Technik des Electronic Commerce im Internet mittels Java/JDBC-Anbindung an die relationale Datenbank Oracle

Vorwort

Datenbanken kommen beim Einstieg von Unternehmen in das Inter- oder Intranet eine besondere Bedeutung zu, denn dort sind für den Mitarbeiter bzw. für den Kunden wichtige **Informationen** vorhanden: Kundenverzeichnisse, Warenwirtschaftssysteme, Rohdaten für Marketinganalysen, Produktkataloge etc. .

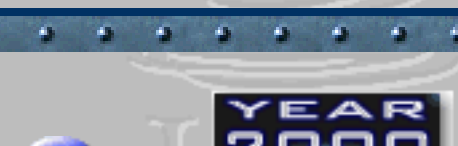
Die einheitliche Einbettung, d.h. Veröffentlichung, dieser Daten war bislang aufgrund der Heterogenität der Betriebssysteme, Datenbanksysteme und der Anwendungen, die mit ihnen arbeiten, oftmals zu aufwendig.

Ziel zur Effizienzsteigerung und Wirtschaftlichkeit sollte es jedoch sein, daß alle Mitarbeiter jederzeit Zugriff auf alle Informationen haben, die sie für ihre Aufgaben benötigen, ohne ihren Arbeitsplatz verlassen zu müssen.

Dank des World Wide Webs (WWW) und Java ist eine einheitliche Wahl des Betriebssystems heute nicht mehr zwingend, da WWW und Java Anwendungen ermöglichen, die auf beliebigen Rechnern lauffähig sind. Somit lassen sich **Informationen** im Inter- oder Intranet überall **darstellen**.

Auf der anderen Seite des **Informationskanal**s stehen verschiedenste Arten von Datenbankmanagementsystemen (**DBMS**), die die benötigten **Informationen bereithalten**. Dies sind vor allem Adabas, DB2, Informix, MS SQL-Server, Oracle, Sybase, sowie einige kleinere Systeme.

Um die **Lücke** von der Informationsbereithaltung bis hin zur Informationsdarstellung zu schließen, hat Sun Microsystems einen Standard definiert, der es Java-Programmen ermöglicht, über das Inter- bzw. Intranet auf relationale DBMS zugreifen zu können. Dieser Standard wird als **JDBC** (Java Database Connectivity) bezeichet. Hierdurch werden dem Programmierer Klassen, Interfaces und Methoden zur Verfügung gestellt, die ihm den Informationszugriff über das Netz hinweg ermöglichen sollen.



1. Einleitung

(Electronic Commerce im Internet)

Das Internet gewinnt zunehmend an Einfluß auf das alltägliche Leben. Die Nutzung des neuen Mediums verändert sich und gewinnt eine breitere demographische Basis, die als Grundlage des Electronic Commerce unerläßlich ist. Das "Internet Publikum" ist oft weniger an Unterhaltung als am praktischen Nutzen interessiert. Dieser Umstand wird beispielhaft durch eine Statistik von dem amerikanischen Internetnetprovider "America Online" untermauert, die besagt, daß sich von 1995 – 1998 die Zahl der Chat-Room-Besucher von 40% auf 20% reduzierte, während im gleichen Zeitraum die Suche nach Preis- und Produktinformationen von beinahe Null auf 23% anstieg, gemessen an der insgesamt online verbrachten Zeit. Dieser Umstand zeigt, daß sich das Internet tendenziell zu einem Informations- und Vertriebsmedium entwickelt. Das Marktvolumen steigt nach einer realistischen Schätzung von Hagen/Zagler [1] innerhalb von Deutschland von 500 Mio. DM im Jahr 1996 auf 5 Mrd. DM bis zum Jahr 2001.

Ein wesentlicher Grund für die Zunahme von Online-Geschäften sind die damit verbundenen Kostenvorteile, insbesondere hinsichtlich der Transaktionskosten. Aus Unternehmersicht können die Kosten der Abwicklung einer Transaktion erheblich reduziert werden, insbesondere bei Finanzdienstleistungen.

Dem Kunden bieten beispielsweise Online-Transaktionen neben möglicherweise günstigeren Konditionen oft ein hohes Maß an Bequemlichkeit und Zeitersparnis, die häufig entscheidungsrelevant sind.

Welche Geschäftsvorteile für deutsche Unternehmen durch den Einsatz von Electronic Commerce zu erwarten sind, zeigt folgendes Beispiel:

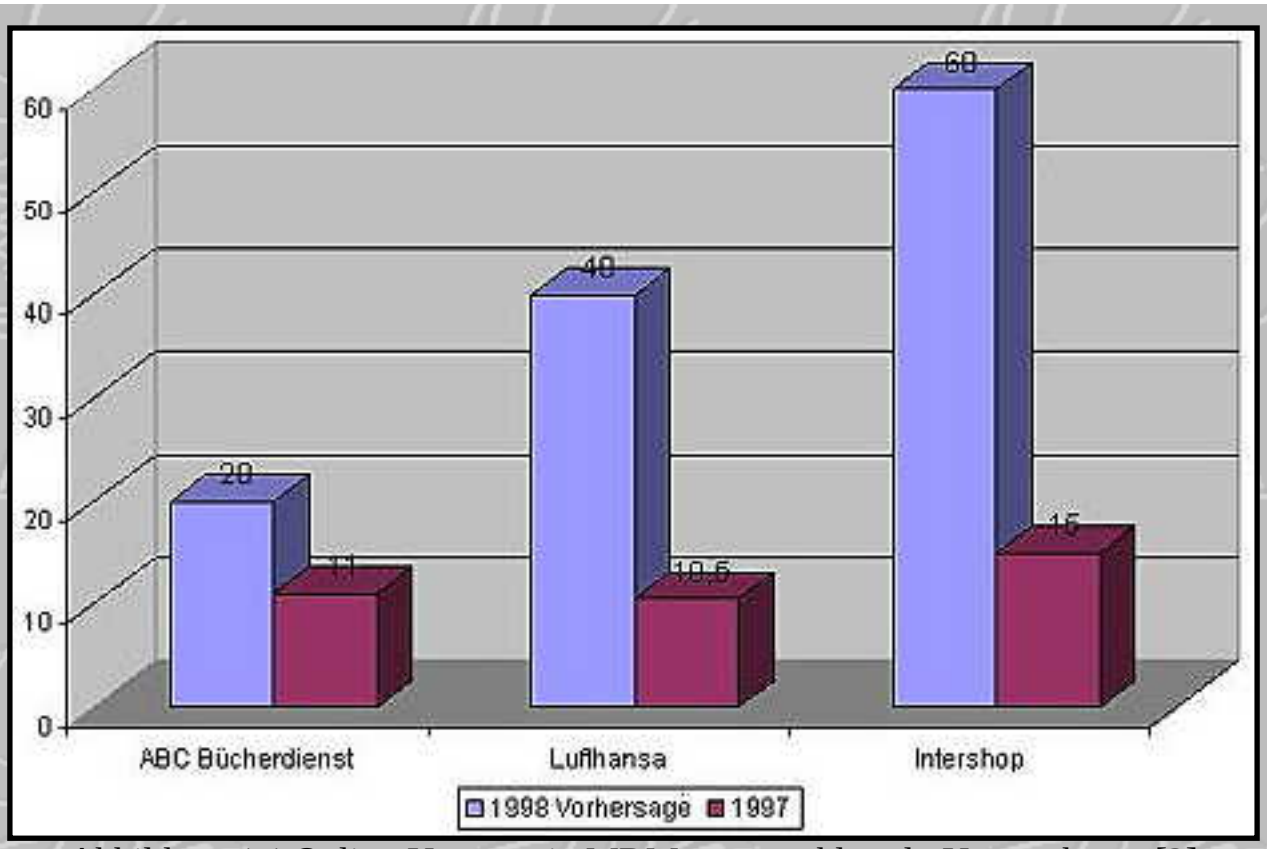


Abbildung 1.1 Online-Umsätze in MDM ausgewählter dt. Unternehmen [2]

Alle drei Unternehmen haben demnach beachtliche Umsatzzuwächse zu verzeichnen. ABC Bücherdienst (inzwischen von Amazon übernommen) verkauft seit 1995 unter der Bezeichnung Telebuch online Bücher. Die Deutsche Lufthansa vertreibt Flugtickes online und bedient sich dabei elektronischer Auktionen. Die Firma Intershop bezeichnet sich selbst als der "weltweit erfolgreichste Anbieter von Standardsoftware für das eBusiness".

Unter "Electronic Commerce" wird im deutschen der "Elektronische Handel" verstanden. Diese Bezeichnung ist implizit in dem mehr geläufigen Ausdruck "elektronischer Markt" vorhanden. Allgemein ist ein Markt ein Ort, an dem Angebot und Nachfrage aufeinandertreffen und Transaktionen möglich sind. In diesem Sinne ist das Internet ein weltumspannender elektronischer Marktplatz für Güter und Dienstleistungen. Das Datennetz selbst dient dabei als Trägermedium, auf dem aufbauend verschiedene Dienste existieren, welche die Funktionen eines Marktes unterstützen. Transaktionen, d.h. alle Prozesse im Dialog von Anbietern mit Kunden, werden virtualisiert. Dies umfaßt die Information/Präsentation, sowie die Vereinbarung von Leistungen und die Abwicklung von Bestellungen inkl. Zahlungsverkehr bis hin zu gegebenenfalls (virtueller) Lieferung von Waren (z.B. Software) auf elektronischem Weg. [3]

Die hier vorliegende Arbeit beschäftigt sich hauptsächlich mit der Technik, die für den Prozeß der Information/Präsentation notwendig ist. Dies meint einerseits den vom Nutzer unmittelbar im Internet-Browser wahrgenommenen Teil der Anwendung (Front-End), der mittels Java-Applets interagiert. Andererseits wird auch der im Hintergrund aktive Prozeß (Back-End) betrachtet, der für die Datenbankanbindung und Kommunikation mit dieser zuständig ist.

Beispielhaft wird eine Datenbank mit hochschulspezifischen Inhalten dem Nutzer via Internet zur Verfügung gestellt. Selbiges, später ausführlich beschriebenes, System kann in leicht abgewandelter

Form als elektronischer Warenkorb oder elektronischer Katalog Verwendung finden.

Solche elektronischen Kataloge bieten weit mehr als ihre auf Papier gedruckten Vorläufer. Über sie bezieht der Kunde aktuelle Produktinformationen, führt Bestellungen und Zahlungen aus, nimmt den Kundendienst in Anspruch und gibt dem Unternehmen Rückmeldungen darüber, ob er mit der ihm angebotenen Leistung zufrieden ist. Aus der Sicht des Unternehmens stellen sie einen neuen Kanal für Werbung, Marketing, Distribution und Support dar.

[1] Hagen, Jan; Zagler, Hans; Sicherer Zahlungsverkehr im Electronic Commerce; Die Bank; April 1998; S.217-219
[2] Burkhardt, Th.; Lohmann, K.; Banking und Electronic Commerce im Internet; Berlin 1998; Berlin Verlag; A.Spitz; S.30
[3] Assfalg, R.; Goebels, U.; Welter, H.; Internet Datenbanken; Bonn; 1998 Addison-Wesley-Longman; S.326

2. Die Programmiersprache Java

Sun Microsystems beschreibt die von ihnen entwickelte Programmiersprache Java folgendermaßen: "Java soll eine einfache, objektorientierte, verteilte, interpretierte, robuste, sichere, architekturneutrale, portable, performante, nebenläufige, dynamische Programmiersprache sein."

Tatsächlich hat zuvor keine Programmiersprache in so kurzer Zeit so viele Entwickler begeistern können und eine derart weite Verbreitung gefunden wie das von Sun Microsystems entwickelte Java. Einige Gründe, die eng mit der obigen Beschreibung zusammenhängen, werden im folgenden kurz vorgestellt:

Einfachheit

Java ist eine vollständig einfach gehaltene Programmiersprache. Es werden keine "Header Files" benötigt und da Java objektorientiert ist, wurde auf Konstrukte wie "struct" und "union" verzichtet. Zudem gibt kein Operator-overloading und keine explizite multiple Vererbung. Vor allem jedoch gibt es keine Pointer, wodurch die Sprache von zahlreichen Fehlermöglichkeiten befreit wurde.

Das Speichermanagement in Java erfolgt automatisch. Ein "Garbage-Collector", der als niedrigpriorisierter Hintergrundprozeß läuft, sucht in regelmäßigen Abständen nach nicht mehr referenzierten Objekten und gibt den durch sie belegten Speicher an das Laufzeitsystem zurück.

Die Syntax und die objektorientierte Struktur von Java wurde sehr stark an das Vorbild von C++ angelehnt, was eine leichte Erlernbarkeit garantiert.

Objektorientierung

Java verfügt über ein in objektorientierten Sprachen übliches Klassen-Konzept, wonach eine Klasse eine Sammlung von Daten und Methoden ist, aus der ein Objekt erzeugt werden kann. Klassen sind hierarchisch aufgebaut. Subklassen erben damit die Eigenschaften ihrer Vaterklassen.

Alle Methodenaufrufe in Java sind dynamisch. Methoden können überladen werden, Operatoren jedoch nicht. Java erlaubt zwar Einfach-, aber keine Mehrfachvererbung. Mit Hilfe von Interfaces (das sind abstrakte Klassendefinitionen, die nur aus Methoden bestehen) ist dennoch eine restriktive Form der Mehrfachvererbung möglich, die einen Kompromiß zwischen beiden Alternativen darstellt. Java erlaubt zudem die Definition abstrakter Basisklassen, die neben konkreten auch abstrakte Methoden enthalten, die in abgeleiteten Klassen nachträglich definiert werden müssen.

Klassen

Klassen "verkapseln" ihre Daten und Methoden, so daß sie nach außen nicht sichtbar sind. Sie kommunizieren mit ihrer Außenwelt nur über definierte Interfaces mit definierten Methoden. Vordefinierte Java-Klassen sind in "Packages" organisiert. Besonders erwähnenswert sind hierbei vor allem das graphische Benutzerinterface (java.awt), sowie das Package zur Verwendung von Datenbank-, d.h. JDBC-Funktionen (java.sql).

Netzwerkorientierung

Java ist eine verteilte Sprache (distributed). Die URL-Klasse z.B. erlaubt den Zugriff auf das weltweite Webseiten-System aus jedem Programm heraus und die Bibliothek RMI (Remote Method Invocation) ermöglicht den netzweiten Zugriff auf verteilte Objekte.

Applets

Mit Hilfe von Java ist es möglich, Programme zu entwickeln, die über das World Wide Web verbreitet und mit Hilfe eines Browsers wie Netscape Navigator, Sun HotJava oder Microsoft Internet Explorer gestartet werden können. Dazu wurde die Sprache HTML um das APPLET-Tag erweitert und bietet so die Möglichkeit, kompilierten Java-Code in normale Web-Seiten einzubinden.

Im Gegensatz zu den eingeschränkten Möglichkeiten, die Script-Sprachen wie JavaScript bieten, sind Applets vollständige Java-Programme, die alle Merkmale der Sprache nutzen können. Insbesondere besitzt ein Applet alle Eigenschaften eines grafischen Ausgabefensters und kann zur Anzeige von Text, Grafik und Dialogelementen verwendet werden.

Plattformunabhängigkeit

Die Übersetzung eines Java-Quellprogramms erzeugt keinen Maschinencode für eine bestimmte Zielplattform, sondern einen Bytecode. Dieser kann von einer virtuellen Maschine (VM) aufgenommen und in Kommandos, die das jeweilige Betriebssystem verarbeiten kann, konvertiert werden. Da die genaue Beschreibung der virtuellen Maschine Bestandteil der Java-Spezifikation ist, existieren Java-VMs bereits auf einer große Anzahl unterschiedlicher Plattformen. Diese Tatsache verleiht Java die gewünschte Plattformunabhängigkeit.

Einmal geschrieben sind Java-Programme theoretisch auf jeder Hardwarearchitektur lauffähig. Jeglicher Portierungsaufwand entfällt, was einen enormen Zeit- und Kostenvorteil mit sich bringt. Dies gilt in besonderem Maße auch für die Softwarepflege, die sich nur noch auf eine einzige zu wartende Version beschränkt.

