Drei- Schichten- Architektur. Die Problematik des Zugriffs auf eine relationale Datenbank aus einer JAVA-Applikation.

(wl, fj) Im Unterricht wird der Zugriff auf eine Relationale Datenbank aus einer Objektorientierten Anwendung meist über ein Beispiel realisiert, in welchem es nur eine Datenbanktabelle gibt und diese exakt den Bauplan für die Klasse vorgibt. Ausgehend von einer einfachen Problemstellung wird dargestellt, dass der Zugriff auf eine Relationale Datenbank (RDB) sehr komplex ist. Im Fazit wird vorgeschlagen, die Thematik der Objektpersistenz in Lehrplänen ohne RDB zu realisieren.

Vorbemerkung

Um die Softwarentwicklung planbar, überschaubar und auch unter ökonomischen Aspekten kalkulierbar gestalten zu können, entwickelte die Informatik grundlegende Vorgehensweisen. Eine dieser Vorgehensweisen besteht darin, „ […] eine software-technische Lösung im Sinne einer **Softwarearchitektur** zu entwickeln.“[5]

Eine Softwarearchitektur beschreibt die Struktur des Software-Systems. Dieses besteht aus mehreren Komponenten, welche zueinander Beziehungen haben. Die Praxis lehrt, dass bei der Softwareentwicklung bereits in einer frühen Phase mit dieser Strukturierung begonnen werden sollte, um arbeitsteilig, kostensparend und zielsicher arbeiten zu können. Die Intensität der Strukturierung nimmt mit der Anzahl der Komponenten, die das Software-System erhalten soll, zu. Werden die Komponenten zu verschiedenen Schichten zusammengefasst, spricht man von einem ***Schichtenmodell***[6] oder einer ***Schichtenarchitektur***. Schichten definieren Komponenten, die meist unproblematisch innerhalb einer Schicht aufeinander zugreifen können, während der Zugriff zwischen den Schichten strengeren Regeln folgt. In Schichten werden Komponenten zusammengefasst, die zueinander eine Affinität haben. Diese Affinität ist meist sachlogisch begründet.

Für den Informatikunterricht ergibt sich als Konsequenz, dass das Schichtenmodell in didaktisch reduzierter Form Gegenstand des zu vermittelnden Lernstoffes sein sollte. Die Methode der Schichtung ist ein Problemlösungsverfahren, bei dem versucht wird, das Gesamtproblem in überschaubare Einzelaspekte aufzuteilen.

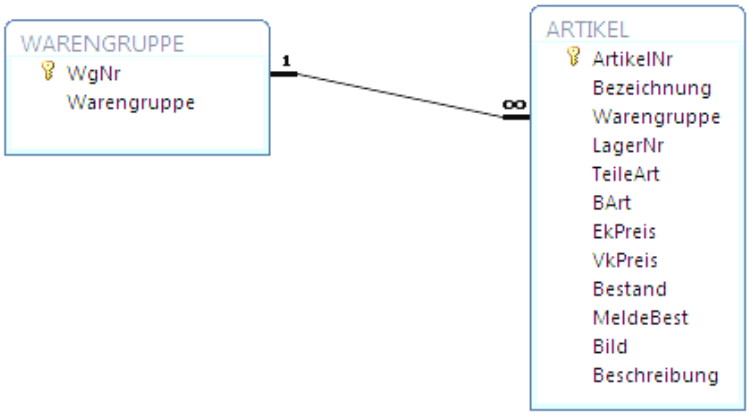
[5] Balzert, Helmut: Lehrbuch der Softwaretechnik, Bd. 1, 2. Auflage, 686ff.

[6] Ebd.: 696ff

Es reicht auch für Unterrichtszwecke nicht, wenn Software funktioniert, sie sollte auch überschaubar sein, auf einer Konzeption beruhen, wartungsfreundlich sein und nach allgemein anerkannten Regeln aufgebaut sein (d. h.: eine Architektur haben). Der Informatikunterricht sollte die Schüler nicht zu Codierern machen, die mehr oder weniger tief mit einer Programmiersprache vertraut sind, die Schüler sollten vielmehr die verschiedenen Phasen der Softwareentwicklung und deren Bedeutung kennenlernen.

