Schriftliche Hausarbeit zur Abschlussprüfung der erweiterten Studien für Lehrer im Fach Informatik

VIII. Weiterbildungskurs in Zusammenarbeit mit der Fernuniversität Hagen

Eingereicht dem Amt für Lehrerfortbildung –Außenstelle Gießen – Vorsitzender des Prüfungsausschusses: G. Jungermann

Thema:

Einführung in die objektorientierte Modellierung mit UML

Verfasser: StR Werner Steffens

Gutachter: StD Gerhard Röhner

Inhaltsverzeichnis

Einleitung

„Das primäre Produkt eines Projektteams sind nicht 'schöne Dokumente', Konferenzen, gute Werbung oder preisverdächtige Source-Code-Längen, sondern gute Software, die den Anforderungen des Benutzers vollständig gerecht wird. Alles andere ist sekundär!"[1] Bei dieser Aussage ist mit „sekundär" aber nicht „irrelevant" gemeint. Denn um qualitativ hochwertige Software entwickeln zu können, benötigt man gute Programmierer, geeignete Entwicklungstools und vor allem einen definierten Weg, also ein Prozessmodell, nach dem Software gut, schnell und effektiv entwickelt werden kann.

Als geeignetes Prozessmodell zur Entwicklung von Software hat sich in den letzten Jahren die objektorientierte Modellierung mit Hilfe der Unified Modeling Language herauskristallisiert. Dies gilt nicht nur für die Praxis, sondern auch für den Informatikunterricht. Die Zeit eines hardware- und programmiersprachenorientierten Informatikunterrichts ist schon lange vorbei.[2] Methoden, Prinzipien und Konzepte sind heute von zentraler Bedeutung. Dies lässt sich eindeutig an den einheitlichen Prüfungsanforderungen für das Fach Informatik in der gymnasialen Oberstufe erkennen. Sie fordern „grundlegende Modellierungstechniken" als einen von drei verbindlichen Lern- und Prüfungsbereichen sowohl im Grund- als auch im Leistungskursfach. Vermittelt werden soll das Grundprinzip des Modellierens als zielgerichtetes Vereinfachen und strukturiertes Darstellen von Ausschnitten der Wirklichkeit, das Erstellen eines Modells auf der Grundlage der Problemanalyse, Kenntnisse der objektorientierten Modellierung, hier insbesondere: Objekt, Klasse, Beziehungen zwischen Klassen, Interaktion von Objekten, Klassendiagramm (z. B. mit UML). Mit diesen Themen soll der Informatikunterricht übergeordnete Kompetenzen weiterentwickeln und Arbeitsweisen und Methoden bereitstellen, die im Alltag, in Studium und Beruf sowie in Wissenschaft und Wirtschaft erforderlich und von Nutzen sind. [3]

Sucht man Unterlagen für die Erarbeitung und Umsetzung der Thematik in der gymnasialen Oberstufe, so hat sich im Kreis meiner Kollegen herausgestellt, dass das Ergebnis der Recherche meist unbefriedigend war. Veröffentlichungen sind eher für die universitäre Ausbildung gedacht. Diese Tatsache war der entscheidende Grund dafür, mich in der vorliegenden Hausarbeit mit einer für die gymnasiale Oberstufe gemäßen Einführung in die objektorientierte Modellierung mit UML zu befassen.

Die Examensarbeit gliedert sich in drei Teile: Im ersten Teil findet eine Analyse der Ausgangssituation statt. In ihr werden die Leistungsvoraussetzungen, materiellen Rahmenbedingungen sowie die Vorga-

[1] Buchberger, Michael: S. 5

[2] Hubwieser, Peter S. 68

[3] vgl. KMK, 2004, S. 3-5

ben durch den Lehrplan analysiert. Auf der Grundlage dieser Erkenntnisse wird anschließend die Zielsetzung der Examensarbeit definiert. Im zweiten Teil der Arbeit erfolgt eine didaktische Analyse der Konzepte und Notationen der objektorientierten Modellierung. Sie beinhaltet die verbindlich zu vermittelnden Themen sowohl im beruflichen als auch im allgemeinbildenden Gymnasium. Lehrerinnen und Lehrern soll dieser Teil als Grundlage dienen, sich in die Thematik der objektorientierten Modellierung einzuarbeiten. Anschließend erfolgt im dritten Teil die Übertragung dieser Lerninhalte in eine Unterrichtsreihe für die gymnasiale Oberstufe 12 I. Hier werden mögliche Stundenverläufe mit Verlaufsplanung, Tafelbild, Medieneinsatz, Arbeitsblättern etc. vorgestellt. Dieser Teil der Arbeit soll Lehrerinnen und Lehrern Anregungen geben, eine Unterrichtreihe zur objektorientierten Modellierung zu gestalten. Die Unterrichtreihe ist für drei Unterrichtsstunden pro Woche konzipiert. Somit besteht die Möglichkeit, noch ausstehende Themen des Lehrplans in die Unterrichtsreihe aufzunehmen. Diese Aufteilung war erforderlich, da nicht alle verbindlichen Inhalte des beruflichen mit denen des allgemeinbildenden Gymnasiums übereinstimmen. Die Unterrichtsreihe begleitet das Projekt „Wetterstation auf dem Felsberg“. Dies ermöglicht, die erworbenen Kenntnisse an einem durchgehenden Beispiel anzuwenden. Anhand dieses umfangreichen Projektes sollen sich die Schülerinnen und Schüler der Notwendigkeit der einzelnen Schritte der objektorientierten Modellierung bewusst werden. Die Aufgaben des Projektes lassen sich ohne die objektorientierte Modellierung nur unzureichend lösen. Für Lehrerinnen und Lehrer bietet das Projekt die Möglichkeit zusätzliche Inhalte zu vermitteln. Da ich an einem beruflichen Gymnasium unterrichte, habe ich weitere Inhalte wie die Datenkommunikation über das Ethernet, die parallele oder serielle Schnittstelle, die Datenhaltung in einer Datenbank oder in der Registry mit in das Projekt aufgenommen. In der Unterrichtreihe werden zu diesen Themen nur die für die Modellierung und anschließende Implementierung relevanten Aspekte angesprochen. Das entwickelte Programm ist eine Maximalversion für Lehrerinnen und Lehrer. Sie soll zeigen, welche Anforderungen an eine Entwicklung gestellt werden können.[4] In der Praxis gilt es, die Anforderung dem Leistungsvermögen der jeweiligen Klasse anzupassen.

[4] Das Programm wurde von drei Schülern am Ende der Jahrgangsstufe 13 entwickelt.

Teil I

1. Rahmenbedingungen

- **Leistungsvoraussetzungen**

Grundsätzlich bestehen zwei Möglichkeiten, in die Thematik der Programmierung einzusteigen. Man beginnt mit der Vermittlung der grundlegenden Konzepte der strukturierten Programmierung und führt dann - aufbauend auf den o. g. Erkenntnissen - in die objektorientierte Programmierung ein. Oder man beginnt gleich mit der objektorientierten Programmierung.[5]

Die Examensarbeit setzt voraus, dass den Schülerinnen und Schülern die grundlegenden Konzepte der strukturierten Programmierung bekannt sind. Dabei handelt es sich um Datentypen, Auswahlstrukturen, Schleifen, strukturierte Datentypen, Funktionen, Zeiger, Referenzen, dynamische Speicherverwaltung, Dateibearbeitung sowie Threads. Weiterhin sollten die Schülerinnen und Schüler in der Lage sein, mit dem Visual Studio 7.0 für C++ und der MSDN-Bibliothek (Microsoft Developer Network) zu arbeiten. Erste Erfahrungen bei der Erstellung von Fensterprogrammen mit den MFC-Assistenten[6] wären von Vorteil, sind aber nicht unbedingt notwendig.

- **Materielle Rahmenbedingungen**

Der Kurs sollte in einem Rechnerraum stattfinden, in dem jedem Schüler ein Rechnerarbeitsplatz zur Verfügung steht. Der Lehrerarbeitsplatz sollte mit Rechner, Beamer, Drucker und Overhead-Projektor ausgestattet sein.
Unabdingbar für den Unterricht ist die Möglichkeit, auf das Internet als Informationsquelle zugreifen zu können, da für die Thematik kein Schulbuch zur Verfügung steht.

Die Schüler sind bei der Programmierung teilweise auf Administratorenrechte angewiesen. Daher ist es von Vorteil, wenn die Rechner mit Wechselfestplatten ausgestattet sind. So ist gewährleistet, dass nach Beendigung des Unterrichts und mit Wechsel der Festplatte der Raum für andere Kurse funktionsfähig bleibt.

---

[5] vgl. Schuber/Schwill S. 195 ff.

[6] Microsoft Foundation Classes

Die Hardwareausstattung der Rechner sollte angemessen sein, um mit den Programmen Jumli[7] und Visuell Studio 7.0 C++ arbeiten zu können. Erfahrungen haben gezeigt, dass Rechner mit einer Taktfrequenz von 1000 MHz und einem Arbeitsspeicher von 512 MB ausreichen. Dagegen kann die Größe des Bildschirms nicht groß genug sein, um ein ökonomisches Arbeiten zu ermöglichen.

- **Vorgaben durch den Lehrplan**

Die Examensarbeit ist in erster Linie ausgerichtet auf die Leistungskurse „Technikwissenschaften“ an einem beruflichen Gymnasium sowie den Leistungs- und Grundkurs „Informatik“ an einem allgemeinbildenden Gymnasium.

Ebenfalls besteht die Möglichkeit, die Examensarbeit in der informationstechnischen Berufsausbildung an Beruflichen Schulen einzusetzen. Dabei bietet sich das Lernfeld „Anwendungsentwicklung“ im Ausbildungsberuf „Fachinformatiker“ mit der Fachrichtung „Anwendungsentwicklung“ an. Hierzu müsste allerdings noch eine Angleichung der Inhalte erfolgen.

Da die Examenarbeit in erster Linie für den Leistungskurs in der gymnasialen Oberstufe konzipiert ist, werden nur die Vorgaben des Lehrplans für das Fach „Informatik“ und die Vorgaben des Rahmenlehrplans für das Fach „Datenverarbeitungstechnik“ genauer betrachtet.

Der Rahmenlehrplan für das berufliche Gymnasium vom 6. Juni 1998 beinhaltet nicht die Thematik der objektorientierten Modellierung. Dies hat den Hintergrund, dass die objektorientierte Programmierung zum Zeitpunkt des Inkrafttretens des Rahmenlehrplans noch nicht die Bedeutung wie heute hatte. Im Zuge der Einführung der landesweiten Abiturprüfung - Beginn 2007 - wurde gerade mit einer Neufassung des Lehrplans begonnen. Um den Schülerinnen und Schülern einen innovativen Unterricht anbieten zu können, sind an den Schulen - auf der Grundlage des noch gültigen Rahmenlehrplans - aktuelle Themen der Informatik in den schulinternen Lehrplan aufgenommen worden. So ist z. B. an der Karl Kübel Schule in Bensheim die Objektorientierung ein fester Bestandteil des Leistungskurses 12 I „Hard- und Softwareaspekte der Datenkommunikation“. In diesem Leistungskurs sind aus dem Bereich der Programmierung folgende Inhalte zu vermitteln:

- Einführung in die Objektmodellierung
  - Objekt, Klasse, Attribut und Methode
  - Assoziation und Vererbung
  - Anwendungsfall-, Sequenz- und Klassendiagramm

[7] Jumli ist ein Freeware-Programm zur Erstellung von UML-Diagrammen. Die Version 1.4 befindet sich auf der beiliegenden CD-ROM.

- Drei-Schichten-Modell
- Gestaltung von Oberflächen mit MFC
- Programmierung der Kommunikation über
  - die parallele und serielle Schnittstelle
  - TCP-Socket-Programmierung
- Datenhaltung in einer Datenbank.

Hinzu kommen die Inhalte der Datenkommunikation:

- ISO/OSI-Schichtenmodell
- Übertragungsprotokolle der Schicht 1 und Schicht 2
  - Leitungscodierung
  - Fehlersicherungsverfahren
  - Übertragungsmethoden, -parameter und -medien
  - Zugriffsverfahren
- Übertragungsprotokolle der Vermittlungs-, Transport- und Anwendungsschicht.

Einfacher lässt sich die Thematik dem Lehrplan des allgemeinbildenden Gymnasiums zuordnen. Hier gibt es bereits im Leistungskurs „Informatik“ in der Jahrgangsstufe 12 I der Kurs „Objektorientierte Modellierung“. Dazu gehören die verbindlichen Inhalte Objektmodell, Klassen, abstrakte Datentypen, Standardalgorithmen, effiziente Algorithmen, Komplexität von Algorithmen und Grundkonzepte des Software-Engineerings. Hinzu kommen die fakultativen Unterrichtsinhalte abstrakter Datentyp, Graphen, Internetprogrammierung, heuristische Verfahren, Polymorphie und Suche in Texten.[8]

2. Zielbestimmung

Ziele der Examensarbeit:

- Sie soll Lehrerinnen und Lehrern als Grundlage dienen, sich in die Thematik der objektorientierten Modellierung einzuarbeiten.
- Die vorgestellte Unterrichtsreihe soll Anregungen geben, den Leistungskurs der Jahrgangsstufe 12 I zu gestalten.
- Das durchgeführte Programmierbeispiel soll zeigen, welche zusätzlichen Unterrichtsinhalte in die Unterrichtsreihe mit einbezogen werden können.

[8] vgl. HESSISCHES KULTUSMINISTERIUM: Lehrplan Informatik

Teil II

3. Prozessmodelle

Generelles Ziel der Softwareentwicklung ist die Erstellung eines Programms bzw. Programmsystems zur Lösung eines gegebenen Problems. Softwareentwicklung kann dabei als ein Prozess angesehen werden, bei dem die Elemente des Problem- bzw. Anwendungsbereichs durch Elemente des Programms abgebildet werden. Um den organisatorischen Rahmen darstellen zu können, entstanden im Laufe der Zeit unterschiedliche Prozess-Modelle[9]. Dabei versteht man unter einem Modell eine abstrakte Repräsentation einer Spezifikation, eines Designs oder eines Systems aus einem bestimmten Blickwinkel. Es wird meist bildlich in Form von Abbildungen oder Diagrammen dargestellt.

Historisch betrachtet begann die Entwicklung von Software in den 50er Jahren in einem sehr überschaubaren Rahmen. Anfänglich wurden die meisten Programme von Personen entwickelt, die gleichzeitig auch die Anwender waren. Hinzu kam, dass Aufgrund der damaligen Leistungsfähigkeit der Rechner die Programme nicht sehr umfangreich waren und in überschaubarer Zeit entwickelt werden konnten. In dieser Zeit fand die Softwareentwicklung in zwei Stufen statt: „Schreibe ein Programm“ und „Finde und behebe die Fehler im Programm“. Da das Finden und Beheben der Fehler immer aufwendiger wurde, reifte schon sehr früh die Erkenntnis, dass eine Entwurfsphase der Implementierung vorangestellt werden muss. Außerdem kam der Anwender der Software häufig erst mit einer vorläufigen Endversion in Kontakt; Missverständnisse und Änderungswünsche waren nur mit großem Aufwand möglich. Mit der Zunahme der Leistungsfähigkeit der Rechner in den 60er Jahren wurden die zu entwickelnden Programme immer umfangreicher und aufgrund ihrer gewachsenen Komplexität mussten sie nun in Teams entwickelt werden, was eine Aufteilung der Arbeiten auf mehrere Personen erforderlich machte. Da Softwareentwicklung bis dahin aber immer monolithisch betrachtet worden war, fehlten die sauber strukturierten Teilaufgaben, die man auf einzelne Teammitglieder hätte übertragen können. Um aber die Entwicklung von Software plan- und kontrollierbar zu machen, strebte man nach einer Strukturierung von Softwareprojekten. Ziel war die Aufteilung des Projektes in wohl definierte Teilaktivitäten, die mit vorgegebenen Methoden bearbeitet werden sollten.
Die erste Generation von Modellen zur Softwareentwicklung entstand daraufhin zu Beginn der 70er Jahre. Die Entwicklung vom Problem bis zur Software-Lösung erfolgte nicht mehr in einem Schritt, sondern wurde in verschiedene Stufen unterteilt.

[9] In der Literatur wird auch der Begriff „Vorgehensmodelle“ verwendet.

Ein weit verbreitetes Prozessmodell dieser Generation ist das Wasserfallmodell von Boehm. Hierbei durchläuft der Entwicklungsprozess die Stufen „Analyse“, „Definition“, „Entwurf“, „Implementierung“, „Test“ und „Einsatz/Wartung“.[10] Die zeitliche und logische Abfolge der verschiedenen Phasen im Entwicklungsprozess war ursprünglich rein sequentiell und voneinander getrennt.[11] So ist z. B. die Struktur des Analysemodells in der Regel nicht im Entwurfsmodell wiederzufinden, da beide Modelle andere Modellierungskomponenten benutzen. Aufgabe der Analyse- und Definitionsphase ist es, die spezifischen Produktanforderungen zu ermitteln. Als erstes ist zunächst eine Ist-Analyse zu erstellen, auf deren Basis dann eine Schwachstellenermittlung durchgeführt werden kann. Aufbauend auf diesen Erkenntnissen schließt sich eine Zieldefinition an, die schließlich zu einer Definition der Produktanforderungen führt. Das Ergebnis dieser Phase ist die Produktdefinition, die auch als „Anforderungsdefinition“ oder „Pflichtenheft“ bezeichnet wird. In der Entwurfsphase wird dann auf der Basis der Produktdefinition die innere Struktur des Softwaresystems entwickelt. Dazu wird das Gesamtsystem schrittweise in Komponenten zerlegt. Das erste Zwischenergebnis des Entwurfs ist die „Software-Architektur“. Anschließend erfolgt eine genauere Spezifikation der Aufgaben der Komponenten und ihres Zusammenwirkens. Es geht dabei allerdings noch nicht um die Implementierung, sondern um eine klare Definition der Komponenten. In der Phase der Implementierung erfolgt dann die Umsetzung der Komponenten in eine Programmiersprache. Das Ergebnis dieser Phase sind eine Menge dokumentierter und getesteter Komponenten. Da in der Implementierung die einzelnen Komponenten bereits getestet wurden, richtet sich das Augenmerk in der Testphase auf die Integration der Komponenten und damit auf den Integrationstest, der im Allgemeinen sukzessiv durchgeführt wird. Nachdem alle Komponenten in einem Systemtest zusammen getestet wurden, erfolgt die Installation des Systems im Produktionsumfeld und der Abnahmetest. Die Grenzen zwischen dem System- und Abnahmetest sind fließend. Zum Abschluss erfolgt der Einsatz des Systems und damit die Beendigung des eigentlichen Entwicklungsprozesses. Optional kann die Phase der Wartung erfolgen, in der die Korrektur von Fehlern, die Anpassung an andere Systemumgebungen oder Änderungen und Erweiterungen der Funktionalität durchgeführt werden.

In der Praxis ist es allerdings erforderlich, frühere Entscheidungen aufgrund neuer Erkenntnisse zu revidieren oder Anforderungen für die Spezifikation zu ändern, wenn sich herausstellt, dass dem Kunden das implementierte System nicht gefällt. Daher werden in der Praxis Modelle benötigt, die einen iterativen Prozess ermöglichen. Das Spiralmodell von Boehm ging auf diese Forderung ein. Es beinhaltet „Sollbruchstellen“, die die Möglichkeit bieten, aus dem ursprünglichen angestrebten Vorgehen bei Bedarf auch ausbrechen zu können. Somit kann das Spiralmodell als ein Metamodell betrachtet wer-

[10] Für die einzelnen Stufen werden in der Literatur synonyme Bezeichnungen verwendet.

[11] vgl. Stevens, Perdita/ Pooly, Rob, S. 71

den, das für die einzelnen Phasen der Softwareentwicklung wiederum einen allgemeinen Ablauf definiert. Ausgehend vom Zentrum durchläuft eine Softwareentwicklung, die diesem Prozess folgt, hintereinander die Stufen „Risikoanalyse und Planung", „Anforderungsanalyse", „Engineering und Evaluierung".[12] Die Anzahl der Iterationen ist dabei beliebig. In den einzelnen Phasen finden andere Prozessmodelle wieder ihre Anwendung. So kann z. B. die Phase des Engineering im Spiralmodell wiederum die Stufen des Wasserfallmodells „Design", „Implementierung" und „Test" enthalten.

Mit der steigenden Akzeptanz der objektorientierten Programmiersprachen drängte sich das objektorientierte Modell immer mehr in den Vordergrund. Das objektorientierte Modell zeichnet sich durch einen durchgängigen Entwicklungsprozess von der Analyse bis zur Wartung aus. Ziel der angewandten Methoden ist es, die Struktur des Problembereichs möglichst genau auf die Implementierung abzubilden. Programmsysteme werden nicht mehr, wie beim strukturierten Ansatz, dadurch entwickelt, dass Funktionskomplexe in Prozeduren und Module zerlegt werden, sondern indem der Systemkern durch Abstraktionen der Realität gebildet wird. Je nach Entwickler[13] variieren die Anzahl sowie die Bezeichnungen der einzelnen Phasen. In fast allen Modellen[14] existieren die Phasen der „Objektorientierten Analyse (OOA)", des „Objektorientierten Entwurfs / Designs (OOD)" und der „Objektorientierten Implementierung/ Programmierung (OOP)".[15] Weiterhin haben alle Modelle gemeinsam, dass die Grenzen zwischen den Phasen fließend sind. So werden Aktivitäten, die in einigen Modellen noch zur Analyse zählen, in anderen schon dem Entwurf zugeordnet.

Die ersten Modelle zum Entwicklungsprozess in der objektorientierten Programmierung waren die Object Modeling Technique (OMT), das Object-Oriented Design (OOD) und das Object-Oriented Software Engineering (OOSE). Die OMT wurde unter der Leitung von Rumbaugh in einer Arbeitsgruppe von General Electric entwickelt und ist deutlich an traditionellen Modellen wie dem ER-Modell und Datenflussdiagrammen orientiert. Das OOD von Booch ist im Unterschied dazu durch eine ausgeprägtere Berücksichtigung objektorientierter Konzepte gekennzeichnet. Das in Schweden bei Ericsson entstandene OOSE unter der Leitung von Jacobson bietet mit sog. Anwendungsszenarien („use cases") ein Konzept, das eine durchgängige Beschreibung eines Systems aus der Sicht der Nutzer unterstützt. Im Folgenden wurde ein „Krieg" der Modelle erwartet, ähnlich dem „Krieg" der Programmiersprachen.[16] Aber genau das Gegenteil trat ein. Rumbaugh und Booch führten ihre rudimentären Sprach-

[12] Für die einzelnen Stufen werden in der Literatur synonyme Bezeichnungen verwendet.

[13] Shlaer/Mellor, Coad/Yourdon, Booch, Rumbaugh oder Jacobson

[14] In der Literatur wird auch an dieser Stelle von „Methoden" gesprochen.

[15] http://www-is.informatik.uni-oldenburg.de/~dibo/paper/gi98mm/node2.html

[16] vgl. Stevens, Perdita/ Pooly, Rob, S. 73

entwürfe zu einer neuen, gemeinsamen Modellierungssprache Unified Method zusammen und entwickelten diese weiter. Kurze Zeit später schloss sich noch Jacobson den beiden an, so dass das von ihm geprägte Use-Case-Diagramm (Anwendungsfalldiagramm) integriert wurde. Da ihre Modellierungssprache sehr bekannt war, galt sie als Quasi-Standard. Nach einer Reihe von Überarbeitungen entstand 1998 aus ihr die Unified Modeling Language (UML), die von der OMG als Standard dann verabschiedet wurde. „UML selbst beschreibt kein einheitliches Vorgehen, sondern gibt nur ein Werkzeug in die Hand (Modellierungssprache).“[17] Somit stellt UML keine Methode, sondern nur eine Notation zur Verfügung. Eine Methode hat den Anspruch, mehrere Prozesse bzw. Prozessabschnitte zu beschreiben und festzulegen, eine Notation beschreibt nur die Darstellung. UML bedient sich dabei vier verschiedener Sichten bzw. Abstraktionen, die in Diagrammen dargestellt werden:[18]

Systemnutzung:

- Anwendungsfalldiagramm - Use-Case Diagramm
  - Welche Fälle der Systemnutzung gibt es?
  - Wie stellen sich in jedem Fall die Anforderungen an die Systemfunktionalität aus der Sicht des Akteurs dar?
  - Welche Dienste erwartet der Akteur von den verfügbaren Objekten?

Statische Sicht:

- Klassenstrukturdiagramm
  - Welche Klassen sind für die Abbildung des Gegenstandsbereichs geeignet?
  - Welche Attribute und Methoden weisen die Klassen auf?
  - Welche Beziehungen gibt es zwischen den Klassen bzw. Objekten?
- Objektdiagramm
  - Welche Werte besitzen die Attribute zu einem bestimmten Zeitpunkt?

Interaktionssicht:

- Sequenzdiagramm
  - Welche Nachrichten schicken sich Objekte verschiedener Klassen zu?
  - In welcher zeitlichen Reihenfolge werden die Nachrichten ausgetauscht?

---

[17] Rogetzer, Klaus, Moderne Analyse-Methoden. www.infrasoft.at

[18] vgl. Buchberger, S. 18

- Zusammenspieldiagramm
  - Wie können Objekte, die untereinander Nachrichten austauschen (die also „zusammenarbeiten"), sich gegenseitig identifizieren?
- Zustandsdiagramm
  - Welche wesentlichen Ereignisse sind für Objekte einer Klasse zu beachten?
  - Wie sollen die Objekte dieser Klasse auf die Ereignisse reagieren?
  - In welcher Weise ändert sich beim Auftreten eines Ereignisses ggf. der Zustand eines Objekts?
- Aktivitätsdiagramm
  - Welche Prozesse sind für das zu erstellende System von Bedeutung?
  - Wie lassen sich diese Prozesse als zeitliche Folge von Aktivitäten beschreiben?

Implementierungssicht:

- Komponentendiagramm
  - Welche Komponenten (vorhandene Anwendungen, implementierte Klassen und Klassengruppierungen) sind für die Systemarchitektur wesentlich?
  - Welche Beziehungen / Abhängigkeiten gibt es zwischen den Komponenten?
  - Welche Schnittstellen bieten die Komponenten und das System?
- Verteilungsdiagramm
  - Welche Systemressourcen (Computer, Massenspeicher, periphere Geräte) werden von den Komponenten benötigt?

Die Arbeit mit UML-Diagrammen, sie zu zeichnen und konsistent zu halten, kann durch ein CASE-Tool[19] vereinfacht werden. Für kleinere Systeme genügt allerdings oftmals bereits ein Stück Papier. Bei der Erstellung ist zu beachten, dass die Diagramme nicht zu komplex angelegt werden. „Ein Diagramm, das zu komplex ist, um es mit der Hand zu zeichnen, ist vermutlich auch zu komplex, um es geistig klar zu erfassen. In einem solchen Fall sollte es aufgeteilt oder auf einem höheren Abstraktionsniveau gezeichnet werden oder sogar beides."[20]

[19] Computer Aided Software Engineering

[20] Balzert, Heide S. 124

4. Die objektorientierte Analyse

„Das Ziel der objektorientierten Analyse ist es, die Wünsche und Anforderungen eines Auftraggebers an ein neues Softwaresystem zu ermitteln und zu beschreiben.“[21] Produkte der Analysephase sind das Pflichtenheft, das objektorientierte Analyse-Modell, kurz OOA-Modell genannt und der Prototyp der Benutzeroberfläche.

Im „Pflichtenheft“ findet eine umgangssprachliche Beschreibung dessen statt, was das zu realisierende System leisten soll. Es ist weniger detailliert als das OOA-Modell und kann auch einige Informationen beinhalten, die nicht im OOA-Modell dargestellt werden. Das OOA-Modell beinhaltet ein konsistentes, vollständiges, eindeutiges und realisierbares Fachkonzept. In ihm werden Objekte der realen Welt durch Modellbildung und geeignete Abstraktion in Objekte des objektorientierten Modells transformiert. Alle Aspekte der Implementierung werden bewusst ausgeklammert. Man geht von einer perfekten Technik aus, d. h., dass der Prozessor jede Funktion ohne Verzögerung ausführen kann, keine Fehler macht oder gar ausfällt, dass die Speichermedien unendlich viele Informationen aufnehmen können und der Prozessor ohne Zeitverzögerung darauf zugreifen kann.

4.1. Das Pflichtenheft

Der schwierigste Teil in der Entwurfsphase eines jeden Projekts ist es, überhaupt zu verstehen, welche Anforderungen an das zu entwickelnde Softwaresystem gestellt werden. Die Auftragsanfrage ist meist ziemlich vage und ungenau, doch typisch für erste Anfragen im Frühstadium der Projektdefinition. Dies liegt oftmals daran, dass der Auftraggeber keine klar formulierten Vorstellungen hat. Es ist beispielsweise für ihn sehr schwierig, zu unterscheiden, was ein existierendes System macht und was ein geeignetes System können soll. Folglich ist es notwendig, bevor man ein Design entwerfen kann, dass eine viel genauere Untersuchung der eigentlichen Anforderungen der Benutzer erforderlich ist. Für diese Untersuchungen (Systemanalyse) nimmt man einen anwenderorientierten Standpunkt ein. Man identifiziert dabei die Anwender des Systems und die Aufgaben, die sie mit Hilfe des Systems lösen müssen. Weiterhin braucht man Informationen darüber, welche Aufgaben am wichtigsten sind, um die Entwicklung entsprechend planen zu können. Dieser Prozess kann mit Hilfe des Pflichtenheftes abgearbeitet und dokumentiert werden.

[21] Balzert,Heide, S. 8.

Das Pflichtenheft baut auf dem „Lastenheft“ auf, wenn dies erstellt wurde. Im Gegensatz zum Lastenheft muss das Pflichtenheft so abgefasst sein, dass es als Basis eines juristischen Vertrages dienen kann. „Die beschriebenen Anforderungen müssen realisierbar sein. Entwurfs- und Implementierungsentscheidungen sollen nicht vorweggenommen oder unnötig eingeschränkt werden.“[22]

Die Gliederung des Pflichtenhefts sollte so aufgebaut sein, dass es gut lesbar ist und eine leichte Einarbeitung erlaubt. Richtlinien werden von der ISO 9000 vorgegeben, wie z. B. ein grobes Gliederungsschema mit festgelegten Inhalten. Trotz dieser Vorgaben haben sich je nach Anwendungsgebiet unterschiedliche Gliederungen des Pflichtenhefts entwickelt.[23] Ein weit verbreitetes Gliederungsschema in der Softwareentwicklung ist die Einteilung von Balzert[24]. Sie enthält die Punkte „Zielbestimmung“, „Produkteinsatz“, „Produktübersicht“, „Produktfunktionen“, „Produktdaten“, „Produktleistungen“, „Qualitätsanforderungen“, „Benutzeroberfläche“, „nichtfunktionale Anforderungen“, „technische Produktumgebung“, „spezielle Anforderungen an die Entwicklungsumgebung“, „Gliederung der Teilumgebung“ und „Ergänzungen“ mit jeweiligen Unterpunkten. Für den Einsatz im Unterricht sind die aufgezählten Aspekte allerdings zu umfangreich. Daher werden im Folgenden nur Gliederungspunkte erläutert, die für Softwareprojekte im Informatikunterricht relevant sind.

1. Zielbestimmung: In ihr gilt es zu beschreiben, welche Vorhaben durch den Einsatz des Produkts realisiert werden sollen. Hierbei wird unterschieden zwischen Muss-, Wunsch- und Abgrenzungs-Kriterien. Dabei versteht man unter Muss-Kriterien Leistungen, die für das Produkt unabdingbar sind. Sie müssen am Ende des Projekts auf jeden Fall erfüllt sein. Wunsch-Kriterien dagegen beschreiben Wünsche, die am Ende des Projektes nicht unabdingbar sind, deren Erfüllung aber so weit wie möglich angestrebt werden sollte. Abgrenzungs-Kriterien dienen dazu, deutlich zu machen, welche Ziele mit dem Produkt bewusst nicht erreicht werden sollen.
2. Produkteinsatz: In diesem Abschnitt werden der Anwendungsbereich, z. B. Textverarbeitung im Büro, die Zielgruppen, hier: Sekretärinnen oder Schreibkräfte sowie die Betriebsbedingungen wie z. B. physikalische Umgebung des Systems definiert.
3. Produktübersicht: In Form eines Übersichtsdiagramms wie Funktionsbaum, Anwendungsfall- oder Datenflussdiagramm, wird eine grobe Übersicht über das Produkt gegeben. Das bedeutet, dass die typischen Arbeitsabläufe, die mit der zu erstellenden Software durchgeführt werden sollen, in Form eines Diagramms wiedergegeben werden.

---

[22] Balsert, Helmut S. 113

[23] vgl : Prof. Dr. Klüver, Vorlesungsskript Software Engineering, FHA Informatik

[24] vgl Balzert, Helmut, S. 111 ff..

4. Produktfunktionen: In Abhängigkeit von dem gewählten Übersichtsdiagramm im Punkt 3. erfolgt hier eine Konkretisierung und Detaillierung.
5. Produktdaten: Unter diesem Gesichtspunkt erfolgt die detaillierte Beschreibung der langfristig zu speichernden Daten, sowie die Bestimmung des Umfangs der Daten.
6. Produktleistungen: Sie geben Leistungsanforderungen bzgl. Zeit oder Genauigkeit an einzelne Funktionen und Daten. Dabei ist zu prüfen, ob die gewünschten Leistungen mit denen in Punkt 5. genannten Datenmengen und mit der im Punkt 9. beschriebenen technischen Produktumgebung erreicht werden können.

7. Benutzeroberfläche: In diesem Abschnitt werden grundlegende Anforderungen an die Benutzeroberfläche festgelegt, z. B. Fensterlayout, Dialogstruktur und Mausbedienung entsprechend dem Windows-Gestaltungs-Regelwerk (style guide) oder den unternehmenseigenen Gestaltungs-Regelwerken. Gibt es verschiedene Rollenträger (Administrator oder Benutzer), die das Softwaresystem benutzen, dann sind für jede Rolle die Zugriffsrechte aufzuführen.
8. Nichtfunktionale Anforderungen: In diesem Abschnitt werden alle Anforderungen aufgeführt, die sich nicht auf die Funktionalität, die Leistung und die Benutzeroberfläche beziehen wie z.B. einzuhaltende Gesetze und Normen, das Testat durch eine externe Prüfungsgesellschaft, Revisionsfähigkeit oder Sicherheitsanforderungen (Passwortschutz oder Mitlaufen von Protokollen) oder Plattformabhängigkeiten.
9. Technische Produktumgebung: Die technische Produktumgebung beinhaltet
    - die Softwaresysteme, die für den Betrieb zur Verfügung stehen müssen, wenn das Produkt nicht als stand-alone-Produkt geplant ist,
    - die Hardwarevoraussetzungen für den Betrieb,
    - die organisatorischen Voraussetzungen für den Betrieb.

4.2. Das OOA-Modell

Das OOA-Modell besteht aus zwei Säulen, nämlich dem statischen und dem dynamischen Modell. Das statische Modell verwendet zur Modellierung die Basiskonzepte „Objekt“, „Klasse“, „Methode“ und „Attribut“ sowie die statischen Konzepte „Assoziation“, „Vererbung“ und „Paket“. Mit Hilfe dieser Konzepte beschreibt es die Klassen des Softwaresystems, die Assoziationen zwischen den Klassen und die Vererbungsstrukturen. Das dynamische Modell beschreibt das Verhalten des zu entwickelnden Softwaresystems. Es benutzt dazu ebenfalls die Basiskonzepte „Objekt“, „Klasse“, „Methode“ und „Attribut“ sowie die dynamischen Konzepte „Anwendungsfall“, „Szenario“, „Botschaft“ und „Zustandsautomat“.

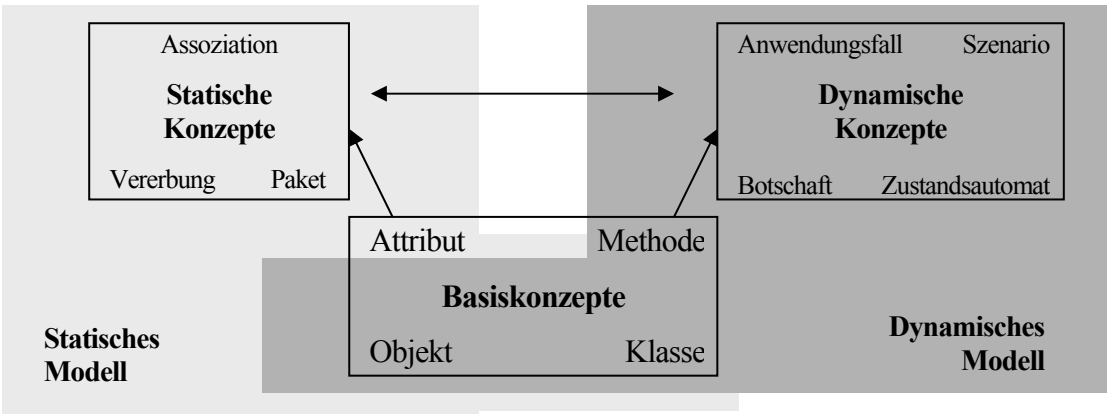


Abb. 1: Statisches und dynamisches Modell.[25]

4.2.1 Basiskonzepte

- Objekt

Ein Objekt ist gekennzeichnet durch einen bestimmten Zustand und ein definiertes Verhalten gegenüber seiner Umgebung. Es zeichnet sich gegenüber anderen Objekten durch seine Identität aus. Sie ist einmalig und kann sich nicht ändern. Der Zustand eines Objekts wird über seine Attribute bzw. deren aktuelle Werte und die jeweiligen Verbindungen zu anderen Objekten gekennzeichnet. Attribute sind inhärente, unveränderliche Eigenschaften des Objekts, während die Attributwerte Änderungen unterliegen können. Allerdings dürfen Attributwerte nicht beliebig verändert werden. Zustandsänderungen werden daher gewöhnlich über die von dem Objekt dafür vorgesehenen Operationen vorgenommen. Man unterscheidet in diesem Zusammenhang zwischen Modifikatoren (Set-Methoden), die den Zustand eines Objekts verändern, und Selektoren (Get-Methoden), die Attributwerte zurückliefern, ohne den Objektzustand zu verändern.

In der UML wird ein Objekt als Rechteck dargestellt. Im Rechteck selbst befindet sich die Bezeichnung des Objektes, die immer unterstrichen wird. Wenn der Objektname ausreicht, um das Objekt zu identifizieren und der Name der Klasse aus dem Kontext ersichtlich ist, kann dieser weggelassen werden. Ansonsten muss die Klasse und der Objektname angegeben werden. Bei einem anonymen Objekt wird nur der Klassenname angegeben. Möchte man zusätzlich die kontextrelevanten Attribute des Objekts mit angeben, wird das Rechteck geteilt und die Attribute werden im unteren Teil eingetragen. Hierbei gibt es die Möglichkeit Attribut, Typ und Wert oder nur Attribut und Wert anzugeben. Die Operationen, die ein Objekt ausführen kann, werden in der UML nicht angegeben.

[25] Balzert, Heide: S.10.

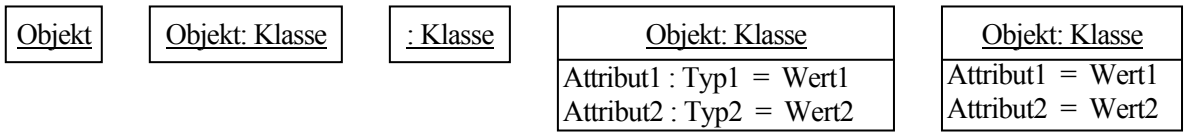


Abb. 2: Beispiele für die Bezeichnung von Objekten.

In der Praxis besteht ein Modell selten aus nur einem Objekt. Vielmehr handelt es sich um Objektgeflechte, in denen jedes Objekt seine Operationen anderen Objekten zur Verfügung stellt oder sich der Operationen anderer Objekte bedient. Gehen die Operationen nur in eine Richtung, spricht man von einer „unidirektionalen Verbindung". Dagegen werden Verbindungen mit wechselnder Aufgabenverteilung als „bidirektionale Verbindungen" bezeichnet. Die Darstellung von Objekten und deren Verbindungen untereinander werden im UML-Objektdiagramm spezifiziert. Objektdiagramme sind Momentaufnahmen und haben ihre Gültigkeit immer nur zu einem bestimmten Zeitpunkt.