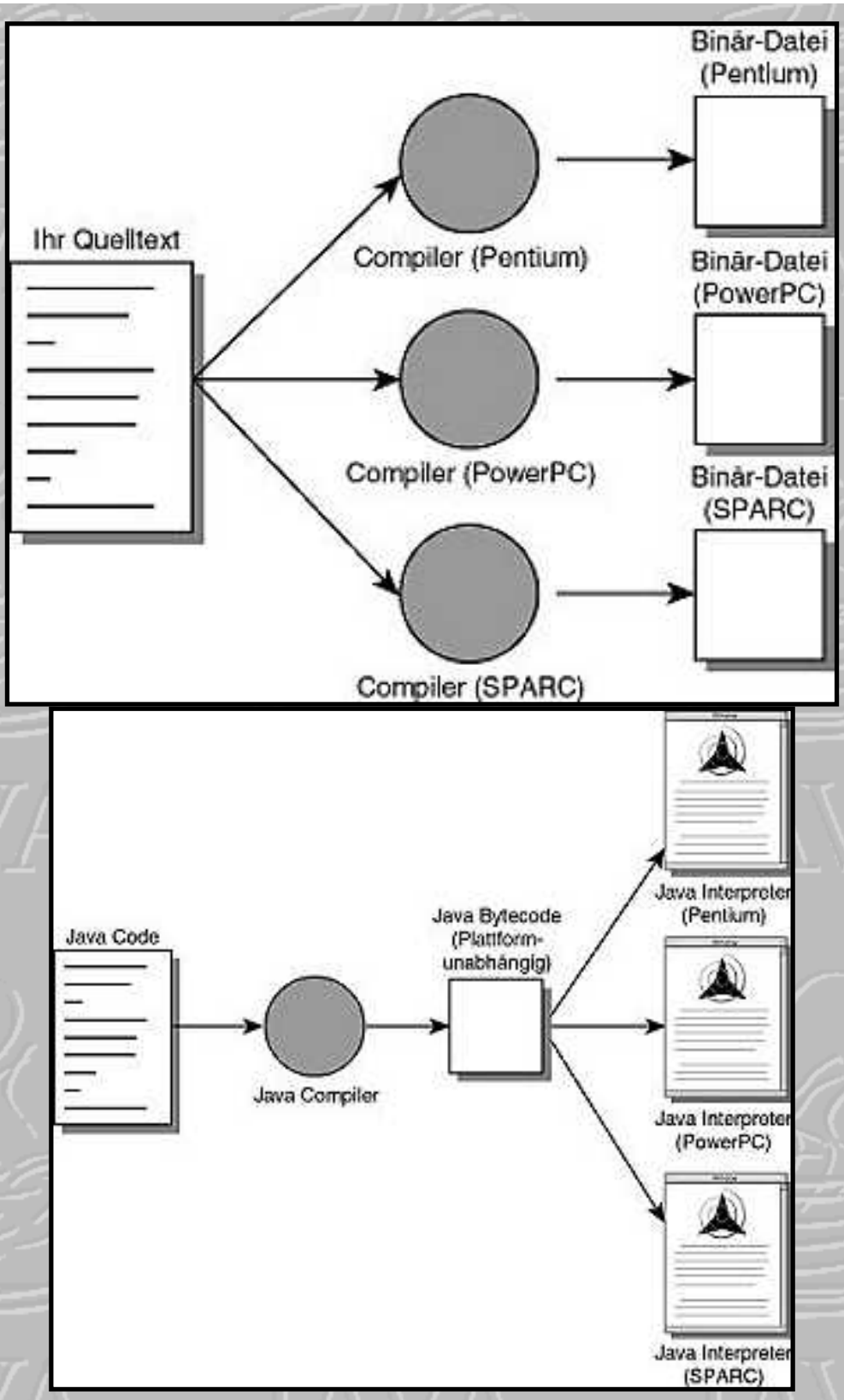


Abbildung 2.1 Vgl. zwischen herkömmlich kompilierten Programmen und Java-Multiplattform-Programmen [4]

Robustheit

Java war zunächst für elektronische Gebrauchsgeräte entwickelt worden und ist daher einfach gehalten. Die Sprache ist demnach streng typgebunden und vermeidet daher Fehler durch Vermischung von Datentypen. Implizite Deklarationen werden nicht geduldet, alles muß vor seiner Benutzung zunächst definiert werden. Der Verzicht auf Pointer eliminiert ebenfalls Fehlermöglichkeiten wie z.B. das "memory overwriting" (Speicherdoppelbelegung). Zudem gibt es in Java ein strukturiertes

"Exceptionhandling". Damit ist es möglich, Laufzeitfehler zu erkennen und in strukturierter Weise zu behandeln. Eine Methode muß jeden Laufzeitfehler, der während ihrer Abarbeitung auftreten kann, entweder abfangen oder durch eine geeignete Deklaration an den Aufrufer weitergeben. Dieser hat dann seinerseits die Pflicht, sich um den Fehler zu kümmern. Exceptions sind normale Objekte, die zugehörigen Klassen können erweitert und als Grundlage für anwendungsspezifische Exception-Handler verwendet werden.

Sicherheit

Dadurch, daß Java über keine Pointer verfügt, kann der Programmierer das "memory management" nicht manipulieren. Außerdem sagen die Deklarationen der Klassen nichts über das "memory layout" aus. Der Byte-Code-Verifikationsprozeß, den die "virtuelle Maschine" ausführt, überprüft, daß der über das Netzwerk geladene Byte-Code keine Sprachrestriktion verletzt. Jede geladene Klasse wird in einen separaten "Namespace" geführt, so daß kein Java-Applet Standardklassen mit seinen eigenen überschreiben kann.

Zudem verhindern zusätzliche Sicherheitsmechanismen, daß Java-Applets während ihrer Ausführung Schaden anrichten. So ist es einem Applet, das in einem Web-Browser läuft, beispielsweise nicht erlaubt, Dateioperationen auf dem lokalen Rechner durchzuführen oder externe Programme zu starten.

Mehrprozeßfähigkeit

Die Implementierung von Threads war eine explizite Anforderung an das Design der Sprache. Ein Thread ist ein eigenständiges Programmfragment, das "nebenläufig" zu seinesgleichen funktioniert. Unter Nebenläufigkeit wird die Fähigkeit eines Systems bezeichnet, zwei oder mehr Vorgänge gleichzeitig bzw. quasi-gleichzeitig auszuführen. Ein Thread ähnelt damit einem Prozeß, arbeitet aber auf einer feineren Ebene. Während ein Prozeß das Instrument zur Ausführung eines kompletten Programms ist, können innerhalb dieses Prozesses mehrere Threads parallel laufen. Ein wichtiger Unterschied zwischen Threads und Prozessen ist der, daß alle Threads eines Programmes sich einen gemeinsamen Adreßraum teilen, also auf dieselben Variablen zugreifen, während die Adreßräume unterschiedlicher Prozesse streng voneinander getrennt sind.

Threads sollen unter anderem die Implementierung grafischer Anwendungen erleichtern, die durch Simulationen komplexer Abläufe oft inhärent nebenläufig sind.

[4] Lemay, Laura; Cadehead, Rogers; Java 1.2 programmieren in 21 Tagen; 1998 Haar bei München; SAMS


3. Datenbankenzugriff mit JDBC

Zu den wichtigsten Anwendungsgebieten von Java zählen User-Interfaces zu Datenbanksystemen. Das Java-Programm kann dabei ein Applet, eine Applikation oder ein Servlet (siehe Kapitel 4) sein und kann am selben Rechner wie die Datenbank laufen oder auch auf einem anderen Rechner und von dort über das Internet oder ein Intranet auf die Datenbank zugreifen.

Die Konzeption einer Datenbankstruktur und das Design der Zugriffsmethode stellen dabei die eine Seite der Medaille dar, das Arbeiten und die Programmentwicklung mit einem konkreten DBMS die andere. Eine Datenbankanwendung sollte daher im doppelten Sinne portabel sein, da sie unterschiedlichen Plattformen beherrschen muß und gleichzeitig mit unterschiedlichsten DBMS konfrontiert wird.

Erleichternd wirkt der Umstand, daß sich nahezu jedes DBMS über einen SQL-Monitor (Structured Query Language) bedienen läßt. Dieser ist ein einfaches Werkzeug, das dem Anwender auf der Ebene der Abfragesprache SQL (siehe Kapitel 6) Zugriff auf Datenbanken und Tabellen gewährt. Der Entwickler erhält hiermit auf der untersten Ebene Zugang zum DBMS.

Innerhalb der Programmentwicklung muß eine Verbindung zur Datenbank aufgebaut werden, SQL-Anweisungen gesendet werden und letztlich Ergebnisse gelesen und verarbeitet werden.

Jeder Hersteller von DBMS stellt genau für diese Schnittstelle ein eigenes Verfahren zur Verfügung. Auf diese Weise versucht er den Anwender zu binden. Dieser kann nun nicht mehr einfach auf ein anderes DBMS wechseln, ohne Teile des Programmcodes neu zu schreiben.

Einen Ausweg zur Beherrschung der doppelten Portabilität schafft JDBC. Das Java Database Connectivity-API (Application Programming Interface), definiert eine strukturierte Schnittstelle zu SQL-Datenbanken. Durch die Unterstützung von SQL ermöglicht JDBC Entwicklern die Interaktion und Unterstützung einer breiten Palette von Datenbanken. Dies bedeutet, daß die spezifischen Eigenschaften der zugrundeliegenden Datenbankplattform in bezug auf JDBC im allgemeinen irrelevant werden. SQL-Datenbanken sind auf dem allgemein anerkannten SQL-Standard aufgebaute Datenbanken, der ein striktes Protokoll für den Zugriff auf und die Manipulation von Daten definiert. Der Ansatz des JDBC-API für den Zugriff auf SQL-Datenbanken ist mit bestehenden Entwicklungstechniken für Datenbanken vergleichbar, so daß Interaktion mit einer SQL-Datenbank per JDBC nicht sehr unterschiedlich zu der Interaktion über SQL-Monitore ist.

Das JDBC-API beinhaltet Klassen für allgemeine Formen von SQL-Datenbanken wie Datenbankverbindungen, SQL-Anweisungen und Resultsets. Java-Programme sind mit JDBC in der Lage, übliche SQL-Anweisungen für die Ausgabe und für die Verarbeitung von Ergebnisdaten zu

benutzen.

Das JDBC-API hängt weitgehend von einem Treibermanager ab, der mehrere Treiber, die die Verbindung zu verschiedenen Datenbanken herstellen, unterstützt. JDBC-Datenbanktreiber können entweder vollständig in Java geschrieben werden oder durch Verwendung systemeigener Methoden implementiert werden, um eine Brücke zwischen Java-Anwendungen und bestehenden Datenbankzugriffsbibliotheken zu schlagen. JDBC beinhaltet ebenso eine Brücke (Bridge) zu ODBC, der allgemeinen Schnittstelle zum Zugriff auf SQL-Datenbanken von Microsoft. Die JDBC-ODBC-Bridge ermöglicht es, JDBC-Treiber aus ODBC-Treibern heraus zu benutzen (siehe Kapitel 7).

4. Java-Programme: Applets, Applikationen und Servlets

Datenbanken, die im Internet, Intranet oder Extranet (Verbindung von Intranets) zur Verfügung stehen, werden Web-Datenbanken genannt. Die Bezeichnungsweise ist auf die technische Anbindung zurückzuführen, die in allen drei Fällen gleich ist.

Da der Datenbank-Client auf beliebigen Betriebssystemen lauffähig sein sollte, kommen für eine plattformunabhängige Benutzeroberfläche im wesentlichen nur zwei Darstellungsformen in Frage: Java und HTML (Hypertext Markup Language).

Mit Java lassen sich wesentlich attraktivere und interaktivere Anwendungen realisieren als mit reinem HTML: Java animiert WEB Seiten und ermöglicht spezialisierte Anwendungen. Es gibt erstmals die Möglichkeit Animationen, sowie komplexe Benutzerinteraktionen in Echtzeit auf WWW-Seiten (World Wide Web) darzustellen. Allerdings kann nicht jeder beliebige Browser diese ausführbaren Inhalte anzeigen, er muß javafähig sein.

Java-Applets unterscheiden sich kaum von **Java-Applikationen**. Bis auf wenige Ausnahmen werden sie mit denselben Werkzeugen und Techniken konstruiert. Vereinfacht kann gesagt werden, daß Java-Applikationen eigenständige Programme sind, die zur Ausführung den Stand-Alone-Java-Interpreter benötigen, während Java-Applets aus HTML-Seiten heraus aufgerufen werden und zur Ausführung einen Web-Browser benötigen.

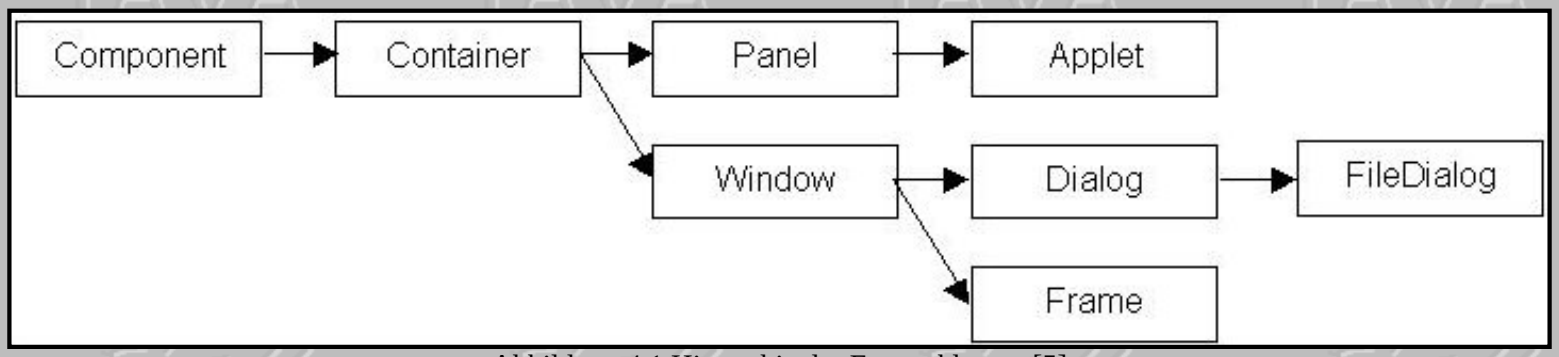


Abbildung 4.1 Hierarchie der Fensterklassen [5]

Die wichtigsten Unterschiede:

- Das Hauptprogramm eines Applets wird immer aus der Klasse "Applet" (siehe Abb. 4.1) abgeleitet. Bei einer Applikation ist es prinzipiell gleichgültig, woraus die Hauptklasse abgeleitet wird.
- Eine Applikation wird gestartet, indem vom Java-Interpreter (Java Runtime Environment, JRE) die Klassenmethode "main" aufgerufen wird. Das Starten eines Applets wird dadurch erreicht, daß der Web-Browser die Applet-Klasse instanziert und die Methoden "init" und "start" aufruft (siehe Kapitel 9, JDBC im Einsatz). Viele WWW-Browser unterstützen die neueste Java-Version jedoch noch nicht.
- Aus Sicherheitsgründen darf ein Applet in der Regel weder auf Dateien des lokalen Rechners zugreifen noch externe Programme auf diesem starten. Eine Ausnahme bilden signierte Applets. Außerdem darf ein Applet nur Netzwerkverbindungen zu dem Rechner öffnen, von dem es geladen wurde (Host). Für eine Applikation gelten diese Beschränkungen nicht.
- Ein Applet arbeitet immer grafik- und ereignisorientiert. Bei einer Applikation dagegen ist es möglich, auf die Verwendung des AWT (Abstract Windowing Toolkit) zu verzichten und alle Ein-/Ausgaben textorientiert zu gestalten.

- Applets bieten einige zusätzliche Möglichkeiten im Bereich des Benutzerschnittstellen-Designs, die so bei Applikationen nicht ohne weiteres zu finden sind. Die Ausgabe von Sound beispielsweise ist standardmäßig auf Applets beschränkt.

Applets werden mit dem <applet>-Tag in HTML-Seiten eingebunden. Ist der Client-Browser Java-kompatibel wird das Applet auf dem Client-Browser problemlos dargestellt.

Um ein Applet in eine HTML-Seiten einzubinden, müssen

- Standort (HTML-Adresse und Verzeichnis z.B. http://www.xyz.de/java/class)
- Name (Name des Applets in ByteCode, also mit Suffix "class")
- Parametereigenschaften

des Applets bekannt sein. Beispielhafter HTML-Text:

1. <applet codebase =URL code = Applet-Name.class archive = Archivname1.jar, Archivname2.zip width = X height = Y>
2. <! Optionale Parameter
3. alt = "Alternativer Text für Textbrowser"
4. name= "Symbolischer Name zur Identifikation der Applets"
5. align ="Ausrichtung: left, top, right, middle, absmiddle, baseline, bottom, absbottom"
6. vspace="Vertikaler Abstand um Applet herum (nur wenn align=left oder right)"
7. hspace="Horizontaler Abstand um Applet, sonst wie vspace"
8. Ab hier appletspezifische Parameter
9. >
10. <param name = Parametername1 value = Parameterwert1>
11. <param name = Parametername2 value = Parameterwert2>
12. <param name = Parametername3 value = Parameterwert3>
13. ..
14. ..
15. ..
16. <param name = ParameternameN value = ParameterwertN>
17. Sie benötigen einen Javakompatiblen Browser!
18. </applet> [6]

Neben den Parametern des Applet-Tags und den optionalen Parametern gibt es die Möglichkeit, Parameter an das Applet selbst zu übergeben. Dazu kann innerhalb von <APPLET> und </APPLET> das optionale Tag <PARAM> verwendet werden. Jedes Param-Tag besitzt die beiden Parameter "name" und "value", die den Namen und den Wert des zu übergebenden Parameters angeben.

Innerhalb des Applets können die Parameter mit der Methode

public String getParameter(String name);

der Klasse Applet abgefragt werden. Für jeden angegebenen Parameter liefert getParameter den zugehörigen Wert als String. Numerische Parameter müssen demnach vor der weiteren Verwendung erst konvertiert werden.

Die dritte Art von Java Programmen, **Servlets**, können in dem Fall zum Einsatz kommen, wenn z.B. ein Web-Server die Interaktion mit einer Datenbank übernehmen soll. Der Web-Server, auf dem traditionell für diese Aufgabe ein CGI-Skript (Common Gateway Interface) benutzt wird, fungiert dabei als Bindeglied zwischen WWW-Browser oder Applikation und der Datenbank.

CGI-Skripte können in jeder beliebigen Programmier- oder Script-Sprache geschrieben werden, auch als Java-Applikation. In diesem Fall besteht das CGI-Skript aus einem Shell-Script, in dem die Java Virtual Machine (JVM, Bytecodeinterpreter) aufgerufen wird, die den Bytecode der Java-Applikation interpretiert. Beispiele:

- java Classname
- java Classname Parameter
- java -Dvariable=wert Classname

Der damit verbundene Nachteil von CGI-Skripts besteht in ihrer Ineffizienz. Der WWW-Server muß für jeden CGI-Aufruf die Java Virtual Machine erneut starten. Dies funktioniert bei geringer Auslastung des Systems, jedoch kommt es bei hohen Zugriffszahlen zu inakzeptablen Antwortzeiten.