Das Kapitel Anbindung einer objektorientierten Software an eine relationale Datenbank[7] wird oft fehlinterpretiert. Nicht das Funktionieren des Zugriffs ist das Entscheidende, vielmehr sollen in der Datenbank Objekte abgespeichert werden können und Entitäten, die aus der Datenbank entnommen werden, in Objekte umgewandelt werden können. Das Kernproblem ist somit die Komposition von Objekten aus Entitäten und die Dekomposition von Objekten zu Entitäten, mithin das Thema Persistenz von Objekten in einer RDB. Die Prinzipien und Gestaltungsmöglichkeiten der Objektorientierung sollten dabei nicht durch die Einschränkungen der relationalen Datenbank verfremdet und vernachlässigt werden.
Im Folgenden soll versucht werden, an Hand eines Beispiels dieser Problematik gerecht zu werden. Die eingesetzte Programmiersprache ist Java, die IDE Eclipse und das Datenbankmanagementsystem ist mySQL.

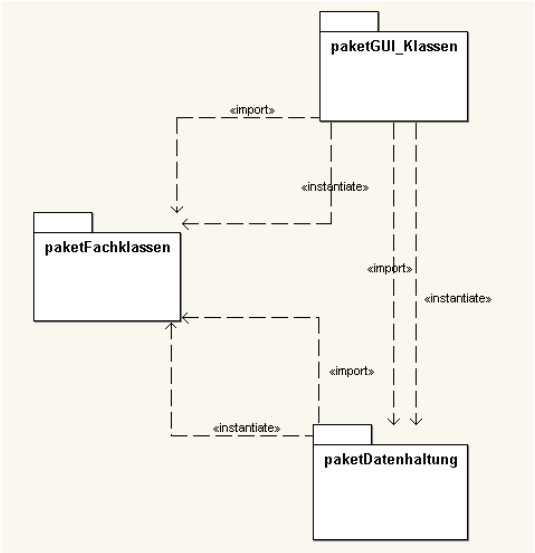
Der objektorientierte Softwareentwicklungsprozess kümmert sich normalerweise erst in einer späten Phase um die Objektpersistenz. Hier sei die Entscheidung schon im Vorfeld über die zu verwendende Datenbank definiert. Die Datenbank „haro“, wie sie im BK/WI zum Einsatz kommt, wird als Grundlage für dieses Beispiel definiert[8].



In der Datenbank gibt es u. a. die Tabellen Artikel und Warengruppe. Die obige Abbildung zeigt die Struktur dieser Tabellen:

Aufgabenstellung
Die zu entwickelnde Software soll in einem Ausgabefenster alle Artikel sowie die Artikel einer Artikelgruppe anzeigen können. Ebenfalls soll ein Artikel über ein Eingabefenster erfasst werden können. Dabei soll eine unidirektionale Assoziation implementiert sein und diese auch datenseitig in die relationale Welt transferiert werden können.

Paketabhängigkeitsdiagramm[9]
Für die Lösung werden drei Schichten entwickelt, die Datensicht, die Darstellungsschicht (GUI) und die Fachklassenschicht. Die Schichten werden in der Software über Pakete realisiert. Alle Klassen einer Schicht werden in Paketen zusammengefasst. Die gegenseitigen Abhängigkeiten der Pakete werden im package-dependency-diagram dargestellt:



[7] Je nach Schulart hat dieses Kapitel einen anderen Namen

[8] Die Autoren sind sich bewusst, dass die Datenbank aus dem Klassendiagramm entstehen müsste. Da aber die Datenbank über viele Projekte fest definiert ist, erscheint der eingeschlagene Weg gerechtfertigt.