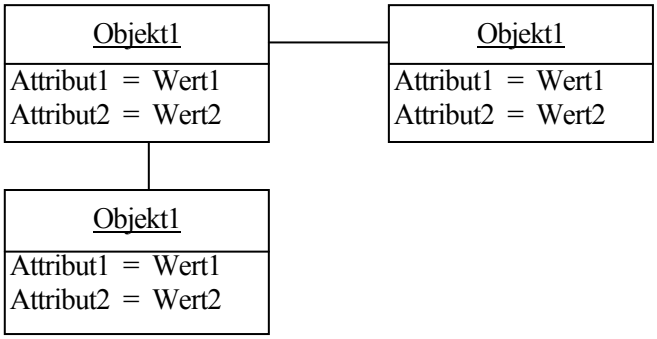


Abb. 3: Beispiele für ein Objektdiagramm.

- Klasse

Klassen sind die wichtigsten Bestandteile jedes objektorientierten Systems. Um ein Softwaresystem zu modellieren ist es nicht ausreichend, einzelne, konkrete Objekte zu identifizieren, sondern es wird ein aussagekräftigeres Modellierungskonzept benötigt; das Konzept der Klasse. Eine Klasse ist eine Beschreibung einer Menge von Objekten, die sich dieselben Attribute, Methoden, Beziehungen und Semantik teilen. Je nach Detaillierungsgrad unterscheidet man zwischen „instanzierbaren Klassen" und „abstrakten Klassen". Instanzierbare Klassen sind Klassen, deren Bauplan so detailliert ist, dass konkrete Objekte mit diesem Plan erzeugt werden können. Der Prozess, mit dem konkrete Objekte aus dem Bauplan erzeugt werden, heißt Instanzierung.

Jede Klasse muss einen Namen besitzen, der sie von anderen Klassen unterscheidet. Er kann aus beliebig vielen Buchstaben, Ziffern und bestimmten Interpunktionszeichen bestehen. Allerdings haben sich in der Praxis als Klassennamen kurze Nomen oder Nominalphrasen, die dem Vokabular des zu modellierenden Systems entnommen werden, durchgesetzt. Dabei ist es üblich, den Anfangsbuchstaben jedes

Wortes in einem Klassennamen als Großbuchstaben zu schreiben. In der UML wird eine Klasse ähnlich wie ein Objekt als Rechteck dargestellt. Der Klassenname wird immer fettgedruckt, zentriert in das Rechteck eingetragen. Da der Klassenname innerhalb eines Systems eindeutig sein muss, wird er bei Bedarf um den Paketnamen bzw. Bibliotheknamen erweitert.

Ein Attribut beschreibt die Daten des zu modellierenden Gegenstands, die allen Objekten der betreffenden Klasse gemeinsam sind. Dabei ist jedes Attribut von einem bestimmten Typ, wobei eine Klasse über beliebig viele oder auch gar keine Attribute verfügen kann. Objekte der Klasse besitzen immer die gleichen Attribute, jedoch sind die Attributwerte unterschiedlich. In der Praxis ist ein Attributname ein kurzes Nomen (oder eine Nominalphrase), das die Eigenschaft der Klasse repräsentieren, die das Attribut einschließt. Es ist üblich, den Anfangsbuchstaben jedes Wortes im Attribut als Großbuchstaben zu schreiben, mit Ausnahme des ersten Buchstabens. In der UML werden Attribute in einem eigenen Abschnitt direkt unterhalb des Klassennamens aufgelistet. Sie beginnen generell mit einem Kleinbuchstaben und müssen im Kontext der Klasse eindeutig sein. Da ein Attributname nur innerhalb der Klasse eindeutig ist, verwendet man außerhalb des Klassenkontextes die Bezeichnung „Klasse.Attribut“. Weiterhin kann optional der Datentyp, ein Anfangswert oder die Sichtbarkeit angegeben werden. Die Sichtbarkeit gibt an, ob von außen auf das Attribut zugegriffen werden darf.

Eine „Methode“ ist die Implementierung eines Dienstes, der von jedem Objekt der Klasse zur Verhaltensbeeinflussung abgerufen werden kann. Sie kann auf alle Attribute eines Objekts der Klasse direkt zugreifen und die Attributwerte lesen oder schreiben, Berechnungen ausführen und Methoden anderer Objekte zur Ausführung bringen. Dabei kann eine Klasse beliebig viele oder auch gar keine Methoden haben. In der UML werden alle Methoden in einem eigenen Abschnitt direkt unterhalb der Klassenattribute aufgelistet. Man kann Methoden in der Weise zeichnen, dass nur ihr Name gezeigt wird. In der Praxis ist der Name einer Methode ein kurzes Verb oder eine Verbalphrase. Es ist üblich, den Anfangsbuchstaben jedes Wortes in einem Methodennamen als Großbuchstaben zu schreiben, mit Ausnahme des ersten Buchstabens.

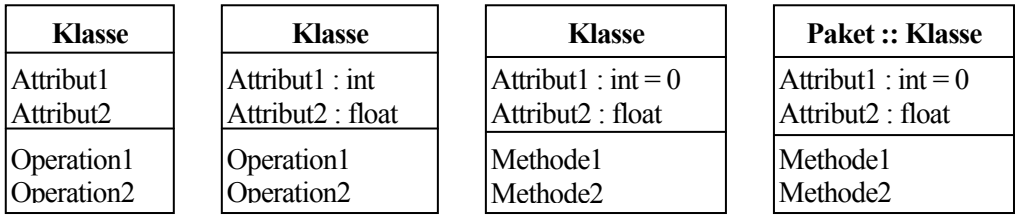


Abb. 4: Beispiele für die Bezeichnung von Klassen.

Grundsätzlich existieren vier Methodenarten. Dies sind die Objektmethoden. Sie werden stets auf ein einzelnes, bereits existierendes Objekt von außen angewendet. Gegenteilig verhalten sich die Klassenmethoden. Sie sind der jeweiligen Klasse zugeordnet und nicht auf ein einzelnes Objekt der Klasse

anwendbar, also nicht von außen nutzbar. Die dritte Art sind Konstruktormethoden. Sie erzeugen ein neues Objekt und führen entsprechende Initialisierungen und Datenerfassungen durch. Sie sind so eng mit der Klasse verbunden, dass sie den gleichen Namen tragen, wie die Klasse. Im Gegensatz zu Objektmethoden besitzen sie keinen Rückgabewert. Weisen sie zusätzlich keine Übergabeparameter auf, so spricht man von der Standardkonstruktormethode oder kurz dem Standardkonstruktor. Allgemeine Konstruktormethoden können dagegen Übergabeparameter besitzen und genau wie Funktionen überladen werden. Dies bedeutet, dass es mehrere allgemeine Konstruktormethoden mit unterschiedlichen Parameterlisten geben kann. Wenn mindestens eine allgemeine Konstruktormethode definiert worden ist, wird vom System kein Standardkonstruktor erzeugt. Dies hat zur Folge, dass es keinen gibt, wenn man ihn nicht selbst geschrieben hat. Neben den Konstruktormethoden kann eine Klasse eine Destruktormethode besitzen. Sie dient dazu, alle im Rahmen der Beseitigung eines Objekts anfallenden Aufräumarbeiten zu erledigen. Bei der Destruktormethode handelt es sich um eine parameterlose Funktion, die ebenso wie eine Konstruktormethode keinen Rückgabewert besitzt. Als Name für die Destruktormethode dient der Klassenname, dem eine Tilde (~) vorangestellt ist.

Die Verwendung von Objekten einer Klasse läuft immer in drei Phasen ab. Zuerst erfolgt die Instanzierung und Initialisierung des Objekts. Dann kann das Objekt verwendet werden. Wird es nicht mehr benötigt, so muss es beseitigt werden.

4.2.2 Statische Konzepte

Das statische Konzept bietet zur Modellierung außer den Basiskonzepten „Objekt“, „Klasse“, „Methode“ und „Attribut“ die Konzepte „Assoziation“, „Vererbung“ und „Paket“.

- Assoziation

Eine Assoziation ist streng genommen eine Verbindung zwischen Objekten einer oder mehrerer Klassen. Jedoch ist es üblich, von einer Assoziation zwischen Klassen zu sprechen. In der UML wird die Assoziation als durchgezogene Linie zwischen Objekten oder Klassen dargestellt. Diese Linie sagt aber noch nichts über die Richtung der Assoziation aus. In der Praxis ist es aber oft der Fall, dass Verbindungen zwischen Objekten nur in eine Richtung ausgelegt werden. In diesem Zusammenhang spricht man von gerichteter und ungerichteter Assoziation.
Während die Linie zunächst nur aussagt, dass sich Objekte der beteiligten Klassen kennen, spezifiziert die „Kardinalität“, wie viele Objekte ein bestimmtes Objekt kennen kann. Dabei unterscheidet man zwischen Kann- und Muss-Assoziationen. Eine Kann-Assoziation hat als Untergrenze die Kardinalität 0, eine Muss-Assoziation dagegen die Kardinalität 1 oder größer. Mögliche Kardinalitäten sind z. B. 1 (genau 1), 0..1 (0 bis 1), * (0 bis viele), 3..* (3 bis viele) oder 2, 4, 6 (2,4, oder 6).

Eine einfache Assoziation zwischen zwei Klassen repräsentiert eine strukturelle Beziehung zwischen Gleichberechtigten, d. h., keine ist wichtiger als die andere. Besteht allerdings eine Hierarchie zwischen den Objekten der beteiligten Klassen, die sich durch „ist Teil von“ bzw. „besteht aus“ beschreiben lässt, so kann dies mit Hilfe der Aggregation in der UML dargestellt werden. Sie ist somit ein Spezialfall der Assoziation. Eine einfache Aggregation ändert weder die Bedeutung der Navigation zwischen dem Ganzen und seinen Teilen in der Assoziation, noch stellt sie eine Beziehung zwischen der Lebensdauer des Ganzen und der seiner Teile her. Ihre Bedeutung ist rein konzeptionell.

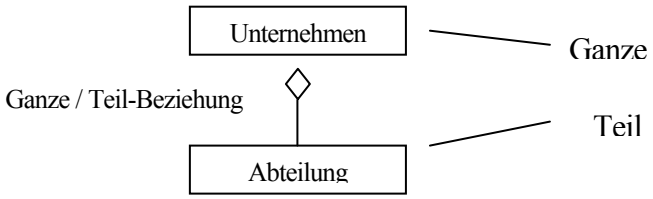


Abb. 5: Beispiel für eine Aggregation

Eine Variante der einfachen Aggregation ist die Komposition, die wichtige semantische Aspekte hinzufügt. Bei ihr herrscht eine stark ausgeprägte Eigentümerschaft des „Ganzen“ über die „Teile“. Teile mit variabler Multiplizität können nach der Erzeugung des Ganzen entstehen, leben und sterben dann aber mit ihm. Solche Teile können auch explizit vor dem Ende des Ganzen entfernt werden.

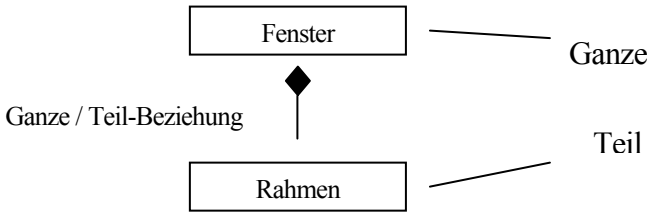


Abb. 6: Beispiel für eine Komposition

- Vererbung

Ein wichtiges Konzept zur Unterstützung der Wiederverwendbarkeit von Programmcode ist die Vererbung. Dabei beschreibt die Vererbung eine Beziehung zwischen einer allgemeinen Klasse und einer abgeleiteten Klasse. Die abgeleitete Klasse ist vollständig konsistent mit der allgemeinen Klasse, enthält aber zusätzliche Informationen (Attribute, Methoden, Assoziationen). Die Klasse, die Eigenschaften weitervererbt, wird auch Oberklasse (bzw. Basisklasse oder Superklasse) genannt, und Klassen, die etwas erben, heißen auch Unterklassen (bzw. Subklassen). Wichtig ist, dass es sich bei der Vererbung um eine Beziehung zwischen Klassen und nicht zwischen Objekten handelt.

Im Zusammenhang mit der Vererbungshierarchie verwendet man je nach Blickwinkel zwei Begriffe: Generalisierung und Spezialisierung. Eine Oberklasse ist eine Generalisierung der Unterklasse und eine Unterklasse ist eine Spezialisierung der Oberklasse. Graphisch wird die Vererbung in UML-Notation durch einen Pfeil von der Unter- zur Oberklasse dargestellt.

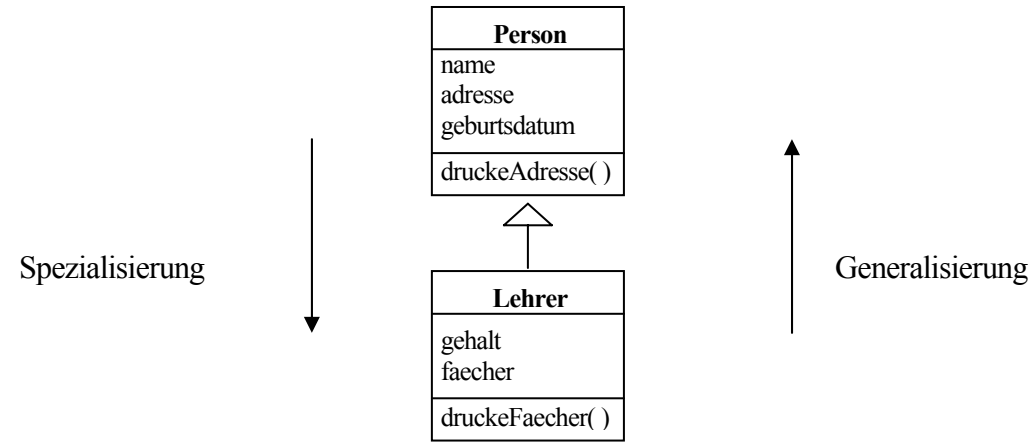


Abb. 7: Beispiel für eine Vererbung

Besitzt jede Klasse, mit Ausnahme der Wurzel, genau eine direkte Oberklasse, so spricht man von einer Einfachvererbung. Bildlich dargestellt entsteht eine Baumhierarchie mit einer Wurzel. Hat eine Klasse mehrere direkte Oberklassen, dann handelt es sich um eine Mehrfachvererbung. Die Struktur entspricht einem azyklischen Graphen, der mehr als eine Wurzel haben kann.

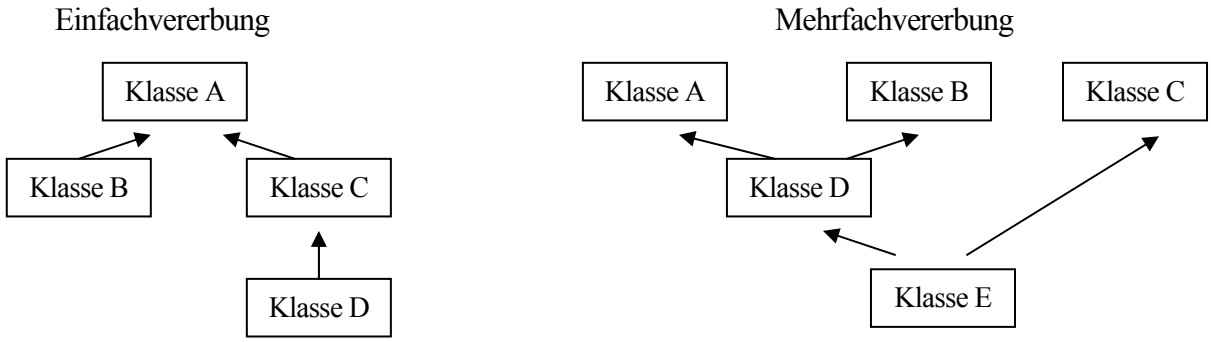


Abb. 8: Beispiel für eine Ein- und Mehrfachvererbung

Bei der Betrachtung der Klassen wurde zwischen der inneren und der äußeren Sicht unterschieden. Bei Einbeziehung der Vererbung kommt jetzt eine dritte Sicht hinzu, nämlich die Sicht der abgeleiteten Klasse auf ihre Basisklasse und deren Vorfahren. Der Zugriff auf die Elemente einer Klasse ist nun von zwei Komponenten abhängig. Erstens von der Zugriffsspezifikation in der Klasse und zweitens von den Zugriffsspezifikation, die bei der Vererbung den Zugriff auf die Oberklasse festlegt. Abgeleitete Klassen haben somit erweiterte Zugriffsrechte im Vergleich zu nicht verwandten Klassen, die jedoch nicht so weit gehen, dass sie auf alle Attribute und Methoden der Oberklasse zugreifen können.

- Klassendiagramm

Um die statischen Konzepte Assoziation und Vererbung der objektorientierten Analyse visualisieren, spezifizieren, konstruieren und dokumentieren zu können, existiert in der UML das Klassendiagramm. Es ist das gebräuchlichste Hilfsmittel beim Modellieren objektorientierter Systeme. Das Erstellen von Klassendiagrammen funktioniert ähnlich wie die ausgewogene Verteilung der Verantwortlichkeiten zwischen Klassen. Übertreibt man bei der Entwicklung, erhält man einen Beziehungswirrwarr, das das Modell unverständlich macht; untertreibt man, unterschlägt man Systemfähigkeiten.

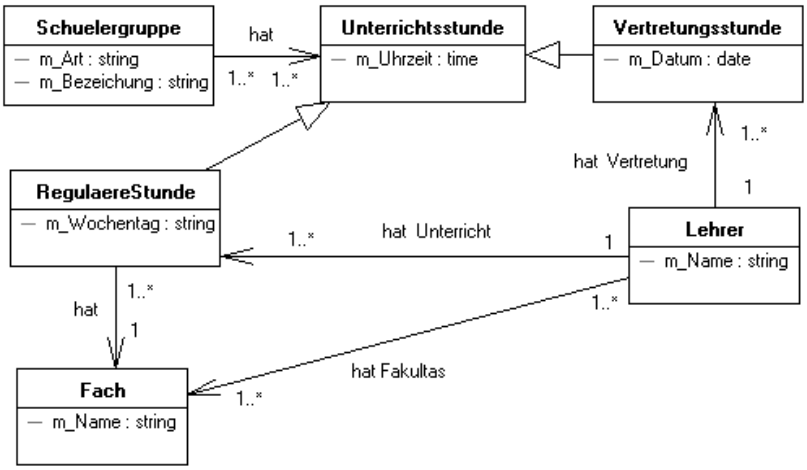


Abb. 9: Beispiel für ein Klassendiagramm

4.2.3 Dynamische Konzepte

- Anwendungsfalldiagramm

Das Anwendungsfalldiagramm (Use-Case-Diagram) dient als Schnittstelle zwischen dem Pflichtenheft und dem OOA-Modell. Im Pflichtenheft unterstützt es die Produktdokumentation und im OOA-Modell trägt es als Entscheidungshilfe zur Ermittlung von Anforderungen und zur Planung von Entwicklungsiterationen bei. Die Syntax des Anwendungsfalldiagramms ist vergleichsweise leicht zu verstehen, selbst wenn man die Notation nicht kennt. Dies hat den Vorteil, dass Diagramme auch mit zukünftigen Systemnutzern besprochen werden können, auch wenn diese sich mit UML nicht auskennen. Als Stilmittel stehen Anwendungsfälle, Akteure und Beziehungen zwischen Akteuren und Anwendungsfällen zur Verfügung. Bei dem Akteur kann es sich um einen Menschen, ein anderes Informationssystem oder ein Hardwaregerät handeln. Dieser Akteur befindet sich außerhalb des zu entwickelnden Systems, d. h., er ist nicht Teil des Anwendungsfallsystems, sondern interagiert mit ihm. Dargestellt wird ein individueller Anwendungsfall mit einem benannten Oval und ein Akteur gewöhnlich als Strichmännchen. Die Systemgrenze wird durch einen Rahmen um die Anwendungsfälle symbolisiert.

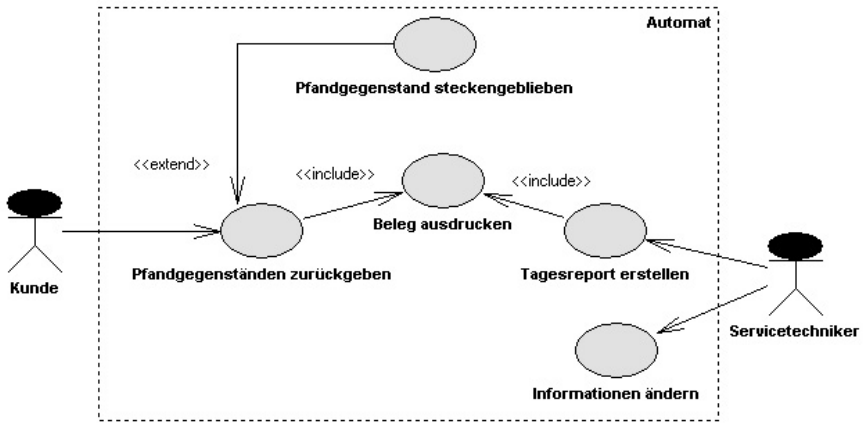


Abb. 10: Beispiel für ein einfaches Anwendungsfalldiagramm

Anwendungsfalldiagramme größerer Softwaresysteme beinhalten meist eine Vielzahl von Anwendungsfällen, einzelne Anwendungsfälle sind sehr komplex oder die Teilfunktionen sind in mehreren Anwendungsfällen identisch. Um dem entgegen zu wirken, bietet die UML-Notation drei Möglichkeiten Anwendungsfälle überschaubarer zu modulieren. Die include-Beziehung wird zur Modellierung eingesetzt, wenn verschiedene Anwendungsfälle dieselbe Teilfunktionalität beinhalten. Zur Redundanzvermeidung und zur besseren Verständlichkeit des Diagramms wird die Teilfunktionalität aus den Anwendungsfällen ausgeklammert und in einem separaten Anwendungsfall dargestellt. Eine include-Beziehung von Anwendungsfall 1 (Pfandgegenstände zurückgeben) zu Anwendungsfall 2 (Beleg ausdrucken) bedeutet also, dass Anwendungsfall 1 die durch Anwendungsfall 2 spezifizierte Teilfunktionalität benutzt. Die extend-Beziehung klammert auch Teilfunktionalität aus, jedoch mit einer anderen Intention. Bei ihr wird das Verhalten eines Anwendungsfalls erweitert. Eine extend-Beziehung von Anwendungsfall 1 (Pfandgegenstände zurückgeben) zu Anwendungsfall 2 (Pfandgegenstand steckengeblieben) bedeutet also, dass Anwendungsfall 1 die durch Anwendungsfall 2 spezifizierte Teilfunktionalität optional benutzt. Durch die extend-Beziehung kann die Komplexität von Anwendungsfällen verringert werden. Die dritte Möglichkeit Anwendungsfälle überschaubarer zu modulieren ist die Generalisierungs-Beziehung. Bei ihr erbt der spezialisierte Anwendungsfall das Verhalten und die Bedeutung des allgemeineren Anwendungsfalls. Der spezialisierte Anwendungsfall kann zusätzliches Verhalten hinzufügen oder Verhalten des allgemeinen Anwendungsfalls überschreiben.

Da das Anwendungsfalldiagramm nur einen Teil der benötigten Informationen beinhaltet, werden die wichtigsten Anwendungsfälle durch zusätzliche Informationen in Textform ergänzt. Während bei einfachen Anwendungsfällen eine umgangssprachliche Beschreibung ausreicht, können bei umfangreicheren Spezifikationen Anwendungsfallschablonen eingesetzt werden. Eine solche Schablone sollte als Checkliste betrachtet werden, d. h. sie ist nicht für jeden Anwendungsfall vollständig auszufüllen.

Anwendungsfall:	Name
Kurzbeschreibung:	Globale Zielsetzung bei erfolgreicher Ausführung des Anwendungsfalls
Vorbedingung:	Erwarteter Zustand, bevor der Anwendungsfall beginnt
Nachbedingung Erfolg:	Erwarteter Zustand nach erfolgreicher Ausführung des Anwendungsfalls, d. h. Ergebnis des Anwendungsfalls
Nachbedingung Fehlschlag:	Erwarteter Zustand, wenn das Ziel nicht erreicht werden kann
Akteure:	Rollen von Personen oder von anderen Systemen, die den Anwendungsfall auslösen oder daran beteiligt sind
Auslösendes Ereignis:	Wenn dieses Ereignis eintritt, dann wird der Anwendungsfall initiiert
Primärszenario:	1 erste Aktion 2 zweite Aktion ...
Erweiterungen:	1a Erweiterung des Funktionsumfangs der ersten Aktion
Alternativszenarien:	1a alternative Ausführung der ersten Aktion 1b weitere Alternative zur ersten Aktion

Abb. 11: Beispiel für ein einfaches Anwendungschablone

- Szenario

Das Anwendungsfalldiagramm ist so aufgebaut, dass es für die zukünftigen Systemnutzer lesbar ist. Dies hat zur Folge , dass es nicht sehr detailliert und präzise formuliert ist und daher als Vorlage für die weitere Entwicklung nicht ausreicht. Im nächsten Schritt gilt es daher die einzelnen Anwendungsfälle zu verfeinern. Hierzu werden Szenarien modelliert. Ein einzelnes Szenario stellt eine Sequenz von Verarbeitungsschritten dar, die unter bestimmten Bedingungen auszuführen sind. Es beginnt mit einem auslösenden Ereignis durch einen Akteur oder einen anderen Anwendungsfall und wird fortgesetzt, bis das Ziel erreicht ist oder aufgegeben wird.

Mit den Szenarien wird es möglich, die Methoden der Klassen zu identifizieren und den Fluss der Nachrichten durch das Softwaresystem zu definieren. Außerdem kann die Vollständigkeit und Korrektheit des statischen Modells kontrolliert werden.

Zu Erstellung eines Szenarios wählt man einen Anwendungsfall aus und überlegt, welche Varianten auftreten können. Dabei stellt jede Variante, die zu einem unterschiedlichen Ergebnis eines Anwendungsfalls führt, ein Szenario dar. Auch wenn ein Softwaresystem eine überschaubare Anzahl von Anwendungsfällen enthält, kann der einzelne Anwendungsfall eine Vielzahl von möglichen Szenarien beinhalten. Da aber nicht alle Szenarien von gleicher Bedeutung für die Modellbildung sind, kann im Gegensatz zum vollständigen statischen Modell das dynamische Modell sich auf die Szenarien beschränken, die wesentlich für das Modellierendes dynamischen Verhaltens sind. Man spricht in diesem Zusammenhang auch von primären und sekundären Szenarien. Primäre Szenarien stellen die fundamentalen bzw. die am häufigsten verwendeten Abläufe des Softwaresystems dar. Sekundäre Szenarien zeigen dagegen Ausnahmesituationen wenig verwendeter Abläufe.

Grundsätzlich können Szenarien in Textform und durch Interaktionsdiagramme dokumentiert werden. Die Textform beinhaltet folgende Punkte: Name des Szenarios, Bedingungen, die zu dieser Variation des Anwendungsfalls führten und das Ergebnis des Szenarios. Zur Darstellung eines Szenarios in einem Interaktionsdiagramm stellt UML zwei Arten von Diagrammen zur Verfügung; das Sequenz- und das Kollaborationsdiagramm. Beide Diagramme haben Vor- und Nachteile. Um zeitliche Abläufe darzustellen, ist das Sequenzdiagramm besonders geeignet. Besitzt ein Sequenzdiagramm allerdings viele Objekte, so lassen sich Kreuzungen von Lebenslinien und Pfeilen nicht vermeiden und das Diagramm wird schnell unübersichtlich. Durch die Möglichkeit, die Objekte beim Kollaborationsdiagramm frei anordnen zu können, tritt dieser Nachteil bei ihm nicht auf. Allerdings sind hier die zeitlichen Abläufe aus dem Diagramm nicht direkt erkennbar.[26]

Ein Sequenzdiagramm besitzt zwei Dimensionen. Auf der Horizontalen werden die Objekte aufgetragen, die an dem Szenario beteiligt sind. Dabei ist es grundsätzlich egal, in welcher Reihenfolge die Objekte dargestellt werden. Zur besseren Lesbarkeit des Diagramms bietet es sich jedoch an, dass die Objekte, die als erste an der Reihe sind, ganz nach links gesetzt werden, so dass die meisten Nachrichten von links nach rechts fließen. Die Vertikale des Diagramms gibt die Zeit aus der Sicht des Objekts an. Geht man von oben nach unten durch das Diagramm, so geht man davon aus, dass die Zeit fortschreitet. Diese Linie wird als „Lebenslinie“ des Objekts bezeichnet und wird gestrichelt im Diagramm dargestellt. Sie beginnt nach dem Erzeugen des Objekts und endet mit dem Löschen des Objekts. Die Bezeichnung eines Objekts wird am oberen Ende der gestrichelten Linie dargestellt.

Die Anzahl der an einer Interaktion beteiligten Objekte ist nicht statisch. Objekte können im Laufe der Interaktion neu erstellt und gelöscht werden. Wird ein Objekt zur Laufzeit erzeugt, so wird sein Kästchen etwas weiter unten auf der Seite - am Punkt seiner Erstellung - platziert. Beim Vernichten eines Objekts endet die Lebenslinie mit einem großen „X“. Ist ein Objekt aktiviert, so wird dies mit einem schmalen Rechteck auf der Lebenslinie dargestellt. Führt das Objekt eine Berechnung aus, so kann dies zusätzlich durch eine Schraffur dargestellt werden.

Eine Nachricht zwischen zwei Objekten wird als Pfeil dargestellt, der von der Lebenslinie des Absenders zu der des Empfängers verläuft. Beschriftet wird der Pfeil mit dem Namen der Nachricht. Auch ist es möglich, Nachrichten nur unter einer bestimmten Bedingung zu senden. Für diesen Fall wird die Bedingung in eckigen Klammern angegeben, d. h. [Bedingung] Nachricht. Mit einem „*“ können Wie-

---

[26] Auf Grund dieser Tatsache habe ich mich entschieden, das Kollaborationsdiagramm als Lerngegenstand für eine Einführung in die objektorientierte Modellierung nicht aufzunehmen.

derholungen von Nachrichten kenntlich gemacht werden. Wenn die Häufigkeit der Iteration fehlt, so bedeutet dies laut UML, dass die Anzahl der Wiederholungen unspezifiziert ist.

Rückführende Nachrichten können, müssen aber nicht in das Diagramm eingetragen werden, da ein Objekt seine Aktivierung verliert, sobald es auf die Nachricht antwortet, die diese Aktivierung ausgelöst hat. Bei der Modulation ist darauf zu achten, dass Objekte nur dann Nachrichten austauschen können, wenn zwischen ihren Klassen eine Assoziation besteht oder sie sich innerhalb derselben Klasse befinden. Sollte dies aus dem Klassendiagramm nicht hervorgehen, ist es fehlerhaft. Der umgekehrte Fall ist allerdings möglich.
Auch kann ein Objekt Nachrichten an sich selbst schicken, was auch häufig der Fall ist. Dabei gibt es jedoch ein Problem. Das Objekt hat in diesem Fall bereits eine Aktivierung. Die neue Aktivierung kann durch eine verschachtelte Aktivierung ausgedrückt werden. Das schmale Rechteck, das die neue Aktivierung wiedergibt, wird leicht versetzt zum Rechteck für die alte Aktivierung gezeichnet, so dass beide sichtbar sind.

Beim reinen objektorientierten Programmieren ist jeder Funktionsaufruf das Ergebnis einer Nachricht, und Objekte können mitunter so oft Nachrichten an sich selbst schicken, dass ein Sequenzdiagramm sehr unübersichtlich werden kann. Zur besseren Lesbarkeit verzichtet man aus diesem Grund auf die Darstellung von Nachrichten, die ein Objekt an sich selbst schickt.

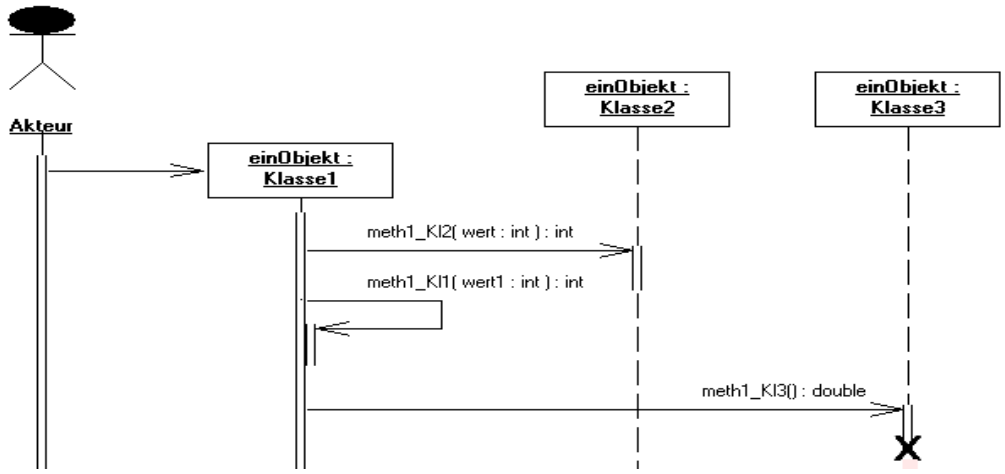


Abb. 12: Notation für ein einfaches Sequenzdiagramm.

4.3 Prototyp der Benutzeroberfläche

Nachdem das OOA-Modell erarbeitet wurde, gilt es aus diesem Modell den Prototyp der Benutzeroberfläche zu gestalten. Als Grundelemente stehen Fenster, Dialoge und Menüs zur Verfügung. Die Transformation kann systematisch aus dem Klassendiagramm erfolgen. Für jede Klasse des OOA-Modells ist zu prüfen, ob die Klasse durch ein Fenster oder einen Dialog abgebildet werden muss.

Als Resultat entsteht ein ablauffähiges Programm, das alle Attribute des OOA-Modells auf die Oberfläche abbildet. In dieser Stufe der Entwicklung beinhaltet das Programm weder Anwendungsfunktionen, noch besitzt es die Fähigkeit, Daten zu speichern. Das Programm soll lediglich dem zukünftigen Benutzer einen Einblick geben, um möglichst frühzeitig Kritik zu üben.

5 Der objektorientierte Entwurf

Das objektorientierte Analyse-Modell beschreibt die essentielle Struktur und Semantik des Problems. Es geht von idealisierten Bedingungen aus und enthält noch keine technische Lösung des Problems. Die Aufgabe des objektorientierten Entwurfs ist es, das Modell der Analyse auf einer Plattform unter den geforderten technischen Rahmenbedingungen zu realisieren. Der Übergang ist fließend, es findet kein Paradigmenwechsel statt.

Das Ergebnis dieser Phase ist wiederum ein statisches und ein dynamisches Modell. Das statische Modell soll in dieser Phase so weit entwickelt werden, dass es im Idealfall ein Abbild des späteren Programms darstellt. Jede Klasse, jedes Attribut und jede Methode des Entwurfs kommen im späteren Programm vor. Um den Übergang in die Implementierung zu erleichtern, werden in der Entwurfsphase die Namenskonventionen für die Implementierungsphase bereits berücksichtigt. Die Anbindung der beiden Phasen ist so stark, dass es üblich ist die modulierten Klassen in die gewünschte Programmiersprache zu implementieren. Die Aufgabe des dynamischen Modells ist die übersichtliche Beschreibung der Kommunikation zwischen den Objekten, die anhand des Quellcodes nur schwer nachvollziehbar ist.

Um das Ziel dieser Phase zu erreichen, gilt es zu Beginn drei Entscheidungen zu treffen. Wie bereits oben beschrieben, ist die Wahl der Programmiersprache von zentraler Bedeutung. Weiterhin steht die Entscheidung an, welches GUI-System für die Implementierung der Prototypen der Benutzeroberfläche verwendet werden soll. Außerdem ist die Art der Datenhaltung festzulegen. Je nach Datenvolumen bietet sich eine Datenbank, eine Datei oder bei Windows-Betriebssystemen die Registry an.

5.1 Umsetzung der Konzepte in C++

Als technische Rahmenbedingung gilt laut Zieldefinition, dass die Implementierung in der Programmiersprache C++ erfolgen soll. Daher müssen nun die Konzepte des OOA-Modells in Kombination mit der Programmiersprache C++ in das OOD-Modell transformiert werden.

5.1.1 Basiskonzepte

- Deklaration einer Klasse

Im Wesentlichen gleicht eine Klassendeklaration in der Programmiersprache C++ sehr der Deklaration einer Struktur, mit dem fundamentalen Unterschied, dass nun auch Funktionen gekapselt werden können. Um den Zugriff kontrollieren zu können, besitzt C++ drei Bereiche mit unterschiedlichen Zugriffsspezifikationen. Im „private"-Bereich werden alle Methoden und Attribute vereinbart, die ausschließlich privat verwendet werden. Es besteht keine Zugriffsmöglichkeit von außen. Im „protected"-Bereich (nur bei Vererbung und auch dann nur in Ausnahmefällen) werden alle Methoden vereinbart, die sowohl privat als auch in abgeleiteten Klassen verwendet werden dürfen. Der „protected"-Bereich ist kaum stärker geschützt als der „public"-Bereich, denn jede abgeleitete Klasse hat Zugriff auf ihn. Deshalb sollten auch hier keine Attribute vereinbart werden. Im „public"-Bereich werden alle öffentlichen Methoden vereinbart. Attribute sind aufgrund des Geheimnisprinzips nicht öffentlich zu machen. Bei den Methoden ist die Reihenfolge Standardkonstruktor (sofern benötigt), sonstige Konstruktoren (sofern benötigt), Destruktor (sofern benötigt) und sonstige Methoden empfehlenswert. Wird bei der Deklaration einer Klasse kein Bereich angegeben, so sind sämtliche Attribute und Methoden privat.

```
class User
{
private:
    string name;
    string password;
    int priv;
public:
    void printUser( );
};
```

Abb. 13: Beispiel für die Deklaration einer Klasse in der Programmiersprache C++.

Im obigen Beispiel sind alle Attribute als „private" deklariert. Sie können also von außen nicht erreicht werden. Lediglich die Methode printUser( ) der Klasse kann auf die Attribute zugreifen. Die Methode printUser( ) ist frei verfügbar.

- Trennung zwischen Deklaration und Definition

Rein theoretisch könnte man sämtliche Programmcodes in eine einzige Datei schreiben. Um aber eine bessere Wiederverwertbarkeit und Datenkapselung zu erreichen, sollte man Programme grundsätzlich modular aufbauen. Aus diesem Grund teilt man den Quellcode einer Klasse in die Deklaration der Klasse und in die dazugehörige Definition auf. Es entsteht eine saubere Trennung zwischen dem, was eine Klasse bietet und wie sie intern aussieht. In C++ wird die Deklaration in einer Header-Datei (klassenname.h) und die Definition (Implementierung) in einer „normalen" Quellcode-Datei (klassenna-

me.cpp) realisiert. Eine Header-Datei wird mittels der Compilerdirektive „#include“ zum einen in die zugehörige Definitionsdatei kopiert und zum anderen in alle Dateien, in denen Objekte der Klasse erzeugt werden müssen. Ebenfalls müssen Header-Dateien #include-Anweisungen enthalten, sofern sie in anderen Header-Dateien definierte Namen (z. B. Typnamen) enthalten. Um bei den daraus resultierenden geschachtelten include-Anweisungen Mehrfachkopien und damit Mehrfachdefinitionen zu verhindern, sind alle Deklarationsdateien mit einem Schutzmechanismus zu versehen. Hierzu wird vor der Deklaration der Klasse mit der Compilerdirektive „#indef _beliebigerName“ abgefragt, ob das Define „_beliebigerName“ existiert. Existiert dieses Define, so wurde die Headerdatei bereits vorher schon einmal compiliert. In diesem Fall wird der Block von #indef bis #endif nicht ein zweites Mal compiliert. Im anderen Fall wird der Block compiliert. Hierbei wird das Define „_beliebigerName“ und die Klasse erstellt.