Dieser Nachteil kann vermieden werden, wenn der zur Verfügung stehende Server die Servlet-API unterstützt, d.h. die Java Virtual Machine bereits integriert enthält (z.B. der Java Web Server oder der Netscape Enterprise Server). Dann kann für die ursprüngliche Aufgabe des CGI-Skripts ein **Servlet** eingesetzt werden. Servlets besitzen gegenüber traditionellen CGI-Skripten zwei wesentliche Vorteile:

- Sie sind in Java geschrieben und somit plattformunabhängig.
- Servlets besitzen eine standardisierte, serverunabhängige Schnittstelle.

Der Name "Servlet" ist analog zu "Applet" gebildet: So wie Applets von einer Java Virtual Machine innerhalb des Web-Browsers ausgeführt werden, so werden Servlets von einer Java Virtual Machine innerhalb des Web-Servers ausgeführt.

Servlets können ähnlich wie Java-Applikationen auch auf lokale Files, Programme und Systemfunktionen zugreifen. Dafür wird ein eigenes Package javax.servlet mit entsprechenden Klassen und Methoden zur Verfügung gestellt, sowie das Java Servlet Development Kit JSDK. Dieses Entwicklungspaket ermöglicht auch Servern, die nicht in Java geschrieben sind, die Java Virtual Machine einmalig zu starten und anschließend im Hintergrund zur Verfügung zu halten.

[5] Krüger, Guido; Java 1.1 lernen; Bonn 1997; 1.Auflage; S.343
[6] Mandel, Andreas; Aufbau und Anwendungspotentiale von Java [Stand: 11.6.1999]

5. Relationale Datenbanken

Relationale Datenbanken bestehen aus Tabellen (Relationen). Die Tabellen entsprechen in etwa den Klassen in der objektorientierten Programmierung, z.B.

- Eine Personaldatenbank enthält Tabellen für Mitarbeiter, Abteilungen, Projekte.
- Eine Literaturdatenbank enthält Tabellen für Bücher, Zeitschriften, Autoren, Verlage.

Diese Tabellen können in Beziehungen zueinander stehen (⇒ "Relation"), z.B.

- Ein Mitarbeiter gehört zu einer Abteilung und eventuell zu einem oder mehreren Projekten.
- Jede Abteilung und jedes Projekt wird von einem Mitarbeiter geleitet.
- Ein Buch ist in einem Verlag erschienen und hat einen oder mehrere Autoren.

Beispielskizze für eine Tabelle "Mitarbeiter":

Angestellter	**Personalnummer**	**Abteilung**
Meier	1400-2	EDV
Müller	1577-1	Instandhaltung

Jede Zeile der Tabelle (row, Tupel, Record) enthält die Eigenschaften eines Elementes dieser Menge, entspricht also einem Objekt. Den obigen Beispielen gemäß jeweils ein bestimmter Mitarbeiter, eine Abteilung, ein Projekt, ein Buch, ein Autor, ein Verlag. Jede Zeile muß eindeutig sein, d.h. verschiedene Mitarbeiter müssen sich durch mindestens ein Datenfeld (eine Eigenschaft) unterscheiden.

Jede Spalte der Tabelle (column, field, entity) enthält die gleichen Eigenschaften der verschiedenen Objekte, entspricht also einem Datenfeld. Beispiele: Angestellter, Personalnummer und Abteilung eines Mitarbeiters, oder Titel, Umfang, Verlag und Erscheinungsjahr eines Buches.

Ein Datenfeld (z.B. die Personalnummer eines Mitarbeiters oder die ISBN eines Buches) oder eine Gruppe von Datenfeldern (z.B. Vorname, Zuname und Geburtsdatum einer Person) muß eindeutig sein, sie ist dann der Schlüssel (key) zum Zugriff auf die Zeilen (Records) in dieser Tabelle. Eventuell müssen dafür eigene Schlüsselfelder eingerichtet werden, z.B. eine eindeutige Projektnummer, falls es mehrere Projekte mit dem gleichen Namen gibt.

Die Beziehungen zwischen den Tabellen können auch durch weitere Tabellen (Relationen, n-m-Beziehungen) dargestellt werden, z.B. eine Buch-Autor-Relation, wobei sowohl ein bestimmtes Buch als auch ein bestimmter Autor eventuell in mehreren Zeilen dieser Tabelle vorkommen kann, denn ein Buch kann mehrere Autoren haben und ein Autor kann mehrere Bücher geschrieben haben.

Alle bereits genannten Eigenschaften gehören bereits mit zu den folgenden **Grundvoraussetzungen für eine relationale Datenbank:**

- Alle Information muß in Tabellen (s. obiges Beispiel) dargestellt werden. Dazu gehören auch alle Beziehungen zwischen den Daten (beispielsweise muß der Aufbau verschiedener Tabellen, die zu einer Datenbank gehören, in einer Tabelle beschrieben sein und nicht über Zeigerstrukturen, wie in hierarchischen Datenbanken). Nur so kann die gewünschte Flexibilität erreicht werden.

- Die Daten müssen mindestens mit den folgenden Operationen bearbeitet werden können:
  ⇒ **Selektion**
  Alle Zeilen der Ausgangstabelle, die eine bestimmte Bedingung erfüllen,
  werden in eine Ergebnistabelle kopiert.
  ⇒ **Verbund von Tabellen (Join)**
  Zwei Tabellen mit mindestens einem gemeinsamen Attribut (Feldnamen) werden verbunden.
  ⇒ **Projektion**
  Die Ergebnistabelle enthält alle Spalten der Ausgangstabelle, die eine bestimmte Bedingung erfüllen.

Die Erfüllung dieser einfachen Bedingungen genügt jedoch noch nicht, um ein Datenbanksystem als relational zu klassifizieren. 1985 wurden von E. F. Codd zwölf Regeln veröffentlicht, die eine relationale Datenbank im strengen Sinne definieren. Diese wurden Ende 1990 von ihm in seinem Buch über relationale Datenbanken (E. F. Codd, The Relational Model for Database Management - Version 2, Addison-Wesley 1990) auf 333 Regeln differenziert, die allgemeine Akzeptanz gefunden haben.

Das Konzept (**Design**) einer relationalen Datenstruktur ist eine komplexe Aufgabe bei der es darum geht alle relevanten Daten (Elemente, Entities) und alle Beziehungen zwischen ihnen festzustellen und dann die logische Struktur der Datenbank entsprechend festzulegen. Die logisch richtige Aufteilung der Daten in die einzelnen Tabellen (Relationen) wird als Normalisierung der Datenbank bezeichnet. Dafür existieren verschiedene Regeln und Grundsätze, ein Beispiel ist die sogenannte dritte Normalform.

Fast alle Datenbanksysteme seit Ende der 70er- oder Beginn der 80er-Jahre sind relationale Datenbanksysteme und haben damit die in den 60er- und 70er-Jahren verwendeten hierarchischen Datenbanksysteme abgelöst.

Eine zukunftweisende Weiterentwicklung der relationalen Datenbanken sind die "Objektrelationalen Datenbanken", bei denen nicht nur primitive Datentypen sondern auch komplexe Objekte als Datenfelder möglich sind, ähnlich wie in der objektorientierten Programmierung.

6. Structured Query Language (SQL)

Für die Definition von Datenbankobjekten und die Rekonstruktion von Anwendungsobjekten werden geeignete Sprachen benötigt, mit deren Hilfe die Anweisungen an ein Datenbanksystem formuliert werden können.

Zur Klassifizierung von Datenbanksprachen werden die Ansätze Datenflußorientierung, Anfragesprache und in Host-Sprachen eingebettete Sprachen benutzt.

Bezüglich des Datenflusses, den Datenbanksprachen generieren, werden zudem mengenorientierte und satzorientierte Sprachen unterschieden. Bei satzorientierten Sprachen werden Daten satzweise aus der Datenbank gelesen und in sie zurückgeschrieben. Zugriffsreihenfolge und Zugriffspfade werden von der Struktur des verarbeitenden Programms definiert. Relationale Systeme arbeiten vorzugsweise mit mengenorientierten oder deskriptiven Sprachen. D.h., der Anwender verlangt nicht nach einem konkreten Datensatz, sondern nach Datensätzen, die bestimmte, von ihm vorgegebene Kriterien, erfüllen. Das Ergebnis dieser deskriptiven Anfragen ist typischerweise eine Datensatzgruppe, also mehrere Datensätze. Wichtigster Vertreter dieser Gattung ist die Structured Query Language (SQL).

SQL ist 'die' Sprache, mit der die meisten relationalen Datenbanken erstellt, manipuliert und abgefragt werden. SQL ist eine Sprache der vierten Generation (4GL, Fourth-Generation Language). Dies bedeutet, daß sie nicht prozedural ist, d.h. der Anwender stellt eine Frage, gibt aber keinen Algorithmus zur Lösung vor. In einer 3GL wie Cobol, Pascal oder C müßte er angeben, wie die gesuchten Informationen gefunden werden können, z. B. vom Öffnen der Datei bis zum schrittweisen Durchgehen der Datensätze.

SQL ist ein ISO- und ANSI-Standard, der mehrfach spezifiziert wurde bzw. noch wird:

- SQL86
  1986 definiert

- SQL 89
  1989 definiert zwei mögliche Ebenen des Sprachumfangs:
  Level 1
  Level 2

- SQL 92 (SQL2)
  1992 definiert vier mögliche Ebenen des Sprachumfangs:
  Entry Level

Transitional
Intermediate Level
Full Level

- SQL3 (1999?)

Neben einem bestimmten SQL-Standard unterstützen Datenbanksysteme meist Teile höherer Standards sowie eigene SQL-Erweiterungen. SQL 89 Level 2 ist auch heute noch Basis des von vielen Datenbanksystemen unterstützen SQL, bei SQL 92 sind die meisten Systeme nur Entry Level-Compliant (z.B. Oracle8).

Um in der Vielfalt der SQL-Spezifikationen eine einheitliche Linie zu beschreiten, fordert Sun Microsystems von JDBC-Treibern bzw. von den dahinterliegenden Datenbanksystemen, mindestens SQL2 Entry Level zu unterstützen. Die JDBC-Spezifikation bezeichnet diesen Standard als "ANSI SQL-2 Entry Level" und bezieht sich damit auf das entsprechende Dokument des American National Standards Institute (ANSI).

In SQL werden mehrere Befehlsgruppen unterschieden:

- die zur Daten**definition** dienen. Diese Befehlsgruppe wird als Data Definition Language (DDL) bezeichnet.
- die zur **Manipulation** der Daten verwendet werden. Diese Gruppe wird Data Manipulation Language (DML) genannt.
- Eine besondere Bedeutung kommt dem sogenannten SELECT-Kommando zu, mit dem **Auswertungen** fast jeder Komplexität durchführt werden können.

Darüber hinaus gibt es Befehlsgruppen, die

- die Vergabe von Zugriffsrechten steuern (DCL, Data Control Language),
- die Struktur und Größe der Datenbank beeinflussen,
- zur Transaktionsverarbeitung notwendig sind,
- Daten aus dem UNIX-Filesystem laden bzw. in das Filesystem entladen.

Alle SQL-Befehle können bei ORACLE mit dem Programm "sqlplus.exe" ausgeführt werden.

7. Datenbankzugriff mit JDBC-Treibern

Eine Java-Anwendung kommuniziert mit einem JDBC-Treibermanager über die in der Standardklassenbibliothek enthaltene JDBC-API (Application Programming Interface). JDBC-Treiber implementieren wiederum die Schnittstellen der JDBC-API und realisieren so eine einheitliche JDBC-Treiber-API. Diese ist im allgemeinen auf ein bestimmtes DBMS und ein bestimmtes Zugriffsverfahren zugeschnitten. Vier Typen von JDBC-Treibern, die im allgemeinen unterschiedliche Verfahren verwenden, um auf ein DBMS zuzugreifen, sind im Prinzip zu unterscheiden und werden nachfolgend behandelt.

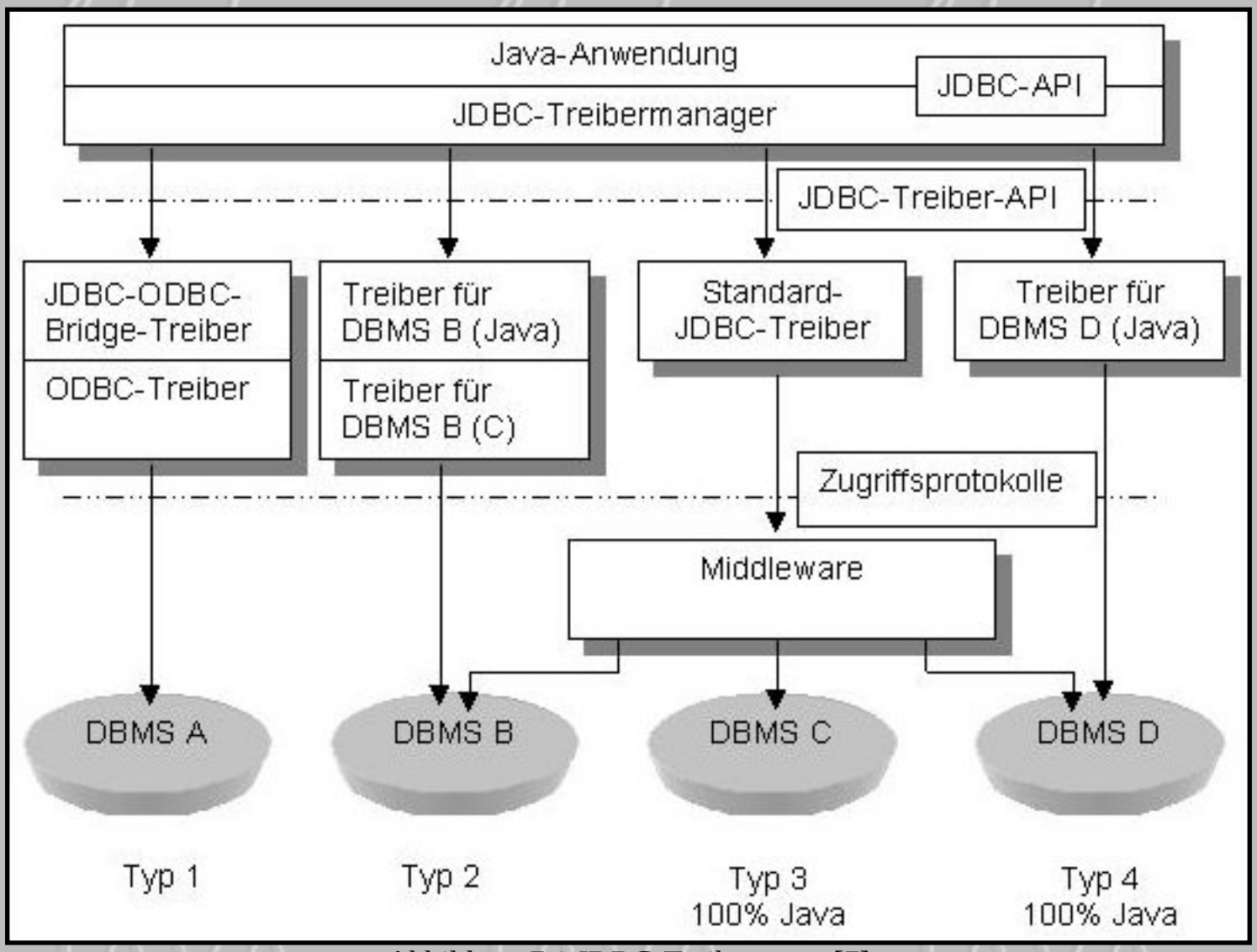


Abbildung 7.1 JDBC-Treibertypen [7]

Die Treibertypen

1. Typ 1: JDBC-ODBC-Bridge
   Da insbesondere für das Betriebssystem Windows zu vielen DBMS ODBC-Treiber (Open Database Connectivity) existieren, ist die JDBC-ODBC-Bridge zu einem festen Bestandteil des JDK (Java Development Kid) geworden. Nachteilig ist jedoch, daß der ODBC-Teil der Bridge in C geschrieben ist, da nur so die Verwendung des ODBC-Treibers möglich ist. Dies bedeutet wiederum, daß die Plattformunabhängigkeit verloren geht und zusätzliche systemabhängige Bibliotheken auf dem Client installiert sein müssen. Letzteres schließt eine Verwendung des Treibers in Applets aus.

2. Typ 2: Native-API partly-Java driver
   Dieser JDBC Treiber setzt die JDBC-API auf eine plattformspezifische API, des DBMS um, die beispielsweise in C geschrieben wurde. Die Portabilität geht hierbei, wie im Falle des Typ 1-Treibers, verloren, da auf jeder Plattform das entsprechende API vorhanden sein muß.

3. Typ 3: JDBC-Net pure Java Driver
   Der Typ 3-Treiber ist ein hundertprozentiger Java-Treiber und bietet von allen vier Treibertypen die größtmögliche Flexibilität. Er kann insbesondere dann verwendet werden, wenn JDBC in Applets eingesetzt wird, da er sich nahtlos in das Dreischichtmodell einfügt. Das bedeutet, daß er über das Netzwerk mit einer Middleware kommuniziert, die sich um den notwendigen Treiber und dessen Ansteuerung kümmert. Die Middleware selbst muß nicht aus Java-Code bestehen, sondern kann in einer beliebigen Programmiersprache geschrieben sein. Für das reibungslose Funktionieren ist lediglich das von Client und Middleware gemeinsam unterstützte Kommunikationsprotokoll erforderlich. Auf diesem Weg können Applets auch auf DBMS zugreifen, die nicht auf dem WWW-Server installiert sind (siehe Kapitel 10).

