumen);
        }
}
```

7.7.3 *Folie* 3: *allgemeine Syntax des Aufrufs von Konstruktoren*

Allgemeine Syntax des Aufrufs eines Konstruktors

Erzeugen eines ADT:

Klasse nameDesObjekts;

Bsp: Quader wuerfel2;

Erzeugen des Objekts:

nameDesObjekts = new Konstruktor;

Bsp: wuerfel2 = new Quader();

Kurzform:

Klasse nameDesObjekts = new Konstruktor;

Bsp.: Quader wuerfel2 = new Quader();

7.7.4 Übung 1: Konstruktor der Klasse Tier

```
public class Tier {

Int anzBeine, anzFluegel;
String fellFarbe, name, laut;

//Konstruktor

…

}
```

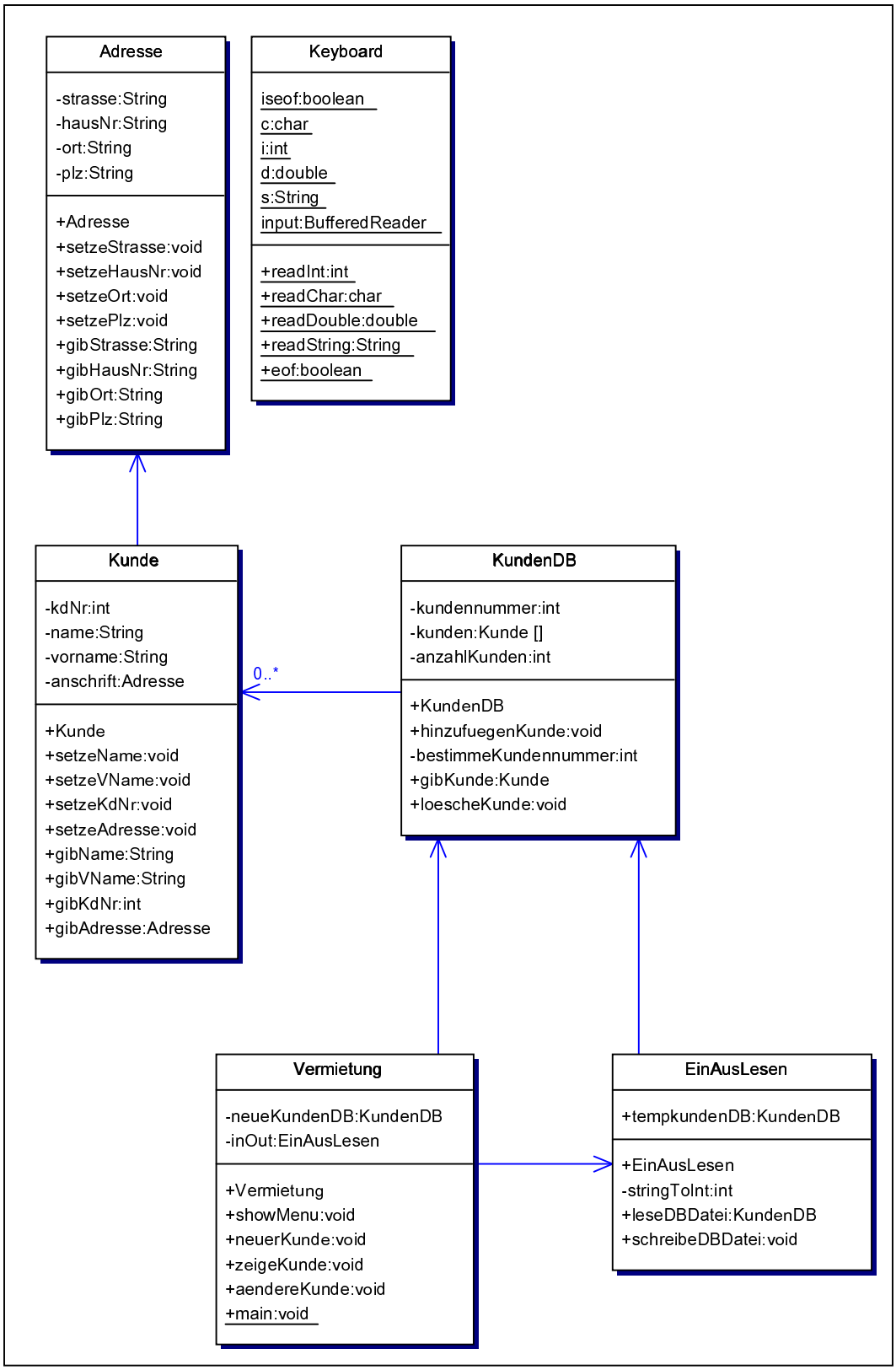
Schreiben Sie zur gegebenen Klasse Tier den Konstruktor. Der Konstruktor soll alle Variablen initialisieren, ohne direkte Werte zuzuweisen.

Lösung:

```
public class Tier {

Int anzBeine, anzFluegel;
String fellFarbe, name, laut;

public Tier ()
{
anzBeine = 0;
anzFluegel = 0;
fellFarbe = ’’;
name = ’’;
laut = ’’;
}

        …
}
```

Anhang

Das UML Modell des (vereinfachten) Fallbeispiels[5]



[5] Die Klassen EinAusLesen und Keyboard werden den Schülern gestellt.

Der Java Code des (vereinfachten) Fallbeispiels

```
public class Vermietung {

   private KundenDB                    neueKundenDB;
   private EinAusLesen         inOut = new EinAusLesen();


   //Konstruktor
    public Vermietung ()
   {
       neueKundenDB = new KundenDB();
   }


           //die "Benutzeroberfläche" anzeigen
           public void showMenu()
   {
                       int zahl;
                       neueKundenDB = inOut.leseDBDatei();

              do
      {
           System.out.println("------------M E N U-----------------");
           System.out.println();
           System.out.println("0: Ende");
           System.out.println("1: neuer Kunde");
           System.out.println("2: Loesche Kunden");
           System.out.println("3: Aendere Kundendaten");
           System.out.println("4: Kundendaten anzeigen");
           System.out.println();
           System.out.println();
           System.out.print("Auswahl: ");
           zahl = Keyboard.readInt();
           System.out.println("-----------------------------------");

           switch(zahl)
         {
           case 0:
              break;
           case 1:
              System.out.println("-----Neuer Kunde----");
              neuerKunde();
              break;
           case 2:
              System.out.println("-----Kunden loeschen----");
              System.out.print("Kundennummer eingeben: ");
               neueKundenDB.loescheKunde(Keyboard.readInt());
              System.out.println("Kunde geloescht");
             System.out.println("-----------------------------------");
              Keyboard.readChar();
               break;
           case 3:
              System.out.println("-----Kundendaten aendern----");
              System.out.print("Kundennummer eingeben: ");
             aendereKunde(Keyboard.readInt());
              break;
           case 4:
              System.out.println("-----Kundendaten anzeigen----");
              System.out.print("Kundennummer eingeben:  ");
               zeigeKunde( Keyboard.readInt());
               break;
           default: break;
                       }
           }
           while (zahl != 0);
```

```
    //speichern der Datenbank in .txt
    inOut.schreibeDBDatei(neueKundenDB);
          }

 //Kunden einlesen (aus .txt oder neuer Kunde) und in Kundendatenbank speichern
 public void neuerKunde()
 {
    Kunde person = new Kunde ();
    Adresse anschrift = new Adresse ();

    System.out.println();
    System.out.print("Nachname: ");
    person.setzeName(Keyboard.readString());
    System.out.print("Vorname: ");
    person.setzeVName(Keyboard.readString());
    System.out.print("Strasse: ");
    anschrift.setzeStrasse(Keyboard.readString());
    System.out.print("Hausnummer: ");
    anschrift.setzeHausNr(Keyboard.readString());
    System.out.print("PLZ: ");
    anschrift.setzePlz(Keyboard.readString());
    System.out.print("Ort: ");
    anschrift.setzeOrt(Keyboard.readString());
    System.out.println("-----------------------------------");

    person.setzeAdresse(anschrift);
    neueKundenDB.hinzufuegenKunde(person);

    System.out.println("Neuer Kunde wurden erfolgreich hinzugefuegt.");
    System.out.println("Kundennummer: " + person.gibKdNr() );
    Keyboard.readChar();
 }