Deklaration in der Datei User.h:

```
#ifndef _User
#define _User
class User
{
private:
        string name;
        string password;
        int priv;
public:
        void printUser();
};
#endif
```

Definition in der Datei User.cpp:

```
#include „User.h“
User::printUser( )
{
        cout << name << endl;
        …
}
```

Abb. 14: Trennung zwischen Deklaration und Definition

Ein weiterer Schutzmechanismus, um Mehrfachdefinitionen zu vermeiden, ist die Compilerdirektive „#pragma once“. Diese Möglichkeit hat allerdings den Nachteil, dass die Umsetzung der Pragma-Direktiven für die Compilerhersteller nicht verpflichtend ist. Im Allgemeinen können Pragma-Direktiven nicht als portabel angesehen werden. Findet ein Compiler eine Pragma-Direktive vor, die er nicht erkennt, gibt er eine Warnung aus, die Kompilierung wird aber fortgesetzt.
Auf Grund der geringeren Schreibarbeit ist die Compilerdirektive „#pragma once“ inzwischen bei fast allen C++-Compilern implementiert.

```
#ifndef _User
#define _User

….
#endif
```

⟺

```
#pragma once
…
```

Abb. 15: Schutzmechanismus vor Mehrfachdefinitionen

- Inline-Funktionen

Grundsätzlich sollte bei der Erstellung von Klassen, wie bereits beschrieben, eine Trennung zwischen der Deklaration der Klasse (Schnittstelle) und der Definition der Methoden (Implementierung) erfolgen. Da aber jeder Funktionsaufruf mit einer Laufzeiterhöhung - aufgrund des erforderlichen Stackmanagements - verbunden ist, bietet C++ die Möglichkeit, so genannte inline-Funktionen zu definieren. Der Code der entsprechenden inline-Funktion wird beim Kompilieren vom Präprozessor an der Aufrufstelle eingefügt, das Stackmanagement entfällt. Grundsätzlich kann dies auf drei Arten erfolgen.
Die erste Variante ist die implizite inline Deklaration der Methoden in der Header-Datei. Die Deklaration der Klasse und die Definition der Methoden erfolgt in einer Datei. Es besteht keine Trennung zwischen Schnittstelle und Implementierung. Folglich ist es nicht möglich Daten zu kapseln oder den Quellcode der Methoden geheim zu halten. Hinzu kommt noch, dass bei umfangreicheren Methoden der Quellcode sehr schnell unübersichtlich wird.

Die zweite Variante ist die explizite inline Deklaration der Methoden in der Header-Datei: Die Definition der Methoden erfolgt ebenfalls in der Header-Datei, allerdings außerhalb der Klassendeklaration. Diese Variante verhält sich bezüglich der Schnelligkeit und der Datenkapselung wie die erste Variante. Der Vorteil dieser zweiten Variante liegt jedoch in der besseren Übersichtlichkeit des Quellcodes, falls eine Klasse viele Methoden besitzt.

implizite inline Deklaration der Methoden

User.h:

```
#pragma once
class User
{
private:
        string name;
        string password;
        int priv;
public:
        void printUser( )
        {
                cout << name << endl;
                …
        }
};
```

explizite inline Deklaration der Methoden

User.h:

```
#pragma once
class User
{
private:
        string name;
        string password;
        int priv;
public:
        void printUser( );
};
inline void User::print( ) {
        cout << name << endl;
        …
}
```

Abb. 16: Beispiel für eine im- und explizite inline Deklaration der Methoden in der Header-Datei

Die dritte Variante ist die explizite inline Deklaration der Methoden in der cpp-Datei: Diese Möglichkeit besitzt durch inline einen Schnelligkeitsgewinn und durch die Trennung von Deklaration und Implementierung die Möglichkeit der Datenkapselung. Damit die inline Methoden anderen cpp-Dateien

zur Verfügung stehen, muss ihr Quellcode bei jedem Compilerdurchlauf neu kompiliert werden, auch wenn er nicht verändert wurde. Standardmäßig erfolgt dies allerdings nicht bei den meisten Compilern, da es die Kompilierungszeit erhöht. Daher sollte diese Variante nach Möglichkeit nicht verwendet werden.

User.h:

```
#pragma once
class User
{
private:
        string name;
        string password;
        int priv;
public:
        void printUser();
};
```

User.cpp:

```
#include „User.h“
inline User::print( )
{
        cout << name << endl;
        …
}
```

Abb. 17: Beispiel für eine explizite inline Deklaration der Methoden in der cpp-Datei

- Generierung von Objekten

Objekte einer Klasse kann man - wie Variablen einer Struktur - automatisch, statisch oder dynamisch anlegen. Automatisch angelegte Objekte werden immer erneut instantiiert, wenn der Programmablauf ihre Definition passiert. Sobald der Block verlassen wird, in dem sie definiert wurden, werden sie beseitigt. Bei statischen Objekten muss zwischen Objekten, die innerhalb und außerhalb von Funktionen angelegt wurden, unterschieden werden. Statische Objekte, die außerhalb von Funktionen angelegt wurden, werden vor dem eigentlichen Programmstart (also noch vor main-Funktion) instantiiert und „leben“ bis zum Programmende. Dagegen werden statische Objekte, die innerhalb von Funktionen angelegt wurden, erst dann instantiiert, wenn der Programmablauf erstmalig ihre Deklaration passiert. Sie „leben“, auch wenn sie innerhalb von Funktionen angelegt wurden, über die gesamte Programmlaufzeit hinweg. Ihre Beseitigung erfolgt erst am Programmende. Dynamisch angelegte Objekte sind Objekte, die man immer dann mit „new“ anlegt, wenn man sie benötigt, und mit „delete“ wieder beseitigt, wenn sie nicht mehr gebraucht werden.

- Konstruktormethode

Wie oben beschrieben, erfolgt die Erzeugung von Objekten einer Klasse analog der Erzeugung von Variablen einer Struktur. Im Gegensatz zur Erzeugung von Variablen einer Struktur können bei der Erzeugung von Objekten den Attributen definierte Anfangswerte zugewiesen werden. Dies kann in unterschiedlicher Weise erfolgen.

Einmal besteht die Möglichkeit die Attribute durch den Standardkonstruktor zu initialisieren. Bei der Erzeugung des Objektes werden den Attributen immer dieselben Werte zugewiesen. Ist dies nicht ge-

wünscht, so besteht die Möglichkeit, eine allgemeine Konstruktormethode zu implementieren. Mit ihr ist es dann möglich, bei der Erzeugung eines Objektes die gewünschten Werte zu übergeben.

Standardkonstruktor:

```
User()
{
  name="";
  password="";
  priv=0;
}
```

allgemeine Konstruktormethode:

```
User(std::string name, std::string password, int priv, int badPwCount)
{
  User::name=name;
  User::password=password;
  User::priv=priv;
}
```

Abb. 18: Beispiel für die Initialisierung von Attributen.

Bei beiden Varianten werden die Werte den Attributen des Objektes immer im Rumpf der Konstruktormethode zugewiesen. Konstruktormethoden werden allerdings in zwei Stufen aufgerufen: Zuerst in der Initialisierungsphase und dann bei Ausführung des Rumpfes. Die meisten Variablen lassen sich in beiden Phasen einrichten. Entweder durch Initialisierung im Initialisierungsteil oder durch Zuweisung im Rumpf der Konstruktormethode. Sauberer und meist auch effizienter ist die Zuweisung im Initialisierungsteil. Auf die schließende Klammer der Parameterliste der Konstruktormethode folgt ein Doppelpunkt. Danach folgt der Name des zu initialisierenden Attributs. Hinter dem Namen steht in Klammern der Ausdruck, mit dem das Attribut initialisiert werden soll. Gibt es mehrere Initialisierungen, sind diese jeweils durch ein Komma zu trennen.

5.1.2 Statische Konzepte

- Vererbung

In der Programmiersprache C++ gilt die folgende Syntax für die Einfach- und Mehrfachvererbung von Klassen:

Einfachvererbung:
class Unterklasse : Zugriffsspezifizierer Oberklasse

Mehrfachvererbung:
class Unterklasse : Zugriffsspezifizierer Oberklasse1, Zugriffsspezifizierer Oberklasse2, ...

„Unterklasse“ ist der Name der neu zu definierenden Klasse. „Oberklasse“ ist der Name der Klasse, deren Eigenschaften übernommen werden sollen. Der Zugriffsspezifizierer gibt an, wie auf die von der Oberklasse geerbten Attribute und Methoden zugegriffen werden darf. Wie in der Klasse selbst stehen für die Vererbung die Zugriffsspezifikationen public, protected oder private zur Verfügung. Fehlt die

Zugriffsspezifikation, so wird vom Compiler die restriktivste Form (die private-Vererbung) angenommen.

```
class User
{
private:
    string name;
    string password;
    . . .
protected:
    int priv;
    . . .
public:
    int unsinn;

    . . .
}
```

```
class Beamte : private  User
{
private:
    double gehalt;
    . . .
}

class Angestellte : protected  User
{
private:
    int alter:

    . . .
}
```

```
class Schueler : public  User
{
private:
   string klasse;
. . .
}
```

Abb. 19: Beispiel für eine privat-, protected- und public-Vererbung

Grundsätzlich kann auf private Attribute und Methoden der Oberklasse nicht zugegriffen werden, unabhängig davon, welcher Zugriffsspezifizierer für die Vererbung angegeben wird. Auf obiges Beispiel bezogen bedeutet dies, dass weder die Methoden der Klasse Beamte, Schueler noch Angestellte auf die als „private“ deklarierten Variablen name und password der Klasse User zugreifen können.

Wird als Zugriffsspezifizierer für die Vererbung „private“ angegeben, dann können die Methoden der abgeleiteten Klassen auf die geerbten „public“ und „protected“ deklarierten Attribute und Methoden der Oberklasse zugreifen. Dies gilt jedoch nicht für Unterklassen, die durch weitere Vererbung der abgeleiteten Klasse entstehen. Die „public“ und „protected“ deklarierten Attribute und Methoden der Oberklasse sind somit zu privaten Attributen und Methoden in der direkt abgeleiteten Unterklasse geworden. Für obiges Beispiel bedeutet dies, dass die Methoden der Klasse Beamte auf die Attribute priv und unsinn der Klasse User zugreifen können. Abgeleitete Klassen der Klasse Beamte können dies allerdings nicht mehr.

Bei einer public-Vererbung können die Methoden der abgeleiteten Klasse auf alle als public und protected deklarierten Attribute und Methoden der Oberklasse zugreifen. Dieses Zugriffsrecht wird auch an eventuelle weitere Unterklassen der abgeleiteten Klasse vererbt. Dies bedeutet wiederum für das obige Beispiel, dass die Methoden der Klasse Schueler auf die Attribute priv und unsinn der Klasse User zugreifen können. Im Gegensatz zu einer private-Vererbung können auch Methoden einer abgeleiteten Klasse von der Klasse Schueler auf die Attribute priv und unsinn der Klasse User zugreifen

Bei einer protected-Vererbung können die Methoden der abgeleiteten Klasse auf alle als public und protected deklarierten Attribute und Methoden der Basisklasse zugreifen. Jedoch stehen die als public deklarierten Attribute und Methoden in der abgeleiteten Klasse als protected zur Verfügung. Für obiges Beispiel bedeutet dies, dass die Methoden der Klasse Angestellte und Methoden einer abgeleiteten Klasse von der Klasse Angestellte auf die Attribute priv und unsinn zugreifen können.

Element in der Basisklasse	***Ableitung***	***Element in der abgeleiteten Klasse***	***Ableitung***	***Element in der abgeleiteten Klasse***
public		*public*		*public*
protected	public	*protected*	public	*protected*
private		*nicht zugreifbar*		*nicht zugreifbar*
public		*protected*		*protected*
protected	proteced	*protected*	proteced	*protected*
private		*nicht zugreifbar*		*nicht zugreifbar*
public		*private*		*nicht zugreifbar*
protected	private	*private*	private	*nicht zugreifbar*
private		*nicht zugreifbar*		*nicht zugreifbar*

Abb. 20: Veränderungen der Zugriffsrechte bei einer private-, protected- und public-Vererbung

An der obigen Übersicht wird auch deutlich, warum das Zugriffsrecht protected für Attribute und Methoden in die Programmiersprache C++ eingeführt wurde. Ohne dieses Schlüsselwort müsste man alle Attribute und Methoden einer Oberklasse, die weitervererbt werden müssen, als public einstufen[27]. Dies würde der Datenkapselung, ein wichtigstes Dogma der Objektorientierung, widersprechen.

Da die Programmiersprache C++ neben der Ein- auch die Mehrfachvererbung unterstützt, besteht bei der Mehrfachvererbung die Möglichkeit, dass eine Klasse von ihren Oberklassen Attribute gleichen Namens erbt. Dies führt dann zu Konflikten, wenn Attribute denselben Datentyp besitzen. Eine einfache Möglichkeit, Konflikte zu vermeiden, wäre, die Attribute der Oberklassen umzubenennen. Da aber die Implementierung der Oberklassen nicht immer im Quellcode zur Verfügung steht, ist diese Möglichkeit nur bedingt einsetzbar. Über eine sehr elegante Möglichkeit verfügt die Programmiersprache C++. Bei ihr ist es in diesem Fall möglich, dass bei der Unterklasse durch Angabe der jeweiligen Oberklasse festgelegt wird, welches Attribut geerbt werden soll. Erbt eine Klasse von ihren Oberklassen Methoden gleichen Namens, aber mit unterschiedlichen Inhalten, so führt dies ebenfalls zu Kon-

[27] vgl. grau unterlegte Zeilen in der Abb. 20

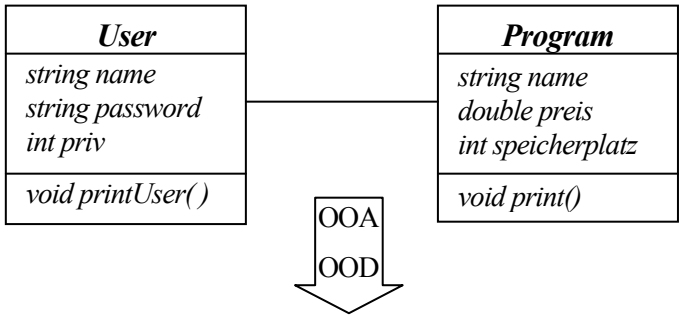
flikten. Wie bei Attributen besteht die Möglichkeit, die Namen der Methoden in den Oberklassen zu verändern, sofern ihr Quellcode zur Verfügung steht. Ist dies nicht der Fall, ist es in der Programmiersprache C++ wie bei Attributen möglich, in der Unterklasse durch Angabe der jeweiligen Oberklasse festzulegen, welche Methoden vererbt werden sollen.

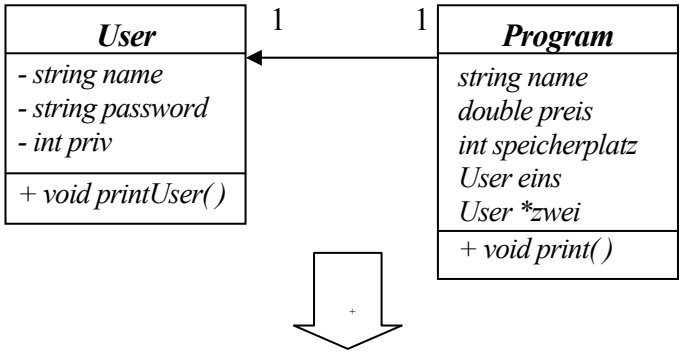
- Assoziation

Eine Assoziation beschreibt die Beziehung zwischen Objekten. Während im OOA-Modell die Assoziation ungerichtet dargestellt wird, muss ihr im objektorientierten Entwurf eine Richtung, auch Navigation genannt, zugeordnet werden. Je nachdem welches Objekt auf welches Objekt zugreift, kann die Richtung uni- oder bidirektional sein. Dabei ist eine Implementierung einer bidirektionalen Navigation komplexer als die einer unidirektionalen Navigation. Dies liegt daran, dass bei einer bidirektionalen Navigation beim Löschen eines Objekts darauf geachtet werden muss, dass auch die Navigation zu ihm im assoziierten Objekt zu entfernen ist.

Die Realisierung einer Assoziation erfolgt in der Programmiersprache C++ mittels eines Zeigers. Dies liegt darin begründet, dass die Assoziation keine Aussage über die Lebensdauer der beteiligten Objekte macht. Durch die dynamische Erzeugung des Objekts über einen Zeiger wird gewährleistet, dass offen bleibt, welches der beteiligten Objekte eine längere Lebensdauer besitzt.

Im folgenden Beispiel wird das Attribut „eins“ der Klasse „Program“ statisch erzeugt, wenn ein Objekt der Klasse „Program“ erzeugt wird. „Stirbt“ das Objekt der Klasse „Program“, so stirbt mit ihm das Attribut„eins“. Dies kann, muss aber bei einer Assoziation nicht der Fall sein. Aus diesem Grund lässt sich eine Assoziation nicht mit Hilfe einer statischen Navigation implementieren. Dagegen wird das Attribut „zwei“ der Klasse „Program“ dynamisch erzeugt, wenn ein Objekt der Klasse „Program“ erzeugt wird. „Stirbt“ das Objekt der Klasse „Program“, so kann die Objektvariable „zwei“ weiterhin existieren.






```
class User
{
private:
        std::string name;
        std::string password;
        int priv;
public:
        void printUser();
};
```

```
class Program
{
private:
        User  eins;
        User *zwei;
        …
public:
        Program( ){ zwei =new User;}
};
```

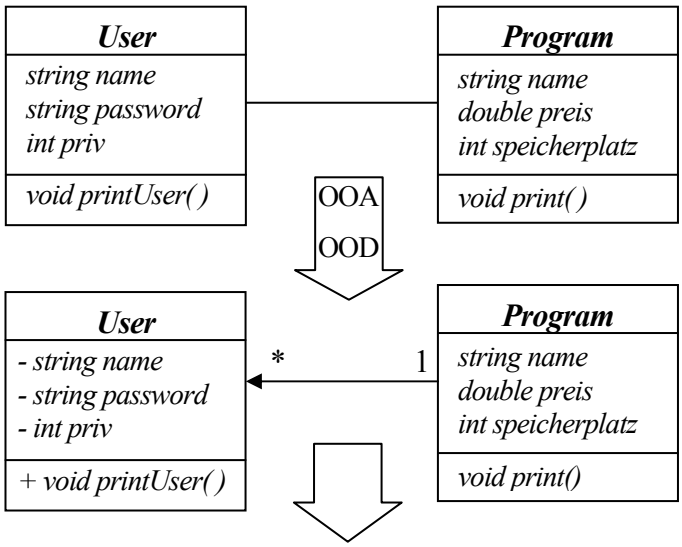
Abb. 21: Beispiel für eine Assoziation mit der Kardinalität 1 zu 1

Liegt bei einer Assoziation zwischen zwei Objekten eine Kardinalität n mit n größer 1 vor, so bestehen grundsätzlich zwei Möglichkeiten, diese Beziehung zu implementieren. Analog der Kardinalität 0..1 kann eine Kardinalität mit der Anzahl n mit n Zeigern oder mit einem Zeiger auf ein Array der Größe n realisiert werden.

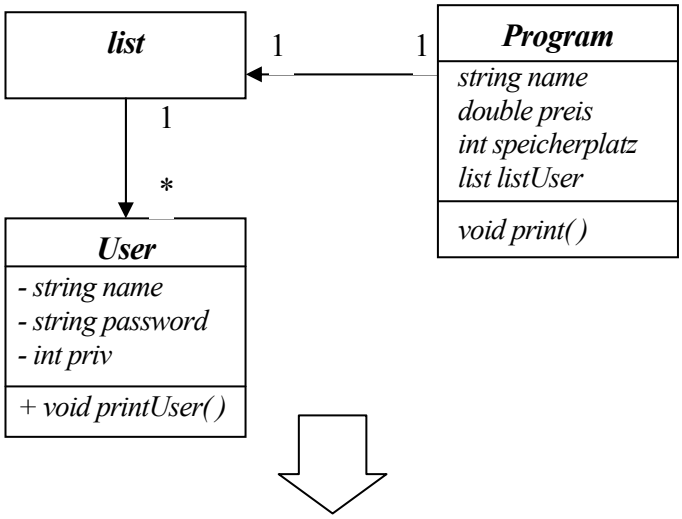
Eine sehr häufige Beziehung zwischen Objekten ist die „größer gleich null Kardinalität“. Z. B. kann ein Programm von keinem, einem, mehreren bis hin zu - theoretisch - unendlich vielen Benutzern verwendet werden. Die Umsetzung dieser Beziehung erfolgt in der Programmiersprache C++ mit Hilfe von Container-Klassen. Hierzu stellt die Standard Template Library die Container-Klassen Vektor, Deque, Liste, Set und Map zur Verfügung.

Container-Klassen speichern Daten, die nur über vorgegebene Iteratoren, Methoden und Algorithmen verändert werden können. Die Verwaltung der Daten innerhalb der Klasse erfolgt entweder mit Hilfe einer Liste oder eines Baumes. Mit dieser kommt der Programmierer allerdings nicht in Berührung. Damit Container-Klassen für beliebige Datentypen verwendet werden können, sind sie als „Template“ implementiert. Daher muss bei der Erzeugung eines Objektes der Datentyp der zu speichernden Daten als Argument mit übergeben werden.

1. Schritt



2. Schritt




```
class User
{
private:
        std::string name;
        std::string password;
        int priv;
public:
        void printUser();
};
```

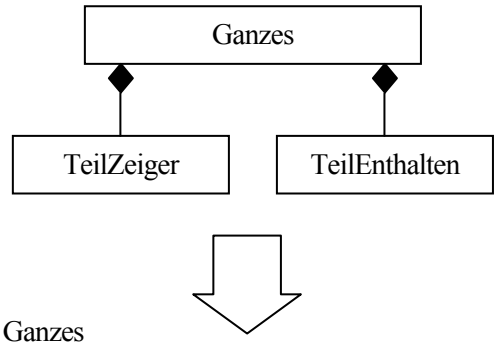
```
class Program
{
private:
        list <User> listUser
        …
public:
        …
};
```

*Abb. 22: Beispiel für eine Assoziation mit der Kardinalität 1 zu **

Ein Spezialfall der Assoziation ist die Aggregation. Sie wird im Prinzip genauso realisiert wie die Assoziation. Bei einer Aggregation muss das Ganze stets seine Teile kennen. Daher muss bei der Implementierung immer eine Navigation vom Ganzen zu seinen Teilen möglich sein.

Ein weiterer Spezialfall ist die Komposition. Auch bei ihr muss eine Navigation vom Ganzen zu den Teilen existieren. Jedoch ist darauf zu achten, dass die Teile nie länger existieren können als das Ganze.

Wie die Assoziation und die Aggregation kann die Komposition über einen Zeiger implementiert werden. Da die Teile nie länger existieren dürfen als das Ganze, ist es jedoch von Vorteil die Aggregation über echtes „physisches Enthaltensein“ zu realisieren. Bei dieser Variante werden die Teile automatisch mit dem Ganzen erzeugt bzw. gelöscht. Dagegen muss bei der Realisierung mit einem Zeiger das Erzeugen und das Löschen der Teile durch den Kon- bzw. Destruktor durchgeführt werden.




```
class Ganzes
{
private:
        TeilEnthalten      einTeilEnthalten
        TeilZeiger         *einTeilZeiger
public:
        Ganzes( ){ einTeilZeiger = new TeilZeiger;}
        ~Ganzes( ){ delete einTeilZeiger}
};
```

Abb. 23: Beispiel für eine Aggregation

5.2 Mehr-Schichten-Architektur

Um ein Softwaresystem später besser realisieren und warten zu können, wird das System in drei logische Teile bzw. Schichten unterteilt. Analog zum OSI-Schichtenmodell können nur jeweils zwei unmittelbar benachbarte Schichten über fest definierte Schnittstellen miteinander kommunizieren. Die obererste Schicht, auch „GUI-Schicht“[28] genannt, stellt die unmittelbare Schnittstelle zum Benutzer der Anwendung dar. Diese Schicht ist absolut unabhängig von der Art der Oberfläche. Es kann sich z. B. um eine Web-Seite handeln, die Formularelemente von HTML enthält, oder auch um eine Einzelanwendung, die vom Web unabhängig läuft. Die darunter liegende Schicht, oftmals „Fachkonzeptschicht“ genannt, enthält die gesamte Logik der Software und stellt das Zentrum dar. Ihr ist es auch möglich, auf die unterste Schicht – „Datenhaltungsschicht“ – zuzugreifen. Bei der Datenhaltungsschicht handelt es sich in den meisten Fällen um gängige Datenbankmanagementsysteme, welche die Daten der Anwendungen aufnehmen. Den Vorteilen der bessern Wiederverwendbarkeit, Änderbarkeit, Wartbarkeit und Portabilität steht der Nachteil gegenüber, dass sich die Schichten-Architektur negativ

[28] Graphical User Interface

auf die Performance auswirken kann, da Daten von der GUI-Schicht nicht direkt in die Datenhaltungsschicht und umgekehrt gelangen können.

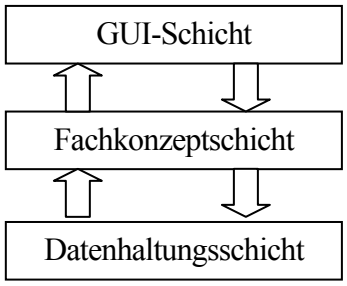


Abb. 24: Drei-Schichten-Architektur

Bei genauerer Betrachtung erfüllt die GUI-Schicht zwei unterschiedliche Aufgaben. Sie präsentiert einerseits die Informationen für den Benutzer und andererseits ist sie für die Kommunikation mit der Fachkonzeptschicht zuständig. Um eine bessere Austauschbarkeit zu erreichen, bietet es sich an, die GUI-Schicht nochmals aufzuteilen. In diesem Fall befasst sich die GUI-Schicht dann nur noch mit der Präsentation der Informationen und die zusätzliche Schicht „Fachkonzept-Zugriffsschicht" ist verantwortlich für alle Zugriffe auf die Fachkonzeptschicht. Neben ihrer eigentlichen Aufgabe - der Modellierung des Problembereichs – beinhaltet die Fachkonzeptschicht beim Drei-Schichten-Modell auch den Zugriff auf die Datenhaltung. Hier bietet sich eine zusätzliche Schicht die „Datenhaltungs-Zugriffsschicht" an, die die Fachkonzeptschicht mit Daten aus der Datenbank füllt und die Datenbank bei Änderungen aktualisiert. Bei einer solchen Aufteilung spricht man dann von einer Mehr-Schichten-Architektur (multi-tier architecture).

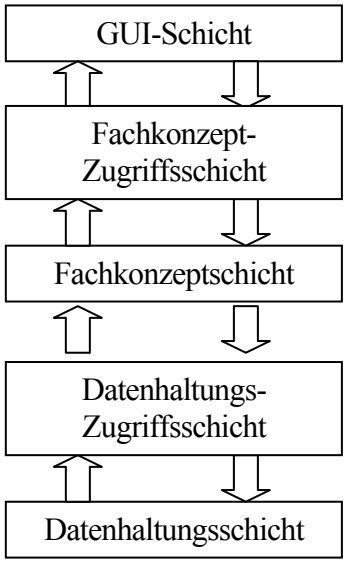


Abb. 25: Mehr-Schichten-Architektur

Teil III

Im dritten Teil der Examensarbeit werden nun die erarbeiteten Inhalte auf eine Unterrichtsreihe von achtzehn Einheiten á drei Schulstunden verteilt. Nach Beendigung der Unterrichtsreihe sollen die Schüler …

- verstehen, womit sich die Disziplin der Softwaretechnik beschäftigt,
- die historische Entwicklung der Vorgehensmodelle in der Softwareentwicklung nennen können,
- den Grundgedanken der objektorientierten Programmierung erläutern können,
- wissen, welche objektorientierten Konzepte es gibt,
- für vorgegebene Problemstellungen ein OOA-Modell systematisch erstellen und überprüfen können,
- Anwendungsfall-, Klassen-, Objekt- und Sequenzdiagramme in UML-Notation zeichnen können,
- CASE-Werkzeuge zur Darstellung der UML-Notation sowie zur Generierung von Anwendungen einsetzen können,
- vorgegebene Diagramme in die Programmiersprache C++ umsetzen können.

Die Beschreibung der Unterrichtsreihe gliedert sich in die Schwerpunkte „Lernziele“, „Verlaufsplanung“ und „Unterrichtsmaterialien“. Lösungen der einzelnen Übungsphasen, Arbeitsblätter und Klausuren befinden sich im Anhang oder auf der CD-ROM. Die Lernzielplanung beschränkt sich auf die kognitiven Lernziele. Sozial-affektive Lernziele werden nicht aufgeführt.

Den Abschluss der Unterrichtseinheit bildet eine dreiwöchige Projektphase. In ihr werden die in der Unterrichtsreihe erarbeiteten Diagramme zur Problemstellung „Wetterstation auf dem Felsberg“ aufgenommen, weiterentwickelt und implementiert. Im Mittelpunkt dieser Phase steht die Umsetzung der Problemstellung in der Programmiersprache C++. Die Umsetzung erfolgt selbstständig in Gruppen. Auf der CD-ROM im Verzeichnis „Musterlösung des Projekts“ befindet sich ein Lösungsvorschlag für das Projekt. Er kann nur zeigen, wie eine Lösung aussehen kann, da der Umfang des Projekts dem Leistungsniveau der Klasse angepasst werden muss.

6. Hinführung zur objektorientierten Modellierung

6.1. Prozessmodelle

6.1.1. Lernziele

Die Schüler sollen …

- die Inhalte des Studiengangs „Informatik“ kennen,
- die Inhalte der Leistungs- sowie Grundkurse der Jahrgangsstufe 12 und 13 kennen,
- Veränderungen, denen die Softwareentwicklung in den letzten zehn Jahren unterworfen war, erörtern können,
- die Faktoren, die Entwicklung von Software zusätzlich erschweren, nennen können,
- erklären können, was ein Prozessmodell ist und was in ihm definiert wird,
- die Vorgehensweise beim Planungsprozess und die jeweils durchzuführenden Aktivitäten erläutern können,
- erklären können, welche Ergebnisse in den Phasen des Wasserfall- und des Spiralmodells entstehen,
- erklären können, worin sich das Wasserfall- vom Spiralmodell unterscheidet.

6.1.2. Verlaufsplanung

1. UNTERRICHTSEINHEIT		
Phase/Zeit	**Inhalt/Beschreibung**	**Medien/ Sozialformen**
Einstieg ca. 45`	• Vorstellung der Inhalte des Studiengangs „Informatik“ an der TUD, FHD , FH Gießen und Fernuniversität Hagen • Vorstellung der Inhalte der Grund- und Leistungskurse in der Qualifikationsphase an der KKS Bensheim • Vergleich der Inhalte	Homepage der Fachhochschulen und Universitäten LV[29]
Hinführung ca. 15`  ca. 10`	• Einordnung bzw. Abgrenzung der Softwaretechnik als Teilgebiet der Informatik • Was versteht man unter einem Prozessmodell?	F1[30] LV T1[31] FR-ENT[32]

[29] LV Lehrervortrag

[30] F Folie

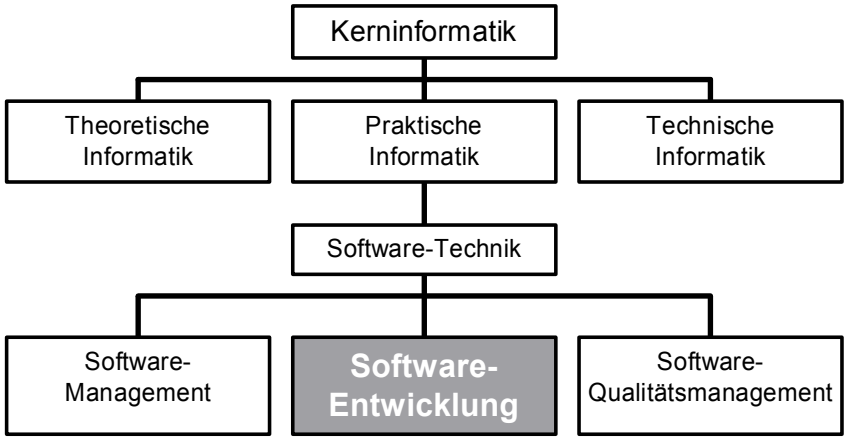
[31] T Tafelbild

[32] FR-ENT fragend entwickelnd

Erarbeitung ca. 45`	• Erarbeitung der Merkmale des - „Urmodells“ - Wasserfallmodells - Spiralmodells	Internet AB1[33] GA[34]
Festigung ca. 20`	• Diskussion der Ergebnisse	F2-F4 SSG[35]

6.1.3. Unterrichtsmaterialien

Die Softwaretechnik als Teilgebiet der Informatik



F1

Was versteht man unter einem Prozessmodell?

Merke:

Ein Prozessmodell ist eine Beschreibung einer koordinierten Vorgehensweise bei der Softwareentwicklung. Es definiert sowohl den Input, der zur Abwicklung der Aktivität notwendig ist, als auch den Output, der als Ergebnis der Aktivität produziert wird.

Bekannte Prozessmodelle sind:

- Wasserfallmodell
- Spiralmodell
- V-Modell
- OO-Modell

T1

[33] AB Arbeitsblatt

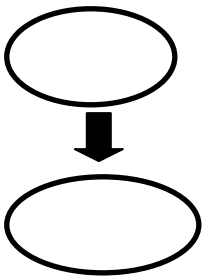
[34] GA Gruppenarbeit

[35] SSG Schüler-Schüler-Gespräch

Aufgabe:

- Erarbeiten Sie in Gruppenarbeit die Merkmale, die Vor- sowie die Nachteile der aufgeführten Prozessmodelle.
- Diskutieren Sie diese in der Gruppe und halten Sie ihre Ergebnisse auf dem Arbeitsblatt fest. Zum Vergleich der Gruppenergebnisse übertragen Sie bitte zusätzlich Ihre Lösung auf die OH-Folie.
- Als Hilfe zur Erarbeitung steht Ihnen das Internet zur Verfügung.

„Urmodell“:



Merkmale:

- .....

Vorteile:

- .....
- .....

Nachteile:

- .....
- .....
- .....

Wasserfallmodell:

Merkmale:

- .....
- .....
- .....

Vorteile:

- .....
- .....

Nachteile:

- .....
- .....
- .....

Spiralmodell:

Eigenschaften:

- .....
- .....
- ....

Vorteile:

- .....
- .....
- .....
- .....

Nachteile:

- .....
- .....

Thema: Der Entwicklungsprozess	AB1
	© W. Steffens

7. Phasen und Produkte des objektorientierten Modells

7.1. Das Pflichtenheft

7.1.1. Lernziele

Die Schüler sollen …

- die Prozessschritte der einzelnen Phasen des objektorientierten Prozessmodells erklären können,
- wissen, welche Produkte in der Phase der Analyse und des Entwurfs zu erstellen sind,
- die Funktion eines Pflichtenhefts erläutern können,
- für vorgegebene Aufgabenstellungen ein Pflichtenheft entsprechend dem beschriebenen Pflichtenheft-Muster erstellen können.

7.1.2. Verlaufsplanung

2. UNTERRICHTSEINHEIT		
Phase/Zeit	**Inhalt/Beschreibung**	**Medien / Sozialformen**
Einstieg ca. 5`	• Brainstorming zum Thema „objektorientierte Modellierung“	LSG[36]
Erarbeitung I ca. 10`	• Warum hat sich die objektorientierte Modellierung durchgesetzt? • Phasen des objektorientierten Prozessmodells	T1 F1, F2 LV
Festigung I ca. 20`  ca. 10`	• Erstellung einer Übersicht über die Aufgabenstellungen, Produkte und Ziele der einzelnen Phasen • Diskussion der Ergebnisse	AB2, Internet GA SSG
Erarbeitung II ca. 10`	• Aufgabe und Gliederung des Pflichtenhefts	T2 LV
Festigung II ca. 50`  ca. 15`	• Erstellung eines Pflichtenhefts für das unterrichtsbegleitende Projekt „Wetterstation auf dem Felsberg“ • Diskussion der Ergebnisse	AB3 GA
Abschluss ca. 15`	• Einigung auf ein gemeinsames Pflichtenheft für die weitere Planung	LSG
Hausaufgabe	• Wiederholung: Aufgabenstellungen, Produkte und Ziele der einzelnen Phasen der objektorientierten Modellierung	

[36] LSG Lehrer-Schüler-Gespräch

7.1.3. Unterrichtsmaterialien

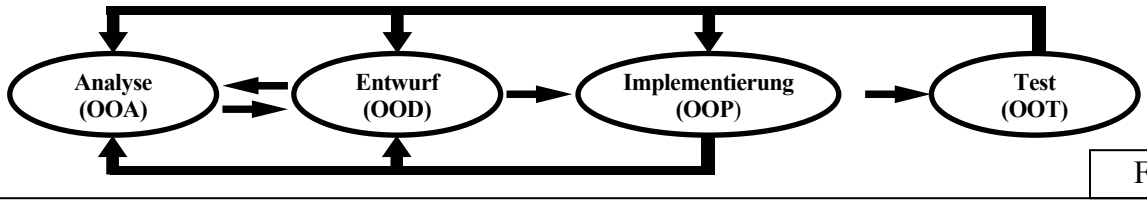
Warum objektorientierte Modellierung?

Vorteile der objektorientierten Modellierung:

- Verkürzung der Entwicklungszeit
- Senkung der Fehlerrate
- verbesserte Erweiterbarkeit und Anpassungsfähigkeit

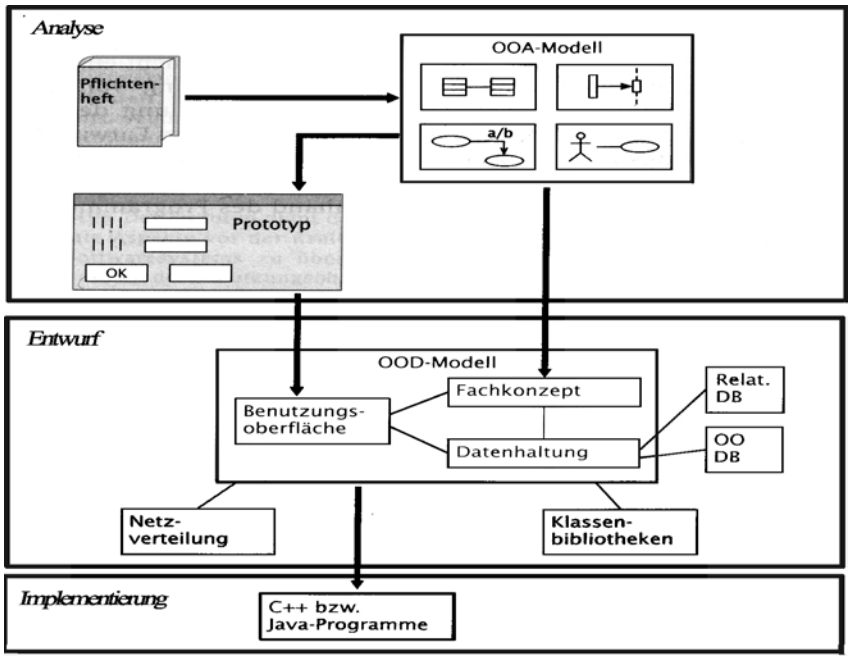
T1

Phasen der objektorientierten Modellierung



F1

Produkte der einzelnen Phasen[37]



F2

Das Pflichtenheft

- Ausgangsbasis für die objektorientierte Modellbildung
- Zusammenfassung aller fachlichen Anforderungen aus der Sicht des Auftraggebers
- Adressaten sind Auftraggeber, Auftragnehmer und evtl. Benutzer
- Beschreibung des „Was“ nicht des „Wie“
- Vertragliche Grundlage zwischen Auftraggeber und Auftragnehmer
- Grundlage der Systemabnahme
- Gliederungsschema Pflichtenheft:
  1. Zielbestimmung
  2. Produkteinsatz
  3. Produktübersicht
  4. Produktfunktionen
  5. Produktdaten
  6. Produktleistungen
  7. Benutzeroberfläche
  8. nichtfunktionale Anforderungen
  9. technische Produktumgebung

T2

[37] vgl. Balzert, Heide S. 13.