4. Typ 4: Native-protocol pure Java driver
   Der vierte Treibertyp ist ebenfalls ein reiner Java-Treiber, der somit in Applets zum Einsatz kommen kann. Er entspricht dem Zweischichtenmodell, was bedeutet, daß er ohne Umwege mit dem DBMS kommunizieren kann. – Genaugenommen kann der Typ 4-Treiber nicht direkt die Methoden des DBMS aufrufen, da diese bislang noch nicht in Java implementiert sind. Zu diesem Zweck existiert eine herstellerspezifische interne Schicht auf dem DBMS, die wie eine Middleware fungiert und nach außen unsichtbar bleibt.

Der JDBC-Treibermanager

Die JDBC-Treiber melden sich selbständig beim JDBC-Treibermanager an, der sie registriert. Durch den Aufruf von

Enumeration enum = DriverManager.getDrivers();

können die geladenen Treiber ausgelesen und anschließend angezeigt werden.

Außerdem ist der Treibermanager verantwortlich für den Verbindungsaufbau zwischen dem anzusprechenden DBMS und einem entsprechenden Treiber. Stehen mehrere Treiber zur Verfügung, die eine Verbindung zu dem angeforderten DBMS herstellen können, so wird mit jedem dieser

Treiber geprüft, ob eine Verbindung aufgebaut werden kann.

Desweiteren stellt der Treibermanager durch die Verwendung von

PrintStream myStream = DriverManager.getLogStream()

Tracinginformationen zur Verfügung.

Vom Datenbankhersteller Oracle zur Verfügung gestellte JDBC-Treiber

Oracle bietet drei verschiedenartige JDBC-Treiber zur Verwendung mit ihren Produkten Oracle 7.2 und höher an. Dies sind der 100%ige Java Thin Driver (Typ 4), der auf dem Oracle Call Interface (OCI) basierende Fat Driver (Typ 2), sowie der serverseitige KPRB-Treiber. Alle diese Treiber genügen mindestens der JDBC 1.2.2 Spezifikation und können zusammen mit dem JDK 1.1.6 genutzt werden. Außerdem besitzen sie ein einheitliches JDBC-Treiber-API, das dem Entwickler maximale Flexibilität gewähren soll.

Der Java Thin Driver ist aufgrund seiner 100%igen Java-Implementation und seiner geringen Größe von ca. 370kB der für Applets von Oracle vorgesehene Treiber (Komprimiert und um die Klasse OracleDatabaseMetaData.class vermindert, sofern keine Metadaten vom Applet abgefragt werden, läßt sich die Größe des Treibers bis auf 175KB reduzieren. Dies funktioniert jedoch nur bei derzeit aktuellen Browsern wie z.B. Netscape 4.51).

Bei dem Fat Driver handelt es sich, wie bei Typ 2-Treibern üblich, um eine JDBC-Ebene auf einem DBMS Treiber in C (hier OCI). Da OCI vorinstalliert werden muß, eignet sich der Fat Driver nicht für Applets, ist aber der ideale Treiber für Java-Servlets.

Der serverseitige Treiber KPRB ist mit Oracle8i ins Leben gerufen worden. Seit dieser Oracle-Version ist es möglich, Java-Applikationen in der Datenbank ablaufen zu lassen. Dies ist dann besonders sinnvoll, wenn sehr viele Datenbankzugriffe erfolgen sollen. Hier muß keine spezielle Verbindung geöffnet werden und Verzögerungen aufgrund des Netzes entfallen ganz, da sich Applikation und Datenbank im selben Speicher befinden.

[7] Klute, Rainer; JDBC in der Praxis; Addison-Wesley; Bonn 1998; Seite 40

8. Web Datenbanken

In diesem Kapitel werden verschiedene Architekturen bzw. Konzepte einer Java-Datenbank-Anwendung erläutert, die zum Einsatz kommen, wenn der Benutzer über das Intra- oder Internet von seinem Client-Rechnern aus auf eine Datenbank zugreifen möchte.

8.1 Direkte Datenbankanbindung

Abbildung 8.1 zeigt die Architektur eines grafischen Datenbank-Clients:

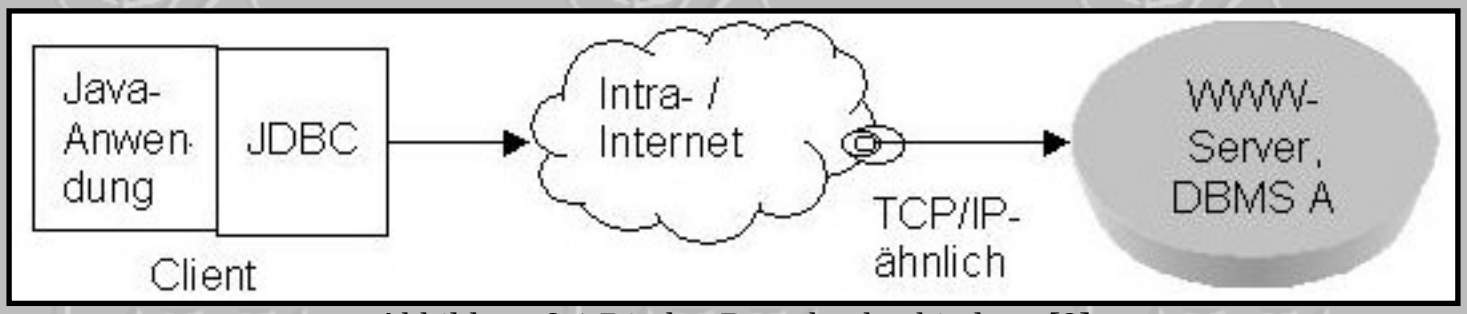


Abbildung 8.1 Direkte Datenbankanbindung [8]

Der Client kann hierbei eine eigenständige Java-Applikation oder ein in eine HTML-Seite integriertes Java-Applet sein. Er präsentiert sich dem Benutzer mit einer in Java programmierten grafischen Oberfläche und greift per JDBC über das Intra- / Internet auf das Datenbankmanagementsystem zu.

Applikationen sind i.a. fest auf dem Client installiert. Sie benötigen lediglich das Java Runtime Environment (JRE) zur Ausführung. Um auf die Datenbank zugreifen zu können, wird zusätzlich noch der für diese Datenbank spezielle JDBC-Treiber benötigt, der entweder, genauso wie für das Applet, vom WWW-Server geladen wird oder bereits fest auf dem Client installiert ist.

Ist die Java-Anwendung mit einem Applet realisiert, so benötigt der Benutzer lediglich einen javafähigen Browser, der mindestens Java 1.1 implementieren sollte, damit die JDBC-Klassen lokal zur Verfügung stehen. Den Vorzug einer grafischen Benutzeroberfläche erkauft sich der Benutzer mit dem Laden des zugehörigen Bytecodes und eventuell des JDBC-Treibers über das Netz.

Im Falle der Applet-Lösung gilt serverseitig zu beachten, daß Applets grundsätzlich nur auf Daten oder Anwendungen zugreifen dürfen (d.h. Netzverbindungen aufbauen), die auf dem Server laufen, von dem das Applet aus geladen wurde (Host). Entfernte Rechner oder Anwendungen können somit nur erreicht werden, wenn ein weiteres Programm auf dem Server seinerseits den Datenaustausch

anbietet, so wie dies beispielsweise beim Einsatz einer Middleware zur Datenbankkommunikation der Fall ist.

Falls sich eine Firewall zwischen dem Client und dem Server befindet, so ist das Konzept der direkten Datenbankanbindung nicht generell lauffähig. Welche Verfahren dann zum Einsatz kommen können erläutert das Kapitel 10.

8.2 Indirekte Datenbankanbindung

Das Konzept der indirekten Datenbankanbindung entlastet den Client erheblich (s. Abbildung 8.2):

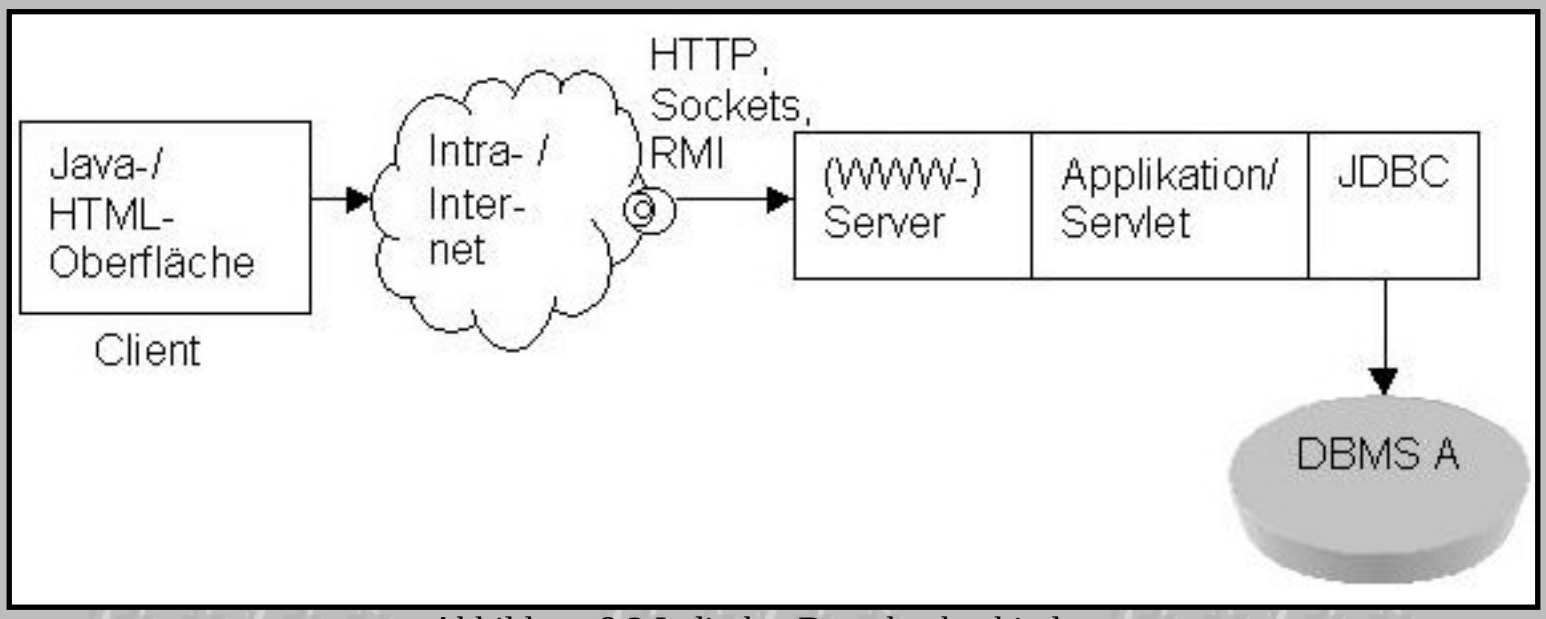


Abbildung 8.2 Indirekte Datenbankanbindung

Die Applikation auf dem Server (bzw. das Servlet) ist für die Interaktion mit der Datenbank zuständig. Sie erhält vom Client Anfragen oder Daten, führt die entsprechenden Datenbank-Abfragen oder Daten-Eingaben durch und liefert die Ergebnisse an den Client. Das Applet bzw. die HTML-Seite stellt das User-Interface auf dem Client dar. Es sendet die Abfragen oder Dateneingaben des Benutzers an den Server und gibt die von dort erhaltenen Daten oder Meldungen an den Benutzer aus. Die Kommunikation funktioniert beispielsweise über das "Hypertext Transfer Protocol" (HTTP), welches standardmäßig z.B. zum Laden von HTML-Seiten benutzt wird, über Sockets, eine Abstraktion von standardmäßigen TCP-Socket-Programmiertechniken, oder über RMI (Remote Method Invocation), eine Methode zur Realisierung verteilter Anwendungen, durch die der Client die Klassen des Anwendungsservers wie lokale Klassen nutzen kann (in eine ähnliche Richtung gehen Lösungsmöglichkeiten mit CORBA (Common Object Request Broker Architecture) und DCOM (Distributed Component Object Model)).

Folgende Kombinationen wären für eine Java-Anwendung auf dem Client denkbar:

- Java-Anwendung — HTTP/RMI/Sockets — Java-Applikation (als CGI-Skript)
- Java-Anwendung — HTTP/Sockets — Servlet

Servlets können zwar auch mittels RMI mit Java-Anwendungen kommunizieren, jedoch muß der RMI Teil im Servlet selber geschrieben werden, da Java keine Mehrfachvererbung unterstützt und Servlets standardmäßig die Klasse HTTP-Servlet erben, ein RMI-Server jedoch von UnicastRemoteObject erben müßte [9].

Die Verwendung eines indirekten Modells mit einem Java-Programm auf der Client-Seite bringt folgende Vorteile:

- Ist der Client ein Applet, so spart er Ladezeit, da die Logik zur Kommunikation mit der Datenbank in dem CGI-Skript bzw. Servlet integriert ist.
- Der Bytecode der kompletten Java-Anwendung kann nicht mehr dekompiliert werden, da er zum Teil auf dem Server verbleibt.
- Das Laden bzw. Installieren des speziellen JDBC-Treibers zur Datenbankanbindung entfällt.
- Ist der Client ein Applet, so entfällt die Einschränkung, daß die Datenbank auf dem WWW-Server (Host) installiert sein muß. Wichtig ist nur noch, daß das Servlet bzw. CGI-Skript auf dem WWW-Server (Host) abläuft.
- Umfangreiche Datenübertragungen, die bei dem Benutzer nicht in der Anzeige erscheinen, werden auf Ergebnisse reduziert, die für das Applet essentiell von Bedeutung sind. Aufwendigere Berechnungen werden bereits durch das Servlet bzw. CGI-Skript ausgeführt.

Weitere Kombinationen sind denkbar, falls auf der Client-Seite eine HTML-Oberfläche zur Anwendung kommt:

- HTML-Oberfläche — HTTP — Servlet/Java-Applikation (als CGI-Skript)

HTML-Oberflächen können grundsätzlich nur über das HTTP-Protokoll mit anderen Anwendungen kommunizieren. Eine solche Lösung ist auch dann noch voll funktionstüchtig, falls eine Firewall zwischen Server und Client arbeitet, da es sich bei der Kommunikation lediglich um ein Textprotokoll handelt, wie es bei Aufrufen von HTML-Seiten üblich ist.

Dem Nachteil einer weniger ausgefeilten und weniger interaktiven Java-Oberfläche steht der Vorteil gegenüber, daß die HTML-Variante einen größeren Benutzerkreis abdeckt, da zur Ausführung keine Java Virtual Machine notwendig ist. Außerdem benötigt eine HTML-Datei zumeist eine geringere Ladezeit und die Darstellung durch den Browser ist zumeist schneller und sicherer als bei Applets.

[8] Klute, Rainer; JDBC in der Praxis; Addison-Wesley; Bonn 1998; Seite 155
[9] Unglauben, Frank; FH-Aachen; Med. Informatik; persönliche Nachricht vom 14.6.1999

9. JDBC im Einsatz

Der Einsatz der JDBC-Klassenbibliothek soll beispielhaft unter Einsatz des Konzeptes der "Direkten Datenbankanbindung" demonstriert werden:

Ein Java-Applet kommuniziert mittels JDBC und dem Java-Thin-Treiber von Oracle direkt mit einer Oracle8-Datenbank auf einem WWW-Server und liest dabei eine Tabelle "Test" aus.

Zur Darstellung ist lediglich ein javafähiger Browser erforderlich.

In dem durch eine eigene URL-Adresse zugänglichen Verzeichnis müssen für dieses Beispiel folgende Dateien und Verzeichnisse zu finden sein:

- index.html (HTML-Seite, die durch die URL angesprochen wird)
- mini.class (das in index.html aufgerufene Java-Applet, erzeugt aus mini.java)
- mini$table.class (Subklasse der Klasse "mini", ebenfalls erzeugt aus mini.java)
- classes111.zip (der von Oracle zur Verfügung gestellte Java-Thin-Treibers)

Damit der Zugriff hierauf erfolgen kann, müssen diese Dateien und Verzeichnisse für den Zugriff freigegeben werden. Dies geschieht auf einem UNIX-Server mit dem "chmod"-Kommando:

chmod 755 *.* (Freigabe der Dateien im aktuellen Verzeichnis)

Der hier vorgestellte Datenbank-Client besitzt folgenden Grundaufbau, um seine Aufgaben mit Hilfe von JDBC auszuführen:

1. Laden des Java-Thin-Treibers
2. Öffnen der Datenbank
3. Senden von SQL-Anweisungen zum Auslesen der Datenbank
4. Auslesen, bearbeiten und darstellen der Treffermenge

Bevor die Datenbank unter Einsatz des Java-Applets ausgelesen werden kann, müssen zunächst die entsprechenden Daten in der Datenbank gespeichert werden. Dazu wurden folgende Anweisungen in den Oracle-Editor "sqlplus.exe" eingegeben:

```
drop table Test;
create table Test(
Nummer number(5) primary key,
Name varchar2(20),
Jahrgang number(2)
);
insert into test values (1, 'Martin', 67);
insert into test values (2, 'Justine', 73);
commit;
```

Hierdurch wird eine Tabelle mit dem Namen "Test" angelegt (create), die die Spalten "Nummer", "Name" und "Jahrgang" spezifiziert. Darin werden zwei Eintragungen (insert) vorgenommen und mit dem "commit"-Kommando in der Datenbank abgelegt.