[9] Das Projekt ist unter www.lowie.de zum Download verfügbar.

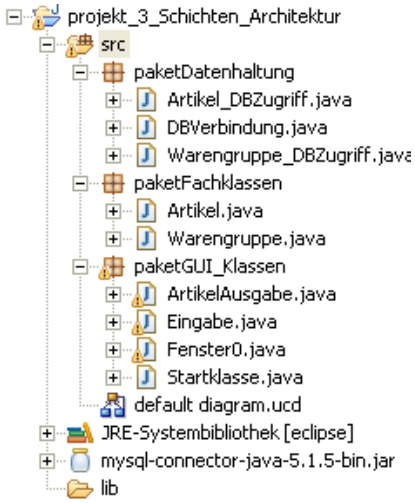
Man erkennt, dass hier ein Schichtenmodell mit strikter Ordnung vorliegt, da die oberste Schicht der GUI-Klassen auch mit der untersten Schicht der Datenhaltung kommunizieren kann. Allerdings kann sie das nur, wenn - im zeitlichen Ablauf gesehen - die Datenhaltungsschicht vorher mit der Fachklasse korrespondiert.

Im Ablauf gesehen geschieht Folgendes:

- *<u>Einfügen eines Artikel:</u> In einem Fenster werden die Attribute eines Objektes erfasst. Dazu benötigt die GUI ein Objekt der Klasse Artikel (instantiate) und den Zugriff auf das Fachklassenpaket (import). Aus dem Artikelobjekt wird nun in der Datenschicht eine Entität erzeugt (paketDatenhaltung). Dazu benötigt die GUI ein Objekt aus der Datenhaltung, welches für die Umwandlung eines Objektes in eine Entität und dessen Speicherung zuständig ist. Diese Aufgabe kann die Fachklasse nicht übernehmen, da sie sonst von der Datenhaltung abhängig wäre.*
- *<u>Anzeigen von Artikeln:</u> Die GUI fordert die Datenhaltungsschicht dazu auf, eine Abfrage auszuführen. Diese Abfrage enthält Tupel, die in Objekte umzuwandeln sind. Somit benötigt die GUI-Schicht für die Abfrage die Datenschicht. Die Datenschicht muss das Ergebnis der Abfrage in Objekte transformieren. Dazu benötigt die Datenschicht die Fachklassen.*

Bei dieser Art der Schichtung sind die Fachklassen völlig unabhängig von der Darstellung und Speicherung. Sie sind nur zuständig für die fachliche Richtigkeit der Applikation. Darstellungs- und Datenschicht haben fest definierte Zuständigkeiten. Die Datenschicht ist von der Fachklassenschicht nur insoweit

abhängig, als die Datensicht Informationen zur Umwandlung der Entitäten in Objekte benötigt. Die Datenbank selbst ist von der Applikation unabhängig. Die hier vorgenommene Schichtung auf Paketebene kommt auch in der Struktur des Projektes zum Ausdruck:

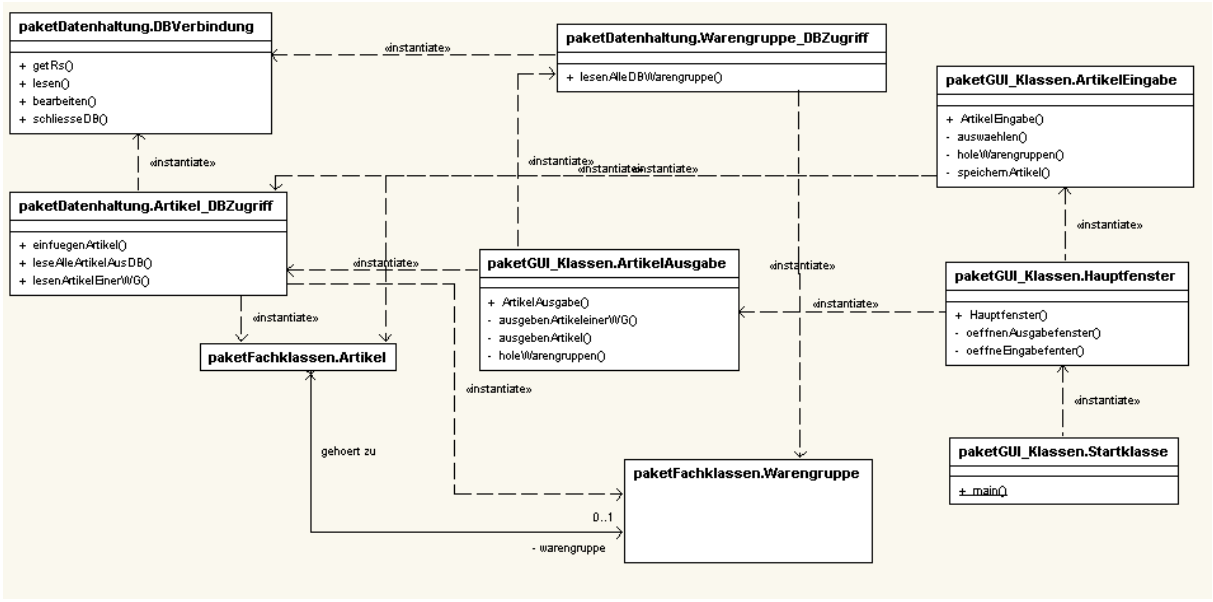


Anmerkung: Die vorgestellte Softwarearchitektur trägt der grundlegenden Idee einer Drei-Schichten-Architektur Rechnung. Allerdings ist die GUI-Schicht hier für die Darstellung der Daten sowie für die Kommunikation mit den anderen Schichten zuständig. Dies ließe sich durch eine sogenannte „indirekte Kommunikation“ mit Hilfe eines Beobachter-Musters im Rahmen der Einführung einer Mehr-Schichten-Architektur dadurch umgehen, dass separate Zugriffschichten definiert werden und die GUI-Schicht somit ausschließlich für die Darstellung der Daten zuständig wäre. Die programmtechnische Realisierung der hierzu notwendigen Kommunikation zwischen den Objekten der verschiedenen Schichten würde den Rahmen des schulischen Informatikunterrichts sprengen.[10]

Das Klassendiagramm

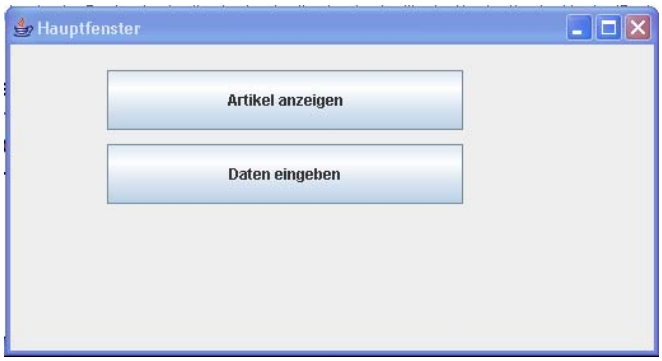
Im Folgenden ist das Klassendiagramm aufgeführt. Abhängigkeiten, die vom Import einer Klasse herrühren, sind aus Gründen der Übersichtlichkeit ausgeblendet. Ebenso werden die Set- und Get-Methoden nicht angezeigt.

[10] Der interessierte Leser sei für einen ersten Einstieg verwiesen auf: Balzert, Heide: UML 2 in 5 Tagen. Bochum 2006, 143ff.



Man kann hier die Abhängigkeiten der Klassen erkennen. Zusätzlich zum Paket-Abhängigkeits-Diagramm sind hier die Abhängigkeiten innerhalb der Pakete auf Klassenebene sichtbar. Die unidirektionale Assoziation zwischen Warengruppe und Artikel ist ebenfalls dargestellt.