Aufgabe:

- Erarbeiten Sie in Gruppenarbeit die Aufgabenstellungen, die Produkte sowie die Ziele der einzelnen Phasen des objektorientierten Prozessmodells.
- Diskutieren Sie diese in der Gruppe und halten Sie ihre Ergebnisse auf dem Arbeitsblatt fest. Zum Vergleich der Gruppenergebnisse übertragen Sie bitte zusätzlich Ihre Ergebnisse auf die OH-Folie.
- Als Hilfe zur Erarbeitung steht Ihnen das Internet zur Verfügung.

Objektorientierte Analyse

Aufgabenstellungen:

- .....
- .....
- .....
- .....

Produkte:

- .....
- .....
- .....
- .....

Ziel: .....

Objektorientierter Entwurf

Aufgabenstellungen:

- .....
- .....

Produkte:

- .....
- .....
- .....
- .....

Ziel: .....

Objektorientierte Implementierung

Aufgabenstellungen:

- .....

Produkte:

- .....

Ziel: .....

Objektorientierter Test

Aufgabenstellungen:

- .....

Produkte:

- .....

Ziel: .....

Problembeschreibung:

Das meteorologische Institut „Schön Wetter“ möchte eine Wetterstation zur Messung von Windrichtung, Windgeschwindigkeit, Luftdruck, Luftfeuchtigkeit sowie der Temperatur auf dem Felsberg in Bensheim einrichten. Vorort befindet sich bereits eine Wetterstation, die vom Institut „Schön Wetter“ aufgekauft wurde. Die vorhandene Hard- und Software soll, wenn möglich, weiter verwendet werden.

Grundsätzlich soll es möglich sein, über eine Internetverbindung auf die gemessenen Daten sowie - mit Ausnahme der Windrichtung - auf ihr 24h-Minimum und 24h-Maximum zuzugreifen. Die Abfrage soll periodisch oder auf Wunsch des Benutzers individuell erfolgen.

Zurzeit werden von der Zentrale in Darmstadt die Sturm- bzw. Glatteiswarnungen manuell ausgelöst. Es wäre wünschenswert, wenn dieser Vorgang automatisiert werden könnte. Hierzu ist es notwendig, die Windgeschwindigkeit und Temperatur zu prognostizieren. Um eine Prognose aufstellen zu können, werden die Daten der Windgeschwindigkeit und Temperatur der letzten 24 Stunden benötigt. Bei jeder neuen Messung der Windgeschwindigkeit und Temperatur soll die Prognose aktualisiert werden. Sofern sich bei der Windgeschwindigkeit ein Prognosewert von mehr als 70 km/h ergibt und gleichzeitig der aktuelle Luftdruck um mehr als 15 Millibar unter den Maximalwert der letzten 24 Stunden gefallen ist, ist ein Sturmwarnungsalarm auszulösen. Eine Glatteiswarnung ist zu geben, wenn bei einer negativen Temperaturprognose der aktuelle Luftfeuchtigkeitswert mindestens 90% beträgt.

Wird eine Warnung ausgelöst, soll diese mit der Angabe des Prognosewertes, dem Benutzernamen desjenigen, der die Warnmeldung ausgelöst hat, und dem Zeitpunkt, an dem sie ausgelöst wurde, protokolliert werden.

Das Programm soll einen Passwortschutz besitzen. Starten darf das Programm jeder. Privilegierte Benutzer können Warnmeldungen auslösen und ihre Protokollierung ansehen.

Der PC der Wetterstation auf dem Felsberg ist folgendermaßen ausgestattet: Am COM-Port des PCs ist ein PC-Wettersensorempfänger angeschlossen. Über ihn können die gewünschten Messwerte abgefragt werden. Weiterhin besitzt der Rechner eine Netzwerkkarte, mit der er dauerhaft an das Internet angebunden ist. Das Betriebssystem des Rechners ist Windows 2000. Bis auf das Office-Paket befindet sich keine weitere Software auf dem Rechner.

Für Fragen steht Ihnen unser Mitarbeiter Herr Müller zur Verfügung.

Aufgabe:

Erstellen Sie ein Pflichtenheft.

7.2. Das Anwendungsfalldiagramm

7.2.1. Lernziele

Die Schüler sollen …

- wissen, wozu man die Unified Modelling Language einsetzen kann,
- erklären können, zu welchem Zweck ein Anwendungsfalldiagramm erstellt wird,
- Beziehungen zwischen Anwendungsfällen erklären können,
- Anwendungsfalldiagramme mit Hilfe eines CASE-Werkzeuges dokumentieren können,
- einen Einblick über die Anwendungsmöglichkeiten der Software Jumli besitzen,
- erklären können, was eine Anwendungsfallschablone ist,
- Anwendungsfallschablonen systematisch erstellen können.

7.2.2. Verlaufsplanung

3. UNTERRICHTSEINHEIT		
Phase/Zeit	**Inhalt/Beschreibung**	**Medien/ Sozialformen**
Einstieg ca. 5`	• Einführung in die Thematik: Wie kann die Funktionalität einer zu entwickelnden Software beschrieben werden, so dass der Kunde in den Entwicklungsprozess mit einbezogen werden kann?	FR-ENT
Erarbeitung I ca. 10`  ca. 10`	• Die Unified Modelling Language als Darstellungsmöglichkeit der Ergebnisse bei der objektorientierten Modellierung • Das Anwendungsfalldiagramm - Ausgangspunkt - Ziel - Notation	 T1 T2 LV
Festigung I ca. 30`     ca. 25` ca. 15`	• Erstellung eines Anwendungsfalldiagramms für das Projekt „Wetterstation auf dem Felsberg“ - Bestimmung der Akteure und Systemgrenzen - Identifizierung der Anwendungsfälle - Erstellung eines Anwendungsfalldiagramms • Diskussion der Ergebnisse • Einigung auf ein gemeinsames Anwendungsfalldiagramm für die weitere Planung	 F1 GA SSG SSG

Erarbeitung II ca. 10`	• Beziehungen zwischen Anwendungsfällen - include-Beziehung - extend-Beziehung - Generalisierung	T3 LV
Festigung II ca. 25`   ca. 10`	• Überarbeitung der Anwendungsfälle - Benutzer verwalten (Server) - Messwerte abfragen (Server) - Einstellungen vornehmen (Client) • Diskussion der Ergebnisse	 GA SSG
Hausaufgabe	• Erstellung von Anwendungsfalldiagrammen	AB4

4. UNTERRICHTSEINHEIT		
Phase/Zeit	**Inhalt/Beschreibung**	**Medien/ Sozialformen**
Einstieg ca. 15`	Kontrolle der Hausaufgabe	AB4 FR-ENT
Erarbeitung I ca. 10`	• Einführung in das Software-Tool Jumli - Installation - Bedienung	Jumli Beamer LV
Festigung I ca. 20`	• Darstellung des Anwendungsfalldiagramms für die Wetterstation aus der vorherigen Unterrichtseinheit mit der Software Jumli	Jumli GA
Erarbeitung II ca. 10`	• Erstellung einer Anwendungsfallschablone	T4 LV
Festigung II ca. 20` ca. 15`	• Erstellung von Anwendungsfallschablonen für das unterrichtsbegleitende Projekt „Wetterstation auf dem Felsberg“ • Diskussion der Ergebnisse	GA SSG
Wiederholung ca. 30`     ca. 15`	• Abschlussbeispiel: - Bestimmung der Akteure des Systems - Bestimmung der Anwendungsfälle des Systems - Beschreibung der Anwendungsfälle an Hand einer Anwendungsfallschablone - Erstellung eines Anwendungsfalldiagramms • Diskussion der Ergebnisse	 AB5 PA[38] SSG
Hausaufgabe	• Informationsbeschaffung zum Visual Studio 7 Ressourcen Editor	

[38] PA Partnerarbeit

7.2.3. Unterrichtsmaterialien

Unified Modelling Language UML

Abgrenzung:

- UML ist kein Software-Entwicklungsprozess, sondern beschreibt Diagramme und Notationen.
- UML ist keine visuelle Programmiersprache, sondern eine Entwurfsprache.

Merkmale:

- UML ist allgemein verwendbar --> sprachen- und prozessunabhängig.
- UML besitzt eine klare Semantik --> formale Basis.

Ziel:

- UML soll Anwendern helfen, aussagekräftige Modelle zu erstellen und damit den Entwicklungsprozess zu unterstützen.

T1

Anwendungsfalldiagramm

Ausgangspunkt: informelle, knappe Problembeschreibung oder ein Pflichtenheft

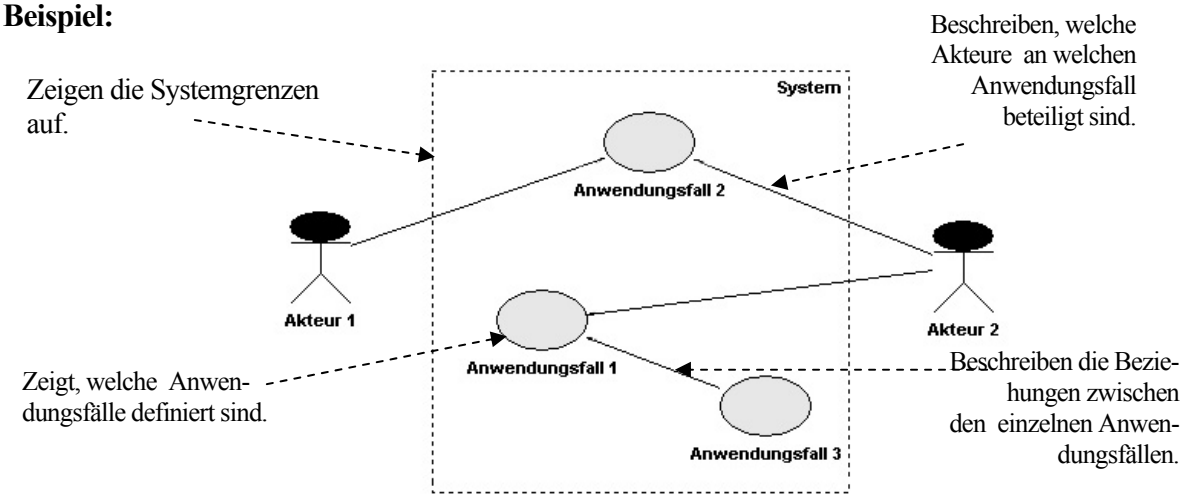
Ziel:

- Beschreibung der gewünschten Funktionalität der zu entwickelnden Software aus externer Sicht.
- Dient als Kommunikationsmedium zwischen Entwickler und Kunde.

Notation:

- Akteure (Aktoren):
  - tauschen von außen Informationen mit dem System aus,
  - werden durch die Rolle, die ein Benutzer gegenüber dem System einnimmt, charakterisiert,
  - können Benutzer, andere Systeme oder Geräte sein.
- **Anwendungsfall (Use Cases)**
  - beschreibt die Interaktionen zwischen einem oder mehreren Akteuren und dem System bei der Bearbeitung einer bestimmten, abgegrenzten Aufgabe.

Beispiel:



T2

Arbeitsauftrag:

Es soll ein Anwendungsfalldiagramm für die Wetterstation entwickelt werden. Gruppieren Sie sich zu dritt oder zu viert. Als Grundlage dienen das von Ihnen entwickelte Pflichtenheft sowie die Problembeschreibung auf dem Arbeitsblatt 3.

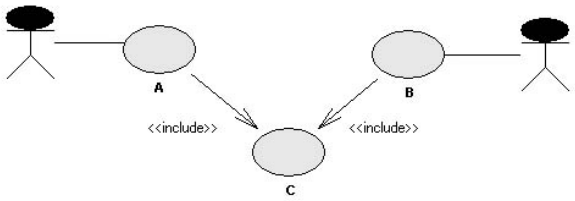
Bei der Entwicklung des Diagramms sind folgende Schritte durchzuführen:

a) Bestimmung der Akteure und Systemgrenzen
b) Identifizierung der Anwendungsfälle
c) Erstellung eines Anwendungsfalldiagramms

F1

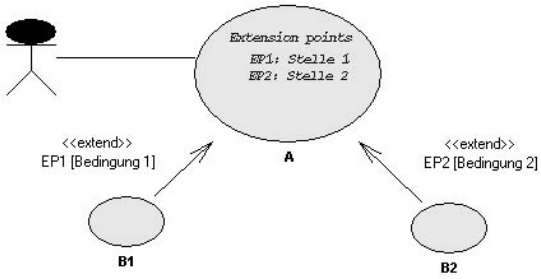
Beziehungen zwischen Anwendungsfällen

- include-Beziehung (Enthältbeziehung)



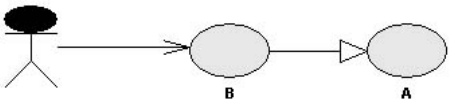
Jede Ausführung des Anwendungsfalls A bzw. B beinhaltet als Teilaufgabe notwendigerweise eine Ausführung von C.

- extend-Beziehung (Erweiterungsbeziehung)



A wird um zusätzliches mögliches Verhalten erweitert, falls die angegebene Bedingung erfüllt ist

- Generalisierung



A ist eine Generalisierung von B. Dies bedeutet, dass B das Verhalten von A erbt.

T3

Eine Anwendungsfallschablone besteht aus:

- Name des Anwendungsfalls
- Kurzbeschreibung
- Vorbedingung: Voraussetzung für eine erfolgreiche Ausführung des Anwendungsfalls
- Nachbedingung: Zustand nach erfolgreicher Ausführung
- Standardablauf (Primärszenario): Schritte bzw. Interaktionen, die im Normalfall bei Ausführung des Anwendungsfalls durchlaufen werden
- Alternativabläufe (Sekundärszenarien): bei Fehlerfällen und Optionen

T4

Aufgabe 1:
Erklären Sie die Begriffe „Anwendungsfalldiagramm“, „Anwendungsfall“ und „Akteur“.

Aufgabe 2:

Projektbeschreibung
Ihre Firma erhält den Auftrag, Software für einen Geldautomaten zu entwickeln. Der Benutzer startet eine Sitzung, indem er die Geldkarte in den Kartenleser einschiebt. Daraufhin verlangt das System einen vierstelligen Code zur Identifikation. Die Bedienung des Automaten erfolgt über eine Tastatur. Nach der Identifikation stehen dem Benutzer folgende Transaktionsmöglichkeiten zur Auswahl:

- Er kann eine Einzahlung durchführen. Nach der Einzahlung gibt der Automat eine Quittung aus, welche das Einzahlungsdatum, den eingezahlten Betrag sowie die Kontonummer enthält.
- Er kann pro Benutzung bis zu 400,00 € abheben. Auch hier erhält er einen Ausdruck, der das Datum der Auszahlung, den ausgezahlten Betrag sowie die Kontonummer mit dem neuen Kontostand enthält. Dabei darf das Konto nicht um mehr als 1000,00 € überzogen werden.
- Er kann sich den Kontostand anzeigen und ausdrucken lassen.
- Er kann Geld von dem eigenen Konto auf ein anderes Konto überweisen. Hierbei darf ebenfalls sein Konto nicht um mehr als 1000,00 € überzogen werden.
- Er hat die Möglichkeit, eine Sitzung jederzeit zu beenden und erhält dann die Karte vom Automaten zurück.
- Kann eine Transaktion nicht durchgeführt werden, so erhält er eine Meldung und kann dann mit einer der anderen Funktionen weitermachen.

Nennen Sie die wesentlichen Anwendungsfälle und Akteure der obigen Problemstellung und stellen Sie diese in einem Anwendungsfalldiagramm dar.

Aufgabe 3:

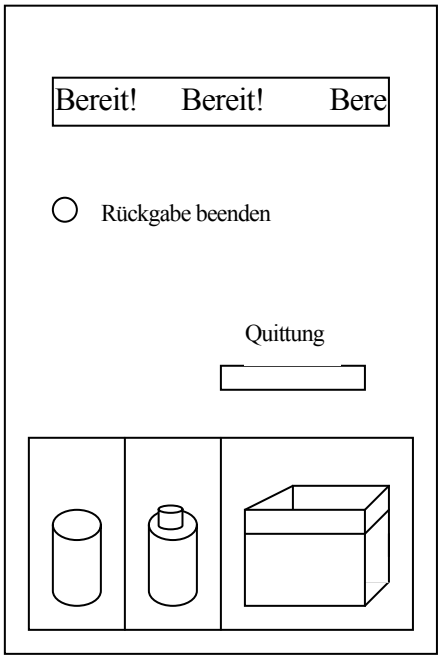
Projektbeschreibung
Der Buchhandel „Mein Buch“ will seinen Verkauf modernisieren. Sie sollen die Kundenkartei von Mein Buch durch ein neu zu entwickelndes Softwaresystem ersetzen. In einem ersten Treffen wurden die folgenden Sachverhalte und Anforderungen identifiziert:

- Mein Buch hat bereits ein EDV-System im Einsatz. Dieses soll an das von Ihnen zu entwickelnde System angeschlossen werden.
- Das zu entwickelnde System soll dem Mitarbeiter erlauben, einem Kunden Angebote zu unterbreiten. Die Angebote werden dann über das alte EDV-System an den Kunden weitergegeben. Ein Kunde kann daraufhin sowohl online über das angeschlossene EDV-System bestellen oder auch per Fax. Die Bestellungen werden von Hand eingegeben.
- Mitarbeiter können neue Kundendateien anlegen und bestehende aktualisieren. Es gibt Geschäfts- und Privatkunden. Nur Geschäftskunden ist ein zuständiger Sachbearbeiter zugeordnet.
- Die Kunden selbst sollen online Zugriff auf die über sie gespeicherten Daten bekommen.
- Zu jeder Bestellung wird eine Liste von Einzelposten gespeichert.
- Um herauszufinden, welche Autoren besonders beliebt sind, wird regelmäßig und vollautomatisch ein Verkaufsbericht erstellt und ausgedruckt. Diese Berichte werden nicht gespeichert, da sie jederzeit reproduzierbar sind.

Nennen Sie die wesentlichen Anwendungsfälle und Akteure der obigen Problemstellung und stellen Sie diese in einem Anwendungsfalldiagramm dar.

Aufgabe:

Projektbeschreibung:
Die folgenden Anforderungen beschreiben einen einfachen Automaten zur Rückgabe von Pfandflaschen, Getränkedosen und Getränkekisten, der objektorientiert zu modellieren ist.
Ein Kunde kann dabei in beliebiger Reihenfolge Pfandflaschen, Getränkedosen und leere Getränkekisten zurückgeben. Dies können verschiedene Typen von Flaschen, Kisten und Dosen sein. Daher muss das System bei jedem zurückgegebenen Gegenstand prüfen, um was für einen Typ es sich handelt, und den Wert der zurückgegebenen Gegenstände erfassen. Nachdem der Kunde den „Beenden"-Knopf gedrückt hat, muss ein entsprechender Beleg ausgedruckt werden.
Gewartet wird das System von einem Servicetechniker. Dieser will wissen, wie viele Gegenstände eines bestimmten Typs und wie viele Gegenstände insgesamt im Laufe des Tages zurückgegeben wurden. Der Servicetechniker sollte auch in der Lage sein, bestimmte Informationen in dem System zu ändern, wie z. B. den Pfandwert der Gegenstände. Bei einer Funktionsstörung, z. B. etwas ist verklemmt oder das Papier für die Belege ist ausgegangen, soll der Servicetechniker durch ein Alarmsignal gerufen werden.



a) Finden Sie die Akteure des Systems
b) Finden Sie die Anwendungsfälle des Systems und erstellen Sie eine Anwendungsfallschablone
c) Stellen Sie das Ganze in einem Anwendungsfalldiagramm dar

Thema: Erstellung von Anwendungsfalldiagrammen | AB5 | *© W. Steffens*

7.3. Der Prototyp der Benutzeroberfläche

7.3.1. Lernziele

Die Schüler sollen …

- begründen können, für welche Anwendungen eine Einzel- (SDI), eine Mehrfachdokumentoberfläche (MDI) oder eine dialogbasierte Oberfläche eingesetzt wird,
- für die Gestaltung von Oberflächen geeignete Steuerelemente auswählen können,
- modale und nicht-modale Dialoge entsprechend der Aufgabenstellung einsetzen können,
- mit dem MFC-Ressourceneditor Oberflächen gestalten können.

7.3.2. Verlaufsplanung

5. UNTERRICHTSEINHEIT		
Phase/Zeit	**Inhalt/Beschreibung**	**Medien/ Sozialformen**
Einstieg ca. 15`	• Vorstellung der Funktionen des Ressourceneditors	Beamer ‚PC LV
Festigung ca. 60 ca. 15 ca. 15	• Gestaltung der Oberflächen für die - Clientanwendung (zwei Gruppen) - Serveranwendung (zwei Gruppen) • Diskussion der Ergebnisse • Einigung über eine einheitliche Oberfläche für den Server und Clienten	F1 GA SSG SSG
Hausaufgabe	• Informationsbeschaffung: Was ist ein Objekt?	

7.3.3. Unterrichtsmaterialien

Arbeitsauftrag:

Es sollen das Menü und die Dialoge der Oberflächen für die Wetterstation entwickelt werden. Gruppieren Sie sich zu dritt oder zu viert. Als Grundlage dienen das von Ihnen entwickelte Pflichtenheft sowie die Problembeschreibung auf dem Arbeitsblatt 3.

Hilfe:

- Welche Vorgaben macht das Pflichtenheft?
- Welche Ein- und Ausgaben hat das Programm?
- Wie lassen sich die Ein- und Ausgaben strukturieren?

.

F1

8. Basiskonzepte

8.1. Klassen, Objekt, Attribute und Methoden in UML-Notation

8.1.1. Lernziele

Die Schüler sollen,

- den Begriff „objektorientierte“ Software-Entwicklung erklären können,
- die Begriffe „Klasse“, „Objekt“, „Attribut“ und „Methoden“ anhand von Beispielen erklären können,
- ein Klassen- von einem Objektattribut unterscheiden können,
- eine Klassen- von einer Objektmethode unterscheiden können,
- Attribute und Methoden in einem Text identifizieren und geeignete Klassen modellieren können,
- die UML-Notation für Objekte, Klassen, Attribute und Methoden anwenden können,
- das Prinzip der Kapselung erklären können,
- Zugriffsrechte spezifizieren können,
- ein CASE-Werkzeug für das Erstellen von Klassen und Objekten einsetzen können.

8.1.2. Verlaufsplanung

6. UNTERRICHTSEINHEIT		
Phase/Zeit	**Inhalt/Beschreibung**	**Medien/ Sozialformen**
Einstieg ca. 5`	• Was ist ein Objekt? - Definition	T1 FR-ENT
Erarbeitung I ca. 20`	- Attribute - Methoden - Darstellung eines Objektes in UML-Notation	T2 T3 T4 FR-ENT
Festigung I ca. 10	• Übung: Bestimmung von Objekten in UML-Notation	F1 FR-ENT
Erarbeitung II ca. 30`	• Was ist eine Klasse? - Definition - Darstellung eines Objektes in UML-Notation - Datenkapselung - Zugriffsrechte	F2 F3 T5 T6 FR-ENT

Festigung II ca. 30` ca. 15`	• Bestimmung von Klassen in UML-Notation • Diskussion der Ergebnisse	AB6 PA SSG
Abschluss ca. 10`	• Darstellung von Klassen in Jumli	PC, Beamer LV
Hausaufgabe	• Installation der Software Jumli zu Hause • Erfahrungen sammeln im Umgang mit der Software	

8.1.3. Unterrichtsmaterialien

Was ist ein Objekt?

Reale Welt

Modell

Computer PC 01

Benutzer Hugo

Definition:

Unter einem Objekt versteht man die softwaretechnische Repräsentation eines realen oder gedachten, klar abgegrenzten Gegenstands oder Begriffs eines Anwendungsgebietes.

T1

Was ist ein Attribut?

Definition:

Die Attribute eines Objekts enthalten Daten, die erforderlich sind, um den jeweiligen Zustand des Objekts vollständig und konsistent zu beschreiben. Die Attribute modellieren so die statischen Aspekte des Objekts.

z. B.:	Benutzer:		PC:	
	Name :	Hugo	Zustand:	an
	Passwort:	geheim	Prozessor:	PIII 2000MHz
	Rechte:	keine Rechte	Programme:	Word

T2

Was ist eine Methode?

Definition:

Die Methoden eines Objekts beschreiben alle Operationen, die das Objekt ausführen kann, und modellieren so das dynamische Verhalten des Objekts.

z. B:	Benutzer:	PC:
	ändere Rechte	beende Programm
	ändere Name	starte Programm
	ändere Passwort	ändere Zustand

T3

Darstellung eines Objektes in UML-Notation

:einBenutzer	
name:	Hugo
passwort:	geheim
rechte :	keine

In der UML-Notation werden für ein Objekt nur die Attribute und nicht die Methoden dargestellt.

T4

Übung

Ausgangssituation:
Die Fluglinie Lufthansa hat u. a. einen Piloten namens „Max Meier“ (Gehalt 14 000,00 €) und u. a. zwei Flugbegleiter namens „Karl Schmidt“ und „Markus Müller“ (Gehalt jeweils 2 700, 00 €). Am 28.4.2004 findet ein Hin- und Rückflug (Flugnummern 321 und 322) von München nach Helsinki statt. Bei diesem Flug ist Max Meier der Pilot und Markus Müller der Flugbegleiter. Der Grundpreis für eine Strecke beträgt 99,00 €. Als Flugzeug wird eine Boing 707 eingesetzt. Sie hat ein Fassungsvermögen von 120 Personen und einen Stauraum von 400m^2. Erbaut wurde sie 1970.

Geben Sie alle Objekte in UML-Notation an.

F1

Was ist eine Klasse?



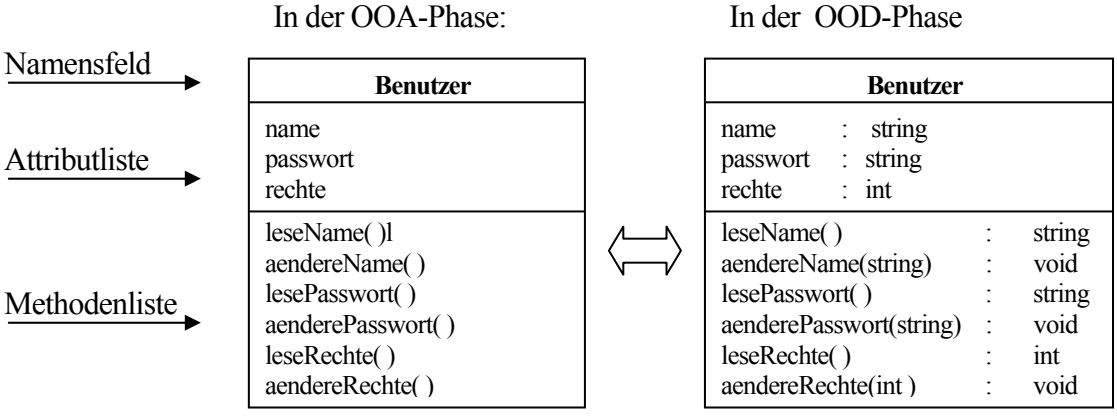
Hugo Franz Eva

Eine Klasse beschreibt die Struktur und das Verhalten einer Menge gleichartiger Objekte.

Alle Benutzer besitzen die gleichen Attribute (Struktur): Name, Passwort und Rechte.
Die Attribute können bei allen Benutzern mit den gleichen Methoden (Verhalten) verändert werden: Name ändern, Passwort ändern und Rechte ändern.

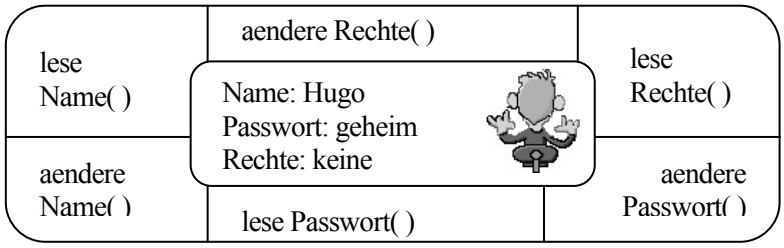
F2

Klasse in UML-Notation



T4

Datenkapselung



Der entscheidende Unterschied zwischen einer Struktur und einer Klasse beseht in der Datenkapselung. Auf die Attribute des Objekts kann nur über die Methoden des Objekts zugegriffen werden. Auf diese Weise stellt das Objekt sicher, dass es immer einen konsistenten Zustand hat, der von außen nicht korrumpiert werden kann.

T5

Zugriffsrechte

Um die Datenkapselung zu gewährleisten besitzt die objektorientierte Programmierung drei Zugriffsrechte.

- public: Attribute und Methoden sind für alle Klassen und Funktionen erreichbar.
- private: Attribute und Methoden nur für die Klasse selbst verfügbar.
- protected: Erklärung erfolgt später!!!

Darstellung in UML-Notation:

- public: +
- private: -
- protected: #

Beispiel:

Benutzer	
- name : string - passwort : string - rechte : int	
+ leseName( )	: string
+ aendereName(string)	: void
+ lesePasswort( )	: string
+ aenderePasswort(string)	: void
+ leseRechte( )	: int
+ aendereRechte(int)	: void

T6

Aufgabe 1:

Eine Getränkedose ist ein Behälter, der mit einer bestimmten Menge Flüssigkeit gefüllt werden kann. Eine Dose wird bei der Erzeugung befüllt. Danach kann sie vom Konsumenten geöffnet, ausgetrunken und weggeworfen werden. Modellieren Sie eine Klasse Dose in UML-Notation.

Aufgabe 2:

Modellieren Sie eine Klasse Uhrzeit in UML-Notation (mit Datentyp der Attribute). Die Uhrzeit bestehend aus Stunden (0-23), Minuten (0-59) und Sekunden (0-59). Ein Benutzer der Klasse darf die Stunden, Minuten und Sekunden eines Objektes abfragen und eine Uhrzeit neu setzen. Jedoch muss die Uhrzeit immer gültige Werte enthalten, es soll also keine Möglichkeit geben, dass die Uhrzeit ungültige Werte aufweist.

Aufgabe 3:

a) Modellieren Sie die Klassen Kunde und Konto in UML-Notation. Gegeben ist folgender Pflichtenheftauszug:
   - ein Kunde besitzt einen Namen, ein Geburtsdatum und einen Wohnort.
   - zu seinem Konto gehören Kontonummer und Kontostand.

b) Geben Sie für folgende Beschreibung die Objekte an. Der Kunde Norbert Meier verfügt über ein Konto mit der Nummer 0815 mit einem Guthaben von 4000,00 €.

Aufgabe 4:

Für das Projekt Wetterstation sind folgende Klassen in UML-Notation zu modellieren.

a) Zur temporären Datenhaltung der Einstellungen des Clients soll die Klasse CAppSettings modelliert werden. Sie speichert die Triggerdauer, die IP-Adresse des Servers und die Portnummer des Zugangs, auf dem der Server auf Anfragen wartet.

b) Zur temporären Datenhaltung der Einstellungen des Servers soll die Klasse CAppSettings modelliert werden. Die Klasse speichert:
   - die Triggerdauer,
   - die Baudrate, Bytegröße, Anzahl der Stopbits, das Paritybit sowie das Interface der COM-Schnittstelle,
   - die Namen der Benutzer-, Messwert- und Logdatenbank,
   - sowie die Portnummer, über die der Server befragt werden kann.

   Hilfe: Überlegen Sie, ob eine vorherige Strukturierung der Daten sinnvoll erscheint!

Aufgabe 5:

Entwickeln Sie aus den gegebenen Objekten die dazugehörigen Klassen!

Person3
nachname : Müller
vorname : Markus
funktion : Flugbegleiter
gehalt : 2700

:Flug2
nummer : 322
von : Helsinki
nach : München
am : 28.04.2204
preis : 99

:Flugzeug1
typ : Boing 707
sitzplaetze : 120
stauraum : 400
baujahr : 1970

8.2. Klassen, Objekt, Attribute und Methoden in C++

8.2.1. Lernziele

Die Schüler sollen …

- erklären können, wie die Konzepte „Objekt“, „Klasse“, „Attribut“ und „Methode“ in C++ umgesetzt werden,
- eine Klassendefinition in der Programmiersprache C++ erklären können,
- den Unterschied zwischen einer Deklaration und einer Definition erklären können,
- die Vor- und Nachteile unterschiedlicher Arten der Implementierung der Methoden begründen können,
- die Aufgaben der Konstruktor- und Destruktormethode erklären können,
- den Unterschied zwischen Objekt- und Klassenattributen erklären können,
- vorgegebene Klassen in UML-Notation in die Programmiersprache C++ überführen können.

8.2.2. Verlaufsplanung

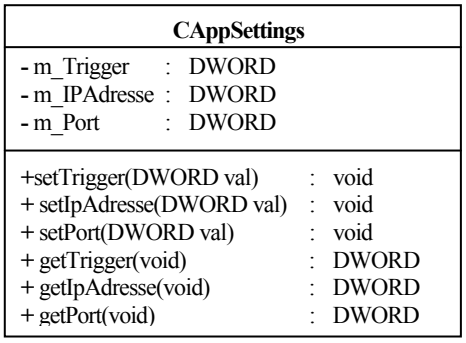
7. UNTERRICHTSEINHEIT		
Phase/Zeit	**Inhalt/Beschreibung**	**Medien/ Sozialformen**
Einstieg ca. 5`	• Einführung in die Thematik: Wie sieht die Syntax einer Struktur[39] in C++ aus?	FR-ENT
Erarbeitung ca. 10` ca. 20` ca. 10`	• Syntax der Klassendeklaration • Implementierung der Methoden - Deklaration und Definition in der Klasse - Deklaration und Definition in der Header-Datei - Trennung zwischen Definition und Deklaration • Konstruktor- und Destruktormethode	T1 T2 T3 T4 LV
Festigung ca. 30` ca. 15` ca. 25` ca. 15` ca. 15` ca. 10`	• Übung 1: Deklaration von Klassen in C++ • Überprüfung und Diskussion der Ergebnisse • Übung 2: Implementierung der Methoden • Überprüfung und Diskussion der Ergebnisse • Übung 3: Erstellung von Testprogrammen • Überprüfung und Diskussion der Ergebnisse	F1, PA SSG F2, PA SSG F3, PA SSG
Hausaufgabe	• Gesamtwiederholung • Notieren von Fragen	

[39] Die grundsätzliche Syntax einer Struktur ist in C++ der einer Klasse sehr ähnlich.

8.2.3. Unterrichtsmaterialien

Von der UML-Notation zum C++-Quellcode

UML-Notation:



Ouellcode:

```
class    CAppSettings
{
private:
     DWORD m_Trigger;
     ...

public:
     void setTrigger(DWORD val);
     DWORD getTrigger(void);
     …
};
```

Merke:

- Im „private“-Bereich werden alle Methoden und Attribute vereinbart, die ausschließlich privat verwendet werden. Es besteht keine Zugriffsmöglichkeit von außen.
- Im „public“-Bereich werden alle öffentlichen Methoden vereinbart. Von außen kann auf sie zugegriffen werden. Attribute sind aufgrund des Geheimnisprinzips nicht öffentlich zu machen.
- Wird bei der Deklaration einer Klasse kein Bereich angegeben, so sind sämtliche Attribute und Methoden privat.
- Wichtig: „Die Klassendeklaration endet mit einem Semikolon.“

T1

Implementierung der Methoden

Deklaration und Definition in der Klasse (implizit inline):

```
#pragma once

class    CAppSettings
{
private:
   DWORD m_Trigger;
   DWORD m_IPAdresse;
   DWORD m_Port;
public:
   void setTrigger(DWORD val)
   {
        m_Trigger = val;
   }
   …
   DWORD getTrigger(void)
   {
        return m_Trigger;
   }
   …
};
```

Deklaration und Definition in der Header-Datei (explizit inline):

```
#pragma once

class    CAppSettings
{
private:
   DWORD m_Trigger;
   DWORD m_IPAdresse;
   DWORD m_Port;
public:
   void        setTrigger(DWORD val);
   …
   DWORD    getTrigger(void);
   …
};

inline void CAppSettings :: setTrigger(DWORD val)
{
   m_Trigger = val;
}
…
```

- Inline deklarierte Methoden werden beim kompilieren vom Präprozessor an der Aufrufstelle eingefügt. Dadurch ergibt sich eine Laufzeitverbesserung.
- Umfangreichere Methoden werden wegen der Übersichtlichkeit der Klassendeklaration nicht impliziert inline deklariert.
- Um Mehrfachdefinitionen zu vermeiden muss jede Deklarationsdatei mit der Kompilerdirektive #pragma once versehen werden.

T2

Trennung zwischen Deklaration und Definition

Deklaration:

//AppSettings.h

```
#pragma once
class    CAppSettings
{
private:
   DWORD m_Trigger;
   …
public:
   void          setTrigger(DWORD val);
   …
   DWORD    getTrigger(void);
   …
};
```

Implementierung:

//AppSettings.cpp

```
#include"AppSettings.h"
void CAppSetting :: setTrigger(DWORD val)
{
   m_Trigger=val;
}
…
DWORD CAppSettings :: getTrigger(void)
{
   return m_Trigger;
}
…
```

Merke:

- Durch die Trennung besteht die Möglichkeit die Definition zu kapseln und damit bei der Weitergabe der Klasse zu verbergen.

T3

Konstruktor- und Destruktormethode

Die Verwendung von Objekten einer Klasse läuft in drei Phasen ab:

Erzeugen und initialisieren des Objekts -> benutzen des Objekts -> beseitigen des Objekts

Zum Erzeugen und Initialisieren eines Objekts wird die Konstruktormethode verwendet:

- Sie trägt den gleichen Namen wie die Klasse.
- Im Gegensatz zu Objektmethoden besitzt sie keinen Rückgabewert.
- Weist sie keinen zusätzlichen Übergabeparameter auf, so spricht man von der Standardkonstruktormethode oder kurz dem Standardkonstruktor.
- Allgemeine Konstruktormethoden können dagegen Übergabeparameter besitzen.
- Wenn mindestens eine allgemeine Konstruktormethode definiert worden ist, wird vom System kein Standardkonstruktor erzeugt.

Um bei der Beseitigung eines Objekts die anfallenden Aufräumarbeiten zu erledigen, wird die Destruktormethode verwendet.

- Als Name für die Destruktormethode dient der Klassenname, dem eine Tilde (~) vorangestellt ist.
- Bei ihr handelt es sich um eine parameterlose Funktion, die ebenso wie eine Konstruktormethode keinen Rückgabewert besitzt.

T4

Übungsaufgabe

Implementieren Sie die Klassen der Aufgaben 1-3 vom Arbeitsblatt 6 aus der UML-Notation in die Programmiersprache C++. Die Umsetzung soll folgendermaßen erfolgen:

a) Deklaration und Definition in der Klasse
b) Deklaration und Definition in der Header-Datei
c) Trennung zwischen Definition und Deklaration

Hinweis: Erstellen Sie die Klassen in einer Konsolenanwendung!

F1

Fortsetzung der Übungsaufgabe

Implementieren Sie nun die Methoden. Verwenden Sie hierzu die in c) erstellten Klassen.

Hinweis zur Aufgabe 1:
- Bei der Herstellung einer Dose ist diese zu Beginn immer voll.
- Man kann nie mehr aus einer Dose trinken als ihren momentanen Inhalt

Hinweis zur Aufgabe 2:
- Schreiben Sie zwei Konstruktoren für die Klasse. Der erste Konstruktor soll immer die Zeit auf 0:0:0 setzen. Der zweite Konstruktor bekommt die Zeit als Parameter übergeben. Gibt ein Benutzer hier eine falsche Zeit an, so wird die Zeit auf 0:0:0 initialisiert.
- Fügen Sie eine Methode hinzu, die die Differenz zu einer anderen Uhrzeit berechnet und die Differenz in Sekunden zurückgibt.

Hinweis zur Aufgabe 3:
- Eine Kontonummer ist eine einmalige Nummer. Um menschliche Fehler zu vermeiden, muss die Kontonummer dem Objekt automatisch bei der Instanziierung aufsteigend zugewiesen werden.

F2

Fortsetzung der Übungsaufgabe

Schreiben Sie zu jeder Aufgabe ein Testprogramm um die Richtigkeit der Klassen zu überprüfen.

F3

9. Klausur

• Verlaufsplanung

8. UNTERRICHTSEINHEIT		
Phase/Zeit	**Inhalt/Beschreibung**	**Medien/ Sozialformen**
Einstieg ca. 5`	• Sammeln von Schülerfragen	SSG
Erarbeitung ca. 40`	• Bearbeitung der Schülerfragen	SSG
ca. 90`	• 1. Klausur[40]	

[40] Klausur und Musterlösung befinden sich im Anhang.