Da Applets in HTML-Seiten eingebettet sind, wird die Datei "index.html" Kapitel 4 gemäß konzipiert:

```
<html>
<head>
<title>MiniConnection</title>
</head>
```

```
<body>
<h1>Mini Connection JDBC Oracle Test</h1>
<applet code="mini.class" archive="classes111.zip" width=500 height=150>
<param name="Treiber" value="oracle.jdbc.driver.OracleDriver">
<param name="Pfad" value="Pfadbezeichnung für das Oracle-DBMS">
<param name="User" value="hier steht der Loginname des Users">
<param name="Password" value="hier steht das Paßwort des Users">
Hier steht das Applet Mini
</applet>
</body>
</html>
```

Um das Testbeispiel flexibel zu gestalten, werden die genaue Treiberbezeichnung, der Pfadname der Oracle-Datenbank, der Loginname des Users und das Paßwort des Users über die HTML-Datei übergeben.

Die genaue Pfadbezeichnung (URL der Datenbank) ist hierbei zumeist die größte Schwierigkeit. Diese hat allgemein folgendes Format:

jdbc:<subprotocol>:<subname>

- <Subprotokoll> bezeichnet dabei den Treibernamen bzw. das DBMS.
- <Subname> hängt von <Subprotokoll> ab.
- falls <Subname> eine Netzwerkadresse enthält, gliedert sie sich auf in:
  "//<Rechnername>:<Port>/<Subsubname>" [10]

Beispielsweise lautet der genaue Pfadname der Oracle-Datenbank der FH-Gummersbach mit der Host-Zeichenfolge "vorlesung" (Alias zu "sid2"):

jdbc:oracle:thin:@cips2.gm.fh-koeln.de:1527:sid2

Der zugehörige Quellcode des Java-Applets (mini.java):

```
//mini.java, copyright by Timo Leimbach
//Programm zum beispielhaften Auslesen eines Tables
//Test in einer Oracle8 Datenbank
//mit den Spalten Nummer (Integer), Name (Varchar2) und
//Jahrgang (Integer)

import java.sql.Connection ; //JDBC-Klasse zum Verbindungsaufbau
 import java.sql.DriverManager ; //JDBC-Klasse zur Implementation des
                                 //Treibermanagers, der Objekte der Klasse
                                 //Driver verwaltet
import java.sql.ResultSet ; //JDBC-Klasse zur Verwaltung einer Ergebnistabelle der Datenbankabfrage

import java.sql.Statement ; //JDBC-Klasse zum Senden von SQL-Anweisungen und Ergebniszugriff

import java.applet.Applet; //Vaterklasse jedes Applets

import java.awt.*; //Grafikpaket, hier für den Ende-Button und für die Klasse Graphics (grafische Textausgabe mit drawString)

import java.awt.event.*; //Paket für Adapter-, Event- und Listenerklassen, hier: ActionListener zum Reagieren auf den Ende-Button

import java.util.*; //Paket Utilities, zumeist Klassen für die Datenverwaltung (hier: Klasse Vektor == lineare Liste)


public class mini extends Applet //Java-Applets erben immer die Klasse Applet
 implements ActionListener //die abstrakten Methoden der Klasse ActionListener werden später implementiert,
                           //zur Reaktion auf den Ende Button
{

private ResultSet rs;//Deklaration von rs, hier werden die Ergebnisse der Datenbankabfrage beim Auslesen geschrieben

private Button ende;//Deklaration des Ende-Buttons

 private Vector elements; //Deklaration der linearen Liste Elements, die zur Aufnahme der Abfrageergebnisse aus der
```

```
                    //Datenbank benutzt wird
public void init()
{
super.init(); //parameterloser Aufruf des Superklassenkonstruktors
elements = new Vector(); //Reservierung des Speicherplatzes für die lineare Liste
setSize(500,150); //definiert die Größe des Grafikfensters
setLayout(new FlowLayout()); //definiert die Art des Layouts für Buttons
ende = new Button("Ende"); //erzeugt Speicherplatz für den Ende-Button
ende.addActionListener(this); //ordnet dem Ende-Button einen ActionListener zu
add(ende); //fügt den Ende-Button dem Layout zu
setBackground(Color.lightGray); //setzt die Hintergrundfarbe des Grafikfensters
setVisible(true); //setzt das Layout auf "sichtbar"
showStatus("JDBC Mini Connection to Oracle8i"); //verfügt über die Statusanzeige des Browsers
} //Ende von (init)
public void start()
{
Graphics g = getGraphics(); //gewährt Zugriff auf die grafische Textausgabe
try {
String s = getParameter("Treiber"); //entnimmt den Wert des Parameters Treiber aus dem aufrufenden HTML-Dokument
Class.forName(s); //liefert das Klassenobjekt von s, hier: der Java-Thin-Treiber
} //Ende von try
catch (Exception e) {
g.drawString("Class not found: "+e.toString(),10,100) ; //gibt eine Ausnahmefehlermeldung an
System.exit (0) ; //beendet das Applet
} //Ende von catch
try {
//Aufmachen der Datenbankverbindung
String pfad = getParameter("Pfad"); //entnimmt den Wert des Parameters Pfad aus dem aufrufenden HTML-Dokument
String user = getParameter("User"); //entnimmt den Wert des Parameters User aus dem aufrufenden HTML-Dokument
 String password = getParameter("Password"); //entnimmt den Wert des Parameters Password
                                             //aus dem aufrufenden HTML Dokument
Connection connection = DriverManager.getConnection(
pfad, // ## Treiber und Pfad
user, // ## User
password // ## Password
 ); //baut eine Verbindung zu der durch den JDBC-URL spezifizierten Datenbank auf
    //und liefert ein Connection-Objekt, welches die
    //Datenbankverbindung repräsentiert, als Ergebnis
String query = "SELECT * From Test" ; //besetzt den String query mit einer SQL-Anweisung
 Statement statement = connection.createStatement (); //erzeugt ein Statement-Objekt, welches
                                                      //zum Senden von SQL-Anweisungen dient
rs = statement.executeQuery (query) ; //führt die SQL-Anweisung aus und liefert ein einzelnes ResultSet zurück
while ( rs.next () ) { //Auslesen der einzelnen Zeilen des ResultSets
table Daten = new table(); //Erzeugung eines Daten-Tables
Daten.Nummer=rs.getInt ("Nummer"); //Auslesen der Spalte "Nummer" (Integer) und Ablegen in der Klasse "Daten"
Daten.Name=rs.getString ("Name"); //Auslesen der Spalte "Name" (String) und Ablegen in der Klasse "Daten"
```

```
Daten.Jahrgang=rs.getInt ("Jahrgang"); //Auslesen der Spalte "Jahrgang" (Integer) und Ablegen in der Klasse "Daten"

elements.addElement(Daten); //der Daten-Table wird der linearen Liste zugeführt
} // Ende von while

connection.close () ; //schließt die Datenbankverbindung und gibt alle mit ihr verbundenen Ressourcen frei

repaint(); //zeichnet die Grafikkomponente erneut durch Aufruf von paint (siehe weiter unten)
}

catch (java.sql.SQLException e) {

g.drawString ("SQLException: "+e.toString(),10,100) ; //gibt eine Ausnahmefehlermeldung grafisch aus

System.exit (0) ;//beendet das Applet

} //Ende von catch
} //Ende con start()

public void paint(Graphics g) //zuständig für die grafische Textausgabe
{

g.drawString("Dies wird eine Mini Connection",10,50); //schreibt den String in das Ausgabefenster

int i=0; //setzt i auf 0

 for (Enumeration el=elements.elements(); el.hasMoreElements();){ //"el" repräsentiert einen Zeiger auf die
                                                                   //lineare Liste "elements"
                                                                   //die "for-Schleife" ist erst dann beendet,
                                                                   //wenn "el" auf ein leeres Element zeigt
i++; //i wird um 1 erhöht

if (i==1) g.drawString("Hier sind sie:",10,70) ; //schreibt den String in das Ausgabefenster

table Daten2 = new table(); //erzeugt den Table "Daten2"

Daten2 = (table)el.nextElement(); //der in der linearen Liste gespeicherte Table wird in den Table "Daten2" kopiert

g.drawString(Daten2.Nummer + " | " +
Daten2.Name + " | " +
Daten2.Jahrgang,10,(80+i*10)) ; //grafische Textausgabe des Tables "Daten2"
} //Ende von for
} //Ende von paint(Graphics g)

 public void actionPerformed(ActionEvent event) { //wird ausgeführt, sobald ein ActionEvent ausgelöst wird,
                                                   //hier: Drücken des Ende-Buttons
String cmd = event.getActionCommand(); //gibt als String zurück, welcher Button gedrückt wurde

if (cmd.equals("Ende")) { //hier hinein verzweigen, falls der Ende-Button gedrückt wurde

setVisible(false); //setzt das Layout auf unsichtbar

Graphics g = getGraphics(); //ermöglicht den Zugriff auf das Grafikfenster

g.dispose(); //gibt alle Grafikressourcen frei und löscht das Grafikfenster

System.exit(0); //beendet das Applet
} //Ende von if
} //Ende von actionPerformed(Action Event event)


class table //definiert die Klasse "table"
{

int Nummer;

String Name;

int Jahrgang;

} //Ende von class Table
} //Ende von class mini
```

Wird der Java-Programmcode "mini.java" kompiliert, so werden automatisch die Dateien mini.class und mini$table.class erzeugt.

Die elementaren Bestandteile des Java-Applets sollen im folgenden aufgegriffen und verdeutlicht werden:

Die JDBC-Klassen (java.sql.*) Connection, Statement und ResultSet, sowie weitere, die nicht in diesem Beispiel verwandt wurden, sind als Interfaces definiert. Sie sind demnach eine Sammlung von Methoden, die erst durch den hier verwendeten Java-Thin-Treiber implementiert werden. Die meisten dieser Methoden lösen SQL-Exceptions aus, sobald beim Zugriff auf die Datenbank ein Fehler auftritt. Deshalb wird bei ihrem Aufruf die "try und catch – Klausel" verwandt.

Das Laden des JDBC-Treibers:

Die Zeile "Class.forName("oracle.jdbc.driver.OracleDriver");" lädt den JDBC-Treiber von Oracle. Diese Klassen werden erst zur Programmlaufzeit mit Hilfe der Methode "Class.forName()" durch den Classloader der Java-Virtual-Machine hinzugeladen. Anschließend meldet sich der Treiber in seinem Initialisierungscode beim JDBC-Treibermanager an. Sollte der Ladeversuch hingegen fehlschlagen, so wird eine "java.lang.ClassNotFoundException" erzeugt, die vom Programm abgefangen werden kann.

Das Öffnen der Datenbank:

Ist der JDBC-Treiber geladen, so kann die Datenbank mit Hilfe der Methode "DriverManager.getConnection()" geöffnet werden. Diese besitzt die drei Paramerter JDBC-URL (Erläuterung siehe oben), Benutzernamen und das zugehörige Paßwort. Der Treibermanager überprüft, welcher der geladenen Treiber, falls mehrere zur Verfügung stehen, den angegebenen JDBC-URL öffnen kann. Anschließend wird die entsprechende Methode des ausgewählten Treibers aufgerufen.

Senden von SQL-Anweisungen zum Auslesen der Datenbank:

Über die nun geöffnete Datenbankverbindung sendet der Client SQL-Anweisungen an das DBMS. Dies geschieht, indem ein Statement-Objekt mit "connection.createStatement ();" erzeugt wird, welches mit Hilfe des JDBC-Treibers und der Statement-Methode "executeQuery()" die Anweisungen an das DBMS übermittelt.

Auslesen, bearbeiten und darstellen der Treffermenge:

Das Ergebnis einer SELECT-Anweisung ist eine Tabelle. Der JDBC-Treiber stellt sie als Instanz der Klasse ResultSet zur Verfügung. Diese Klasse besitzt Methoden zum Durchlaufen der Tabelle und zum Zugriff auf die Ergebnisse. In dem hier betrachteten Beispiel wird das ResultSet "rs" mittels "while (rs.next()) { }" zeilenweise durchlaufen und über die Methoden "getInt()" und "getString()" ausgelesen. Das Ergebnis wird in eine neu gebildete Instanz der Klasse "Daten" geschrieben, die anschließend an die lineare Liste (Vector) "elements" angehängt wird.

Ist die Datenbankverbindung nicht mehr nötig, so wird sie mit "connection.close();" geschlossen. Gleichzeitig werden dadurch das ResultSet und das Statement geschlossen, sowie alle übrigen belegten Ressourcen freigegeben.

Das Darstellen der Treffermenge geschieht in der Methode "paint(Graphics g)", die das Applet von java.awt.component geerbt hat. Hier wird die lineare Liste "elements" ausgelesen und mit der Methode "drawString()" angezeigt.

Allgemein kann das Zusammenspiel der wichtigsten Interfaces der JDBC-Klassenbibliothek anhand Abbildung 9.1 verdeutlicht werden:

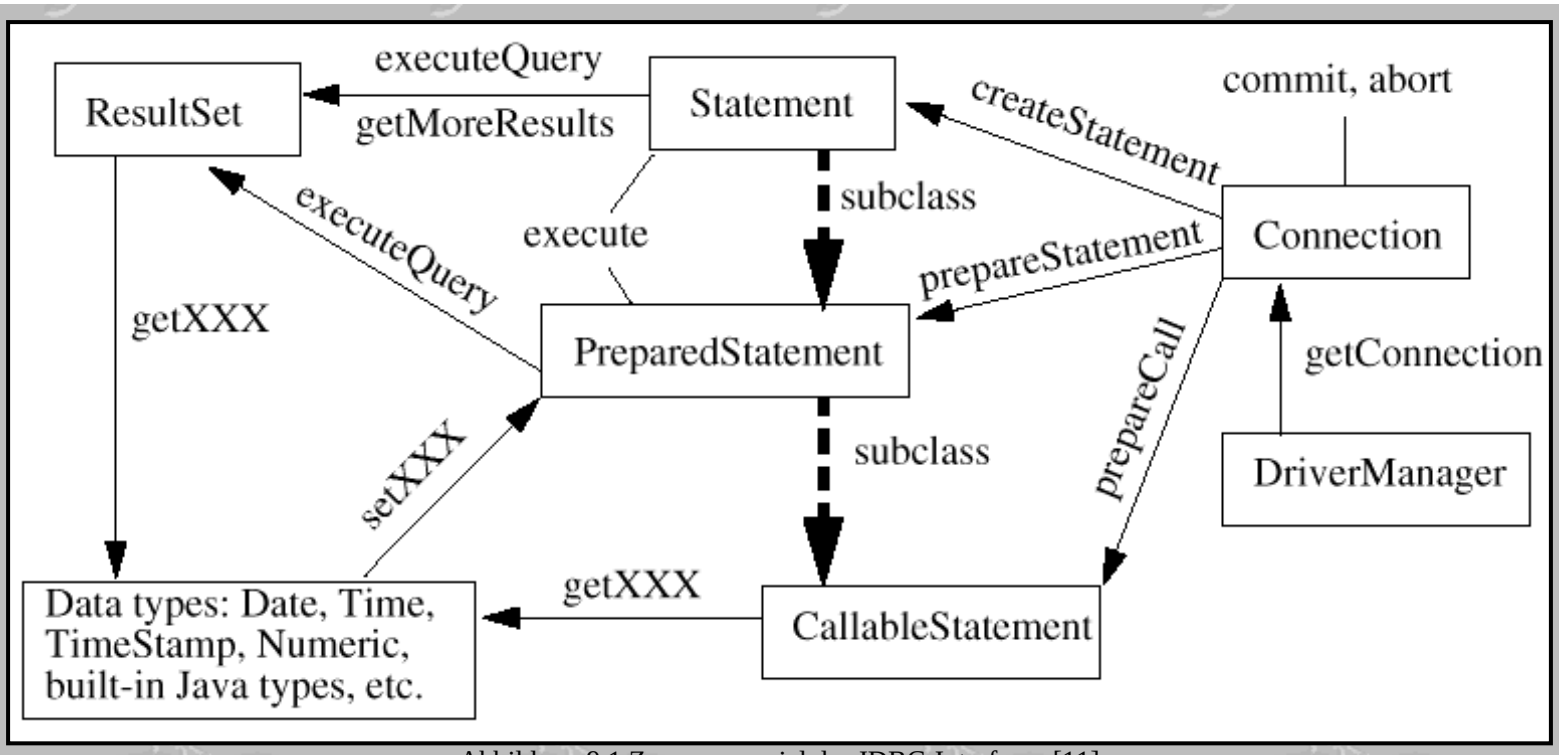


Abbildung 9.1 Zusammenspiel der JDBC-Interfaces [11]

[10] Kowalewski, Daniel; Uni Hannover [Stand 11.6.1999]
[11] Hamilton, Graham; Cattell, Rick; JDBC: A Java SQL API; 10.1.1997; jdbc.spec-0120.pdf; JavaSoft

10. Das Applet-Sandbox/Firewall-Problem

Damit Applets keinen Schaden auf den Client-Rechnern herbeiführen auf denen sie ablaufen, hat JavaSoft ein eigenes Sicherheitskonzept hierfür erarbeitet. Prinzipiell soll hierdurch jede Ausführung einer Operation des Applets untersagt sein, wenn sie nicht eindeutig durch den "Security Manager" erlaubt wird. Dieser kommuniziert zur Laufzeit mit der "**Sandbox**" (hier: die Java Virtual Machine), dem Fenster, in dem das Applet ausgeführt wird. Beispiele für Restriktionen:

- kein Zugriff auf das lokale Dateisystem
- keine Informationen über Dateien erlangen
- keinen Verbindungsaufbau zu anderen Rechnern als dem Host des Applets
- keine Ausführung von Systemaufrufen oder Maschinencode
- erzeugte Fenster werden explizit als "unsigned" bezeichnet
- stark eingeschränkter Zugriff auf Systemkomponenten
- keine Überschreibung des "Class Loaders"/"Security Managers" möglich

Falls eine Sicherheitsverletzung auftreten sollte, so erzeugt der "Security Manager" eine "Security Exception", die innerhalb der "Java Console" des benutzten Browsers festgehalten wird.