 //alle Kundendaten eines Kunden anzeigen lassen
 public void zeigeKunde(int kdNr)
 {
    Kunde person;
    Adresse anschrift;

    person = neueKundenDB.gibKunde(kdNr);
    System.out.println();
    if (person != null)
    {
       System.out.println("Name : " + person.gibName());
       System.out.println("Vornamen : " + person.gibVName());

       anschrift = person.gibAdresse();
       System.out.println("Strasse : " + anschrift.gibStrasse());
       System.out.println("Hausnummer : " + anschrift.gibHausNr());
       System.out.println("Ort : " + anschrift.gibOrt());
       System.out.println("PLZ : " + anschrift.gibPlz());
    }
    else
       System.out.println ("ERROR: Keinen Kunden gefunden!");
       System.out.println("-----------------------------------");
       Keyboard.readChar();
 }

          //Daten eines Kunden aendern
          public void aendereKunde(int kdNr)
 {
    Kunde person;
    Adresse anschrift;
    String tempString;

    person = neueKundenDB.gibKunde(kdNr);

    System.out.println();
    if (person != null)
```

```
        {
                anschrift = person.gibAdresse();

                System.out.println();
                System.out.print("neuer Nachname [ " + person.gibName() + " ] : ");
                tempString = Keyboard.readString();
         if ( tempString != null) person.setzeName(tempString);
           System.out.print("neuer Vorname [ " + person.gibVName() + " ] : ");
           tempString = Keyboard.readString();
         if ( tempString != null) person.setzeVName(tempString);
           System.out.print("neuer Strasse [ " + anschrift.gibStrasse() + " ] : ");
           tempString = Keyboard.readString();
        if ( tempString != null) anschrift.setzeStrasse(tempString);
          System.out.print("neuer Hausnummer [ " + anschrift.gibHausNr() + " ] : ");
          tempString = Keyboard.readString();
        if ( tempString != null) anschrift.setzeHausNr(tempString);
          System.out.print("neuer Ort [ " + anschrift.gibOrt() + " ] : ");
          tempString = Keyboard.readString();
        if ( tempString != null) anschrift.setzeOrt(tempString);
           System.out.print("neue PLZ [ " + anschrift.gibPlz() + " ] : ");
           tempString = Keyboard.readString();
        if ( tempString != null) anschrift.setzePlz(tempString);

           person.setzeAdresse (anschrift);
           neueKundenDB.loescheKunde(kdNr);
           neueKundenDB.hinzufuegenKunde(person);
           System.out.println("Daten wurden erfolgreich geaendert.");
           System.out.println("-------------------------------------");
           Keyboard.readChar();
              }
        else
           System.out.println ("ERROR: Keinen Kunden gefunden!");
    }

    //main methode
            public static void main(String args[])
    {
       Vermietung    v;

       try
       {
            v = new Vermietung ();
            v.showMenu();
       }
       catch (Exception e)
       {
       }
    }
}
```

```
public class KundenDB {

   private int kundennummer;
   private Kunde [] kunden;
   private int anzahlKunden;

   //Konstruktor
   public KundenDB()
   {
      kunden = new Kunde [100];
      anzahlKunden = 0;
   };

   //hinzufügenKunde() erhält komplettes Objekt Kunde und ergänzt - wenn nötig - noch die Kundennummer
   public void hinzufuegenKunde(Kunde kd)
   {

      anzahlKunden++;

     //Test ob noch Platz in Datenbank
     if (anzahlKunden >= kunden.length)
     {
        System.out.println("Datenbank voll! Bitte erweitern!");
        return;
     }

     //Test ob KdNr vorhanden, wenn nicht bestimmen einer KdNr und ergänzen
                    kundennummer = kd.gibKdNr();
      if (kundennummer < 0)  //keine kdNr bekannt
      {
            kundennummer = bestimmeKundennummer();
            kd.setzeKdNr(kundennummer);
      }
      //Kunden an die Stelle seiner KdNr im Array der Kunden setzen
       kunden [kundennummer] = kd;
   }

   private int bestimmeKundennummer()
   {
      int zahl = -1;
      for(int i = 0; i < kunden.length && zahl < 0; i++)
      {
         if (kunden[i] == null || kunden[i].gibKdNr() == -1)
         {
            zahl = i;
         }
      }
      if (zahl < 0)
         zahl = 0;
      return zahl;
   }

   public Kunde gibKunde(int kundennummer)
   {
      return kunden [kundennummer] ;
   }

            public void loescheKunde(int kundennummer)
   {
      int zahl = -1;

      for(int i = 0; i < kunden.length && zahl < 0; i++)
      {
         if (kunden[i] != null && kunden[i].gibKdNr() == kundennummer)
         {
            zahl = i;
         }
      }
      if (zahl >= 0)
            kunden [zahl] = null;
   }
}
```