10. Statische Konzepte

10.1. Klassendiagramm

10.1.1. Lernziele

Die Schüler sollen …

- Klassen, Attribute und Methoden in einer Problemstellung systematisch identifizieren und im Klassendiagramm modellieren können,
- beurteilen können, ob ein geeignetes Klassendiagramm erstellt wurde
- erklären können, wozu Assoziationen in einem Klassendiagramm verwendet werden,
- Assoziationen in einer Problemstellung identifizieren und in UML-Notation darstellen können,
- erklären können, was Kardinalitäten in einem Klassendiagramm aussagen,
- Kardinalitäten problemgerecht identifizieren und in der UML-Notation beschreiben können,
- das Vererbungsprinzip verstehen,
- die Begriffe Oberklasse und Unterklasse erläutern können,
- Vererbungsstrukturen in einer Problemstellung identifizieren und in UML-Notation darstellen können.

10.1.2. Verlaufsplanung

9. UNTERRICHTSEINHEIT		
Phase/Zeit	**Inhalt/Beschreibung**	**Medien/ Sozialformen**
ca. 45`	• Rückgabe und Besprechung der Klausur	FR-ENT
Einstieg ca. 5`	• Wie können Beziehungen zwischen Klassen dargestellt werden?	F1 FR-ENT
Erarbeitung ca. 15` ca. 15`	• Definition der Begriffe - Klassendiagramm, Assoziation und Kardinalität - Vererbung	 T1 F2, T3 FR-ENT
Festigung ca. 45` ca. 20`	• Erstellung von Klassendiagrammen • Diskussion der Ergebnisse	AB7, AB8 PA SSG
Hausaufgabe	• Wiederholung AB7 und AB8	

10.1.3. Unterrichtsmaterialien

Wie können Beziehungen zwischen Klassen dargestellt werden?

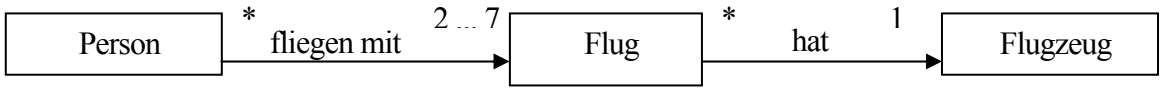
Ausgangssituation:
Die Fluglinie Lufthansa hat u. a. einen Piloten namens „Max Meier“ (Gehalt 14 000,00 €) und u. a. zwei Flugbegleiter namens „Karl Schmidt“ und „Markus Müller“ (Gehalt jeweils 2 700,00 €). Am 28.4.2004 findet ein Hin- und Rückflug (Flugnummern 321 und 322) von München nach Helsinki statt. Bei diesem Flug **ist** Max Meier **der Pilot** und **Markus Müller der Flugbegleiter**. Der Grundpreis für eine Strecke beträgt 99,00 €. Als Flugzeug **wird** eine Boing 707 **eingesetzt**. Sie hat ein Fassungsvermögen von 120 Personen und einen Stauraum von 400m^2. Erbaut wurde sie 1970.

Welche Beziehungen bestehen zwischen den Klassen Flug, Person und Flugzeug?

F1

Klassendiagramm

- Klassendiagramme werden aus Klassen, Assoziationen, Abhängigkeiten und Vererbungsbeziehungen gebildet.
- Dabei beschreibt eine Assoziation eine Menge gleichartiger Beziehungen zwischen Objekten bestimmter Klassen.
- An jedem Ende der Assoziation steht die Kardinalität (z. B. 1, 0 ... 1, *, usw.). Sie gibt an, wie viele Objekte einer Klasse mit wie vielen Objekten einer anderen Klasse in Beziehung stehen können.
- Weiterhin besitzt eine Assoziation einen Namen, der die Beziehung in eine Richtung beschreibt.



T1

Erweiterung der Aufgabenstellung

In das Klassendiagramm für die Fluglinie Lufthansa sollen neben den Daten des Piloten und des Flugbegleiters auch die Daten der Passagiere mit aufgenommen werden.

Übersicht der benötigten Daten:

Pilot:	**Flugbegleiter**	**Passagier**
Vorname	Vorname	Vorname
Nachname	Nachname	Nachname
Anschrift	Anschrift	Anschrift
Gehalt	Gehalt	Anzahl der Bonuspunkte
Trainingseinheiten	Dauer des Zeitvertrags	Zusatzgepäck?
Gewerkschaftsnagehöriger?		Raucher?

Wie sieht das Klassendiagramm aus?[41]

F2

[41] Die Lösung befindet sich im Anhang.

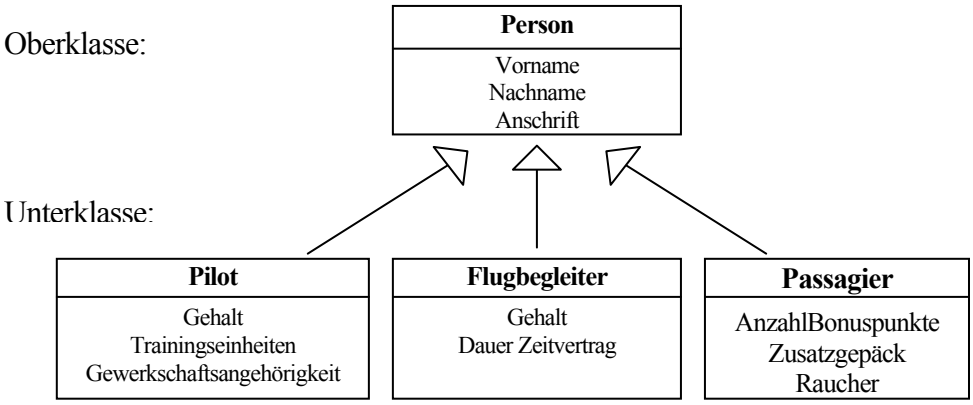
Vererbung

Vorteile der Vererbung:

- Konzeptionelle Vereinfachung durch das Zusammenfassen gemeinsamer Attribute und Methoden verwandter Klassen in einer Oberklasse (Generalisierung).
- Wiederverwendung von bereits vorhandenen Klassen durch Unterklassenbildung (Spezialisierung).
- Einfache Erweiterbarkeit von Vererbungshierarchien durch Hinzunahme von Unterklassen.

Bezogen auf das Beispiel bedeutet dies:

Gemeinsame Attribute und Methoden der Klassen „Pilot“, „Flugbegleiter“ und „Passagier“ werden in einer neuen Klasse „Person“ zusammengefasst und von dieser dann abgeleitet.



Wie sieht das Klassendiagramm aus?[42]

F3

[42] Die Lösung befindet sich im Anhang.

Aufgabe 1

Gesucht ist ein Klassendiagramm für folgenden Sachverhalt.

- Laut Gesetz kann nur eine juristische Person eine Immobilie kaufen. Dabei versteht das Gesetz unter einer juristischen Person eine Privatperson oder ein Unternehmen.
- Zum Kauf stehen Häuser, Eigentumswohnungen oder Baugrundstücke. Für den Kauf einer solchen Immobilie nimmt der Käufer evtl. Bankdarlehen auf.

Bestimmen Sie im ersten Schritt die Attribute der Klassen. Als Anhaltspunkt soll die folgende Situationsbeschreibung für den Kauf einer Eigentumswohnung dienen:
Bei der Immobilie handelt es sich um eine Eigentumswohnung. Der Käufer der Immobilie ist die Sparkasse Bensheim. Die Adresse der Immobilie lautet: Berliner Ring 5, Wohnungsnummer 7 in 64625 Bensheim. Der Kaufpreis beträgt 300.000,00 €. Für den Kauf nimmt die Bank ein Darlehen in Höhe von 100.000,00 € bei sich selbst auf. Die Konditionen sind 4,5% Zinsen und 1,0% Tilgung. Die Darlehensnummer ist 4711.

Stellen Sie nun den oben beschriebenen Sachverhalt in einem Klassendiagramm dar. Verwenden Sie dazu geeignete Assoziationen und Vererbungsbeziehungen. Benennen Sie die Assoziationen und geben Sie die Kardinalitäten an.

Aufgabe 2

Sie werden aufgefordert ein Informationssystem zu entwickeln, mit dem man die Gebäudesituation von Universitäten planen kann. Als Grundlage soll folgender Sachverhalt dienen:

- Universitäten bestehen aus mehreren Abteilungen, wobei eine Abteilung eindeutig einer Universität zugeordnet werden kann. Universitäten sowie Abteilungen werden durch ihren eindeutigen Namen charakterisiert. Darüber hinaus besitzen Abteilungen eine Adresse, bestehend aus Angaben zur Straße, Hausnummer, Stadt und Postleitzahl.
- Jede Abteilung kann mehrere Gebäude umfassen, die ebenfalls durch einen eindeutigen Namen gekennzeichnet sind. Da Universitäten permanent ausgebaut werden, werden auch ständig neue Gebäude erstellt. Daher muss für jedes Gebäude erkennbar sein, ob es sich noch im Bau befindet.
- Die Gebäude einer Abteilung werden unterschieden in Hörsaalgebäude und Institutsgebäude. Sie umfassen unterschiedliche, jeweils mit einer eindeutigen Raumnummer versehene Räume. Hörsaalgebäude sowie Institutsgebäude besitzen Seminarräume, wobei ein Hörsaalgebäude mindestens einen Seminarraum besitzt. Für Seminarräume ist zusätzlich die Anzahl der Sitzplätze zu vermerken.
- An Universitäten studieren Studierende und arbeiten Mitarbeiter. Beide sollen über ihren Vor- und Nachnamen registriert werden. Weiterhin besitzen Studierende eine Matrikelnummer und Mitarbeiter eine Personalnummer.
- Bei Mitarbeitern wird unterscheiden zwischen Professoren, Assistenten und studentische Mitarbeiter. Die studentischen Mitarbeiter sind natürlich auch Studierende.
- Eine Universität wird von einem Präsidenten und jede Abteilung einer Universität von einem Vizepräsidenten geleitet. Präsident und Vizepräsident sind natürlich Professoren der jeweiligen Universität. Es ist aber nicht möglich, dass eine Person Präsident und Vizepräsident gleichzeitig ist.

Erstellen Sie ein UML Klassendiagramm zur Darstellung dieses Sachverhalts.

Thema: Erstellung von Klassendiagrammen	AB7
	© W. Steffens

Aufgabe 1

Im Folgenden soll ein historisches Fuhrunternehmen modelliert werden.

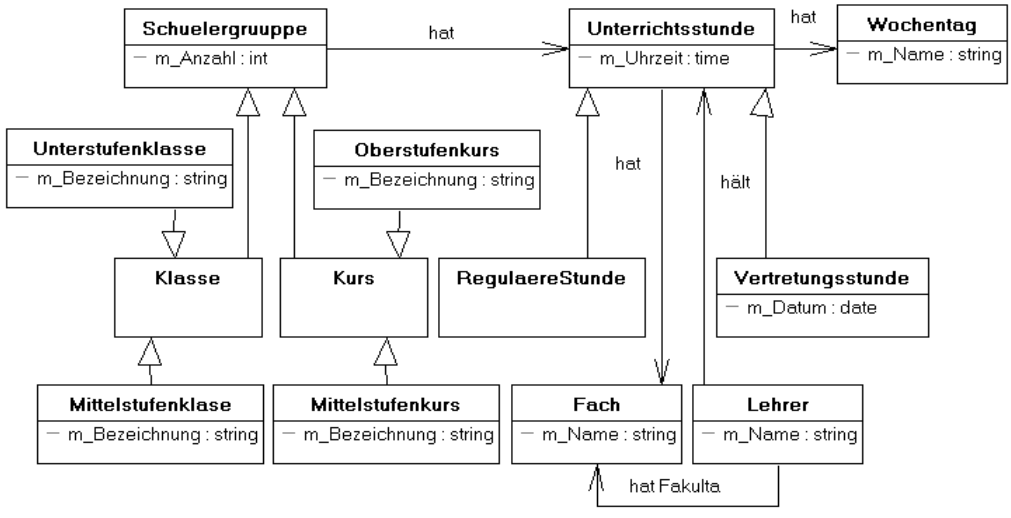
- Es hat wie ein heutiges Unternehmen einen Namen und eine Adresse. Um die anfallenden Aufgaben zu bewältigen, besitzt es Wagen, Pferde und diverse Angestellte.
- Bei den Wagen handelt es sich um Kutschen zur Personenbeförderung und Transportwagen zur Lastenbeförderung. Hinzu kommen noch Transportkutschen, mit denen Personen und Lasten befördert werden können. Kutschen, Transportwagen und Transportkutschen besitzen ein Kennzeichen. Weiterhin ist bei einer Kutsche die Anzahl der Sitzplätze, bei einem Transportwagen die maximale Zuladung und bei Transportkutschen sowohl die Ladekapazität als auch die Sitzplatzanzahl von Interesse.
- Eine Kutsche kann von mindestens einem und von höchstens vier Pferden gezogen werden. Dagegen benötigt ein Transportwagen mindestens zwei und höchstens acht Pferde. Für eine Transportkutsche ist eine feste Anzahl von sechs Pferden vorgesehen. Die Pferde des Fuhrunternehmens sind so ausgewählt, dass sie grundsätzlich jeden Wagen ziehen können.
- Zu den Angestellten des Fuhrunternehmens gehören ein Stellmacher und eine Reihe von Kutschern.
- Jeder Angestellte, der nur bei einem Fuhrunternehmen angestellt sein kann, wird mit seinen Namen und seiner Personalnummer geführt. Für jeden Kutscher ist darüber hinaus wichtig, ob er einen Personenbeförderungsschein besitzt.
- Jedem Kutscher ist eine Reihe von Pferden zugeordnet, für die er verantwortlich ist. Der Stellmacher ist für die Wartung aller Wagen verantwortlich.

Modellieren Sie das historische Transportunternehmen als UML-Klassendiagramm.

Aufgabe 2

Das folgende Klassendiagramm zeigt den Versuch die Verwaltung von Stunden- und Vertretungspläne einer Schule dazustellen. Allerdings enthält das Diagramm einige Fehler. Es enthält überflüssige Klassen und Modellierungsfehler!

- Ändern Sie das Klassendiagramm so, dass möglichst wenige Klassen definiert werden müssen, keine Redundanzen entstehen und dass der Lehrerstundenplan und der Vertretungsplan getrennt modelliert sind.
- Begründen Sie jede Ihrer Änderungen!
- Ergänzen Sie das neue Klassendiagramm mit entsprechenden Kardinalitäten.



Thema: Erstellung von Klassendiagrammen	AB8
	© W. Steffens

10.2. Vererbung in C++

10.2.1. Lernziele

Die Schüler sollen …

- die Syntax einer Vererbung in der Programmiersprache C++ erklären können,
- den Unterschied zwischen einer „private“, „protected“ und „public Vererbung“ erklären können,
- den Unterschied zwischen überladenen und überschriebenen Methoden erklären können,
- eine vorgegebene Vererbung in UML-Notation in die Programmiersprache C++ übertragen können,
- den Begriff der „abstrakten Klasse“ erläutern können.

10.2.2. Verlaufsplanung

10. UNTERRICHTSEINHEIT		
Phase/Zeit	**Inhalt/Beschreibung**	**Medien/ Sozialformen**
Einstieg ca. 10`	• Einführung in die Thematik: Wie überführt man eine modulierte Vererbung in UML-Notaion in einen C++-Quellcode?	T1 FR-ENT
Erarbeitung I ca. 20`	• Arten der Vererbung: public, protected und private	T2, T3, T4 LV
Festigung I ca. 15` ca. 10`	• Übungen zur public, protected und private Vererbung • Diskussion der Ergebnisse	AB9 PA SSG
Erarbeitung II ca. 10`	• Überladen bzw. Überschreiben von Methoden?	T5 LV
Festigung I ca. 10` ca. 10`	• Übungen zur Vererbung • Diskussion der Ergebnisse	AB10 PA SSG
Erarbeitung II ca. 10`	• Abstrakte Klassen	T6 LV
Festigung I ca. 30` ca. 10`	• Übungen zu abstrakten Klassen • Diskussion der Ergebnisse	AB11 PA SSG
Hausaufgabe	• Wiederholung: AB10 und AB11	

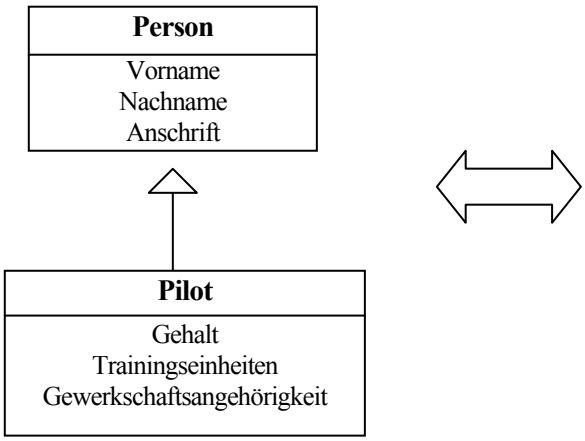
10.2.3. Unterrichtsmaterialien

Vererbung in C++

Syntax der Vererbung : class Unterklasse : Zugriffsspezifizierer Oberklasse

- Der Zugriffsspezifizierer gibt an, wie auf die von der Oberklasse geerbten Attribute und Methoden zugegriffen werden darf.
- Als Zugriffsspezifizierer stehen public, protected und private zur Verfügung.

Beispiel:




```
class Person
{
private:
        string vorname;
        string nachname;
        string anschrift
};

class Pilot : public Person
{
private:
        int gehalt;
        list trainingseinheiten;
        bool gewerkschaftsangehoerigkeit
};
```

T1

public Vererbung

- Bei einer public Vererbung können die Methoden der abgeleiteten Klasse auf alle public und protected deklarierten Attribute und Methoden der Oberklasse zugreifen.
- Dieses Zugriffsrecht wird auch an eventuelle weitere Unterklassen der abgeleiteten Klasse vererbt.

Beispiel:

```
class A
{
private:
    int privatA
protected:
    int protectedA
public:
    int publicA
};
```

verfügbar in A:

```
private:
    privatA
protected:
    protectedA
public:
    publicA
```

```
class B : private A
{
private:
    int privatB
protected:
    int protectedB
public:
    int publicB
};
```

verfügbar in B:

```
private:
    privatB
protected:
    protectedA
    protectedB
public:
    publicA
    publicB
```

```
class C : private B
{
private:
    int privatC
protected:
    int protectedC
public:
    int publicC
};
```

verfügbar in C:

```
private:
    privatC
protected:
    protectedA
    protectedB
    protectedC
public:
    publicA
    publicB
    publicC
```

T2

private Vererbung

- Wird als Zugriffsspezifizierer „private“ angegeben, dann können die Methoden der abgeleiteten Klassen auf die geerbten public und protected deklarierten Attribute und Methoden der Oberklasse zugreifen.
- Dies gilt jedoch nicht für Unterklassen, die durch weitere Vererbung der abgeleiteten Klasse entstehen. Die public und protected deklarierten Attribute und Methoden der Oberklasse sind somit zu privaten Attributen und Methoden in der direkt abgeleiteten Unterklasse geworden.

Beispiel:

```
class A
{
private:
    int privatA
protected:
    int protectedA
public:
    int publicA
};
```

verfügbar in A:

```
private:
    privatA
protected:
    protectedA
public:
    publicA
```

```
class B : private A
{
private:
    int privatB
protected:
    int protectedB
public:
    int publicB
};
```

verfügbar in B:

```
private:
    privatB
    protectedA
    publicA
protected:
    protectedB
public:
    publicB
```

```
class C : private B
{
private:
    int privatC
protected:
    int protectedC
public:
    int publicC
};
```

verfügbar in C:

```
private:
    privatC
protected:
    protectedA
    protectedB
    protectedC
public:
    publicA
    publicB
    publicC
```

T3

protected Vererbung

- Bei einer protected-Vererbung können die Methoden der abgeleiteten Klasse auf alle public und protected deklarierten Attribute und Methoden der Basisklasse zugreifen.
- Jedoch stehen die public deklarierten Attribute und Methoden in der abgeleiteten Klasse als protected zur Verfügung.

Beispiel:

```
class A
{
private:
    int privatA
protected:
    int protectedA
public:
    int publicA
};
```

verfügbar in A:

```
private:
    privatA
protected:
    protectedA
public:
    publicA
```

```
class B : private A
{
private:
    int privatB
protected:
    int protectedB
public:
    int publicB
};
```

verfügbar in B:

```
private:
    privatB
    protectedA
    publicA
protected:
    protectedB
public:
    publicB
```

```
class C : private B
{
private:
    int privatC
protected:
    int protectedC
public:
    int publicC
};
```

verfügbar in C:

```
private:
    privatC
protected:
    protectedA
    protectedB
    protectedC
public:
    publicA
    publicB
    publicC
```

T4

Überladen bzw. Überschreiben von Methoden

- Man sagt, eine „Methode ist überladen“, wenn es innerhalb einer Klasse mehrere Methoden gleichen Namens gibt, die sich durch ihre formalen Argumente unterscheiden.
- Überschriebene Methoden können nur im Zusammenhang mit einer Vererbung auftreten. Eine abgeleitete Klasse deklariert dabei eine Methode, die denselben Namen und dieselben Übergabeparameter wie eine Methode der Basisklasse besitzt.

Überladen:

```
class A
{
private:
   int privatA
public:
   int berechnung( );            // (1)
   int berechnung( int wert);  // (2)
};
```

Hauptprogramm

```
…
A test;
erg1 = test.berechnung( )     // (1)
erg2 = test.berechnung( 10 ) // (2)
...
```

Überschreiben:

```
class A
{
private:
   int privatA
public:
   int berechnung( int wert)      // (1)
};
```

```
class B : A
{
private:
   int privatB
public:
   int berechnung( int wert);     // (2)
};
```

Hauptprogramm

```
…
A testA;
B testB;
erg1 = testA.berechnung(10)          // (1)
erg2 = testB.berechnung(10)          // (2)
...
```

T5

Abstrakte Klassen

- Klassen, von denen keine Objekte erzeugt werden können, bezeichnet man als „abstrakte Klassen“.
- Von abstrakten Methoden sind nur die Methodenköpfe definiert. Der Rumpf muss von der abgeleiteten Klasse implementiert werden.
- Eine Klasse mit mindestens einer abstrakten Methode ist abstrakt.
- Implementierung von abstrakten Klassen:

Beispiel 1:

```
class A
{
public:
    virtual void a( )= 0;
    virtual void a( )= 0;
};
class B : public B
{
public:
    void a( );
};
class C: public B
{
public:
    void b( );
};
```

Hauptprogramm

```
…
A *testA= new A( );   // nicht zulässig
B *testB= new B( );   // nicht zulässig
C *testB= new C( );   //zulässig
...
```

Beispiel 2:

```
class A
{
protected:
       A( ):
public:
    void a( );
    void b( );
};
class B : public B
{
public:
    void B( );
};
```

Hauptprogramm

```
…
A *testA= new A( );      // nicht zulässig
B *testB= new B( );      // nicht zulässig
...
```

T6

Aufgabe 1

Gegeben sei die folgende C++-Klasse:

```
class CAuto
{
private:
      string m_Art;
public:
      CAuto(string autoart){ }
      CAuto(string autoart){m_Art = autoart;}
      string gibArt(void){return m_Art);}
};
```

a) Deklarieren Sie eine Unterklasse der Klasse „Auto“ mit dem Namen „Van“. Die Klasse „Van“ soll
   - ein weiteres ganzzahliges Attribut m_SitzAnzahl haben,
   - eine Methode getSitzAnzahl( ) haben, die die Sitzanzahl zurückgibt,
   - einen Konstruktor mit den Parametern m_Art und m_SitzAnzahl haben, der die gleichnamigen Attribute mit den übergebenen Werten initialisiert.

b) Welche Attribute und Methoden sind in der Klasse „Auto“ und welche in der Klasse „Van“ gültig?

Aufgabe 2:

Welche Werte werden ausgegeben? (Begründung)

```
class CBasis
{
public:
  int  x;
  CBasis ( ) : x(10) { }
  void f ( ) { ++x; }
};
```

```
class CAbgeleitet : public CBasis
{
public:
  CAbgeleitet( ) { }
  void f( ) {--x;}
};
```

```
int main()
{
  CAbgeleitet * pa = new CAbgeleitet;
  CBasis      * pb = pa;
  pa->f();
  pb->f();
  cout << pa->x << endl;
  cout << pb->x << endl;
}
```

Aufgabe 3:

Welche Werte werden ausgegeben? (Begründung)

```
class CBasis
{
public:
   int x;
   int y;
   int z;
   CBasis ( ) : x(1), y(2), z(3) { }
   CBasis (int px, int py, int pz) : x(px), y(py), z(pz) { }
   void print()
   {
        cout << x << ", " << y << ", " << z << endl;
   }
};
int main( )
{
   CBasis *pa;
   CAbgeleitet a;
   pa = &a;
   a.print();
   pa->print();
}
```

```
class CAbgeleitet : public CBasis
{
public:
   int y;
   int x;
   CAbgeleitet( ) : CBasis (111, 112, 113), x(CBasis::x+1000),
                    y(CBasis::y+1000) { }
   void print()
   {
        cout << x << ", " << y <<", " << z << endl;
   }
};
```

Aufgabe 1

Geben Sie an, welche der überladenen Methoden aufgerufen werden bzw. bei welchen Aufrufen ein Fehler auftritt. Begründen Sie den Fehler.

```
class Blub
{
public:
    m(int wert );                  // 1
    m(float wert);                 // 2
    m(double wert);                // 3
    m(char wert);                  // 4
    m(double wert1, float wert2);  // 5
    m(float wert1, double wert2);  // 6
};
```

```
int main( )
{
    int a = 0; short b = 0; long c = 0; float d = 0; double e = 0;
    Blub bla;
    bla.m(a);
    bla.m(b);
    bla.m(c);
    bla.m(d);
    bla.m(e);
    bla.m(e,d);
    bla.m(e,e);
}
```

Aufgabe 2

a) Kreuzen Sie die Aussage an, die für die jeweilige Methodendeklaration gültig ist:

```
class AKlasse{
    int a;
    void aMethode(int a);
    void aMethode(int a, int b, int c);      Überladung:___ Überschreibung:___ nichts von beidem:___
};
class B1Klasse : public AKlasse{
    int b;
    void aMethode(int a, int b);             Überladung:___ Überschreibung:___ nichts von beidem:___
    void bMethode(int c);                    Überladung:___ Überschreibung:___ nichts von beidem:___
};
class B2Klasse : public AKlasse{
    void aMethode(int a);                    Überladung:___ Überschreibung:___ nichts von beidem:___
};
```

b) Geben Sie für die markierten Zeilen an, welche der Attribute a, b, c dort gültig sind:

```
class AKlasse{
    int a;
    void aMethode(int a);
    void aMethode(int a, int b, int c);      Hier sind folgende Attribute gültig: _____
}
class B1Klasse : public AKlasse{
    int b;
    void aMethode(int a, int b);
    void bMethode(int c);                    Hier sind folgende Attribute gültig: _____
}
class B2Klasse : public AKlasse{
    void aMethode(int a);                    Hier sind folgende Attribute gültig: _____
}
class C1Klasse : public B1Klasse{
    int c;                                   Hier sind folgende Attribute gültig: _____
}
```

Aufgabe 1

Gegeben seien folgende Begriffe aus der zu modellierenden Realität:

PKW, LKW, LKW mit Anhänger, Taxi, Autobus, Containerschiff, Fähre, Floss, Yacht

- Identifizieren Sie die Klassen und entwerfen Sie eine Vererbungshierarchie.
- Finden Sie Verallgemeinerungen der Klassen und ergänzen Sie die Vererbungshierarchie.
- Teilen Sie Ihre Klassen in konkrete und abstrakte Klassen ein. Kennzeichnen Sie dies im Klassendiagramm.

Aufgabe 2

Die Firma Kreis möchte ihre Lagerverwaltung von verschiedenartigen Bauteilen am PC koordinieren. Folgende Notizen wurden im ersten Gespräch festgehalten:
Jedes Bauteil ist rechteckig und hat eine Höhe und eine Breite in Millimetern und eine eindeutige Nummer. Keine zwei Bauteile haben die gleiche Nummer. Das Programm kennt drei Varianten des Bauteils. Variante A hat zusätzlich eine Farbe, B ein Gewicht in Gramm und Variante C ist immer quadratisch. Jede Variante hat nochmals eine eindeutige Variantennummer.

a)

- Identifizieren Sie die Klassen.
- Identifizieren Sie die Attribute und Methoden der Klassen.
- Finden Sie eine Verallgemeinerung der Klassen und führen Sie eine Vererbungshierarchie ein.
- Teilen Sie Ihre Klassen in konkrete und abstrakte Klassen ein. Kennzeichnen Sie dies im Klassendiagramm.

b) Implementieren Sie die Klassen in C++.

- Zusätzlich soll jede Klasse eine print()-Methode besitzen, um die Eigenschaften des Bauteils auszugeben.

c) Schreiben Sie ein Hauptprogramm zu Testzwecken:

- Erstellen Sie hierzu von jeder Klasse zwei Objekte.
- Kontrollieren Sie, ob die Nummerierung der Objekte ordnungsgemäß erfolgte.

10.3. Mehrschichtenarchitektur

10.3.1. Lernziele

Die Schüler sollen …

- die grundlegende Software-Architektur beim objektorientierten Entwurf erklären können,
- eine Drei- von einer Mehrschichten-Architektur unterscheiden können,
- Klassen für den Zugriff auf die Daten der GUI-Schicht entwerfen können,
- die Aufgabe und den Aufbau der Registry erklären können,
- die Funktionsweise zum Speichern und Lesen von Daten aus der Registry erklären können,
- mit Hilfe der MFC den Zugriff auf die Registry implementieren können
- den Zugriff auf eine Access-Datenbank[43] über eine ODBC-Schnittstelle erklären können,
- einen ODBC-Treiber installieren können,
- mit Hilfe der MFC den Zugriff auf eine Access-Datenbank implementieren können.

10.3.2. Verlaufsplanung

11. UNTERRICHTSEINHEIT		
Phase/Zeit	**Inhalt/Beschreibung**	**Medien/ Sozialformen**
Einstieg ca. 10`	• Warum wurde das OSI-Schichtenmodell[44] entwickelt? • Gibt es ein OSI-Schichtenmodell in der Software-Entwicklung?	
Erarbeitung ca. 10` ca. 5` ca. 10`	• Drei-Schichten-Architektur • Mehr-Schichten-Architektur • Vorstellung der Vorgehensweise bei der Implementierung von Klassen der Fachkonzept-Zugriffsschicht mit dem Visual Studio 7	T1 T2 PC, Beamer LV
Festigung ca. 60` ca. 20`	• Entwicklung der Klassen für die Fachkonzept-Zugriffsschicht • Vorstellung und Diskussion der Ergebnisse	F1, GA SSG
Abschluss ca. 20`	• Einigung auf ein gemeinsames Ergebnis für die weitere Entwicklung	SSG
Hausaufgabe	• Wiederholung: Erstellung der Klassen für die Fachkonzept-Zugriffsschicht	

[43] Im beruflichen Gymnasium findet in der Jahrgangsstufe 12 I der Grundkurs „Datenbanken“ statt. Von daher ist den Schülern der Umgang mit Datenbanken bekannt.

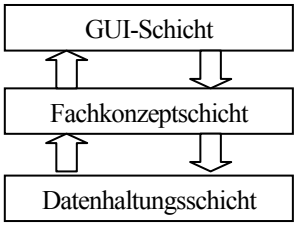
[44] Das OSI-Schichtenmodell ist den Schülerinnen und Schülern bekannt.

12. UNTERRICHTSEINHEIT		
Phase/Zeit	**Inhalt/Beschreibung**	**Medien/ Sozialformen**
Einstieg ca. 5`	• Wie können Daten dauerhaft gespeichert werden?	FR-ENT
Erarbeitung I ca. 10`	• Datenhaltung in der Registry	F2
Festigung I ca. 70` ca. 20`	• Entwicklung der Klassen für die Datenhaltungs-Zugriffsschicht • Vorstellung und Diskussion der Ergebnisse	F3, GA SSG
Erarbeitung II ca. 10` ca. 10`	• Datenhaltung in einer Datenbank mit ODBC • Vorstellung der Vorgehensweise bei der Implementierung von Klassen der Datenhaltungs-Zugriffsschicht für den Zugriff auf Access-Datenbanken über einen ODBC-Treiber	T3 PC, Beamer LV
Festigung II ca. 70` ca. 20`	• Entwicklung der Klassen für die Datenhaltungs-Zugriffsschicht • Vorstellung und Diskussion der Ergebnisse	F4, GA SSG
Abschluss ca. 20`	• Einigung auf ein gemeinsames Ergebnis für die weitere Entwicklung	SSG
Hausaufgabe	• Wiederholung: Zugriff auf die Registry und auf eine Access-Datenbank	

10.3.3. Unterrichtsmaterialien

Drei-Schichten-Architektur

- Um ein Softwaresystem besser implementieren und später warten zu können, wird das System in drei logische Teile bzw. Schichten unterteilt.
- Analog zum OSI-Schichtenmodell können nur jeweils zwei unmittelbar benachbarte Schichten über fest definierte Schnittstellen (Methoden) miteinander kommunizieren.



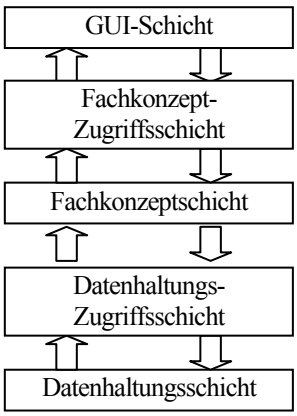
← stellt die unmittelbare Schnittstelle zum Benutzer der Anwendung dar.

← enthält die gesamte Logik der Software und stellt das Zentrum dar.

← hat die Aufgabe die Daten der Anwendungen aufzunehmen und stellt die Datenhaltung dar.

T1

Mehr-Schichten-Architektur



- GUI-Schicht: Präsentation der Informationen
- Fachkonzept-Zugriffsschicht: hat die Aufgabe die Fachkonzeptschicht mit Daten aus der GUI zu füllen und die GUI bei Änderungen zu aktualisieren.
- Fachkonzeptschicht: enthält die gesamte Logik der Software und stellt das Zentrum dar.
- Datenhaltungs-Zugriffsschicht: hat die Aufgabe die Fachkonzeptschicht mit Daten aus der Datenbank zu füllen und die Datenbank bei Änderungen zu aktualisieren.
- Datenhaltungsschicht: hat die Aufgabe die Daten der Anwendungen aufzunehmen.

T2

Arbeitsauftrag:

Es sollen die Klassen der Fachkonzept-Zugriffsschicht für die Wetterstation entwickelt werden.
Als Grundlage dienen die bereits entwickelten Benutzeroberflächen.

Client: AppSettingsNetworkDlg, AppSettingsTriggerDlg, ChangePasswdDlg, UserLoginDlg und WarningsProtocolDlg

Server: AppSettingsDBsDlg, AppSettingsMeasurementDlg, AppSettingsNetworkDlg, UserLoginDlg, UserManagementDlg und UserAddUserDlg

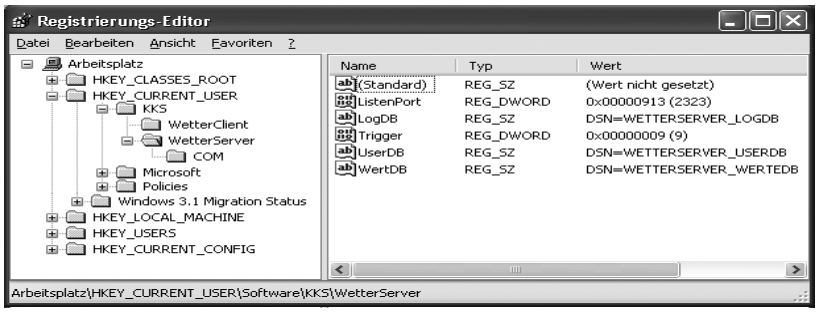
a) Entwickeln Sie die Klassen in UML-Notation.
b) Implementieren Sie die Klassen.

F1

Datenhaltung in der Registry

- Die Registry ist eine zentrale Datenbank, in der Anwendungs- und Systemeinstellungen gespeichert werden können.
- Der Aufbau ist analog der einer Verzeichnisstruktur. Das Äquivalent zu einem Ordner ist der Schlüssel und zu einer Datei der Wert. Dabei kann ein Schlüssel wiederum aus einer Menge von Unterschlüsseln und Werten bestehen.

Beispiel:



Befehle:

RegOpenKey Ex:	Öffnet einen existierenden Registry-Schlüssel.
RegCreateKeyEx:	Existiert der Schlüssel, so wird er geöffnet und ansonsten erzeugt und geöffnet.
RegSetValueEx:	Setzt einen benannten Wert eines Unterschlüssels.
RegQueryValueEx:	Gibt einen benannten Wert zurück, der einem geöffneten Schlüssel zugeordnet ist.
RegCloseKey:	Gibt das Handle des geöffneten Schlüssels wieder frei.

F2

Arbeitsauftrag 1:

Es sollen ein Teil der Klassen für die Datenhaltungs-Zugriffsschicht der Wetterstation entwickelt werden. Hierzu muss die Klasse CAppSettings der Server- und der Clientanwendung um die Methoden SaveDataToReg und LoadDataFromReg erweitert werden. Ihre Aufgabe ist es, die Einstellungsdaten der Anwendung in der Registry abzuspeichern bzw. abzurufen.

Server:

- Die Portnummer, die Triggerzeit und die Namen der Datenbanken sind unter dem Schlüssel HKEY_CURRENT_USER \ KKS \ WetterServer abzulegen.
- Die Einstellungen der seriellen Schnittstelle sind unter dem Schlüssel HKEY_CURRENT_USER \ KKS \ WetterServer \ COM abzulegen.

Client:

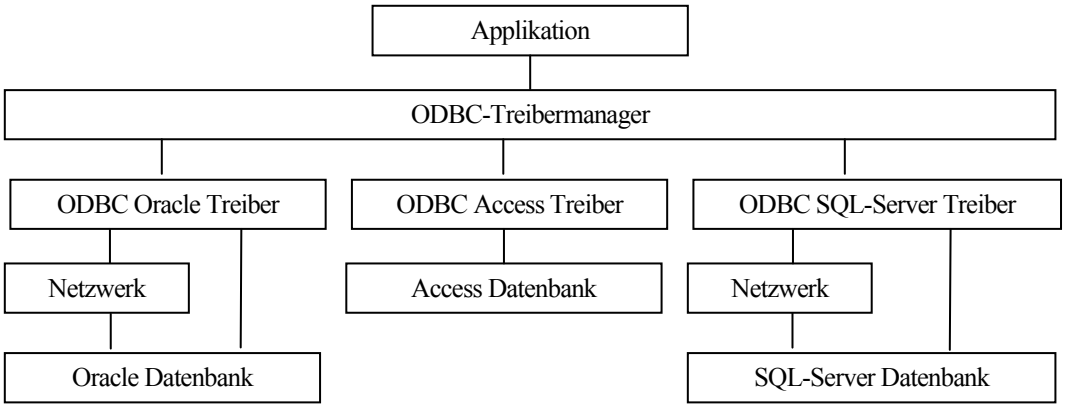
- Die Triggerzeit sowie die Portnummer und die IP-Adresse des Servers sind unter dem Schlüssel HKEY_CURRENT_USER \ KKS \ WetterClient abzulegen.

a) Implementieren Sie die Methoden.
b) Schreiben Sie eine Testanwendung, um die Funktion der Methoden zu überprüfen.

F3

Datenbankanbindung mit ODBC

Die ODBC-Architektur.



MFC-Klassen zur ODBC-Programmierung:

- Objekte der Klasse CDatabase: Sie sind für den Auf- und Abbau von Verbindungen, Timeouts und die Durchführung von Transaktionen zuständig.
- Objekte der Klasse CRecordset: Sie dienen zum Durchführen von Datenbankabfragen und zur Verwaltung von Satzgruppen. Auf die Datensätze einer Satzgruppe kann nur zeilenweise zugegriffen werden.
- Mit Objekten der Klasse CRecordView lassen sich Datensätze einer ODBC-Quelle in Form von einfachen Masken anzeigen und verändern.
- Möchte man selbst ein eigenes Exception-Handling implementieren, so muss man Ausnahmen der Klasse CDBException abfangen, die von den ODBC-Klassen der MFC für Fehlermeldungen verwendet werden.