Beispiel für einen Programmdialog
Nach dem Start des Programmes ist folgender Eingabeschirm zu sehen:



Bei Druck auf die Befehlsschaltfläche „Artikel anzeigen“ öffnet sich das Fenster:

Betätigt man die Schaltfläche „Auswahl der Warengruppe“, wählt im Kombinationsfeld die Warengruppe aus und drückt die Schaltfläche „Ausgabe der Artikel einer Gruppe“, dann erscheint z. B. folgendes Fenster:

Im Ablauf geschieht Folgendes:
Nachdem der Button “Auswahl der Warengruppe“ betätigt wurde, wird das Kombinationsfeld angezeigt, das seinen Inhalt aus der Datenbank entnimmt. Die Abfrage lautet: SELECT * FROM Warengruppe ORDER BY 2;

Das Ergebnis (ResultSet) wird in der Methode lesenalleDBWarengruppe() aus der Klasse Warengruppe_DBZugriff Satz für Satz jeweils in ein Objekt umgewandelt und in den Vector warengruppeSet geschrieben. Dieser Vector wird der GUI übergeben. (Klasse ArtikelAusgabe, Methode holeWarengruppen()). Die GUI sieht als Ergebnis der Datenbankabfrage nur Objekte.

Das ausgewählte Item des Kombinationsfeldes (im Beispiel Saegen) wird von der GUI in der Methode auswaehlen() ermittelt und in einen auf Klassenebene sichtbaren String (bez) geschrieben. Die Methode ausgebenArtikeleinerWG() wird durch die Schaltfläche „Ausgabe der Artikel einer Gruppe“ aufgerufen. Diese Methode steht im Dialog mit der Klasse Artikel_DBZugriff und übergibt als Parameter die Bezeichnung (bez) der Warengruppe. Es entsteht in der Methode lesenArtikelEinerWG(String wgbezeichnung) der SQL-String, der die WHERE-Klausel entsprechend der Auswahl zusammensetzt. Das Ergebnis der Abfrage wird wiederum Satz für Satz jeweils in ein Objekt umgewandelt und in den Vector „ArtikelSet“ geschrieben. Dieser wird der GUI übergeben.

Man beachte hierbei insbesondere, dass die Assoziation, die in der Datenbank durch Fremdschlüssel realisiert wird, in der objektorientierten Sicht durch eine Referenz auf ein Objekt der Klasse Warengruppe erzeugt wird. Nur in der Datenschicht kommt das Paradigma der relationalen Datenbank zum Tragen. Da diese Schicht auch die Umwandlung von Entitäten in Objekte und beim Schreiben in die Datenbank von Objekten in Entitäten vornimmt, bleibt die eigentliche Anwendung (Fachklassen und GUI-Klassen) objektorientiert.

Das Schreiben in die Datenbank wurde analog programmiert und kann anhand des Quellcodes nachvollzogen werden.

Ein Vorteil der Schichtung ist die Trennung der Zuständigkeiten. Im Beispiel sind der Datenzugriff, die GUI und die Fachklassen gekapselt. Die Architektur erfüllt somit auch das Prinzip der Kapselung: Auf die Daten kann nur über Objekte, die den Anforderungen der Fachklassen entsprechen, zugegriffen werden. Sämtliche datenrelevante Operationen außerhalb der Datenschicht sind in Methoden von Klassen beschrieben. Alle Attribute sind außerhalb der Klasse, in der sie definiert werden, nur mit zugelassenen Operationen manipulierbar.

Die relationale Datenbank behält bei der Datenmanipulation alle ihre Funktionen. In ihr sind die Zugriffsrechte definiert, sie hat die Prüfung auf referenzielle Integrität zu leisten und kann auch jenseits der Applikation Verwendung finden. Ebenfalls kann das Fachklassenkonzept auch auf andere Speicherparadigmen angewandt werden.