```
public class Kunde {

    private int kdNr;
    private String name;
    private String vorname;
    private Adresse anschrift;

    //default Konstruktor
    public Kunde(){
        this.kdNr = -1;
        this.name  = "";
        this.vorname = "";
    }

    public void setzeName(String name) {
        this.name = name;
    }

    public void setzeVName(String vorname) {
        this.vorname = vorname;
    }

    public void setzeKdNr(int kdNr){
        this.kdNr = kdNr;
    }

    public void setzeAdresse(Adresse adresse) {
        anschrift = adresse;
     }

    public String gibName() {
        return name;
    }

    public String gibVName() {
        return vorname;
    }

    public int gibKdNr(){
        return kdNr;
    }

    public Adresse gibAdresse() {
        return anschrift;
     }
}
```

```
public class Adresse {

    private String strasse;
    private String hausNr;
    private String ort;
    private String plz;

    //default Konstruktor
    public Adresse()
    {
        strasse = "";
        hausNr = "";
        ort = "";
        plz = "";
    }

    public void setzeStrasse(String strasse)
    {
        this.strasse = strasse;
    }

    public void setzeHausNr(String nummer)
    {
        hausNr = nummer;
    }

    public void setzeOrt(String ort)
    {
        this.ort =ort;
    }

    public void setzePlz(String plz)
    {
        this.plz = plz;
    }

    public String gibStrasse()
    {
        return strasse;
    }

    public String gibHausNr()
    {
        return hausNr;
    }

    public String gibOrt()
    {
        return ort;
    }

    public String gibPlz()
    {
        return plz;
    }

}
```

```
import java.io.*;

public class EinAusLesen {

   public KundenDB tempkundenDB;

          public EinAusLesen()
   {
                    tempkundenDB = new KundenDB();
   }

          private int stringToInt(String str)
   {
                    int         startIndex = 0;
                    boolean   bIsNegative = false;
                    int         val = 0;

                    if (str.charAt(0) == '-'  )
                    {
                              bIsNegative = true;
                              startIndex = 1;
                    }
                    else
        if (str.charAt(0) == '  +'  )
                                        {
                                        startIndex = 1;
                                        }

                    for (int i = startIndex; i < str.length(); i++)
                    {
                              if (str.charAt(i) >= '  0'   && str.charAt(i) <= '  9'  )
                              {
                                        val *= 10;
                                        val += str.charAt(i) - '  0'  ;
                              }
                              else
                                        break;
                    }
                    if (bIsNegative)
                    val *= -1;

                    return val;
          }


   public KundenDB leseDBDatei ()
   {
      BufferedReader     f;
      int                                 index;
      String                                        zeile;

      try
      {
          f = new BufferedReader(new FileReader("xxx.txt"));
        while (f.ready())
        {
          try
          {
                    zeile = f.readLine();
                                                  if (zeile != null && zeile.length() > 0)
                                                  {
                                                            String      wort;
                                                            Kunde person = new Kunde ();
                                                            Adresse anschrift = new Adresse ();

                index = zeile.indexOf ('\t'  );
                if (index >= 0)
                {
                   int kdNr;
```

```
                    kdNr = stringToInt(zeile.substring(0, index));
                    person.setzeKdNr (kdNr);
                    zeile = zeile.substring (index + 1);
                }
                index = zeile.indexOf ('\t'  );
                if (index >= 0)
                {
                    person.setzeName (zeile.substring(0, index));
                    zeile = zeile.substring (index + 1);
                }
                index = zeile.indexOf ('\t'  );
                if (index >= 0)
                {
                    person.setzeVName (zeile.substring(0, index));
                    zeile = zeile.substring (index + 1);
                }
                index = zeile.indexOf ('\t'  );
                if (index >= 0)
                {
                    anschrift.setzeStrasse (zeile.substring(0, index));
                    zeile = zeile.substring (index + 1);
                }
                index = zeile.indexOf ('\t'  );
                if (index >= 0)
                {
                    anschrift.setzeHausNr (zeile.substring(0, index));
                    zeile = zeile.substring (index + 1);
                }
                index = zeile.indexOf ('\t'  );
                if (index >= 0)
                {
                    anschrift.setzePlz (zeile.substring(0, index));
                    zeile = zeile.substring (index + 1);
                }
                index = zeile.indexOf ('\t'  );
                if (index >= 0)
                {
                    anschrift.setzeOrt (zeile.substring(0, index));
                    zeile = zeile.substring (index + 1);
                }
                person.setzeAdresse (anschrift);
                tempkundenDB.hinzufuegenKunde(person);
                }
            }
            catch (IOException e)
            {
                break;
            }
        }
        f.close();
    }
    catch (IOException e)
    {
    }
    return tempkundenDB;
}