T4

Arbeitsauftrag 3:

Es sollen die Klassen der Datenhaltungs-Zugriffsschicht für die Wetterstation entwickelt werden.

a) Entwickeln Sie hierzu zuerst die Access-Datenbanken LogDB.mdb, UserDB.mdb und WertDB.mdb.
b) Installieren Sie die ODBC-Treiber.
c) Erstellen Sie eine MFC-Anwendung und fügen Sie ihr die MFC-ODBC-Consumer Klassen hinzu, um auf die Tabellen der Access-Datenbank zugreifen zu können.
d) Schreiben Sie eine Testanwendung, die exemplarisch die Benutzer in die Datenbank UserDB einfügen und aus ihr löschen kann.

F4

10.4. Aggregation und Komposition als Spezialfall der Assoziation

10.4.1. Lernziele

Die Schüler sollen …

- den Unterschied zwischen einer einfachen „Assoziation“, „Aggregation“ und „Komposition“ erklären können,
- den Einsatzbereich einer Assoziation, Aggregation und Komposition erklären können,
- bei einer vorgegebenen Problemstellung entscheiden können, ob der Sachverhalt durch eine Assoziation, Aggregation oder Komposition modelliert wird
- erklären können, wie eine Aggregation bzw. Komposition in C++ implementiert wird

10.4.2. Verlaufsplanung

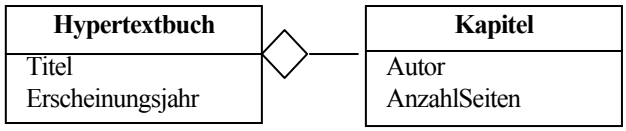
13. UNTERRICHTSEINHEIT		
Phase/Zeit	**Inhalt/Beschreibung**	**Medien/ Sozialformen**
Einstieg ca. 5`	• Welche Beziehung existiert zwischen der Klasse „Vater“ und „Kind“ bzw. zwischen der Klasse „Kind“ und „Familie“?	FR-ENT
Erarbeitung I ca. 20`	• Merkmale und Eigenschaften der - Aggregation - Komposition	 T1 T2 LV
Festigung I ca. 15` ca. 10`	• Arbeitsauftrag: Assoziation, Aggregation, Komposition oder Vererbung? • Diskussion der Ergebnisse	F1 PA SSG
Festigung II ca. 30`	• Erstellung von Klassendiagrammen für die Wetterstation • Diskussion der Ergebnisse	AB12, GA SSG
Hausaufgabe	• Wiederholung: Informationsbeschaffung - Sequenzdiagramm	

10.4.3. Unterrichtsmaterialien

Aggregation

- Die Komposition ist eine Ganze-Teil-Beziehung.
- Das Ganze kennt seine Teile und eine Navigation vom Ganzen zu den Teilen ist möglich.
- Wird das Ganze gelöscht, so können seine Teile weiter „existieren".
- Teile können nach dem Ganzen erzeugt werden und vor dem Ganzen „sterben".

Beispiel:

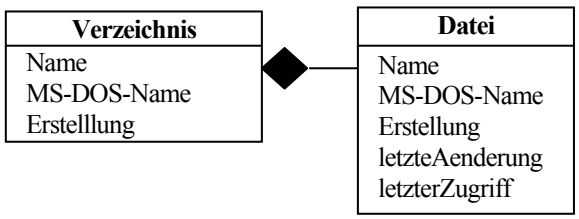


T1

Komposition

- Die Komposition ist ebenfalls eine Ganze-Teil-Beziehung.
- Auch hier kennt das Ganze seine Teile und eine Navigation vom Ganzen zu den Teilen ist möglich.
- Im Gegensatz zur Aggregation werden bei der Komposition alle Teile gelöscht, wenn das Ganze gelöscht wird. Die Teile leben und sterben mit dem Ganzen.
- Wird das Ganze kopiert, werden alle „lebenden" Teile mit kopiert.

Beispiel:



T2

Arbeitsauftrag:

Handelt es sich bei den Beziehungen um eine Assoziation, Komposition, Aggregation oder um eine Vererbung?

a) Unternehmen — Abteilung — Mitarbeiter

b) Rechnung — Rechnungsposition

c) Person — Student

d) Stuhl — Lehne

e) Lehrerschaft — Lehrer

f) ICE-Lok — Motoren

g) Fenster — Scrollbar

h) Gegeben seien folgende Anforderungen an ein Textverarbeitungssystem:

- Ein Textverarbeitungssystem erlaubt es Benutzern Dokumente anzulegen und zu editieren.
- Es kann Text und Bitmap-Grafik enthalten. Ein Text besteht aus Abschnitten, jeder Abschnitt wiederum aus Zeichen.
- Ein Dokument enthält außerdem verschiedene administrative Informationen wie seinen Titel, seinen Autor, seinen Pfad, in dem es abgelegt wurde, sowie das Datum seiner letzen Änderung.

Zeichnen Sie das entsprechende Klassendiagramm in UML-Notation.

F1

Um die Performance eines Programms zu steigern, besteht die Möglichkeit häufig benutzte Klassen in einer globalen Klasse zu kapseln. Die globale Klasse stellt dann die Methoden bereit, um auf die Attribute der einzelnen Klassen zugreifen zu können.

Zu den häufig benötigten Klassen der Serveranwendung gehören die Klassen CUserLoginData, UserManagement, LogManagement und AppSettings.

- Die Aufgabe der Klasse CUserLoginData ist:
  - die Anmeldedaten eines Benutzers zu speichern.

- Die Aufgaben der Klasse UserManagement sind:
  - die benötigten Datenbankverbindung auf- und abzubauen,
  - neue Benutzer in die Datenbank einzutragen,
  - Benutzer aus der Datenbank zu löschen, sofern es sich nicht um den Administrator sowie den gerade am Server angemeldete Benutzer handelt,
  - Benutzerdaten zu verändern,
  - eine Liste mit allen Benutzern aus der Datenbank anzulegen und diese der Dialogklasse UserManagementDlg zur Verfügung zu stellen (Mehrschichtenarchitektur),
  - zu überprüfen, ob ein Benutzer existiert, ob das Passwort richtig ist und ob der Benutzer Administratorenrechte besitzt (zur übersichtlicheren Darstellung überlegen Sie sich geeignete „defines“ zur Rechteverwaltung),
  - die Rechte eines Benutzers aus der Datenbank zu lesen.

- Die Aufgaben der Klasse LogManagement sind:
  - die benötigten Datenbankverbindung auf- und abzubauen,
  - ausgelöste Warnmeldungen in die Datenbank einzutragen. Hierbei soll der Warnungstyp, der Benutzername und das Entstehungsdatum mit Uhrzeit in die Datenbank eingetragen werden,
  - Log-Einträge durchzuführen (dies soll in der Log-Liste des Dialogfeldes der Serveranwendung sowie in der entsprechenden Datenbank erfolgen; eingetragen werden die ausgeführte Aktion, die Priorität der Aktion sowie das Auslösedatum mit Uhrzeit),.
  - dafür zu sorgen, dass zu Beginn und am Ende des Programms keine Log-Einträge in das Dialogfeld geschrieben werden. Zu diesen Zeitpunkten steht kein Dialogfeld zur Verfügung.

- Die Klasse AppSettings wurde bereits modelliert.

a) Entwickeln Sie die Klassen in UML-Notation.

b) Modellieren Sie ein Klassendiagramm, dass alle modellierten Klassen enthält.

c) Untersuchen Sie das Klassendiagramm bezüglich der Beziehungen untereinander. Handelt es sich bei den Beziehungen um Assoziationen, Kompositionen, Aggregationen oder Vererbungen?

d) Implementieren Sie die Klassen.

11. Dynamische Konzepte

11.1. Sequenzdiagramm

11.1.1. Lernziele

Die Schüler sollen …

- die UML-Notation für Sequenzdiagramme erklären können,
- Abläufe in einer Problemstellung systematisch identifizieren und in einem Sequenzdiagramm modellieren können,
- erklären können, wie das Klassen- mit dem Sequenzdiagramm zusammenwirkt,
- Sequenzdiagramme mit einem CASE-Werkzeug erstellen können,
- aus Sequenzdiagramm den entsprechenden Quellcode erstellen können.

11.1.2. Verlaufsplanung

14. UNTERRICHTSEINHEIT		
Phase/Zeit	**Inhalt/Beschreibung**	**Medien/ Sozialformen**
Einstieg ca. 10`	• Was ist ein Szenario?	T1 FR-ENT
Erarbeitung ca. 20`	• Notation des Sequenzdiagramms	F1 LV
Festigung I ca. 30`	• Gemeinsame Interpretation des vorgegebenen Sequenzdiagramms für das Framework	F2 FR-ENT
Festigung II ca. 45` ca. 30`	• Erstellung von Sequenzdiagrammen • Diskussion der Ergebnisse	F3 PA SSG
Hausaufgabe	• Wiederholung für die Klassenarbeit	

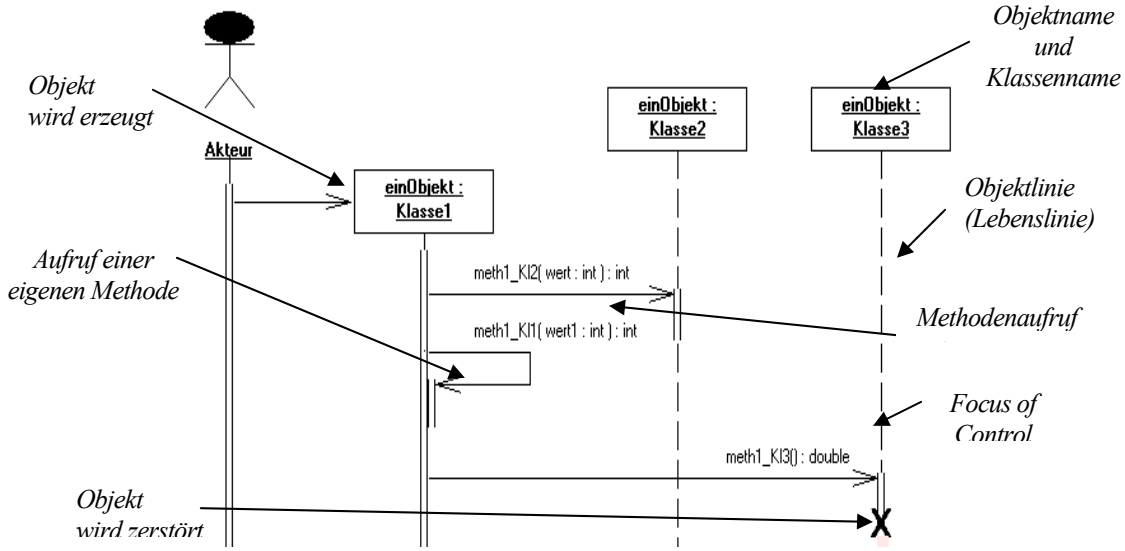
11.1.3. Unterrichtsmaterialien

Was ist ein Szenario?

- Es stellt eine Sequenz von Verarbeitungsschritten dar, die unter bestimmten Bedingungen auszuführen ist.
- Es beginnt mit einem auslösenden Ereignis durch einen Akteur oder einen anderen Anwendungsfall und wird fortgesetzt, bis das Ziel erreicht ist oder aufgegeben wird.
- Mit Szenarien wird es möglich, die Methoden der Klassen zu identifizieren und den Fluss der Nachrichten durch das Softwaresystem zu definieren.
- Außerdem kann die Vollständigkeit und Korrektheit des statischen Modells kontrolliert werden.
- Da nicht alle Szenarien von gleicher Bedeutung für die Modellbildung sind, kann im Gegensatz zum vollständigen statischen Modell das dynamische Modell sich auf die Szenarien beschränken, die wesentlich für das Modellieren des dynamischen Verhaltens sind.
- Szenarien können in Textform und durch Interaktionsdiagramme dokumentiert werden. Zur Darstellung in einem Interaktionsdiagramm stellt UML das Sequenz- und das Kollaborationsdiagramm zur Verfügung.

T1

Notation des Sequenzdiagramms

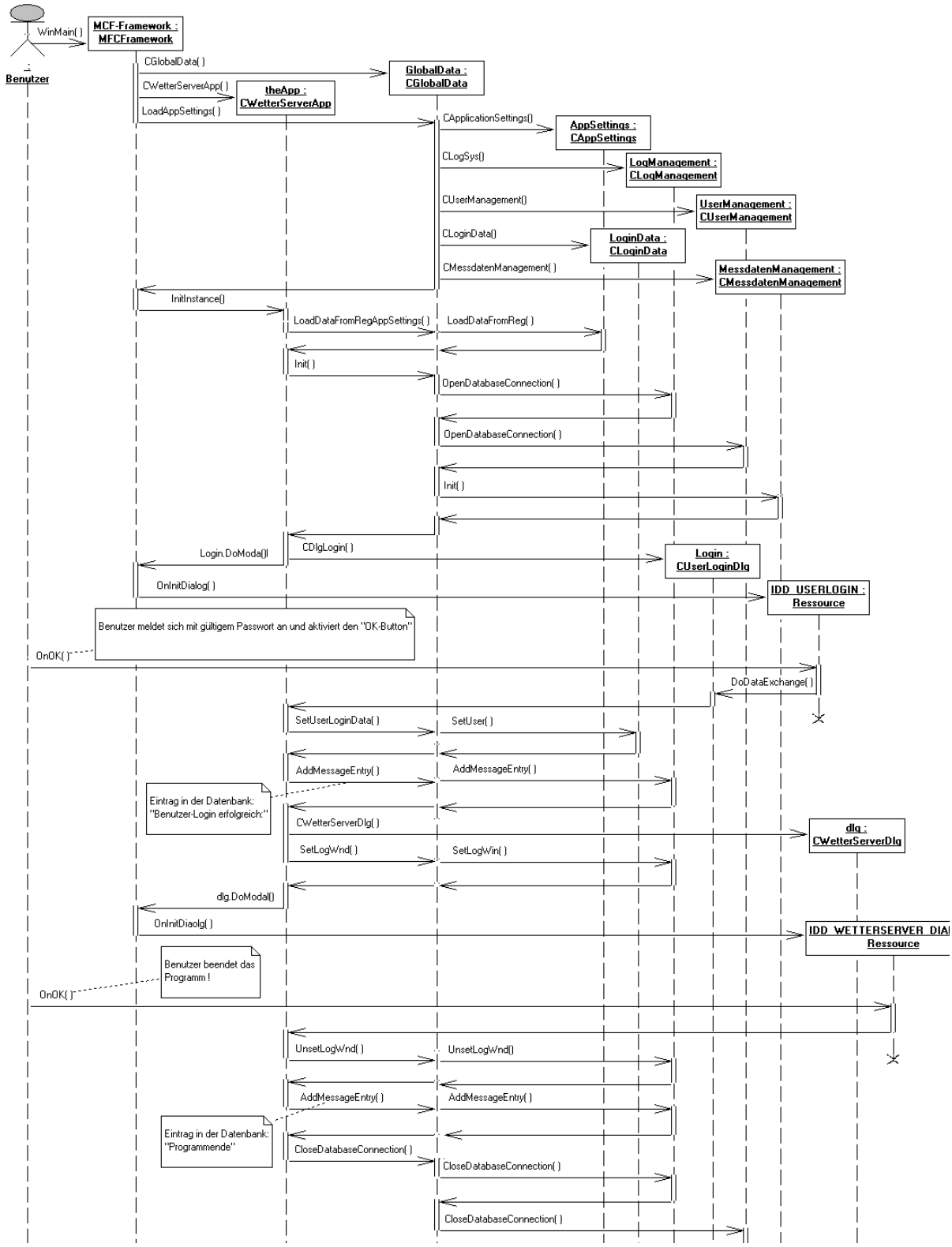


- Die Methodenrückgabe muss nicht eingezeichnet werden.[45]
- Wird nur ein Objekt einer Klasse verwendet, reicht auch die Angabe der Klasse aus.

F1

[45] Das CASE-Tool Jumli zeichnet z.B. die Rückgaben nicht automatisch ein.

Sequenzdiagramm für das Framework



F2

Arbeitsauftrag:

Für die Serveranwendung sind folgende Sequenzdiagramme zu erstellen:

- Der Hauptdialog des Servers ist gestartet. Nun möchte der Benutzer die Portnummer verändern. Die Eingaben erfolgen ohne Fehler.
- Der Hauptdialog des Servers ist gestartet. Nun möchte der Benutzer die Namen der Datenbanken ändern. Die Eingaben erfolgen ohne Fehler.
- Der Hauptdialog des Servers ist gestartet. Nun möchte der Benutzer Daten der Messwertabfrage ändern. Die Eingaben erfolgen ohne Fehler.

Für die Clientanwendung sind folgende Sequenzdiagramme zu erstellen:

- Der Hauptdialog des Client ist gestartet. Nun möchte der Benutzer die IP-Adresse und Potnummer verändern. Die Eingaben erfolgen ohne Fehler.
- Der Hauptdialog des Clients ist gestartet. Nun möchte der Benutzer das Triggerintervall verändern. Die Eingaben erfolgen ohne Fehler.

Hilfe:
Überlegen Sie sich wodurch sich die Sequenzdiagramme unterscheiden!

F3

12. Klausur

• Verlaufsplanung

15. UNTERRICHTSEINHEIT		
Phase/Zeit	**Inhalt/Beschreibung**	**Medien/ Sozialformen**
Einstieg ca. 5`	• Sammeln von Schülerfragen	SSG
Erarbeitung ca. 40`	• Bearbeitung der Schülerfragen	LSG
ca. 90`	• 2. Klausur[46]	F1

[46] Klausur und Musterlösung befinden sich im Anhang.

13. Projektphase

16. UNTERRICHTSEINHEIT		
Phase/Zeit	**Inhalt/Beschreibung**	**Medien/ Sozialformen**
ca. 45`	• Rückgabe und Besprechung der Klausur	LV
Projektphase ca. 90` ca. 20` ca. 10`	• Programmierung der Server- und Clientanwendung • Vorstellung der Gruppenergebnisse • Abstimmung der weiteren Vorgehensweise	GA GA SSG

17. UNTERRICHTSEINHEIT		
Phase/Zeit	**Inhalt/Beschreibung**	**Medien/ Sozialformen**
Projektphase ca. 105` ca. 20` ca. 10`	• Programmierung der Server- und Clientanwendung • Vorstellung der Gruppenergebnisse • Abstimmung der weiteren Vorgehensweise	**GA** **GA** **SSG**

18. UNTERRICHTSEINHEIT		
Phase/Zeit	**Inhalt/Beschreibung**	**Medien/ Sozialformen**
Projektphase ca. 85` ca. 20` ca. 30`	• Programmierung der Server- und Clientanwendung • Vorstellung der Gruppenergebnisse • Abschlussbesprechung	GA GA SSG

14. Reflexion

Am Anfang dieser kritischen Reflexion soll die positive Feststellung stehen, dass mir die Erstellung dieser Examensarbeit einen fundierten Einblick in die objektorientierte Modellierung mit UML gegeben hat. Teile der dargestellten Unterrichtsreihe wurden bereits in meinem Unterricht eingesetzt und die daraus gewonnenen Erfahrungen sind wiederum in die Examensarbeit eingeflossen. Im nächsten Schuljahr werde ich die Unterrichtreihe, so wie ich sie dargestellt habe, im Leistungskurs der Jahrgangsstufe 12 I des beruflichen Gymnasiums einsetzen.

Die im Teil II aufgearbeitete Thematik zur objektorientierten Modellierung hat sich für die Planung der Unterrichtreihe als ein stabiles Fundament bewehrt. Die Themen konnten - mit einer teilweisen didaktischen Reduktion - in die Unterrichtsreihe aufgenommen werden. Von daher wird sie dem Ziel gerecht Lehrerinnen und Lehrern als Grundlage zu dienen, sich in die objektorientierte Modellierung einzuarbeiten. Als umfangreichere und tiefgehendere Literatur empfiehlt sich das Lehrbuch der Objektorientierung von Heide Balzert[47].

Die Vorgehensweise in den bereits gehaltenen Stunden hat sich als sinnvoll und dem beruflichen Gymnasium angemessen erwiesen. Gerade die Verzahnung von Lehrervortrag mit anschließender Partner- oder Gruppenarbeit hat sich als sehr motivierend auf die Lerngruppe erwiesen. Um die Stärken der Schülerinnen und Schüler sowie der Lehrerin bzw. des Lehrers zu nutzen, sollte die Unterrichtsreihe auf die jeweilige Lernsituation abgestimmt werden. Von daher sind die dargestellten Verlaufsplanungen als Vorschlag anzusehen.

Die inhaltliche Planung der Stunden hat gezeigt, dass die Zeit zur Vermittlung der Inhalte des Lehrplans sehr knapp bemessen ist. Dies lässt sich anhand der umfangreichen Lernziele pro Lerneinheit erkennen. Da für die Unterrichtsreihe drei - statt der fünf zur Verfügung stehenden - Unterrichtstunden verplant wurden, besteht die Möglichkeit, in den verbleibenden beiden Stunden noch auf einzelne Punkte intensiver einzugehen bzw. Themen, die zusätzlich durch den Lehrplan vorgeschrieben sind zu bearbeiten. Für das berufliche Gymnasium wären dies z. B. eine intensivere Thematisierung von Listen und Bäumen sowie die Datenkommunikation über TCP/IP und über die serielle Schnittstelle. Gerade die Bearbeitung dieser Themen sind wichtig für die Projektphase, gehören aber nicht in eine Einführung in die objektorientierte Modellierung mit UML.

[47] Siehe Literaturverzeichnis

15. Literatur

BALZERT, HEIDE: Lehrbuch der Objektmodellierung: Analyse und Entwurf. Spektrum, Akademischer Verlag. Heidelberg, 1999.

BALZERT, HELMUT: Lehrbuch der Software-Technik: Software-Entwicklung. Spektrum, Akademischer Verlag. Heidelberg, 2000.

BOOCH, CRADY / RUMBAUGH, JIM / JACOBSON, IVAN: Das UML-Benutzerhandbuch. Addison-Wesley. München, 1999.

BUDSZUHN, FRANK / REICHEL, THOMAS: VISUAL C++ 6. Addison-Wesley. München, 1999.

BUNSE CHRISTIAN / VON KNETHEN ANTJE: Vorgehensmodelle kompakt. Spektrum, Akademischer Verlag. Heidelberg, 2002.

HEROLD, HELMUT / KLAR, MICHAEL / KLAR, SUSANNE: GoTo Objektorietierung. Addison-Wesley. München, 2001.

HUBWIESER, PETER: Didaktik der Informatik Grundlagen, Konzepte, Beispiele. Springer-Verlag. Berlin, Heidelberg, New York, 2000.

SCHUBERT, SIGRID / SCHWILL, Andreas: Didaktik der Informatik. Spektrum, Akademischer Verlag. Heidelberg, 2004.

STEVENS, PERDITA / POOLEY, ROB: UML – Softwareentwicklung mit Objekten und Komponenten. Addison-Wesley. München, 2000.

16. Internetliteratur[48]

BOLES, DIETRICH: Objektorientierte Softwareentwicklung. Universität Oldenburg, 1998.

BUCHBERGER, MICHAEL: Einführung in die UML, Fachhochschule Hagenberg, Studiengang Software Engineering, Sommersemester 2003.

DAHLMANNS, THOMAS: Software Engineering. Fachhochschule Lübeck, 2001.

GATZEMEIER, FELIX: Software Engineering. RWTH Aachen, 2003

HENNICKER, ROLF: Objektorientierte Softwareentwicklung, Universität München, 2003.

[48] Die Quelltexte befinden sich auf der CD-ROM im Verzeichnis Internetliteratur.

HESSISCHES KULTUSMINISTERIUM: Lehrplan Informatik Gymnasialer Bildungsgang Jahrgangsstufe 11 bis 13. Stand: 03.06.2002

HIRSCH, DANIEL: Kursus Java / C++. Universität Heidelberg, 2003.

KULTUSMINISTERKONFERENZ: Einheitliche Prüfungsanforderungen Informatik, Beschluss der Kultusministerkonferenz vom 01.12.1989 i.d.F. vom 05.02.2004.

SCHULTE, CARSTEN: Vom Modellieren zum Gestalten – Objektorientierung im Informatikunterricht. Universität GH Paderborn, 2001.

SIX, HANS-WERNER / WINTER, MARIO: Software Engineering I. Fernuniversität Hagen, 2004.

TELLIOGLU, HILDA: PM: Prozessmodelle in der Softwareentwicklung. Technische Universität Wien, 2004.

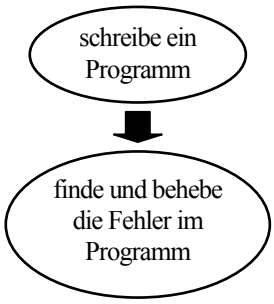
WAHL, GÜNTER: UML kompakt, OBJEKTspektrum, 2/1998.

Anhang

Lösungen zur 1. Unterrichtseinheit

Arbeitsblatt 1: Der Entwicklungsprozess

„Urmodell:

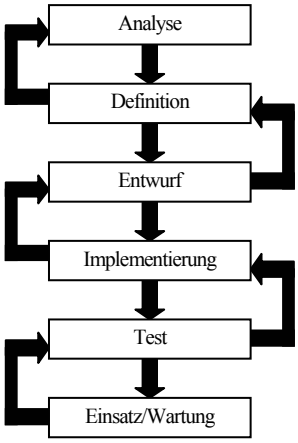


Merkmale: - keine Iteration möglich

Vorteile: - einfaches Modell
- wenig Management-Aufwand

Nachteile: - kein Risikomanagement möglich, da Probleme erst am Ende sichtbar werden
- keine Benutzerbeteiligung

Wasserfallmodell:

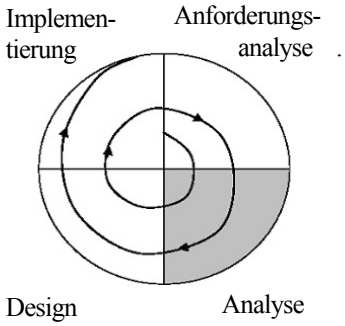


Merkmale: - klare Trennung der verschieden Phasen
- das Modell ist streng sequentiell
- Iterationen sind nur zwischen zwei Stufen erlaubt

Vorteile: - einfaches Modell
- wenig Management-Aufwand

Nachteile: - strenge sequentielle Vorgehensweise ist in der Praxis kaum möglich, da Fehler aus früheren Phasen häufig erst später erkannt werden und Anforderungen sich ändern können
- Schwierigkeiten in einer Phase verzögern das Gesamtprojekt
- keine Benutzerbeteiligung, da Anerkennung des Produkts durch den Kunden erst nach Abschluss der gesamten Entwicklung möglich ist

Spiralmodell:



Merkmale: - risikogetriebenes Modell
- iterative Entwicklung basierend auf dem letzten Durchlauf
- separate Zyklen für verschiedene Komponenten
- keine Trennung von Entwicklung und Wartung

Vorteile: - flexibles Modell und leichte Umdirigierung
- periodische Überprüfung des Modells
- integrierte Risikoabwägung
- frühzeitige Eliminierung von Fehlern
- Betrachtung von Alternativen

Nachteile: - hoher Managementaufwand
- nicht so gut geeignet für kleine und mittlere Projekte

Lösungen zur 2. Unterrichtseinheit:

Arbeitsblatt 2: Das objektorientierte Prozessmodell

Objektorientierte Analyse

Aufgabenstellungen:	- Analyse der Aufgabenstellung - Festlegung der Anforderungen an das System - Festlegung organisatorischer Richtlinien und Rahmenbedingungen
Produkte:	- Pflichtenheft - Erstellung des statischen und dynamischen Modells (OOA-Modell) - Prototyp der Benutzeroberfläche
Ziel:	Eine „Vereinbarung“ zwischen Anwender und Entwickler, die als Grundlage für die weitere Systementwicklung dient.

Objektorientierter Entwurf

Aufgabenstellung:	Weiterentwicklung des statischen und dynamischen Modells
Produkte:	Klassendiagramm, Objektdiagramm und Sequenzdiagramm
Ziel:	Abbildung der realen Welt der Aufgabenstellung in einem Modell.

Objektorientierte Implementierung

Aufgabenstellungen:	Codierung des Entwurfmodells in eine Programmiersprache
Ziel:	Ablauffähiges Programm

Objektorientierter Test

Aufgabenstellungen:	- Test der einzelnen Komponenten - schrittweises Zusammenfügen einzelner Komponenten mit anschießendem Test - Abnahmetest in der Anwendungsumgebung
Ziel:	Ablauffähiges Programm

Erarbeitungsphase II: Erstellung eines Pflichtenhefts

1. Zielbestimmung
   - Muss-Kriterien
     - periodische und individuelle Abfrage der Messdaten
     - automatisches Auslösen von Warnmeldungen
     - manuelles Auslösen von Warnmeldungen durch autorisierte Benutzer
     - Administratoren können Benutzer anlegen, löschen sowie Benutzerdaten ändern
     - privilegierte Benutzer können ihr Passwort ändern
   - Soll-Kriterien
     - Protokollierung der ausgelösten Warnmeldungen in einer Datenbank
     - Protokollierung kann nur von autorisierten Benutzern eingesehen werden
     - Schnittstellenparameter, Triggerintervall und Verzeichnispfade der Datenbank sollen in der Registry angelegt werden
     - Ereignisse am Server werden geloggt
   - Abgrenzungskriterien
     - die Administration der Benutzer kann nur am Server erfolgen
     - die Software ist nur mit dem Betriebssystem Windows 2000 lauffähig
     - es kann sich nur ein Client am Server anmelden (Singleuserbetrieb)
2. Produkteinsatz
   - Client-Software: Office-Anwender ohne besondere Vorkenntnisse

- Server-Software: Fachinformatiker – Anwendungsentwicklung

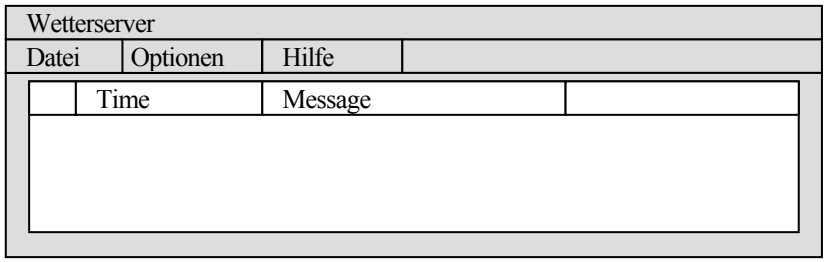
3. Produktübersicht
   die Software dient zum Ablesen von telemetrischen Daten

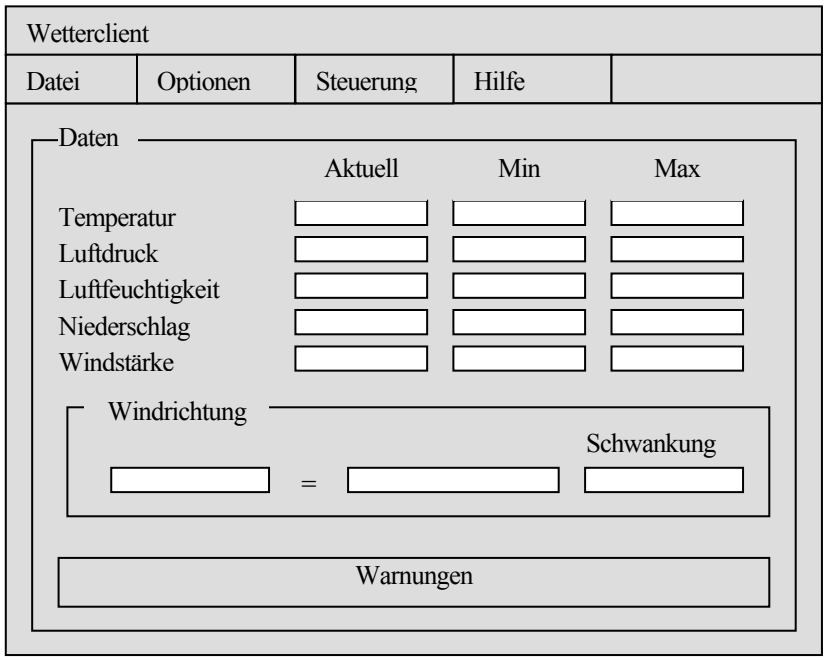
4. Produktfunktionen
   - Server:
     - Protokollierung aller Ereignisse
     - Messung und Bereitstellung der Windrichtung, Windgeschwindigkeit, Luftdruck, Luftfeuchtigkeit sowie der Temperatur
     - Benutzerverwaltung
     - Auslösen und Protokollieren von Warnmeldungen
   - Client:
     - Darstellung der Messwerte
     - individuelles Auslösen von Warnmeldungen
     - Betrachtung der Protokollierung der Warnmeldungen

5. Produktdaten
   - Datenhaltung für die Ermittlung des 24-Stunden-Maxi- und -Minimums, die Benutzerverwaltung, die Protokollierung der Alarmereignisse, die Ereignisse am Server und die Konfigurationsdaten der serielle Schnittstellen, der TCP/IP-Verbindung, des Triggers sowie die Namen der Datenbanken

6. Produktleistungen
   - Einschränkungen sind aufgrund der Rechnerausstattung nicht zu erwarten.
   - bei Ausfall der Standleitung zwischen der Wetterstation auf dem Felsberg in Bensheim und der Zentrale in Darmstadt können keine Daten übermittelt werden.

7. Benutzeroberfläche





8. Nichtfunktionale Anforderungen
   - keine

9. Technische Produktumgebung
   - Hardware:
     - Client-Rechner: Intel PII 333 MHz, 256 MB Arbeitsspeicher, 8 GB Festplatte
     - Server-Rechner: Intel PI 166 MHz, 64 MB Arbeitsspeicher, 2 GB Festplatte
   - Software
     - Client-Rechner: Windows 2000 Betriebssystem und Office 2000
     - Server-Rechner: Windows 2000 Betriebssystem und Office 2000

Lösungen zur 3. Unterrichtseinheit:

Festigungsphase I: Erstellung von Anwendungsfalldiagrammen

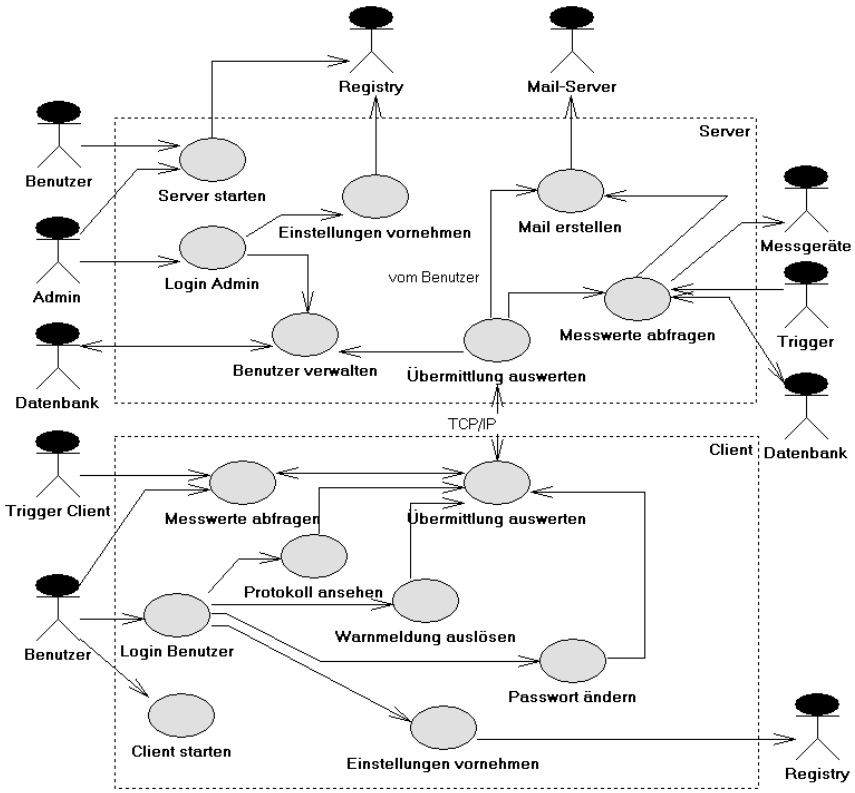
- Bestimmung der Akteure und Systemgrenzen:

  Akteure: Admin, Benutzer, Trigger, Datenbank, Registry, Messgerät und Mail-Server
  Systemgrenzen: Client, Server

- Identifizierung der Anwendungsfälle:

  Server: Server starten, Login Admin, Benutzer verwalten, Einstellungen vornehmen, Messwerte abfragen, Übermittlung auswerten und Mail erstellen
  Client: Client starten, Login Benutzer, Einstellungen vornehmen, Messwert abfragen, Protokoll ansehen, Warnmeldung auslösen, Passwort ändern, Übermittlung auswerten

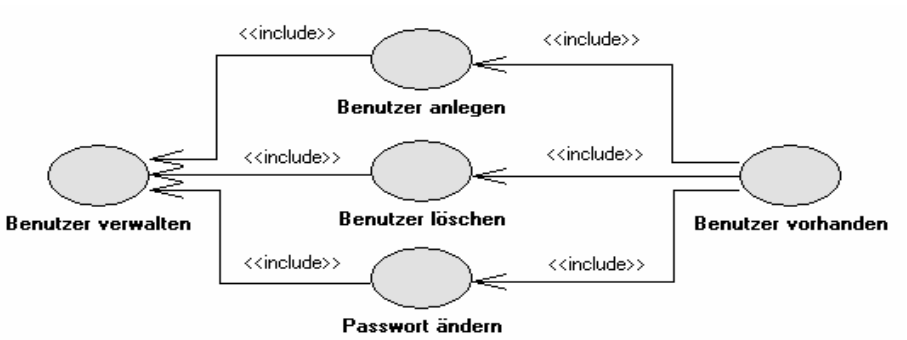
- Anwendungsfalldiagramm[49]



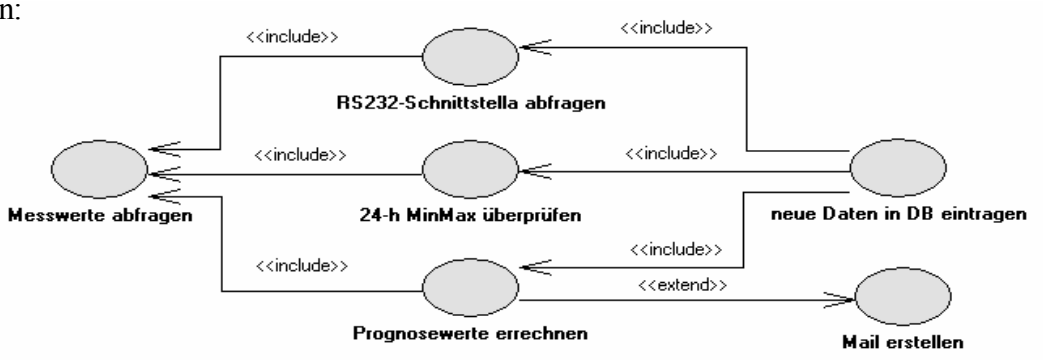
[49] Die Darstellung des Anwendungsfalldiagramms erfolgt bei den Schülern von Hand.