Fazit zur schulischen Umsetzung

Das zu lösende informatische Problem ist im Kern trivial. In der Datenbank existieren zwei Tabellen, die unidirektional verbunden sind (1 : n). Da aber die Paradigmen der Objektorientierung und der Relationen Datenbanken zusammentreffen, steigt die Komplexität sprunghaft.

Im Unterricht versucht man dem Problem durch didaktische Reduktion gerecht zu werden. Dabei wird meist auf eine Datenbank mit nur einer Tabelle zugegriffen. Die Klasse entspricht dabei exakt dem Entitätstyp. Unseres Erachtens ist diese didaktische Reduktion nicht zu empfehlen, weil sie fachwissenschaftliche Inhalte stark verzerrt und kaum haltbar wiedergibt. Man wird auf diese Weise weder der Objektorientierung noch den Relationalen Datenbanken gerecht. Einerseits kann beim Zugriff auf nur eine Tabelle nicht von einem, Zugriff auf eine Datenbank sprechen, wie man andererseits die Möglichkeiten der Objektorientierung nur sehr eingeschränkt nutzen kann, wenn man zwischen Objekten und Entitäten nicht unterscheidet.

Eine andere Version der didaktischen Reduktion besteht oft darin, dass kein Schichtenmodell eingebaut wird. Die grafische Oberfläche greift hierbei über SQL-Statements direkt auf die Datenbank zu. Es gibt dann zu keinem Zeitpunkt Objekte, die abgespeichert werden müssen. Die objektorientierte Software reduziert sich in diesem Falle auf die Programmierung von grafischen Oberflächen.

Ein Beispiel, das aus nur einer Tabelle und einer dazu korrespondierenden gleichen Klasse besteht, zeigt keinen „Zugriff auf eine Datenbank“, sondern zeigt nur, dass man eine Verbindung aus einem objektorientierten Pro-

gramm zu einer Datenbank herstellen kann. Diese Erkenntnis rechtfertigt aber nicht den großen Aufwand. Ebenfalls liefert diese Erkenntnis keinen Beitrag zur Studierfähigkeit. Sobald aber beide Paradigmen in einer Anwendung ausgelebt werden sollen, verlässt man den propädeutischen Bereich und begibt sich in Sphären, die in einem Informatikstudium den höheren Semestern vorbehalten bleiben.

Die Informatik bietet bis heute keine zufriedenstellende und einfache Lösung für die Abspeicherung von Objekten über relationale Datenbanken. Stattdessen könnten Objekte über Objektorientierte Datenbanken (einfacher Weg) persistent gemacht werden oder über eine Serialisierung auf einen Datenträger geschrieben werden. Die Relationale und Objektorientierte Welt könnten in der Schule jeweils isoliert thematisiert werden. Beide Paradigmen tragen zur Studierfähigkeit bei und können propädeutisch behandelt werden.

Dabei darf man aber eine Lösung über Objektorientierte Datenbanken nicht ohne die ihr innewohnende Problematik anbieten. Ebenso ist zu zeigen, dass Objektorientierte Datenbanken eine andere Zielsetzung haben als Relationale Datenbanken. Vergleiche über Zugriffszeiten und ähnlichen Tests sind unzulässig, weil die Modellierung und die Problemstellung einen entscheidenden Einfluss auf Benchmark-Tests haben, die somit je nach Interessenlage beeinflussbar sind. Die Zusammenführung des objektorientierten und relationalen Paradigmas in einer Applikation führt zum Impedance Mismatch und ist ein großes ungelöstes informatisches Problem, das den Rahmen der Propädeutik eindeutig sprengt.

Wikipedia zum Stichwort Impedance Mismatch: „Als **"Object-relational Impedance Mismatch"** - oft auch nur **Impedance Mismatch** - (z.dt. etwa *objekt-relationale Unverträglichkeit*) bezeichnet man ein Problem der Informatik in der Anwendungsentwicklung, das auftritt, wenn Objekte aus einer objektorientierten Programmiersprache in einer relationalen Datenbank gespeichert werden."