Ein weiteres, eng mit der "Sandbox" zusammenhängendes Sicherheitskonzept, ist das des "Class Loaders". Dieser ist für den Ladevorgang des Appletcodes verantwortlich und erzeugt die "Namespace Hierarchy" (Festlegung der Speichergrenzen für die "Sandbox" des jeweiligen Applets). Zudem wird überprüft, ob zusätzliche Klassen nachgeladen werden dürfen (jedoch nur eigene oder Standard API-Klassen). Durch Einsatz des "Class Loaders" wird sichergestellt, daß Applets keine Standard-API-Klassen durch ihre eigenen ersetzen und damit Schaden anrichten können.

Für eine **direkte Verbindung** von Applets zu einer Datenbank bedeutet die Sicherheitseinschränkung durch das "Sandbox-Modell", daß sich die Datenbank auf dem Host des Applets (WWW-Server)

befinden muß. Es existieren jedoch zwei Möglichkeiten diese spezielle Einschränkung zu umgehen:

- Verwendung von "Signed-"Applets
- Verwendung einer speziellen Web-Server-Software

"Signed-"Applets:

Ein Applet gilt als "signed", sofern es durch eine digitale Signatur begleitet wird. Dies ist eine verschlüsselte Datei, die genau angibt, woher das Programm stammt. Benutzer, denen bekannt ist, wer ein bestimmtes Programm hergestellt hat, können nun selber darüber entscheiden, ob sie dieser Quelle vertrauen möchten.

Damit ein Applet-Hersteller diese digitale Signatur erhält, muß er seine Identität gegenüber einer Gruppe, die als Zertifizierungsautorität bezeichnet wird, verifizieren. Diese sind derzeit jedoch nach Art des Browsers, in dem das Applet ablaufen soll, zu unterscheiden.

Wird ein "signed-" Applet auf dem Client gestartet, so werden alle zuvor geltenden Sicherheitsmaßnahmen aufgehoben.

"Spezielle Web-Server-Software":

Firmen, die eine solche Software anbieten, sind z.B. Weblogic (Weblogic Application Server) oder IDS Software (IDS Server). Speziell für die Datenbank Oracle wird hier in Verbindung mit der Web-Server-Software, die gleichzeitig eine Middleware repräsentiert, ein Typ 3 Treiber angeboten. Damit schrumpft die Größe des Treibers, den das Applet benötigt, auf etwa 80Kbyte zusammen.

Die Middleware übernimmt nun, wie in Kapitel 7 beschrieben, die Kommunikation mit der Datenbank. Gleichzeitig werden zusätzliche Verbesserungen im Umgang mit der Treibersoftware erzielt:

- Sicherheit beim Datentransfer durch Einsatz der Sicherheitsschicht im Kommunikationsprotokoll (SSL)
- erweiterter SQL-Sprachschatz
- HTTP-firewall tunneling (Erklärung: s.u.)
- Connection pooling (Erklärung: s.u.)

Ein weiteres Problem von Datenbankanbindungen an das Internet mittels Applets besteht in einer den Server und/oder den Client umgebenden **Firewall**.

Firewalls dienen i.a. der Abschottung, Protokollierung und Verschlüsselung des Datenverkehrs, sowohl in Richtung der eigenen Infrastruktur als auch von ihr kommend. Ob die Erlaubnis oder die Verhinderung des Datenverkehrs im Vordergrund steht, hängt dabei von dem konkret verwendeten Produkt ab.

Grundsätzlich sind zwei Ebenen zu unterscheiden, auf denen Firewalls zum Einsatz kommen:

- Die Netzwerkebene (Protokollebene)
  Hier werden Datenflüsse auf der Ebene der TCP/IP-Verbindungen kontrolliert. Die Sperrung, die Erlaubnis oder die Protokollierung wird hierbei von der Absenderadresse, der Adresse des Adressaten und der Port-Nummer des Adressaten beeinflußt.

- Die Applikationsebene
  Auf Applikationsebene werden, im Gegensatz zu Firewall-Techniken der Netzwerkebene, die Inhalte der Datenpakete und deren Zuordnung zu den auf dem Netzwerk laufenden Diensten mit einbezogen. Firewalls auf Applikationsebene sind als sogenannte Proxy-Server realisiert und dienen als Brücke zwischen dem offenen und dem zu schützenden Netz. Um beispielsweise die interne technische Infrastruktur nach außen hin zu verschleiern, werden Adreßteile der Rechner des Subnetzes durch den Proxy-Server maskiert und umgesetzt.

Der Einsatz einer Firewall kann demnach grundsätzlich dazu führen, daß Applets mittels JDBC keine Verbindung zu der angefragten Datenbank aufnehmen können (Ausnahme: im Fall einer vollen Kontrolle über die Firewall können spezifizierte Verbindungen zulassen werden).

Um dennoch mit der Datenbank kommunizieren zu können, muß auf eine **indirekte Datenbankanbindung** zurückgegriffen werden. (Eine Ausnahme bildet die zuvor beschriebene "spezielle Web-Server-Software", die durch Einsatz eines Typ3-Treibers und einer Middleware noch zu der "direkten Datenbankanbindung" gerechnet werden kann.) Dadurch wird erreicht , daß andere Protokolle als bei der Kommunikation mittels JDBC eingesetzt werden können, um die Firewall zu passieren.

Auf Protokollebene können "Sockets", eine Abstraktion von standardmäßigen TCP-Sockets, wie sie von UNIX her bekannt sind, zur Verbindung mit einem CGI-Skript oder Servlet genutzt werden. Diese Funktionalität wird von dem Paket "java.net" bereitgestellt, das es Java-Programmen grundsätzlich ermöglicht, über ein Netzwerk zu kommunizieren. Dazu müssen allerdings auch "Sockets" auf der Firewall aktiviert werden können, über die Server und Client miteinander kommunizieren.

Auf Applikationsebene existieren diverse Proxy-Server, die den Einsatz von RMI (Remote Method Invocation, Kommunikationssprache von verteilten Java-Anwendungen) oder IIOP (Kommunikationssprache von "Object Request Brokern" (CORBA)), gestatten. RMI kann zudem mittels spezieller Klassenbibliotheken nach IIOP umgesetzt werden (in Zukunft wird wahrscheinlich der Standard von IIOP den des RMI ersetzen [12]).

Die sicherste Methode eine Firewall zu passieren, ist die, das Hyper Text Transfer Protokoll, wie es beim Laden von Web-Seiten genutzt wird, zu verwenden. Damit kann i.a. jede Firewall "durchtunnelt" werden ("**HTTP-Tunneling**").

Das auf dem Server ablaufende Servlet erbt dabei standardmäßig von HTTPServlet. Dies wird i.a. dazu benutzt, durch das Servlet auf "get-requests" von HTML-Seiten hin, weitere HTML-Seiten auszugeben. Für den Client ist eine solche Methode nicht in der Standard-API enthalten. Ein nützliches Beispiel (mit den notwendigen Klassen) ist jedoch unter www.servletguru.com in einer Anwendung, die einen einfachen Taschenrechner simuliert (MathLiteApplet), enthalten. Der Autor Karl Moss beschreibt hieran die Technik des "HTTP-tunneling" in Kapitel 11 seines Buches "Java Servlets".

Bei dem Einsatz einer Applet-Servlet-Datenbankverbindung ist des weiteren der

Punkt der "multiple hits" zu beachten. Hierunter ist zu verstehen, daß das Servlet von mehreren Applets gleichzeitig angesprochen werden kann, was in mehreren miteinander konkurrierenden Datenbankverbindungen resultiert. Ausweg bieten "connection broker", die das Prinzip des "connection pooling" einsetzen. Dies bedeutet, daß das Servlet einen Pool von Verbindungen zur Datenbank aufbaut. Bei Eintreffen eines Applet-"requests" wird eine dieser bereits bestehenden Verbindungen zur Datenbankkommunikation genutzt und später wieder in den Pool zurückgegeben.

Somit hat der Einsatz eines Servlets einige entscheidende Vorteile gegenüber der "direkten Datenbankanbindung":

- die Datenbankverbindung ist im Falle einer bekannten Verbindung bereits existent
- das Applet kommuniziert über HTTP und benötigt keinen JDBC-Treiber mehr, der ansonsten zusätzlich über das Netz geladen werden muß
- durch den Einsatz des "HTTP-tunneling"-Verfahrens kann i.a. jede Firewall passiert werden
- die Daten können zur Übertragung verschlüsselt werden
- die Datenbank muß sich nicht auf dem WWW-Server befinden

Produkte, wie das zuvor beschriebene Servlet, werden auch von kommerziellen Firmen angeboten. Das Applet benutzt bei diesem Verfahren lediglich die mitgelieferte Treibersoftware, die die JDBC-Spezifikation erfüllt. Kommunikationsaufrufe werden hierbei zunächst versucht über "Sockets" abzuwickeln und falls dieser Versuch fehlschlägt, automatisch nach HTTP konvertiert. Das Servlet übernimmt im Anschluß die Verbindung zu der Datenbank. Ein Beispiel hierfür ist das Produkt JDataConnect von der Firma Software Synergy. Der hier benutzte Treiber besitzt eine Größe von 20KByte und kostet im Paket mit zwei Lizenzen für Hochschulen $424. Ein Nachteil ist jedoch, daß das Produkt derzeit serverseitig nur für auf Windows basierenden Plattformen verfügbar ist.

Ein ähnliches Produkt mit einer Treibergröße von 26KByte, welches zusätzlich eine Verschlüsselung der Daten durch SSL erzielt, liefert die Firma ThinWeb. Hierfür existiert auch eine Version, die z.B auf den Servern der FH-Gummersbach (RS6000, AIX 4.1.5 (Power PC) mit Apache Software) lauffähig ist. Fünf "Runtime"-Lizenzen werden hierbei zum Preis von $995 angeboten.

Eine weitere Alternative, die den Ansprüchen des Praxisbeispiels aus Kapitel 11 genügt, ist ein Produkt der Firma OpenLink (Treibergröße 45KByte). Dieses wird als Testversion, ohne zeitliche Begrenzung, mit zwei "Runtime"-Lizenzen und maximal zehn Datenbankverbindungen ("connection pool"), umsonst zur Verfügung gestellt. Firewalls werden hierbei nicht durch "HTTP-tunneling", sondern durch "in-build-proxying capabilities" passiert. Hierunter wird eine Einstellung verstanden, die auf den meisten Firewalls vorgenommen werden kann, die für die Verbindung genutzt wird.

[12] Graf, Olaf; IBM Deutschland; persönliche Nachricht vom 14.6.1999

11. Datenbankanbindung an das Internet

(Praxisbeispiel)

Als Praxisbeispiel wurde die Anbindung der Oracle-Datenbank der Fachhochschule Gummersbach an das Internet ausgewählt. Mit einem mindestens Java 1.1.5-fähigen Browser soll es dabei für Dozenten der Einrichtung möglich sein, die Inhalte ihrer Lehrveranstaltungen über vordefinierte Eingabemasken komfortabel zu beschreiben. Ziel ist es, daß Studenten über geeignete Suchkriterien Einsicht in diese Eintragungen erhalten können.

Zur Lösung dieser Aufgabenstellung wurde das Prinzip der "direkten Datenbankanbindung" (siehe Kapitel 8.1) ausgewählt, bei der ein Applet mittels Java-Thin-Treiber mit der Oracle-Datenbank kommuniziert. Diese Entscheidung begründet sich mit dem komfortablen Design, der Benutzerfreundlichkeit, sowie der Kürze der Entwicklungszeit für diese Anwendung. Der Nachteil einer relativ langen Ladezeit bei Aufruf der Anwendung im Internet fiel hierbei nicht ins Gewicht, da dieser durch die Benutzung eines Produktes, wie z.B. von OpenLink (siehe Kapitel 10), leicht eliminiert werden kann.

Eine besondere Bedeutung kam bei der Entwicklung der Flexibilität des Produktes zu, welche durch "Soft-Coding" erreicht wurde. Mit dieser Technik wird ein Programmierstil bezeichnet, der das Umstellen von Anwendungen auf geänderte Bedingungen erlaubt, ohne daß der Programmcode umgeschrieben werden muß. Im Falle der vorliegenden Arbeit wurde dies durch die nachfolgend beschriebene Parameterübergabe des HTML-Codes an das Applet, sowie einer Textdatei "login.txt", die im selben Verzeichnis wie der HTML-Code zu finden ist, erreicht.

Parameterübergabe der HTML-Datei an das Applet:

(Auszug)

```
<APPLET
CODEBASE = "."
CODE = "Hochschule.MainCode.class"
ARCHIVE = "classes111.zip,Hochschule.jar"
WIDTH = 400
HEIGHT = 300
HSPACE = 0
VSPACE = 0
ALIGN = "absmiddle"
>
<!-- unter Alias stehen alle Platten des DB-Systems mit Synonymen -->
<!-- diese Liste kann beliebig erweitert werden -->
<PARAM NAME = "Alias1" VALUE = "sid1,Diplom">
<PARAM NAME = "Alias2" VALUE = "sid2,Vorlesung">
<!-- die Liste der akzeptierten Dozenten ist in login.txt gespeichert -->
<PARAM NAME = "Studiengang" VALUE = "AI,TI,WI,WZ">
<!-- Werte des Parameters 'Studiengang' bitte alphabetisch sortieren -->
<PARAM NAME = "Treiber" VALUE = "oracle.jdbc.driver.OracleDriver">
<PARAM NAME = "Pfad" VALUE = "jdbc:oracle:thin:@cips2.gm.fh-koeln.de:1527:">
<PARAM NAME = "Betreuer" VALUE = "Faeskorn-Woyke">
<PARAM NAME = "Studentenpfad" VALUE = "jdbc:oracle:thin:@cips2.gm.fh-koeln.de:1527:sid2">
Verwenden Sie einen java-fähigen (JDK 1.1.5) Browser oder schalten Sie Java zu.<BR>
```

```
</APPLET>
```

Mit den Parametern "Alias1" und "Alias2" werden die Bezeichnungen, sowie die Synonyme für die einzelne Festplatten des Datenbanksystems übergeben. Sollte das System erweitert werden, so genügt es, die Liste der Alias kohärent fortzuführen.

Über "Studiengang" werden die Abkürzungen für die einzelnen, in das System aufgenommenen Fachrichtungen, definiert. Diese stehen dann dem Dozenten bei der Fächerbeschreibung innerhalb einer "Choice-Box" zur Verfügung.

Mittels "Treiber" und "Pfad" wird der derzeit benutzte Treiber, sowie die URL der anzusprechenden Datenbank (siehe Kapitel 9), jedoch ohne Festplattenspezifikation, festgelegt. Würde bspw. der Treiber auf ein kommerzielles Produkt (siehe Kapitel 10) umgestellt, so müßte im Idealfall lediglich der Treibereintrag an dieser Stelle geändert werden.

Dem Parameter "Betreuer" kommt eine besondere Bedeutung zu. Er verschleiert den Benutzernamen und das Paßwort, getrennt durch ein "-", da HTML-Seiten grundsätzlich frei im Internet eingesehen werden können. Das Applet benötigt diese Angaben, um sich für den lesenden Teil der Anwendung mit der Datenbank verbinden zu können. "Studentenpfad" liefert hierzu den erforderlichen vollständigen URL der Datenbank.

Ein weiteres Problem der Anwendung würde darin bestehen, daß auf der hier vorliegenden Datenbank Dozenten und Studenten gleichermaßen Zugriff besitzen. Das System kann zwischen diesen zwei Personenkreisen nicht unterscheiden. Da jedoch nur Dozenten das Recht besitzen dürfen Fächer zu beschreiben, wurden ihre Benutzernamen in einer Textdatei "login.txt", um sie nicht frei im Internet zugänglich zu machen, abgelegt. Diese besitzt folgende grundsätzliche Struktur:

```
<!-- Bitte Namen in vorgegebener Form (Benutzername,==Alias) eintragen !! -->
<!-- Alias bitte nicht länger als 25 Zeichen !! -->>
Faeskorn
==Prof. Dr. Faeskorn-Woyke
Scott
==Herr Scott
...
```

Gleichzeitig wird hier der Name festgehalten (Alias) mit dem Dozenten später innerhalb von Suchanfragen im Programm erscheinen. Diese Liste liefert zudem die Grundlage für die maximale Anzahl von zu durchsuchenden Tabellen.

Durch den Aufruf vorangegangener HTML-Seite wird das Applet durch den Browser geladen und die Virtual Machine (VM) beginnt mit der Ausführung des Java Byte Codes. Über die automatisch angesprochene Methode init() der Klasse MainCode (siehe HTML-Code, sowie Kapitel 4) werden die Parameter ausgelesen und die "Hauptseite" des Applets aufgebaut:

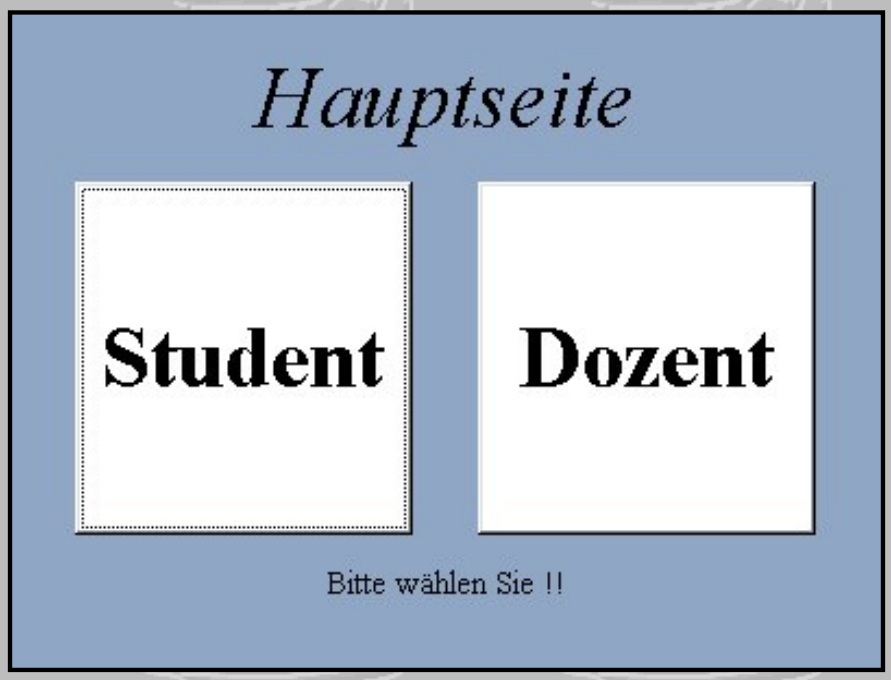


Abbildung 11.1 Hauptseite des Applets

Im Falle von fehlenden Parametern wird automatisch eine Ausnahme ("Exception") erzeugt, die zur Ausgabe einer zugeschnittenen Fehlermeldung führt und das Applet beendet.