Festigungsphase II: include- und extend-Beziehungen

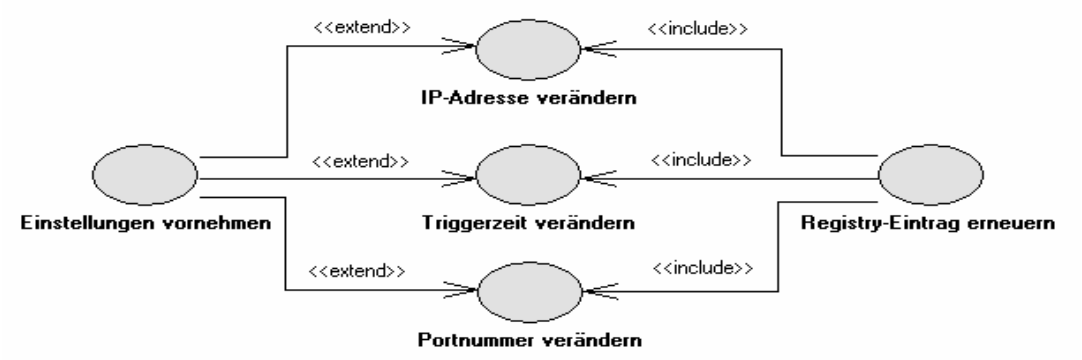
Benutzer verwalten



Messwerte abfragen:



Einstellungen vornehmen (Server):



Lösungen zur 3. und 4. Unterrichtseinheit:

Arbeitsblatt 4: Erstellung von Anwendungsfalldiagrammen

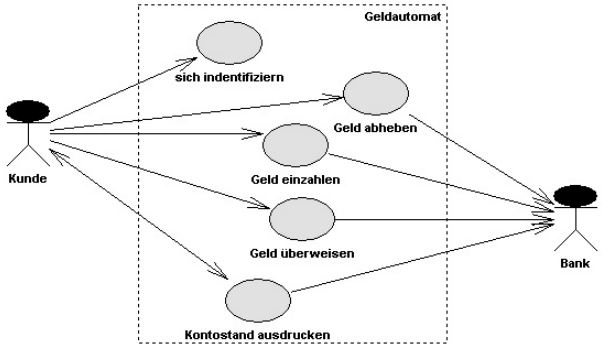
Aufgabe 1:

Ein Anwendungsfalldiagramm beschreibt die möglichen Interaktionen in Form von Anwendungsfällen von externen Akteuren mit dem System. Ein Anwendungsfall beschreibt, wie Anwender das System nutzen, um eine Aufgabe zu erledigen, die ein definiertes Ziel hat. In dem Anwendungsfall ist für ein Ziel eines solchen Akteurs in Textform zusammengefasst, wie Akteure und System kommunizieren und kooperieren, um dieses Ziel zu erreichen.

Aufgabe 2:

Akteure:	Kunde und Banksystem
Anwendungsfälle:	Kontostand anzeigen und ausdrucken, Geld abheben, Geld einzahlen, Geld überweisen, und sich identifizieren.

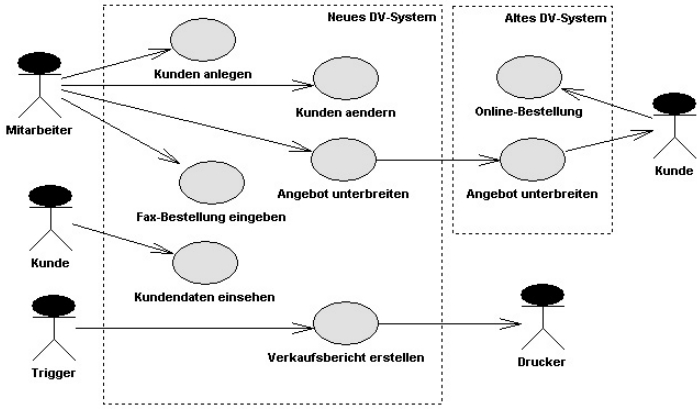
Anwendungsfalldiagramm



Aufgabe 3:

Akteure: Kunde, Mitarbeiter, Kunde Trigger und Drucker
Anwendungsfälle: Kunden anlegen, Kunden ändern, Angebot unterbreiten, Faxbestellung eingeben, Kundendaten einsehen und Verkaufsbericht erstellen

Anwendungsfalldiagramm:



Lösungen zur Unterrichtseinheit 3 und 4:

Festigungsphase I: Erstellung von Anwendungsfalldiagrammen mit der Software Jumli

Die Darstellung der Lösungen zur Festigungsphase I und II der 3. Unterrichtseinheit erfolgte bereits mit der Software Jumli und entspricht somit der Lösung dieser Phase.

Festigungsphase II: Erstellung von Anwendugsfallschablonen

- **Anwendungsfall „Messwerte lesen“**

Kurzbeschreibung: Es werden Messungen der Windrichtung, des Luftdrucks, der Luftfeuchtigkeit, der Windgeschwindigkeit und der Temperatur durchgeführt.

Vorbedingung: Anfrage vom Anwendungsfall „Messwerte abfragen“.

Nachbedingung: Es wurden Messdaten vom COM-Port gelesen und die aufgearbeiteten Werte an den Anwendungsfall „Messwerte abfragen“ übermittelt.

Primärszenario:

1. Der Anwendungsfall „Messwerte abfragen“ baut eine TCP/IP Verbindung zum Anwendungsfall „Messwerte lesen“ auf.
2. Windrichtung, Luftdruck, Luftfeuchtigkeit, Windgeschwindigkeit und Temperatur werden vom COM-Port gelesen und aufgearbeitet.
3. Min- bzw. Maxwerte werden aktualisiert.
4. Messdaten werden an den Anwendungsfall „Messwerte abfragen“ übermittelt.
5. TCP/IP Verbindung wird getrennt.

Sekundärszenarien: Eine der fünf Messungen konnte nicht durchgeführt werden.

- **Anwendungsfall „24h-Extremwerte aktualisieren“**

Kurzbeschreibung: Nachdem der Luftdruck, Windgeschwindigkeit und die Luftfeuchtigkeit gemessen wurde, wird der Minimal- bzw. Maximalwert der letzten 24 Stunden ggf. aktualisiert und alle Werte werden angezeigt.

Vorbedingung: Temperatur, Luftdruck, Windgeschwindigkeit und die Luftfeuchtigkeit wurden gemessen.

Nachbedingung: Die Extremwerte sind ggf. aktualisiert und alle neuen Werte werden angezeigt.

Primärszenario

1.Datenbank mit den Werten der letzten 24 Stunden wird geöffnet.
2.Daten, die älter als 24 Stunden sind, werden gelöscht und die neuen Daten werden eingetragen.
3. Neue Min- und Maxwerte werden ermittelt.
4. Datenbank wird geschlossen.

Sekundärszenarien: Fehlerhafter Zugriff auf den Luftdruckmesser.

Arbeitsblatt 5: Erstellung von Anwendungsfalldiagrammen

a) Akteure: Kunde und Servicetechniker

b) Anwendungsfälle

- **Pfandgegenständen zurückgeben**

Kurzbeschreibung: Der Kunde gibt einen Pfandgegenstand zurück und der Pfandwert erhöht sich entsprechend.

Vorbedingung: Kunde will eine Getränkedose, Pfandflasche oder Getränkekiste zurückgeben.

Nachbedingung: Der Automat ist bereit für neue Rückgaben.

Primärszenario:

1. Gibt der Kunde einen Pfandgegenstand zurück, wird vom System durch eine Vermessung festgestellt, um welchen Typ Getränkedose, Pfandflasche bzw. Kiste es sich handelt.
2. Wird der Pfandgegenstand angenommen, erhöht das System den Wert der zurückgegebenen Pfandgegenstände des Kunden sowie die Anzahl aller am Tag zurückgegebenen Pfandgegenstände.
3. Der Kunde drückt den Knopf zum Beenden der Rückgabe und erhält einen Beleg, auf dem die Anzahl und der Pfandwert der zurückgegebenen Pfandgegenstände aufgedruckt sind.

Sekundärszenarien: Wird der Pfandgegenstand nicht angenommen, wird "Ungültig" auf dem Display ausgegeben.

- **Tagesreport erstellen**

Kurzbeschreibung: Servicetechniker fordert das System auf, den Tagesreport über die Menge der zurückgegebenen Pfandgegenstände auszudrucken.

Vorbedingung: Servicetechniker will Tagesreport abfragen.

Nachbedingung: Zähler werden auf Null zurückgesetzt und ein neuer Report startet.

Primärszenario: Das System druckt die Anzahl der Pfandgegenstände eines Typs sowie die Summe aller zurückgegebenen Pfandgegenstände aus.

- **Informationen ändern**

  Kurzbeschreibung: Servicetechniker will Daten des System ändern, wie z.B. Pfandpreis, Größe für jeden Pfandgegenstand.

  Vorbedingung: Servicetechniker will Änderungen vornehmen.

  Nachbedingung: Daten sind aktualisiert.

  Primärszenario: Servicetechniker gibt neue Daten ein.

- **Pfandgegenstand stecken geblieben**

  Kurzbeschreibung: Ein Pfandgegenstand ist beim Einzug stecken geblieben.

  Vorbedingung: Pfandgegenstand ist stecken geblieben.

  Nachbedingung: Kunde kann mit der Rückgabe fortfahren.

  Primärszenario:
  1. Alarm wird aktiviert, um den Servicetechniker zu rufen.
  2. Der Servicetechniker behebt den Fehler und stellt den Alarm aus.
  3. Der Kunde kann mit der Rückgabe der Pfandgegenstände fortfahren.
  4. Der Pfandwert der bisher vom Kunden zurückgegebenen Pfandgegenstände bleibt gültig.
  5. Der Kunde erhält keinen Pfand für den stecken gebliebenen Pfandgegenstand, da er erneut zurückgegeben werden muss.

  Sekundärszenarien: Fehler kann nicht behoben werden. Auf dem Display wird „Defekt“ ausgegeben.

- **Beleg ausdrucken**

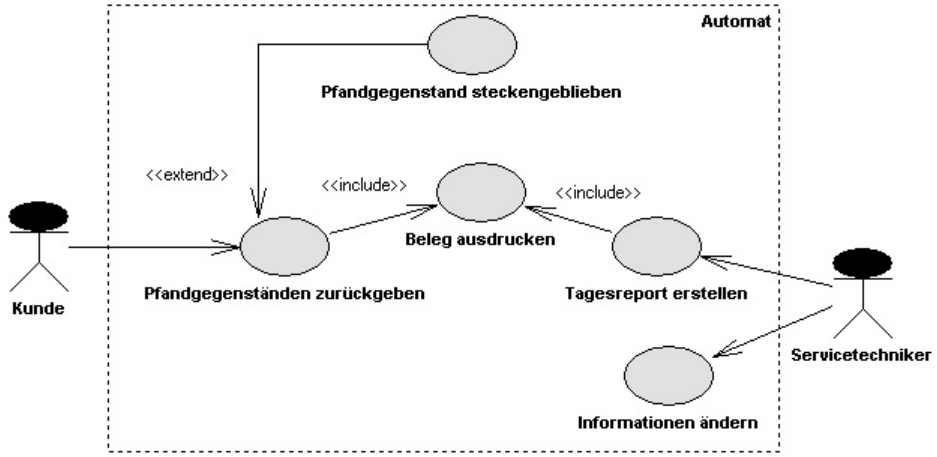
  Kurzbeschreibung: In beiden Anwendungsfällen „Rückgabe von Pfandgegenstände“ und „Erstelle Tagesreport“ wird ein Beleg ausgedruckt. Zur besseren Strukturierung und Wiederverwendbarkeit wird diese Funktion in einen neuen Anwendungsfall „Beleg drucken“ ausgelagert.

  Vorbedingung: Kunde oder Servicetechniker fordern Beleg an.

  Nachbedingung: Beleg wurde ausgedruckt.

  Primärszenario:
  1. Überprüfen der Hardware (Papier vorhanden etc.).
  2. Beleg wird ausgedruckt.

c) Anwendungsfalldiagramm

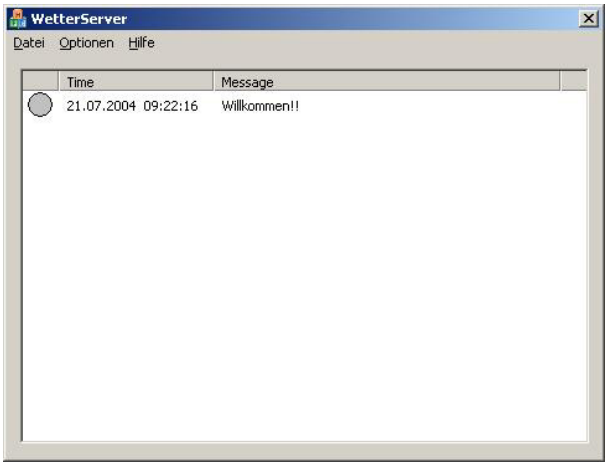


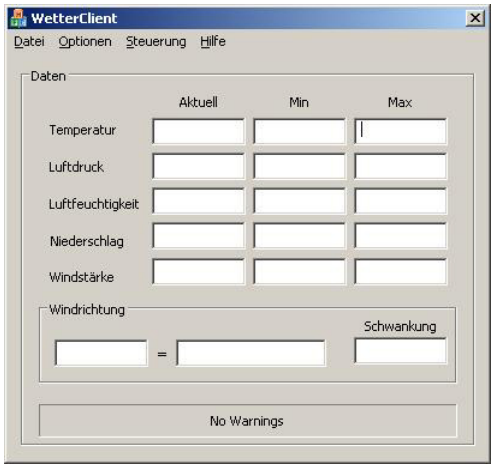
Lösungen zur 5. Unterrichtseinheit:

Festigungsphase I: Erstellung der Oberflächen

Die Lösungen befinden sich auf der CD-ROM im Ordner „Lösungen / 5.Unterrichtseinheit / WetterServer“ bzw. „Lösungen / 5.Unterrichtseinheit / WetterClient“. Zur Ansicht muss das Projekt geöffnet werden. Im Projektverzeichnis „Ressourcen / Dialog“ befinden sich die erstellten Dialoge. Mit Hilfe des Ressourcenbetrachters können diese aufgerufen werden.

Exemplarisch folgt an dieser Stelle die Darstellung des Startmenüs der Server- und Clientanwendung.





Lösungen zur 6. Unterrichtseinheit:

Festigungsphase I: Bestimmung von Objekten in UML-Notation

:Person1	
nachname	: Meier
vorname	: Max
funktion	: Pilot
gehalt	: 14000

:Person2	
nachname	: Schnidt
vorname	: Karl
funktion	: Flugbegleiter
gehalt	: 2700

:Person3	
nachname	: Müller
vorname	: Markus
funktion	: Flugbegleiter
gehalt	: 2700

:Flug1	
nummer	: 321
von	: München
nach	: Helsinki
am	: 28.04.2204
preis	: 99

:Flug2	
nummer	: 322
von	: Helsinki
nach	: München
am	: 28.04.2204
preis	: 99

:Flugzeug1	
typ	: Boing 707
sitzplaetze	: 120
stauraum	: 400
baujahr	: 1970

Arbeitsblatt 6: Erstellung von Klassen in UML-Notation

Aufgabe 1:

Dose	
- inhalt	: float
- maxInhalt	: float
+ trinken(float)	: bool
+ wieVoll(void)	: float

Aufgabe 2:

Uhr	
- stunden	: int
- minuten	: int
- sekunden:	: int
+ setZeit(int stunden, int minuten, int sekunden)	: bool
+ getStunden(void)	: int
+ getMinuten(void)	: int
+ getSekunden(void)	: int

Aufgabe 3:

Konto
- nummer : int - stand : double
+ getStand(void) : double + getNummer(void) : int + setStand(double) : void

Kunde
- name : string - geburtsort : string - wohnort : string
+ getName(void) : sting + setName(string) : void + getGeburtsdatum(void) : string + setGeburtsdatum(string): void + getWohnort(void) : string + setWohnort(string) : void

Aufgabe 4:

a)

CAppSettings
- Trigger : DWORD - IPAdresse : DWORD - Port : DWORD
+ SetTrigger(DWORD val) : void + SetIpAdresse(DWORD val): void + SetPort(DWORD val) : void + GetTrigger(void) : DWORD + GetIpAdresse(void) : DWORD + GetPort(void) : DWORD

b)

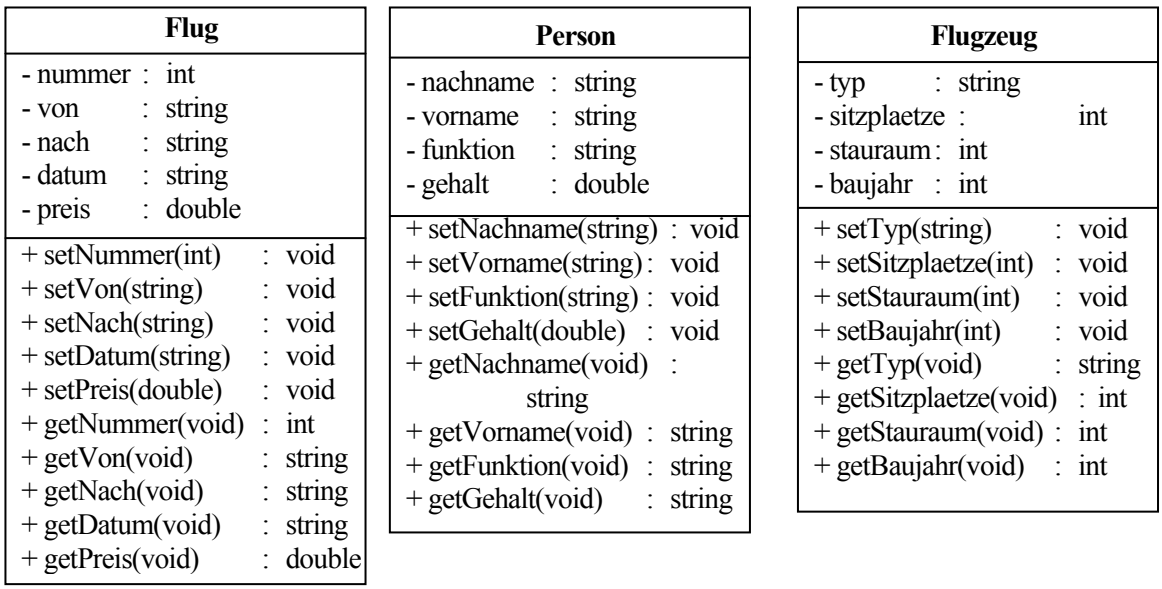
AppSettingsCom
+ Baudrate : int + BytSize : int + StopBit : int + Parity : int + Interface : int

AppSettingsDBs
+ UserDBPath : CString + MeasurementDBPath : CString + LogDBPath : CString

AppSettingsMeasurement
+ SettingsCOM : AppSettingsCOM + Trigger : DWORD

CAppSettings
- SettingsMeasurement : AppSettingsMeasurement - SettingsDBs : AppSettingsDBs - SettingsListenPort : int
+ SetMeasurement(AppSettingsMeasurement &Data) : void + SetDBs(AppSettingsDBs &DBs) : void + SetListenPort(int Port) : void + GetMeasurement(void) : &AppSettingsMeasurement + GetDBs(void) : &AppSettingsDBs + GetListenPort(void) : int

Aufgabe 5:



Lösungen zur 7. Unterrichtseinheit:

Übungsaufgaben:

Der Quellcode der Übungsaufgaben befindet sich im Ordner „Lösungen / 7.Unterrichtseinheit“.

Hinweis zur Aufgabe 3: Der Unterschied zwischen einem Objektattribut und einem Klassenattribut wurde noch nicht besprochen. Darum muss bei Besprechung der Aufgaben dieser Unterschied klar herausgearbeitet werden.

Lösungen zur 8. Unterrichtseinheit:

Klausuraufgaben:

Aufgabe 1

Die Firma SoftTec möchte für eine Firma, die Werkzeuge herstellt, eine Software erstellen, die die Geschäftsabläufe unterstützt. Das relativ unerfahrene Team von SoftTec entscheidet sich, nach dem Wasserfallmodell vorzugehen. Geben Sie an, welche Phasen dabei durchlaufen werden. Formulieren Sie für jede Phase ein knappes Dokument (z.B. in Stichpunkten, 1-2 Sätze), das als Ergebnis dieser Phase auftreten könnte.

Aufgabe 2

Welche Inhalte sollte ein Pflichtenheft beinhalten? Geben Sie zu jedem Punkt eine kurze Erläuterung (nicht länger als drei Sätze).

Aufgabe 3

Problembeschreibung:
Eine Firma will ihre Artikel und Bestellungen verwalten. Für jeden Artikel werden die Artikelnummer, der Lieferant, die Artikelbezeichnung und der Verkaufspreis festgehalten. Jeder Artikel gehört zu einer Artikelgruppe. Neue Artikel werden vom Sachbearbeiter in den Bestand aufgenommen.
Die Lagerung der Artikel kann an mehreren Orten erfolgen. Beispielsweise lagert die Firma Artikel in ihren Filialen in Darmstadt, Bensheim und Frankfurt. An jedem Lagerort werden für alle Artikel der Maximal- und Mindestbestand, der aktuelle Bestand und der Lagerort gespeichert. Eine Lagerliste soll Auskunft über die Bestände geben.

Jeden Tag werden die Lagerbestände vom Lagerverwalter kontrolliert. Wird der Mindestbestand unterschritten, so wird ein Bestellvorschlag erstellt. Für jedes Lager wird errechnet, wie viele Artikel nachbestellt werden müssen. Die vorgeschlagene Anzahl errechnet sich aus dem Maximalbestand und dem aktuellen Bestand. Die Anzahl ist auf ein n-faches der Verpackungseinheit des Lieferanten abzurunden. Ein Kunde kann eine oder mehrere Bestellungen erteilen, die erfasst werden müssen. Jede Bestellung erhält eine eindeutige Bestellnummer. Außerdem werden das Bestelldatum, das Lieferdatum und die Portokosten festgehalten.
Jede Kundenbestellung kann sich auf mehrere Artikel beziehen. Für jeden bestellten Artikel ist die gewünschte Anzahl festzuhalten. Außerdem kann der Gesamtpreis für mehrere Artikel ungleich Anzahl * Verkaufspreis sein. Jede Kundenbestellung wird von einem Sachbearbeiter bearbeitet. Für jeden Sachbearbeiter sollen dessen Personalnummer, ein Kürzel, der Name und die Telefonnummer gespeichert werden.
Außerdem sollen folgende Statistiken erstellt werden:

- Vom Sachbearbeiter soll ermittelt werden, welchen Umsatz ein Kunde X im aktuellen Jahr erzielt hat.
- Vom Personalchef soll ermittelt werden, welchen Umsatz ein Sachbearbeiter X mit seinen verschiedenen Kunden erzielt hat.

a) Finden Sie die Akteure des Systems und geeignete Anwendungsfälle.
b) Stellen Sie den Sachverhalt in einem Anwendungsfalldiagramm dar.

Aufgabe 4

a) Was ist eine Methode?
b) Was ist eine Klasse?
c) Was ist der Unterschied zwischen Objekt und Klasse?
d) Was versteht man unter einem Konstruktor und was ist der Default-Konstruktor?
e) Stimmt es, dass ...
   - eine Klasse immer mindestens / genau / höchstens einen Konstruktor haben muss?
   - eine Klasse immer einen Default-Konstruktor haben muss?
   - der Compiler einen Default-Konstruktor erzeugt, wenn für eine Klasse keiner definiert wurde.
   - Objekte immer mit einem Konstruktor initialisiert werden?
   - ein Objekt, deren Klasse mindestens einen Konstruktor hat, immer mit einem Konstruktor initialisiert wird.
   - Klassen mit Konstruktor auch einen Destruktor haben müssen?
   - Klassen mit Destruktor auch einen Konstruktor haben müssen?

Lösungen

Aufgabe 1

- Analyse: Grobe Formulierung der Ziele, Erfassung des Problembereichs, Machbarkeitsstudie und Analyse des Ist-Zustandes.
- Definition: Erstellung des Lastenheftes, der Systemspezifikation und des Pflichtenheftes.
- Entwurf: Entwicklung der Systemarchitektur, Schnittstellen der Komponenten festlegen und Modulkonzept erarbeiten.
- Implementierung: Strukturierung der Komponenten in Module, Spezifikation einzelner Module sowie Codierung, Generierung und Wiederverwendung.
- Test: stufenweise Integration der einzelnen Module und Komponenten mit anschießendem Test. Anschließend testen, ob die Implementierung zusammen mit der Umgebung funktioniert.
- Einsatz/Wartung: Korrektur von Fehlern und Erweiterung oder Anpassung der Funktionalität.

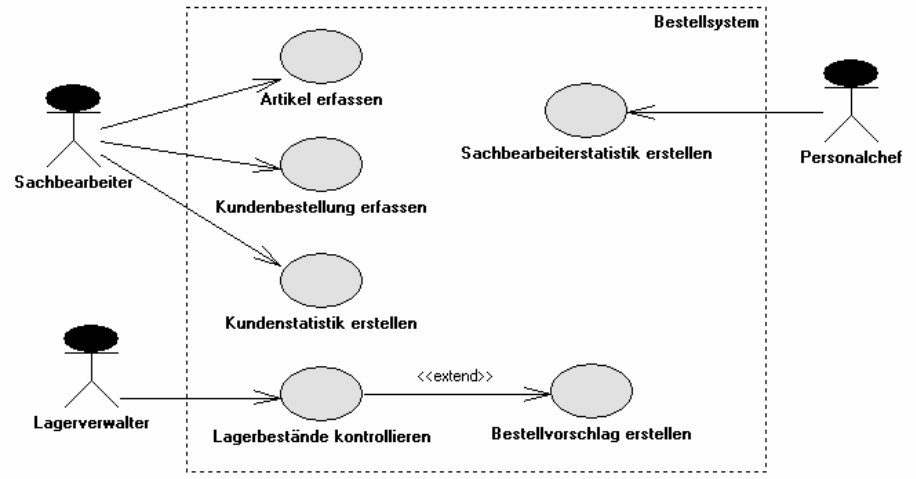
Aufgabe 2

1. Zielbestimmung: In diesem Abschnitt gilt es zu beschreiben, welche Vorhaben durch den Einsatz des Produkts erreicht werden sollen. Hierbei wird unterschieden zwischen Muss-, Wunsch- und Abgrenzungs-Kriterien.
2. Produkteinsatz: In diesem Abschnitt werden der Anwendungsbereich und die Betriebsbedingungen definiert.
3. Produktübersicht: In Form eines Übersichtsdiagramms wie Funktionsbaum, Anwendungsfall- oder Datenflussdiagramm wird eine grobe Übersicht über das Produkt gegeben.
4. Produktfunktionen: In Abhängigkeit von dem gewählten Übersichtsdiagramm im Punkt 3 erfolgt hier eine Konkretisierung und Detaillierung.
5. Produktdaten: In diesem Abschnitt erfolgt die detaillierte Beschreibung und deren voraussichtlicher Umfang der langfristig zu speichernden Daten.
6. Produktleistungen: Produktleistungen geben Leistungsanforderungen bzgl. Zeit oder Genauigkeit an einzelne Funktionen und Daten.
7. Benutzeroberfläche: In diesem Abschnitt werden - entsprechend dem Windows-Gestaltungs-Regelwerk oder unternehmenseigener Gestaltungs-Regelwerke - grundlegende Anforderungen an die Benutzungsoberfläche festgelegt, z.B. Fensterlayout, Dialogstruktur und Mausbedienung.
8. Nichtfunktionale Anforderungen: In diesem Abschnitt werden alle Anforderungen aufgeführt, die sich nicht auf die Funktionalität, die Leistung und die Benutzeroberfläche beziehen wie z.B. einzuhaltende Gesetze und Normen, die die Entwicklung entscheidend beeinflussen.
9. Technische Produktumgebung: Die technische Produktumgebung beinhaltet die Softwaresysteme, die für den Betrieb zur Verfügung stehen müssen, die Hardwarevoraussetzungen sowie die organisatorischen Voraussetzungen für den Betrieb.

Aufgabe 3

a) Akteure sind: Sachbearbeiter, Lagerverwalter und der Personalchef.
Anwendungsfälle sind: Artikel erfassen, Kundenbestellung erfassen, Lagerbestände kontrollieren, Bestellvorschlag erstellen und Kundenstatistik erstellen

b) Anwendungsfalldiagramm:

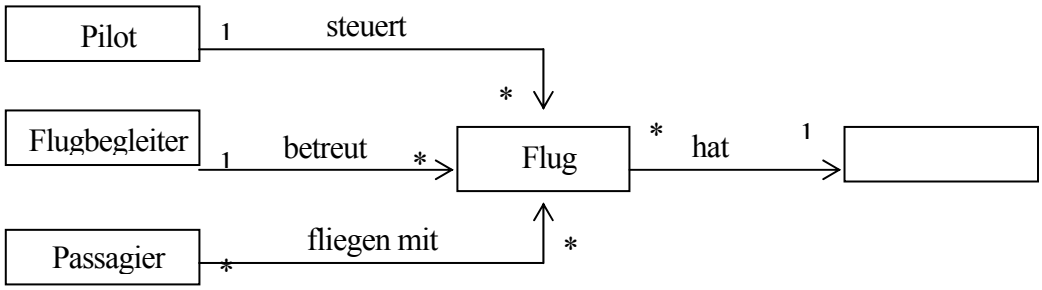


Aufgabe 4

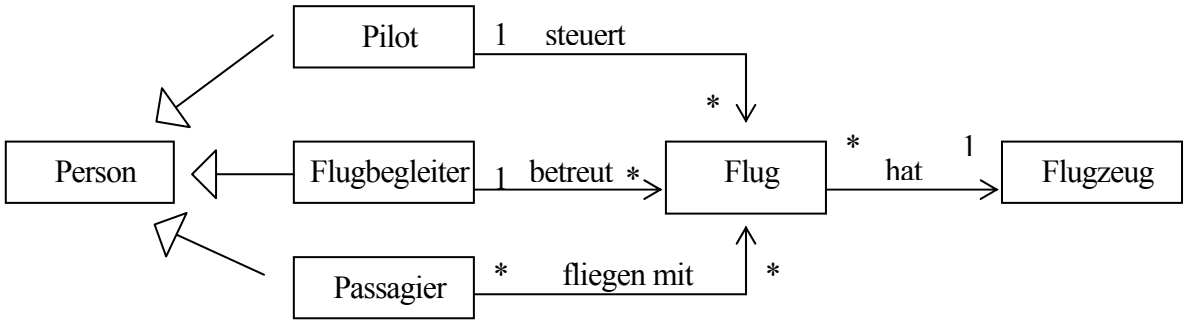
a) „Methoden“ sind Handlungsvorschriften zur Ausführung von Diensten.
b) Eine „Klasse“ ist eine Menge von „Objekten“ mit gleichen Merkmalen und gleichem Verhalten. „Merkmale“ sind die von einem Objekt ausführbaren „Methoden“ zusammen mit den „Attributen“ des Objekts. Die Klasse repräsentiert das Konzept oder den abstrakten Begriff, den man sich von ihren Objekten macht. Technisch ist eine Klasse ein Baumuster ihrer Objekte. Da eine Klasse die Merkmale und das Verhalten ihrer Objekte charakterisiert, bezeichnet man die Klasse auch als den „Typ“ der Objekte.
c) Ein Objekt ist eine Ausprägung einer Klasse, wenn man die Klasse als Typ (Baumuster) betrachtet und ein Objekt ist Element einer Klasse, wenn man eine Klasse als Menge gleichartiger Objekte betrachtet.
d) Ein Konstruktor ist eine Initialisierungsfunktion die automatisch aktiviert wird, wenn ein Objekt vom entsprechenden Typ angelegt wird. Der Default-Konstruktor wird aufgerufen, wenn es keine Hinweise des Programmieres gibt, die den Aufruf eines anderen Konstruktor fordern. Z. B. keine Parameter bei einer Variablendefinition.
e) Stimmt es, dass ...
   - eine Klasse immer mindestens ... Nein
   - eine Klasse immer einen Default-Konstruktor ... Nein
   - der Compiler einen Default-Konstruktor ... Nein
   - Objekte immer mit einem Konstruktor ... Nein
   - ein Objekt, deren Klasse mindestens ... Ja
   - Klasssen mit Konstruktor auch einen ... Nein
   - Klassen mit Destruktor auch einen ... Nein

Lösungen zur Unterrichtseinheit 9:

Folie 2: Wie sieht das Klassendiagramm aus?

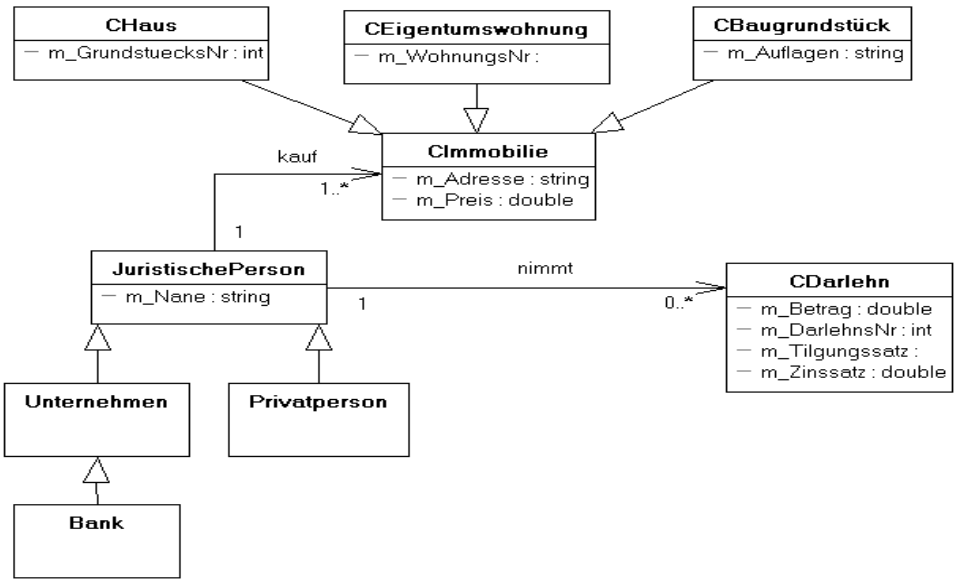


Folie 2: Wie sieht das Klassendiagramm aus?

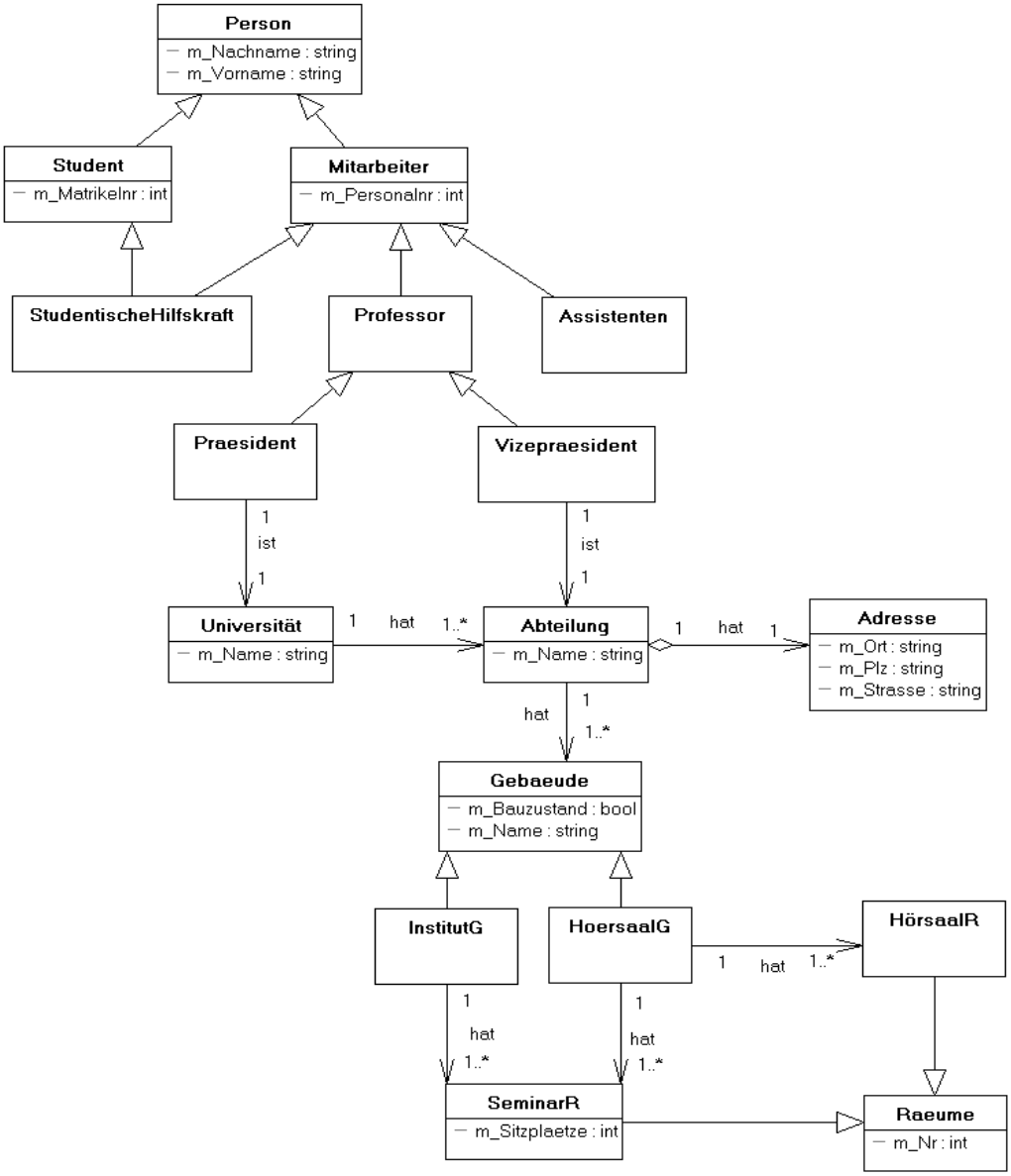


Arbeitsblatt 7: Erstellung von Klassendiagrammen

Aufgabe 1

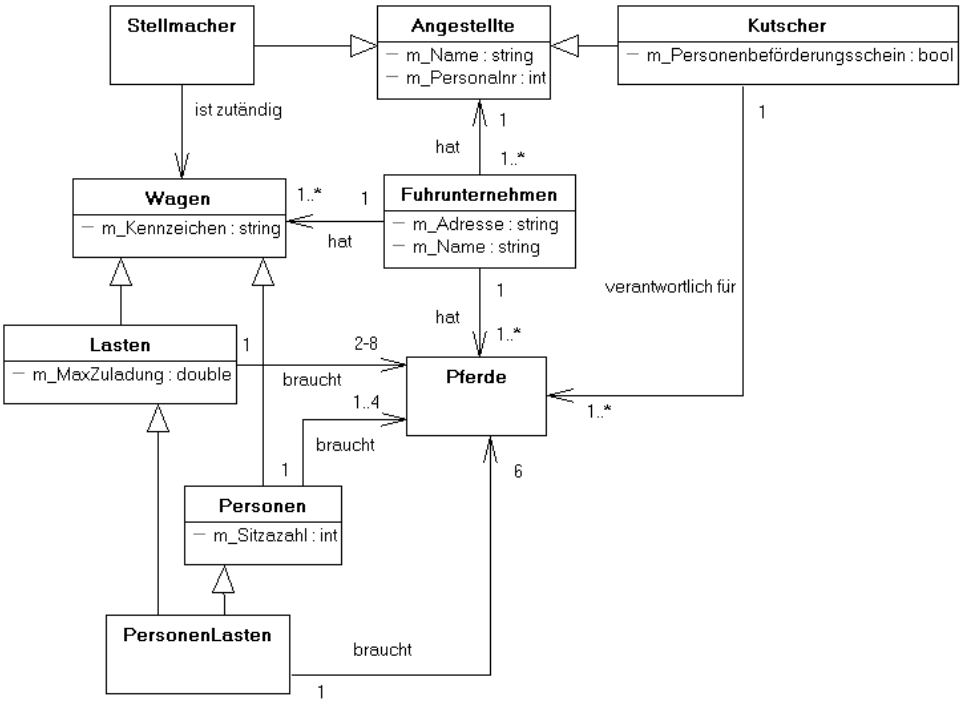


Aufgabe 2

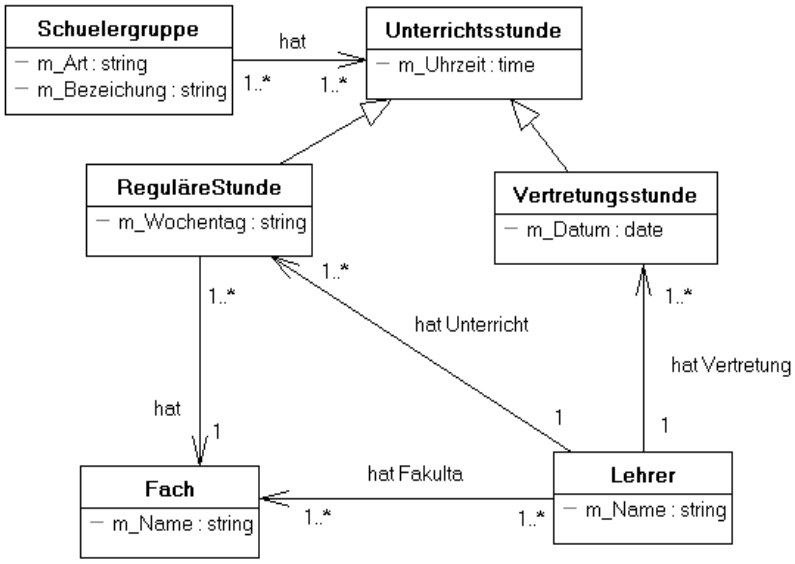


Arbeitsblatt 8: Erstellung von Klassendiagrammen

Aufgabe 1



Aufgabe 2



- Die Information, ob es sich bei einer Schülergruppe um eine Klasse oder einen Kurs handelt, kann einfacher in einem Attribut festgehalten werden.
- Da sich das Verhalten der Klassen Unterstufenklasse, Oberstufenklasse, Oberstufenkurs und Mittelstufenkurs nicht voneinander unterscheidet, können sie zusammengefasst werden.
- Die Angabe des Wochentags ist nicht für alle Unterrichtsstunden, sondern nur für reguläre Unterrichtsstunden erforderlich, da bei Vertretungsstunden der Wochentag durch das Datum gegeben ist.
- Die Assoziation zwischen Lehrer und Unterrichtstunde ist zu stark generalisiert. Sie enthält sowohl Stunden aus dem Lehrerstundenplan als auch Vertretungsstunden. Da der Lehrerstundenplan und der Vertretungsplan in der Anwendung separat gehandhabt werden, ist es sinnvoll, die Assoziation „hält" in der Aufgabenstellung durch die Assoziationen „Lehrerstundenplan" und „Vertretungsplan" zu ersetzen.