        public void schreibeDBDatei (KundenDB tempkundenDB_2)
        {
                BufferedWriter      f;
                int         index;
                Kunde       person;
                Adresse     anschrift;

                try
                {
                        f = new BufferedWriter(new FileWriter("xxx.txt"));
```

```
                    for (int i = 0; i < 100; i++)
                    {
                        person = tempkundenDB_2.gibKunde(i);

                        if (person != null)
                        {
                            anschrift = person.gibAdresse ();
                            f.write(String.valueOf(person.gibKdNr()));
                            f.write('\t'  );
                            f.write(person.gibName ());
                            f.write('\t'  );
                            f.write(person.gibVName ());
                            f.write('\t'  );
                            f.write(anschrift.gibStrasse ());
                            f.write('\t'  );
                            f.write(anschrift.gibHausNr ());
                            f.write('\t'  );
                            f.write(anschrift.gibPlz ());
                            f.write('\t'  );
                            f.write(anschrift.gibOrt ());
                            f.write('\t'  );
                            f.newLine();
                        }
                        else
                        {
                        f.newLine();
                        }
                    }
                    f.flush();
                    f.close();
                }
                catch (IOException e)
                {
                }
        }


}
```

```
import java.io.*;

class Keyboard {
// Primitive Keyboard input of integers, reals,
// strings, and characters.

static boolean iseof = false;
static char c;
static int i;
static double d;
static String s;

static BufferedReader input
      = new BufferedReader (
            new InputStreamReader(System.in),1);

 public static int readInt () {
  if (iseof) return 0;
  System.out.flush();
  try {
    s = input.readLine();
  }
  catch (IOException e) {
    System.exit(-1);
  }
  if (s==null) {
    iseof=true;
    return 0;
  }
  i = new Integer(s.trim()).intValue();
  return i;
 }

 public static char readChar () {
  if (iseof) return (char)0;
  System.out.flush();
  try {
    i = input.read();
  }
  catch (IOException e) {
    System.exit(-1);
  }
  if (i == -1) {
    iseof=true;
    return (char)0;
  }
  return (char)i;
 }

 public static double readDouble () {
  if (iseof) return 0.0;
  System.out.flush();
  try {
    s = input.readLine();
  }
  catch (IOException e) {
    System.exit(-1);
  }
  if (s==null) {
    iseof=true;
    return 0.0;
  }
  d = new Double(s.trim()).doubleValue();
  return d;
 }

 public static String readString () {
  if (iseof) return null;
  System.out.flush();
```

```
    try {
      s=input.readLine();
    }
    catch (IOException e) {
      System.exit(-1);
    }
    if (s==null) {
      iseof=true;
      return null;
    }
    return s;
  }

  public static boolean eof () {
    return iseof;
  }

}
```

Literaturverzeichnis

Booch, Jacobsen, Rumbaugh: *Das UML Benutzerhandbuch.* Bonn: Addison Wesley, 1999.

Echtle, Goedicke: *Lehrbuch der Programmierung mit Java.* Heidelberg: dpunkt.verlag, 2000.

Hubwieser, Peter: *Didaktik der Informatik. Grundlagen, Konzepte, Beispiele.* Berlin, Heidelberg, New York: Springer Verlag, 2000.

Oesterreich, Bernd: *Objektorientierte Softwareentwicklung. Analyse und Design mit der Unified Modelling Language.* München: Oldenbourg Verlag, 1999.

Rumbaugh, Blaha, Premerlani, Eddy, Lorensen: *Objektorientiertes Modellieren und Entwerfen.* Bonn: Hanser Verlag, 1993.