Auf der Hauptseite entscheidet sich der Benutzer, zu welchem Personenkreis er gehört, durch Aktivierung des entsprechenden "Buttons" per Mouseclick oder durch die Tastatur. Das Programm realisiert diese Ereignisse als Action- bzw. Key-Events und ruft die im Anschluß kurz erläuterten Methoden und Klassen auf.

Auswahl von Dozent:

Folgendes Ablaufschema (siehe Abbildung 11.3 und 11.7) soll die Funktion des Applets widerspiegeln, die durch Betätigung des "Buttons Dozent" aktiviert wird:

Zunächst wird eine neue Klasse "PasswordDialog" instanziert, die von "Dialog" abgeleitet ist. Somit stellt sie ein spezialisiertes Übergangsfenster dar, das dazu dient, den Benutzer über das Ereignis "Anmelden auf der Oracle Datenbank" zu informieren und Verifikationseingaben von ihm entgegen zu nehmen (siehe Abbildung 11.2).

Abbildung 11.2 Paßwort-Dialog

Darüber hinaus übernimmt die Klasse auch die Aufgabe, den in der HTML-Seite festgelegten JDBC-Treiber zu laden, mit den Benutzereingaben eine Verbindung zur Datenbank ("Connection") aufzubauen, den Benutzer mit Hilfe von "login.txt" auf seine Eigenschaft als Dozent zu prüfen, sowie alle bereits beschriebenen Fächertitel des Dozenten auszulesen. Beim ersten Aufruf des Applets durch einen Dozenten wird die Tabelle "FHGMinfo" auf der Datenbank angelegt und den Studenten lesende Rechte hierauf eingeräumt. Alle dazu nötigen Anweisungen sind in "try()-" und "catch()-Blöcken" verankert.

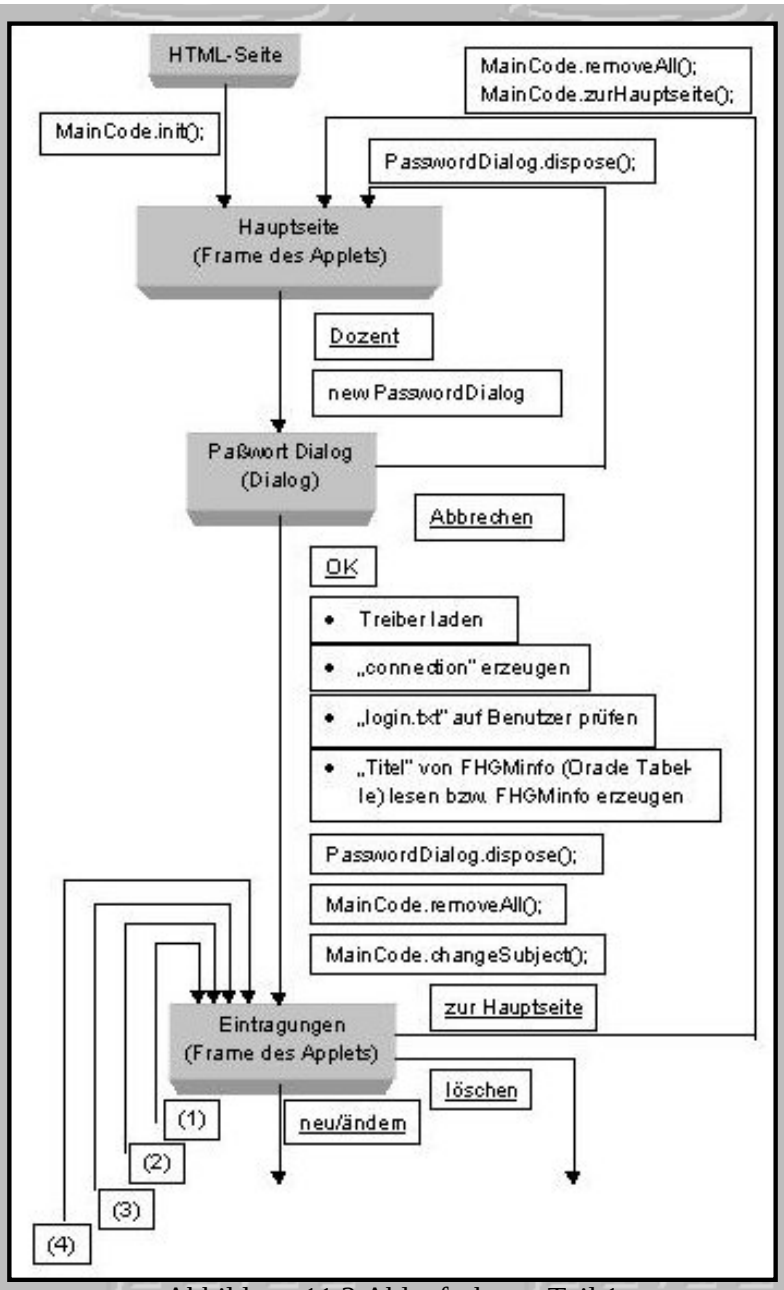


Abbildung 11.3 Ablaufschema Teil 1

Diese Technik ("Exception-Handling") dient zum Aufruf von Fehlerbehandlungsmethoden, falls bspw. die Verbindung zur Datenbank abreißt, nicht erzeugt werden kann etc.. In allen diesen Fällen wird, durch Instanzierung der Klasse "NoConnectionDialog" und Übergabe des Strings "Error" an deren Konstruktor, eine konkrete Fehlermeldung auf der Bildschirm angezeigt.

Meldet sich der Benutzer erfolgreich auf der Datenbank an, so erfolgt der nächste Teil des Programmsablaufs im "Frame des Applets", d.h. in dem Teil der HTML-Seite, der für das Applet reserviert wurde. Der Bildschirm wird hierzu durch Aufruf der Methode "removeAll()" gelöscht und "changeSubject()" übernimmt den neuen Aufbau der Anzeige (siehe Abbildung 11.4).

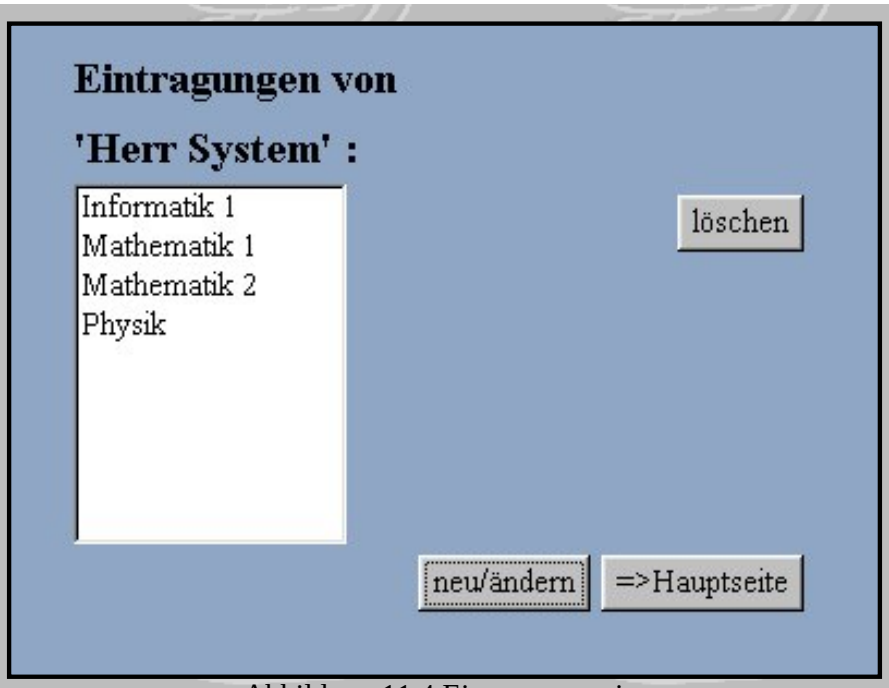


Abbildung 11.4 Eintragungsseite

Hier hat der Benutzer die Möglichkeit neue Eintragungen vorzunehmen, bereits bestehende Eintragungen zu ändern oder zu löschen, sowie zur Hauptseite des Programms zurückzukehren, was gleichzeitig ein Schließen der Datenbankverbindung bewirkt ("connection.close()"). Aus Sicherheitsgründen wird vor dem Löschen eines selektierten Eintrags aus der Fächerliste durch Instanzierung der Klasse "JaNeinDialog" ein Dialog aufgerufen, in dem der Löschvorgang vor Ausführung noch einmal bestätigt werden muß (siehe Abbildung 11.5).

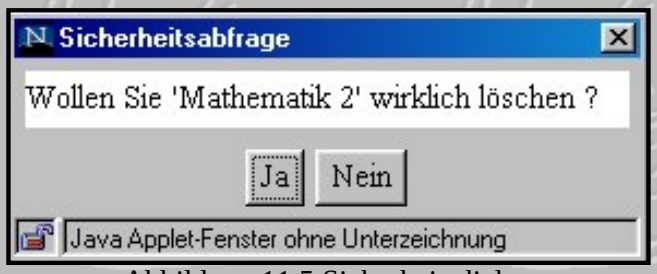


Abbildung 11.5 Sicherheitsdialog

Ist dies der Fall, so wird die entsprechende SQL-Anweisung (DDL) zum Löschen des Eintrags in einem String formuliert, der über Aufruf von "Statement.executeUpdate(String)" an die Datenbank übergeben wird. Die "commit"-Anweisung wird automatisch ausgeführt, da das Programm zuvor den Befehl "connection.setAutoCommit(true)" durchläuft.

Falls der Button "neu/ändern" aktiviert wird, wird geprüft, ob ein Eintrag aus der Fächerliste zuvor ausgewählt worden ist. Ist dies der Fall, werden alle zugehörigen Eintragungen aus der Datenbank ausgelesen und in einem Objekt der Klasse "infoTable" abgelegt. Anschließend wird, wie beim Anlegen eines neuen Eintrags, die Klasse "Maske" instanziert, der eine Referenz auf "infoTable" übergeben wird.

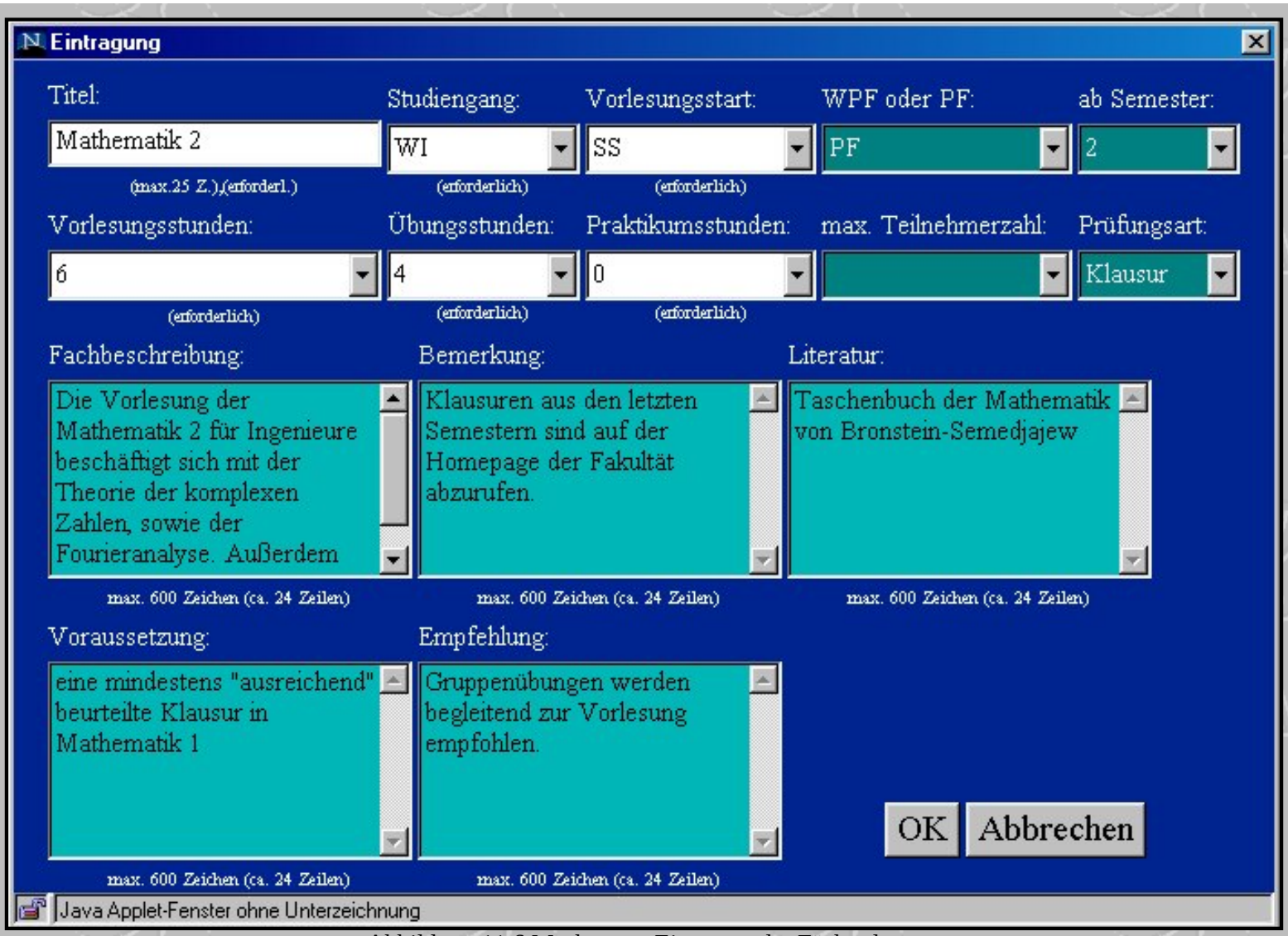


Abbildung 11.6 Maske zum Eintragen der Fächerdaten

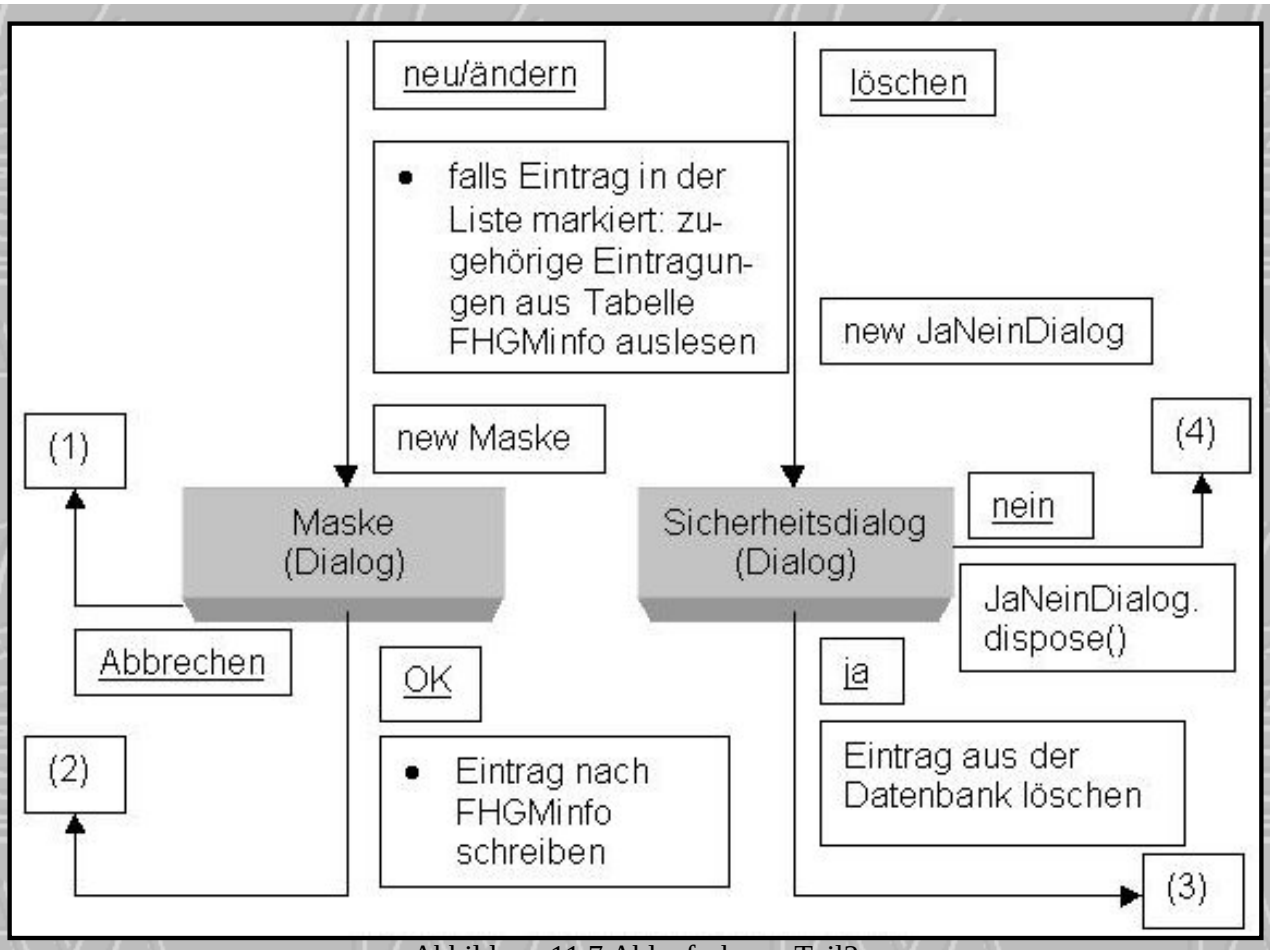


Abbildung 11.7 Ablaufschema Teil2

Innerhalb des Dialogs "Maske" sind sechs Felder als "erforderlich" gekennzeichnet. Die hier erfolgenden Eintragungen dienen dem späteren Auffinden dieses Faches in der "Suchanfrage" des Studenten. Sollte die maximal definierte Zeichenlänge der Eingabefelder Titel, Beschreibung, Bemerkung, Literatur, Voraussetzung oder Empfehlung überschritten werden, so erfolgt bei der Bestätigung durch den Button "OK" eine detaillierte Fehlermeldung, die ein Abspeichern der Eingabe vorerst unterbindet.