Lösungen zur Unterrichtseinheit 10:

Arbeitsblatt 9: Vererbung in C++

Aufgabe 1

a)
```
class CVan : public CAuto
{
private:
   int m_SitzAnzahl;
public:
      CVan(string art, int sitzAnzahl) : CAuto(art) {m_SitzAnzahl = sitzAnzahl;}
      int getSitzAnzahl(){return m_SitzAnzahl;}
}
```

b) in der Klasse Auto: Attribut m_Art und Methode string getArt()
in der Klasse Van: Attribut m_SitzAnzahl, Methode string getArt() und Methode int getSitzAnzahl()

Aufgabe 2

```
int main( )
{
 Abgeleitet * pa = new Abgeleitet;
 Basis     * pb = pa;
 pa->f( );
 pb->f( );
 cout << pa->x << endl;      // Ausgabe: 10
 cout << pb->x << endl;      // Ausgabe: 10
```

Das Programm erzeugt die Ausgabe 10, 10. Zeiger pa und pb zeigen zwar auf das gleiche Objekt, aber mit pa->f( ) und pb->f( ) werden unterschiedliche Methoden dieses Objekts aktiviert. Der Typ von pa und pb entscheidet über die jeweils zu aktivierende Methode.

Aufgabe 3

Das Programm ist korrekt und erzeugt die Ausgabe: 1111, 1112, 113
111, 112, 113

Die Methode a.print( ) aktiviert Abgeleitet::print für a und gibt die abgeleiteten Anteile aus. Der Zeiger pa zeigt auf das gleiche Objekt. Da pa aber vom Typ Basis* ist, wird mit pa->print() die Methode Basis::print aktiviert und die gibt den Basis-Anteil von a aus.

Arbeitsblatt 10: Überladen bzw. Überschreiben von Methoden

Aufgabe1

```
bla.m(a); //1        bla.m(b); //1        bla.m(c); //1        bla.m(d); //2        bla.m(e); //3        bla.m(e,d); //5
bla.m(e,e); // Kompiler kann sich nicht zwischen 5 und 6 entscheiden
```

Aufgabe 2

```
class Aklasse
{
   int a;
   void aMethode(int a){}
   void aMethode(int a, int b, int c){}          Überladung
};
```

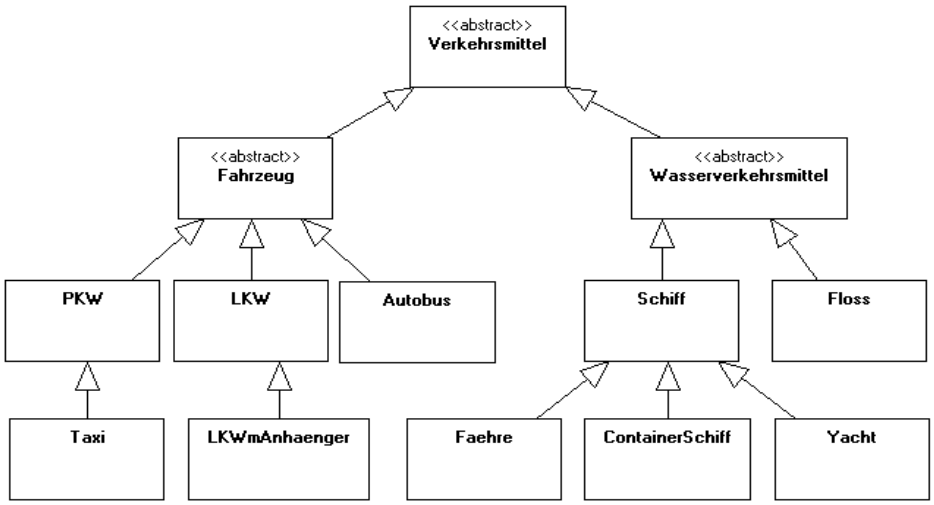
```
class B1Klasse : public AKlasse
{
   int b;
   void aMethode(int a, int b){}          Überladung
   void bMethode(int c){}                 nichts von beidem
};

class B2Klasse : public AKlasse
{
   void aMethode(int a){}                 Überschreibung
};
```

b)

```
class AKlasse                    Hier sind folgende Attribute gültig:  a .
class B1Klasse : AKlasse         Hier sind folgende Attribute gültig: a,b .
class B2Klasse : AKlasse         Hier sind folgende Attribute gültig: a .
class C1Klasse : B1Klasse        Hier sind folgende Attribute gültig: a,b,c .
```

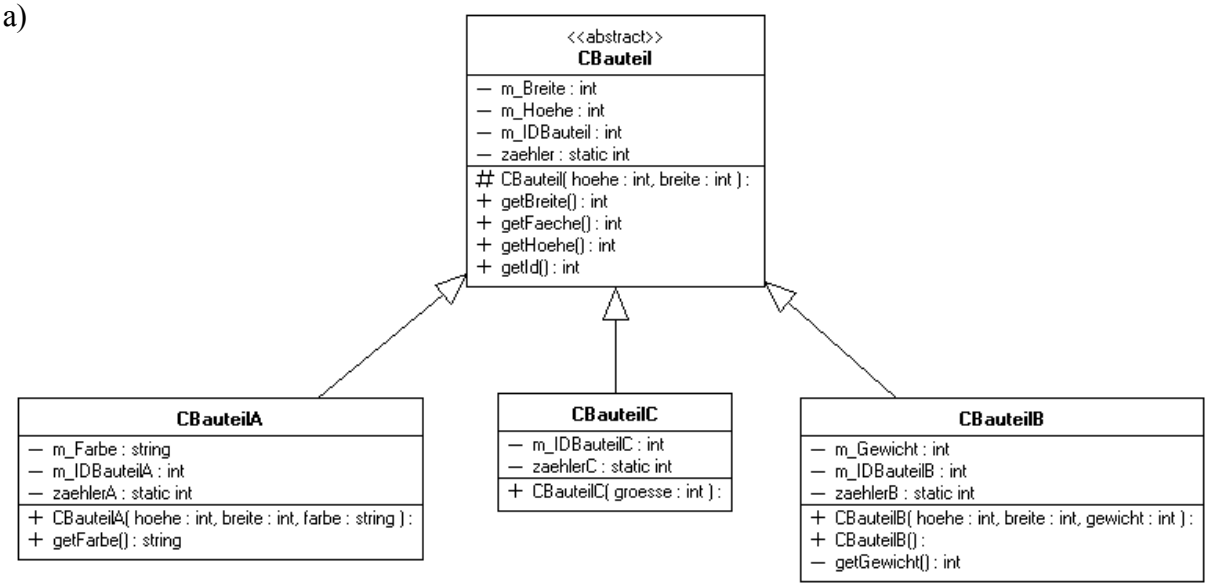
Arbeitsblatt 11: Abstrakte Klassen

Aufgabe 1



Aufgabe 2:

a)



b)

```
class CBauteil
{
private:
   static int zaehler;
   int m_IDBauteil;
   int m_Hoehe;
   int m_Breite;
protected:
   CBauteil( int hoehe, int breite ):
            m_Hoehe(hoehe), m_Breite(breite)
   {
      m_IDBauteil = zaehler;
      zaehler++;
   }
public:
   int getIDBauteil( ){return m_IDBauteil;}
   int getHoehe( ){return m_Hoehe;}
   int getFaeche( ){return (m_Hoehe*m_Breite);}
   int getBreite( ){return m_Breite;}
};
int CBauteil::zaehler = 1;
```

```
class CBauteilA : public CBauteil
{
private:
   static int zaehlerA;
   int m_IDBauteilA;
   string m_Farbe;
public:
   CBauteilA(int hoehe, int breite, string farbe ):
                     CBauteil(hoehe, breite ),m_Farbe(farbe)
   {  m_IDBauteilA = zaehlerA;
      zaehlerA++;
   }
   string getFarbe( )
   {return m_Farbe;
   }
   void print( )
   {  cout<<"Bauteil A"<<endl;
      cout<<"IDBauteil   :"<<getIDBauteil( )<<endl;
      cout<<"IDBauteilA:"<<m_IDBauteilA<<endl;
      cout<<"Breite      : "<<getBreite( )<<endl;
      cout<<"Hoehe       : "<<getHoehe( )<<endl;
      cout<<"Faeche      : "<<getFaeche( )<<endl;
      cout<<"Farbe       : "<<m_Farbe<<endl<<endl;
   }
};
int CBauteilA::zaehlerA = 1;
```

Die Klassen CBauteilB und CBauteilC sind analog der Klasse CBauteilA aufgebaut.

c)

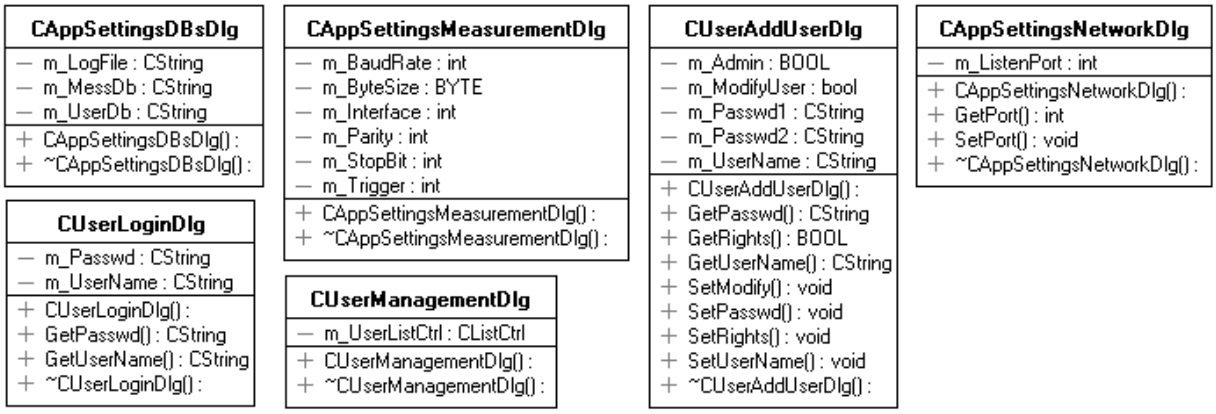
```
int main( )
{
            CBauteilA A1(11,12,"blau"), A2(11,12,"blau");
            CBauteilB B1(21,22,23), B2(21,22,23);
            CBauteilC C1(31), C2(31);
            A1.print( );   B1.print();      C1.print();
            A2.print( );   B2.print();      C2.print();
            return 0;
}
```

Lösungen zur Unterrichtseinheit 11:

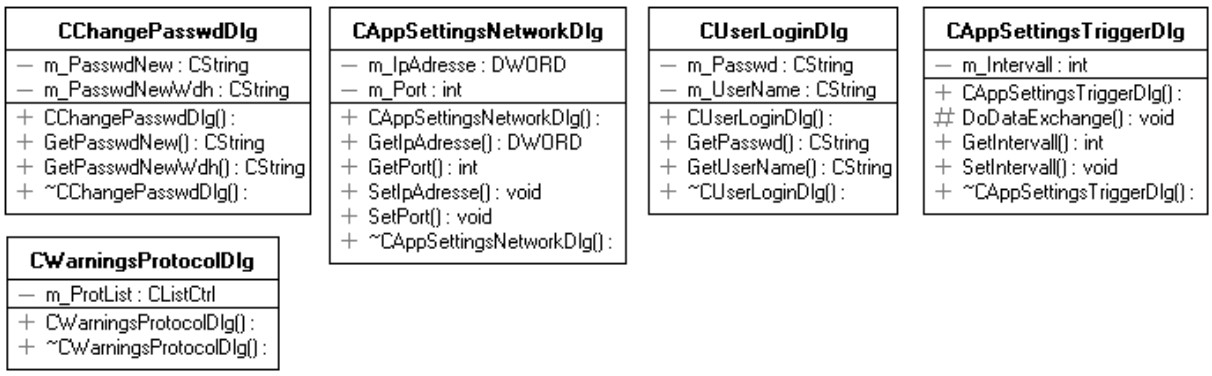
Festigung: Arbeitsauftrag - Entwicklung der Klassen für die Fachkonzept-Zugriffsschicht

a)

Server:



Client:



b) Der Quellcode der Klassen befindet sich auf der CD-ROM im Verzeichnis Lösungen / 11.Unterrichtseinheit / WetterServer bzw. Lösungen / 11.Unterrichtseinheit / WetterClient.

Lösungen zur Unterrichtseinheit 12:

Festigung I: Arbeitsauftrag 1 - Entwicklung der Klassen für die Datenhaltungs-Zugriffsschicht
Der Quellcode der Klassen befindet sich auf der CD-ROM im Verzeichnis Lösungen / 12.Unterrichtseinheit / Arbeitsauftrag 1 / WetterServer bzw. Lösungen / 12.Unterrichtseinheit / Arbeitsauftrag 1 / WetterClient.

Festigung II: Arbeitsauftrag 2 - Entwicklung der Klassen für die Datenhaltungs-Zugriffsschicht

a) Die Datenbanken befinden sich auf der CD-ROM im Verzeichnis Lösungen / 12.Unterrichtseinheit / Arbeitsauftrag 2 / a) Datenbanken.

b) z.B.:

c) Die MFC-ODBC-Consumer Klassen befinden sich auf der CD-ROM im Verzeichnis Lösungen / 12.Unterrichtseinheit / Arbeitsauftrag 2 / c) MFC-ODBC-Consumer Klassen.

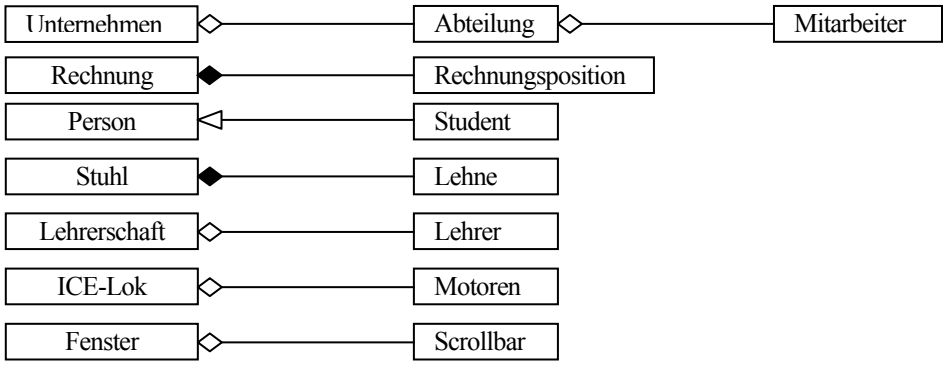
d) Das Testprogramm befindet sich auf der CD-ROM im Verzeichnis Lösungen / 12.Unterrichtseinheit / Arbeitsauftrag 2 / d) Testprogramm.

ACHTUNG: Die Anwendung kann nur dann fehlerfrei ablaufen, wenn der ODBC-Treiber für die Benutzerdatenbank an Ihrem Rechner installiert ist!!!

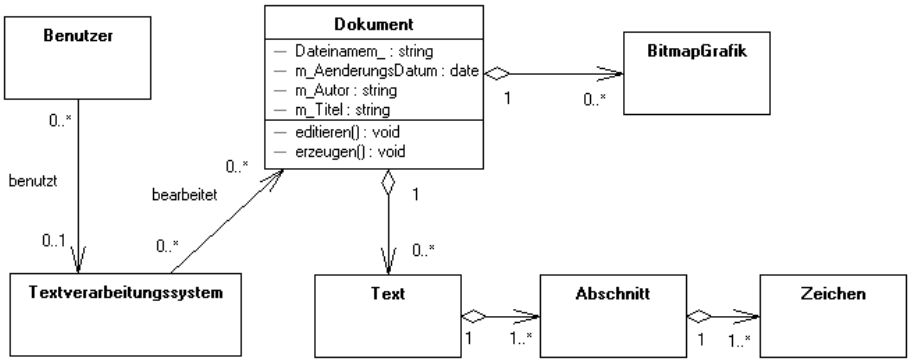
Lösungen zur Unterrichtseinheit 13:

Festigung 1: Arbeitsauftrag

a)

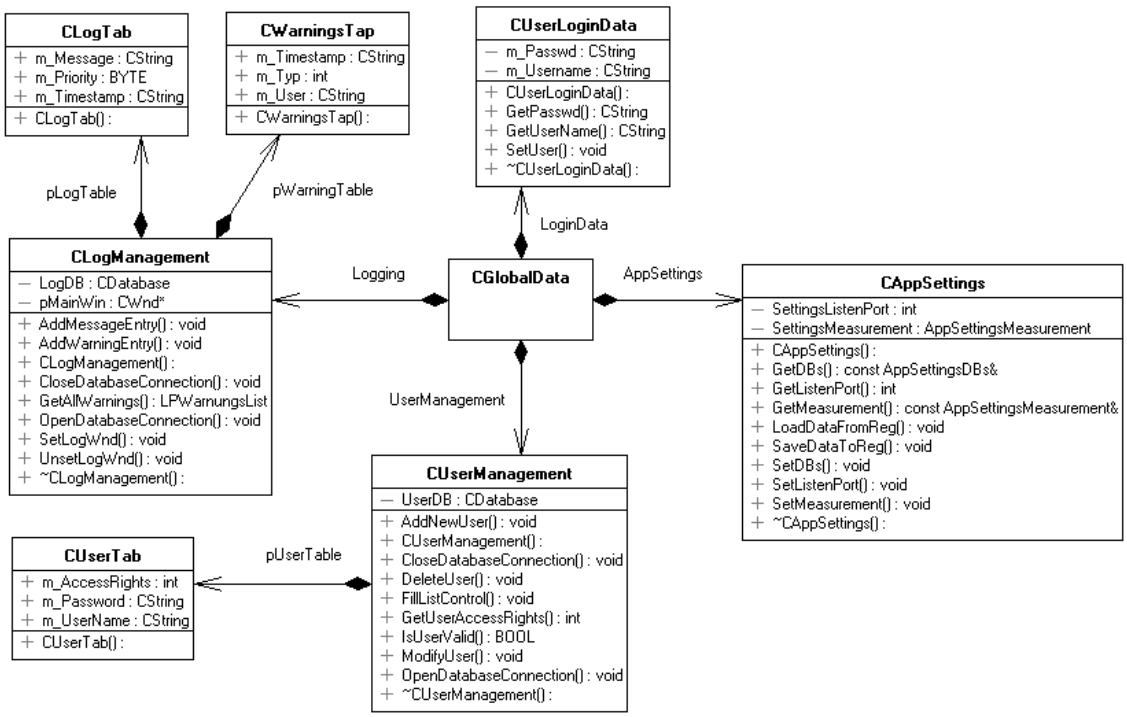


h)



Arbeitsblatt 12: Beziehungen zwischen Klassen

a) - c)

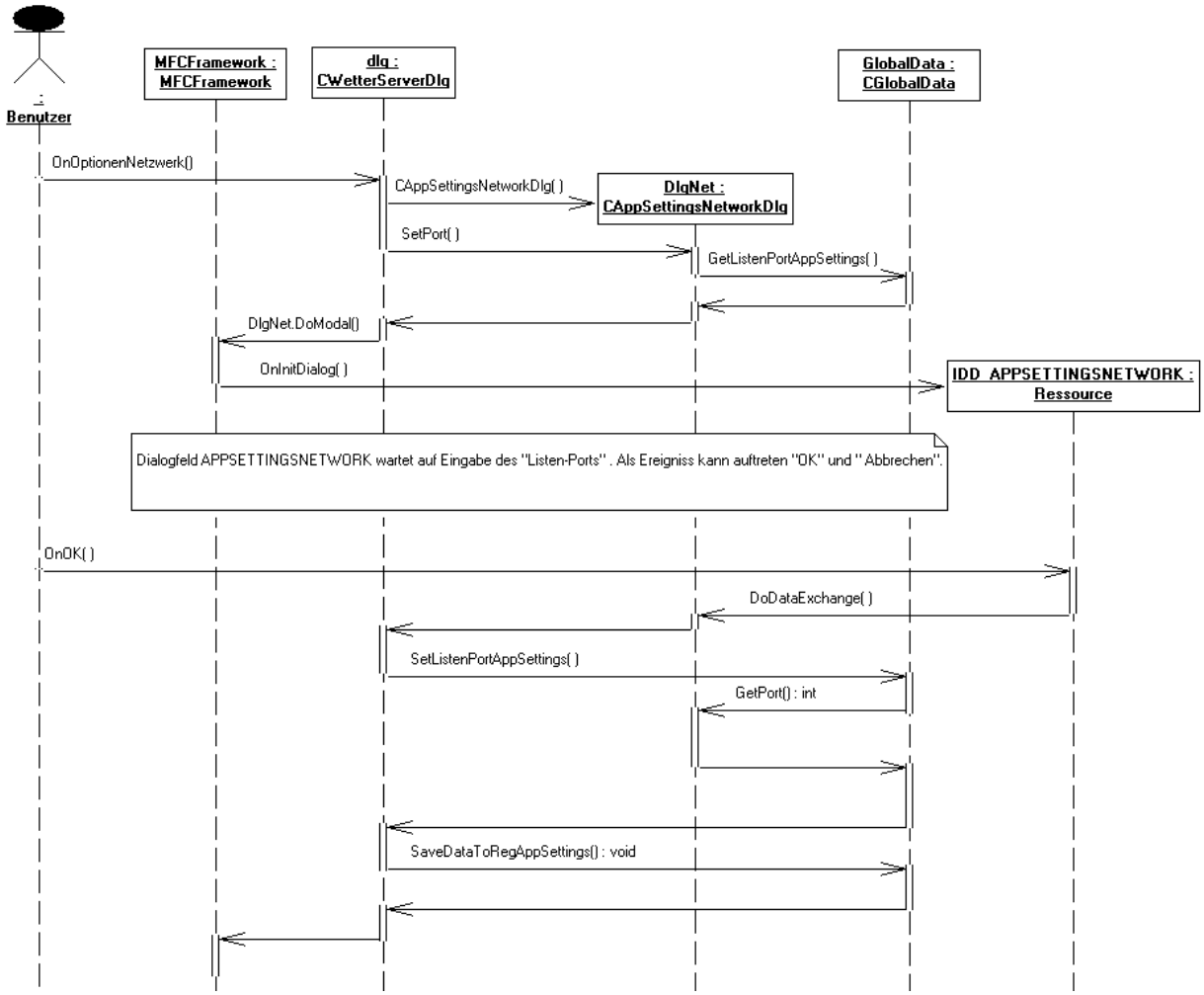


d) Der Quellcode der Klassen befindet sich auf der CD-ROM im Verzeichnis Lösungen / 13.Unterrichtseinheit.

Lösungen zur Unterrichtseinheit 14:

Arbeitsauftrag: Erstellung von Sequenzdiagrammen

Da alle Sequenzdiagramme einen anlogen Ablauf besitzen, wird an dieser Stelle exemplarisch das Sequenzdiagramm für die Änderung der Portnummer dargestellt.

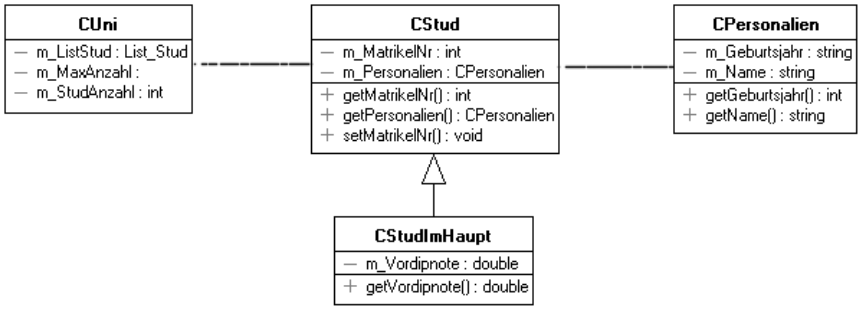


Lösungen zur Unterrichtseinheit 14:

Klausuraufgaben:

Aufgabe 1

Gegeben ist folgendes UML-Klassendiagramm. Es soll den vereinfachten Aufbau einer Universität darstellen.



a) Nennen Sie drei Beziehungen, die zwischen Klassen im Sinne der objektorientierten Softwareentwicklung bestehen können.

b) Offensichtlich besteht zwischen den Klassen CStud und CStudImHaupt eine Vererbungsbeziehung. Welche Beziehung kommt zwischen den Klassen CStud und CPersonalien in Frage?

c) Schreiben Sie C++-Klassen, welche die Klassen des UML-Klassendiagramms implementieren. Da das Diagramm über die Arbeitsweise der Methoden keine Angaben macht, sollen die Blöcke keine Anweisungen enthalten ({ }). Auch Konstruktoren müssen nicht angegeben werden. Es ist ausreichend, wenn Sie pro Klasse ein Attribut und eine Methode Ihrer Wahl aufführen. Beachten Sie bitte die Vererbung!

d) Implementieren Sie einen Konstruktor der Klasse CStud, CPersonalien, CStudImHaupt und CUni, so dass sie das leisten, was in den Kommentaren angegeben ist. Die Implementierung soll nicht inline erfolgen.

- CStud(CPersonalien personalien, int matrikelNr)

```
{
  // Der Konstruktor sorgt dafür, dass bei einer Instantiierung eines Objekts vom Typ CStud das
  // Attribut m_personalien den Wert des Parameters personalien und das Attribut
  // m_MatrikelNummer den Wert des Parameters matrikelNr erhält.
}
```

- CPersonalien(string name, int geburtsjahr)

```
{
  // Der Konstruktor sorgt dafür, dass bei einer Instantiierung eines Objekts vom Typ  Personalien das
  // Attribut m_Name den Wert des Parameters name und das Attribut m_Geburtsjahr den Wert des
  // Parameters geburtsjahr erhält.
}
```

- CStudImHaup(Stud stud, double vordipnote)

```
{
  // Der Konstruktor sorgt dafür, dass bei einer Instantiierung eines Objekts vom Typ CStudImHaupt das
  // Attribut m_Personalien den Wert des Attributs personalien von stud und das Attribut m_MatrikelNr
  //den Wert des Attributs matrikelNr von stud erhält. Dem Attribut m_vordipnote wird der Wert des
  // Parameters vordipnote zugewiesen.
}
```

- CUni(int maxAnzahl)

```
{
  // Der Konstruktor sorgt dafür, dass bei einer Instantiierung eines Objekts vom Typ CUni das Attribut
  // m_MaxAnzahl den Wert des Parameters maxAnzahl erhält. Das Attribut m_StudAnzahl wird auf 0
  // gesetzt.
}
```

e) Erweitern Sie die Klasse CUnit um eine Methode einschreiben, welche Studierende an der Universität immatrikuliert.

- int einschreiben(CStud neuStud)

```
{
  // Die Methode fügt das Objekt neuStud in die Liste m_ListStud am Ende an. Ist dieses nicht
  // mehr möglich, ist der Rückgabewert –1.
}
```

f) Erweitern Sie die Klasse CUnit um eine Methode exmatrikulieren.

- int exmatrikulieren(int matrikelNr)

```
{
  // Die Methode löscht den Studenten mit der übergebenen Matrikelnummer aus der Liste
  // m_ListStud. Ist dieses nicht möglich, weil die Matrikelnummer nicht in der Liste vorhanden ist,
  // so ist der Rückgabewert –1.
}
```

Aufgabe 2

In dieser Aufgabe geht es darum, die Verwaltung von Mietern und Mietobjekten wie Wohnungen oder Gewerberäumen zu modellieren. Die Mietgegenstände werden allgemein durch die abstrakte Klasse „Mietobjekt“ repräsentiert. Die Verwaltung beruht auf Objekten eines Typs „Mietbeziehung“, die eine Verbindung zwischen einem Mieter und einem Mietobjekt herstellt.

Mietobjekt
m_Adresse : string m_Flaeche : double
getAdresse() : string getFlaeche() : double

Mietbeziehung
m_Mieter : string m_Objeket : Mietobjekt
getMieter() : string getMietobjekt() : Mietobjekt

Geschaeft
m_Nebenraeume : int m_ Parkplaetze : int
get Nebenraeume () : int get Parkplaetze () : int

Wohnung
m_Zimmer : int
getZimmer() : int

a) Modellieren Sie ein Klassendiagramm für die Verwaltung von Mietern und Mietobjekten.
   - Entscheiden Sie hierzu, welche Attribute und Methoden privat bzw. public sind.
   - Entscheiden Sie, in welcher Beziehung die Klassen zueinander stehen

b) Implementieren Sie die Klassen des UML-Diagramms. Da das UML-Diagramm über die Arbeitsweise der Methoden keine Angaben macht, sollen in der Klassendefinition nur die Prototypen der Methoden angeben werden. Auch Kon- und Destruktoren müssen nicht angegeben werden.

c) Implementieren Sie einen Konstruktor der Klasse Mietobjekt, Wohnung und Mietbeziehung, so dass sie das leisten, was in den Kommentaren angegeben ist. Die Implementierung soll nicht inline erfolgen.

```
Mietobjekt(string adresse, float flaeche)
{
  // Der Konstruktor sorgt dafür, dass das Attribut m_Adresse den Wert des Parameters adresse und
  //das Attribut m_Flaeche den Wert des Parameters flaeche erhält.
}

Wohnung(String adresse, float wohnflaeche, int zimmerzahl)
{
  // Der Konstruktor sorgt dafür, dass das Attribut m_Adresse den Wert des Parameters adresse,
  // das Attribut m_Flaeche den Wert des Parameters wohnflaeche und das Attribut m_Zimmer den
  // Wert des Parameters zimmerzahl erhält.
}

Mietbeziehung(String einmieter, Mietobjekt einobjekt)
{
  // Der Konstruktor sorgt dafür, dass das Attribut m_Mieter den Wert des Parameters einmieter, und
  // das Attribut m_Objekt den Wert des Parameters einobjekt erhält.
}
```

d) Schreiben Sie ein Hauptprogramm, das ein Objekt vom Typ Mitbeziehung mit Namen eine Mietbeziehung instantiiert, dessen Attribute folgende Angaben speichern:

   Mieter: Hans Hein
   Mietobjekt: eine Wohnung mit der Adresse Samtstrasse 1, 64625 Bensheim
   Fläche der Wohnung: 112 m^2
   Zimmerzahl: 6

Aufgabe 3

a) Was bezeichnet man in C++ als „überladene“ bzw. „überschriebene“ Methode?

b) Gegeben seien die Klassen CBlub, CBla und CHop gemäß folgender Implementierung:

```
virtual class CBlub
{
public:
    void a(int i);          // (1)
    int b(void);            // (2)
};

class CBla : public CBlub
{
    void a(int i);          // (3)
    int b(void);            // (4)
};
```

```
class CHop : public CBla
{
    float b(float x);       // (5)
    int a(void);            // (6)
```

Beantworten Sie folgende Fragen zur oben gegebenen Klassendeklaration.

- In welcher Beziehung steht die Deklaration (4) zur Deklaration (2)?
- In welcher Beziehung steht die Deklaration (5) zur Deklaration (2)?
- In welcher Beziehung steht die Deklaration (6) zur Deklaration (1)?

b) CBlub sei eine Unterklasse von CBla gemäß folgender Implementierung:

```
class CBla
{
public:
    int x;
}
```

```
class CBlub : public CBla
{
public:
    int y;
```

Entscheiden Sie, ob folgende Anweisungen richtig sind, d. h. ohne Fehlermeldung compiliert werden können.

• CBla X = new CBla();	X.x = 7;	richtig	falsch
• CBla X = new CBla();	X.y = 9;	richtig	falsch
• CBlub Y = new CBlub();	Y.x=1;	richtig	falsch
• CBlub Y = new CBlub();	Y.y=2;	richtig	falsch
• CBla A = new CBlub();	A.y=3;	richtig	falsch
• CBla A = new CBlub();	((CBlub)A)x=4;	richtig	falsch
• CBla A = new CBlub();	((CBla)A).y=4;	richtig	falsch

Lösung

Aufgabe 1

a) Assoziation, Aggregation und Komposition

b) Komposition

c)

```
class CUni
{
private:
   List_Stud   m_ListStud;
   int         m_StudAnzahl;
   int         m_MaxAnzahl;
public:
   int getStudAnzahl(void);
   int einschreiben(CStud neuStud);
   int exmatrikulieren(int matrikelnummer);
};
```

```
class CStudImHaupt : public CStud
{
private:
   double      m_Vordipnote;
public:
   double getvordipnote(void);
};
```

```
class CStud
{
private:
   CPersonalien     m_Personalien;
   int              m_MatrikelNr;
public:
   CPersonalien getPersonalien(void);
   int getMatrikelNr(void);
   void setMatrikelNr(int matrikelNr);
};
```

```
class CPersonalien
{
private:
   string     m_Name;
   int        m_Geburtsjahr;
public:
   string getName(void);
   int getGeburtsjahr(void);
};
```

d)

- CStud::CStud(CPersonalien personalien, int matrikelNr)
  { m_Personalien = personalien;
     m_MatrikelNr = matrikelNr;}

- CPersonalien::CPersonalien(string name, int geburtsjahr)
  { m_Name = name;
     m_Geburtsjahr = geburtsjahr;}

- CStudImHaupt::CStudImHaup(Stud stud, double vordipnote) :CStud(stud.getPersonalien( ), stud.getMatrikelNr( ))
  { m_Vordipnote = vordipnote;}

- CUni::CUni(int maxAnzahl)
  { m_StudAnzahl = 0;
     m_MaxAnzahl = maxAnzahl;}

e)

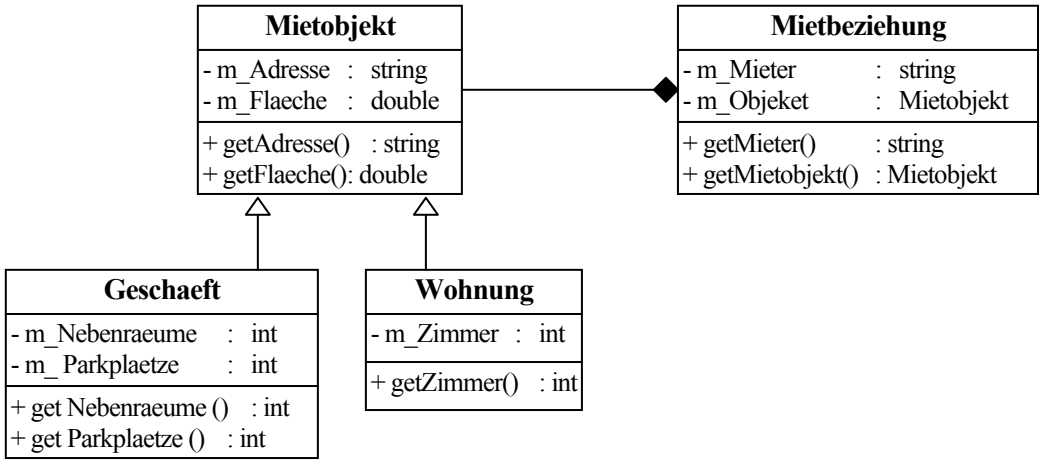
```
int CUni::einschreiben(CStud neuStud)
{
   if(m_MaxAnzahl = = m_StudAnzahl)
        return -1;
   m_ListStud.push_back(neuStud);
   m_StudAnzahl++;
   return 0;
}
```

f)

```
int CUni::exmatrikulieren(int matrikelNr)
{
   List_Stud::iterator it=m_ListStud.begin();
   while(it != m_ListStud.end())
   {
      if((*it).getMatrikelNr() = = matrikelNr){
           m_ListStud.erase(it);
           m_StudAnzahl--;
           return 1;
      }
      it++;
   }
   return 0;
}
```

Aufgabe 2

a)



b)

```
class Mietbeziehung
{
private:
      string mieter;
      Mietobjekt objekt;
public:
      string gibMieter( );
      Mietobjekt gibObjekt( );
};
```

```
class Wohnung : Mietobjekt
{
private:
      int m-Zimmer;
public:
      int getZimmer( );
};
```

```
class Geschaeft : Mietobjekt
{
private:
      int m_Nebenraeume;
      int m_Parkplaetze;
public:
      int getNebenraeume( );
      int getParkplaetze( );
};
```

```
class Mietobjekt
{
private:
      string m_Adresse;
      float m_Flaeche;
public:
      string getAdresse( );
      float getFlaeche( );
}:
```

```
Mietobjekt :: Mietobjekt (string adresse, float flaeche)
      {  m_Adresse = adresse;
         m_Flaeche = flaeche;}
Wohnung :: Wohnung (string adresse, float flaeche, int zimmer) : Mietobjekt(adresse, flaeche)
      {m_Zimmer = zimmer;}
Mietbeziehung(string einmieter, mietobjekt einobjekt)
      {m_Mieter = einmieter;
       m_Objekt = einobjekt;}
```

d)

```
void main( )
{
Wohnung testWohnung = new Wohnung("Samtstrasse 1, 64625 Bensheim ", 112, 6);
Mietbeziehung testMietbeziehung = new Mietbeziehung("Hans Hein", testWohnung);
}
```

Aufgabe 3

a) Überladene Methoden können für dasselbe Objekt aufgerufen werden und besitzen denselben Namen, aber unterschiedliche Signaturen, d. h., die Anzahl oder die Typen der Parameter unterscheiden sich. Der Rückgabetyp spielt keine Rolle. Überschriebene Methoden können nur im Zusammenhang mit Subklassen auftreten. Eine Unterklasse deklariert dabei eine Methode, die denselben Namen und dieselbe Signatur wie eine Methode der Basisklasse besitzt.

b)

- In welcher Beziehung steht die Deklaration (4) zur Deklaration (2)? überschreibt
- In welcher Beziehung steht die Deklaration (5) zur Deklaration (2)? überschreibt
- In welcher Beziehung steht die Deklaration (6) zur Deklaration (1)? überlädt

c)

• CBla X = new CBla();	X.x = 7;	richtig	
• CBla X = new CBla();	X.y = 9;		falsch
• CBlub Y = new CBlub();	Y.x=1;	richtig	
• CBlub Y = new CBlub();	Y.y=2;	richtig	
• CBla A = new CBlub();	A.y=3;		falsch
• CBla A = new CBlub();	((CBlub)A)x=4;	richtig	
• CBla A = new CBlub();	((CBla)A).y=4;		falsch

Ich versichere, diese Arbeit selbständig verfasst, keine anderen als die angegebenen Hilfsmittel verwendet und die Stellen, die in anderen Werken im Wortlaut, als Grafik oder dem Sinne nach entnommen sind, mit Quellenangaben kenntlich gemacht zu haben.

_____