Im Falle einer erfolgreichen Übernahme der Eingabedaten wird die entsprechende SQL-Anweisung (DDL) zum Eintragen der Daten in die Datenbank in einem String formuliert und über Aufruf von "Statement.executeUpdate(String)" ausgeführt. Anschließend kehrt das Programm zur Eingabeseite des Dozenten zurück, wobei der neu erzeugte "Titel" alphabetisch sortiert in die Fächerliste übernommen wird.

Auswahl von Student:

Wird auf der "Hauptseite" des Applets der Button "Student" ausgewählt (siehe Abbildung 11.9), so wird der Frame des Applets gelöscht und die Methode "MainCode.choosePage()" kreiert die Seite Suchanfrage, innerhalb der Titel, Dozentenname, Vorlesungsstart, -art, sowie Studiengang ausgewählt werden können (siehe Abbildung 11.8).

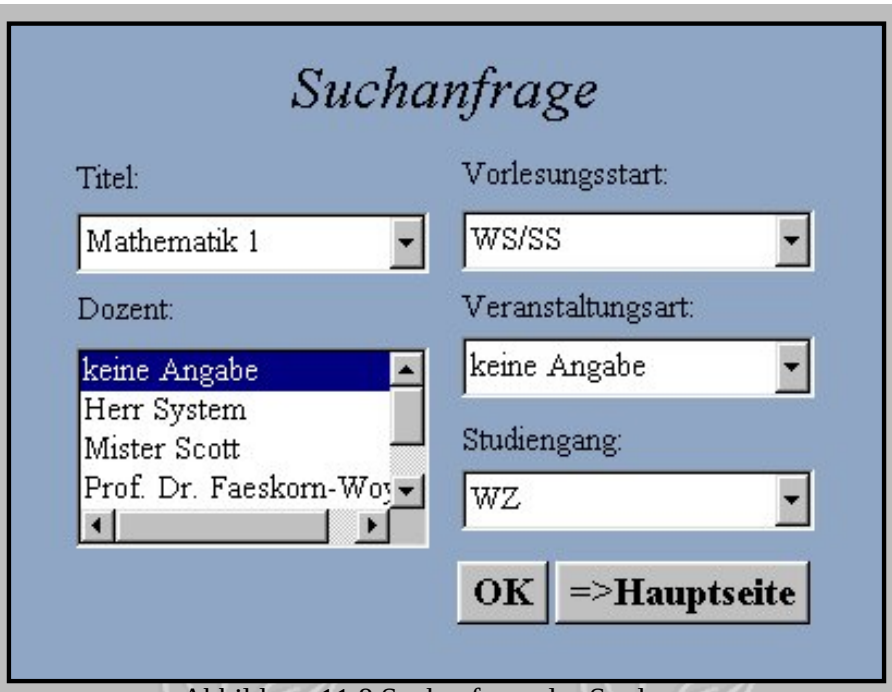
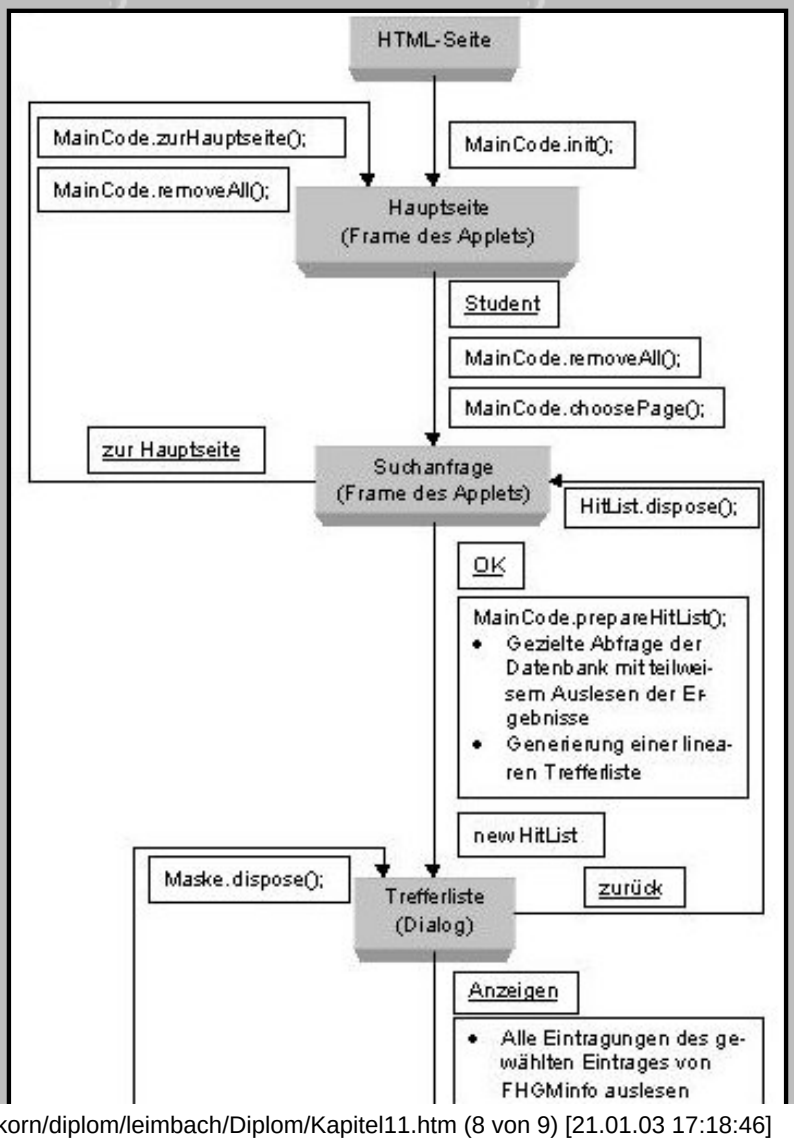


Abbildung 11.8 Suchanfrage des Studenten

Zudem übernimmt die Methode den Verbindungsaufbau zur Datenbank, auf der sie sich mit dem unter Parameter "Betreuer" befindlichen Namen und Paßwort anmeldet. Aus der Liste "login.txt" werden die Namen der Dozenten extrahiert, über die unter anderem alle verfügbaren Tabellen nach unterschiedlichen Vorlesungstiteln durchsucht werden. Anschließend werden diese Daten in den Feldern "Titel" und "Dozent" angezeigt. Wie auch sonst überall im Programm üblich, sind alle kritischen Abfragen in die bereits erwähnten "try()-" und "catch()-Blöcke" eingebettet, um im Ausnahmefall eine Fehlerbehandlungsmethode aufzurufen, die das auftretende Problem durch Instanzierung der Klasse "NoConnectionDialog" auf dem Bildschirm ausgibt.



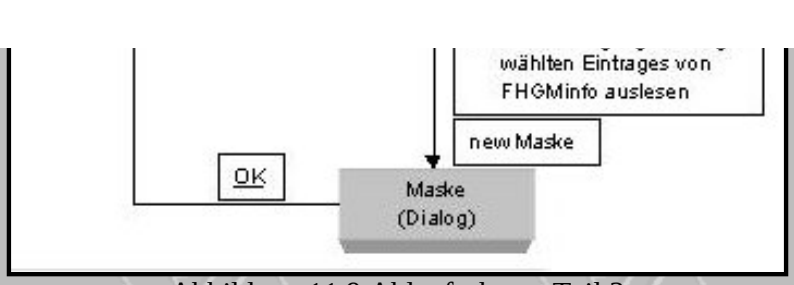


Abbildung 11.9 Ablaufschema Teil 3

Wird eine Suchanfrage durch Drücken des "OK-Buttons" aktiviert, so werden alle fünf Eingabefelder der Suchanfrage innerhalb der Methode "prepareHitList()" ausgelesen und die passende "SELECT"-Anweisung in einem String abgelegt. Diese wird anschließend auf alle Tabellen "FHGMinfo" der ausgewählten Dozenten mit der Methode "statement.executeQuery(String)" angewandt und erzeugt eine lineare Liste ("Vector"). Hierin befinden sich alle Treffer, angeordnet in Objekten der Klasse "infoTable". Diese lineare Liste ("Vector"), primär sortiert nach Dozenten, sekundär sortiert nach Titeln, wird bei der Instanzierung der Klasse "HitList" (siehe Abbildung 11.10) übergeben.

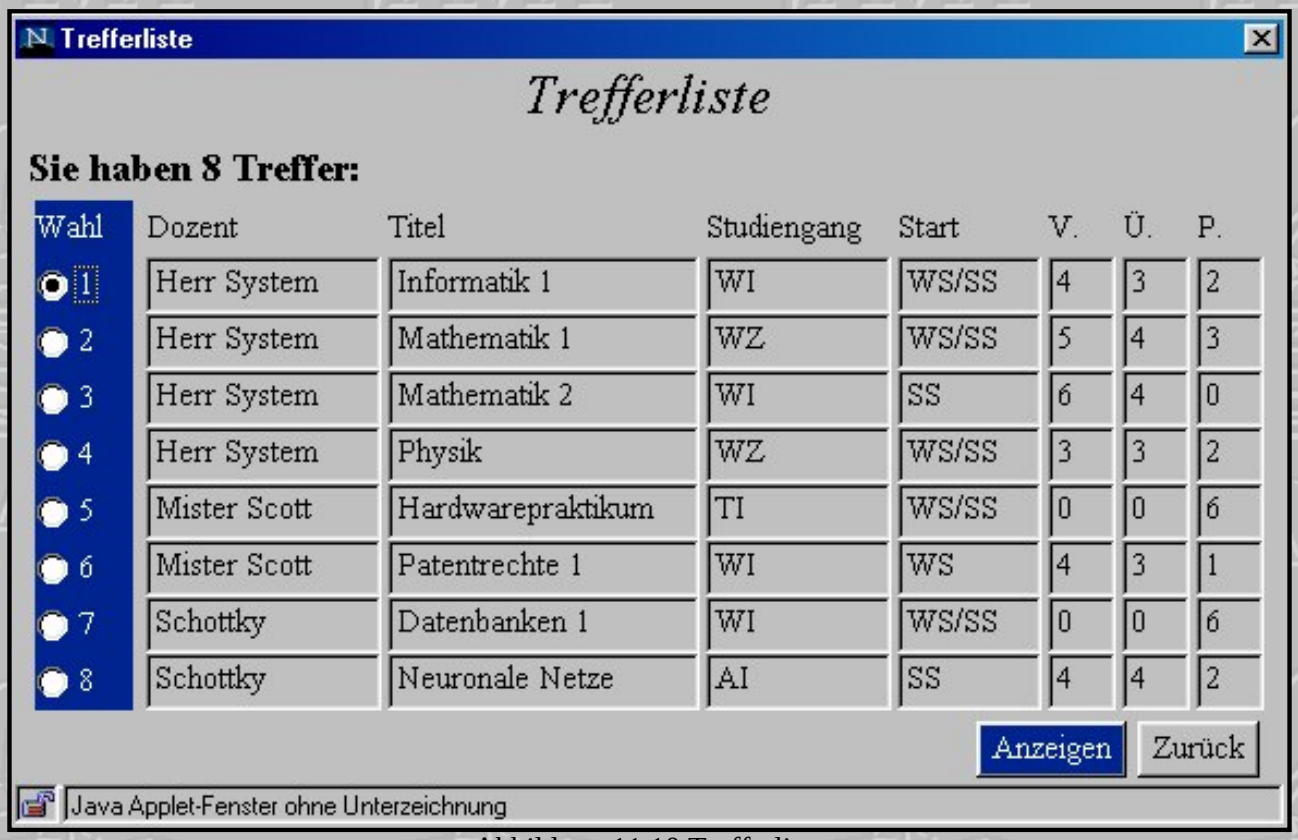


Abbildung 11.10 Trefferliste

Hier werden alle ausgelesenen Informationen bezüglich der generierten Suchanfrage angezeigt. Über die Choice-Box "Wahl" kann ein näher zu betrachtender Eintrag ausgewählt werden. Durch anschließendes Betätigen des Buttons "Anzeigen" wird eine zum Eintrag speziell gehörende "SELECT-Anweisung" generiert und das Ergebnis in ein Objekt der Klasse "infoTable" geschrieben. Diese Daten werden, ähnlich wie bei einem zu ändernden Eintrag des Dozenten, bei der Instanzierung der Klasse "Maske" (siehe Abbildung 11.6) übergeben. Diese ist multifunktionell programmiert, so daß sie einerseits zum Eintragen von Daten (siehe Dozent) und andererseits zur Informationsdarstellung genutzt werden kann.

12. Zusammenfassung

In Zukunft werden Internet und Fernsehen immer mehr zusammenwachsen, was durch Produkte der Firmen Grundig und Bertelsmann bereits heute demonstriert wird. Hieraus folgt die Unabdingbarkeit für Firmen, im Internet beispielsweise mit Produktkatalogen vertreten zu sein, die idealerweise auf schnell zu recherchierenden Datenbanken abgelegt sind. Zu dieser Art der Informationspräsentation, einem Teilaspekt des **Electronic Commerce**, im Zusammenhang mit der Programmiersprache Java, soll die vorliegende Arbeit einen **effektiven Beitrag** leisten. Einzelne Techniken, wie z.B. die direkte und indirekte Datenbankanbindung, werden hierin näher erläutert und die verschiedenen Arten der Kommunikation mittels JDBC-Treibern diskutiert. Desweiteren liegt ein Beispiel mit Quellcode vor, der zeilenweise erläutert ist, um Einsteigern in das Thema Java-Applet-Datenbankanbindung, den Start zu erleichtern. Dabei auftretenden Problemstellungen, ausgelöst durch "Sandboxes" und "Firewalls", sind in einem weiteren Kapitel eingehend beleuchtet und mit Lösungsmöglichkeiten bedacht worden.

Zu den **Hauptaufgaben** dieser Arbeit gehörte unter anderem die Entwicklung eines Hochschulinformationssystems für die FH-Gummersbach, Fachbereich Informatik. Hiermit sollen Dozenten die Möglichkeit erhalten schnell und einfach Informationen bezüglich ihrer Fächerinhalte auf einer Hochschuldatenbank abzulegen. Studenten wiederum wird diese Information über Suchmasken präsentiert. Zur Realisierung wurde ein Java-Applet verwendet, welches beiden Benutzergruppen dient, und im Internet über den Aufruf einer Web-Seite gestartet werden kann. Als Anregung für eine **weiterführende Arbeit** bietet sich an, aus dem hier benutzten zweischichtigen Modell der Datenbankanbindung, mittels Einsatz eines Servlets auf dem WWW-Server, eine dreischichtige Anwendung zu konzipieren. Diese bietet eine erhöhte Kommunikationsgeschwindigkeit und löst gleichzeitig die "Sandbox"/"Firewall"-Problematik.

Da das Internet einen globalen, weltumspannenden Markt repräsentiert und Java eine betriebssystemunabhängige Programmiersprache darstellt, wird die in dieser Arbeit angesprochene Thematik auch in Zukunft aktuell bleiben.

... und sie dreht sich doch ...

Literaturverzeichnis

Assfalg, R.; Goebels, U.; Welter, H.; Internet Datenbanken;Bonn; 1998 Addison-Wesley-Longman

Burkhardt, Th.; Lohmann, K.; Banking und Electronic Commerce im Internet; Berlin 1998; Berlin Verlag A.Spitz;

Hagen, Jan U.; Zagler, Hans; Sicherer Zahlungsverkehr im Electronic Commerce; Die Bank; April 1998

Hamilton, Graham; Cattell, Rick; JDBC ™: A Java SQL API 10.01.1997; jdbc-spec-0120.pdf; JavaSoft

Klute, Rainer; JDBC in der Praxis; Addison-Wesley; Bonn 1998

Kowalewski, Daniel; Uni Hannover [Stand: 11.6.1999]
http://java.rrzn.uni-hannover.de/jdbc/

Krüger, Guido; Java 1.1 lernen; Bonn 1997; 1. Auflage

Lemay, Laura; Cadehead, Rogers; Java 1.2 programmieren in 21 Tagen; 1998; Haar bei München; SAMS

Mandel, Andreas; Aufbau und Anwendungspotentiale von Java [Stand:11.6.1999];
http://www.pure.de/andreas/wi5/www/aufbau.html

Abbildungsverzeichnis